Document Owner: Ilya Kutsenko

Slack: #itmf

ITMF Pilot Project

DESCRIPTION

This document consists of a ITMF pilot solution description 'Phase 3 – Edge prototype' in particular. Document provides description of solution functionality, properties and usage instructions (including testing instruction).

PROJECT SUMMARY

Requirements

*For additional details see 'Contract – ITMF Pilot'

Project - Traffic flow surveillance

Goal(s)
Project goals are to provide edge machine vision solution in single on-site location for client's operation with following features:

- Recognition of vehicles and reading register plates when sensor provides suitable data
- Edge anonymization and encryption of register plate information
- Recognition of vehicles and counting traffic flows according direction

AM-X
Recon AI will implement AM-X platform as part of Client's traffic cameras and provide a count of cars passing the associated section of the road to the Client's ecosystem.

Edge AI
Solution will recognize objects of interest from sensor feed in real-time and on-site, and calculate the amount of passing cars based on this information.

Output information
Output information is sent automatically to the Client's ecosystem using an API specified by the Client.

Hardware and installation
The computational hardware specified by Recon AI will be installed by the Client based on the instructions provided by Recon AI within three weeks after Recon AI has delivered required hardware. On-site maintenance and update operations are the responsibility of the Client.

Methods and Tools Overview

Nvidia Jetson was selected as target platform.

Deepstream SDK 5.0 was selected as a framework to build a solution as it:

- Supports all Jetson devices
- Flexible and capable of running different neural networks (built with Tensorflow, Pytorch, Caffe, TLT)
- Version 5.0 introduced fully functional Python API with allows to write custom Python applications which utilize standard Python libraries

Traffic flow surveillance application has 3 major features:

- License plate detection (with anonymization)
- License plate recognition ("image to text")
- Traffic flow calculation

In addition it application has to:

- Accept RTSP stream as an input
- Output "anonymized" RTSP stream
- Save detection information (flow and license plate data) into a database/file

For these features following approaches were taken:

To detect vehicles - TrafficCamNet model was selected. This object detector model is provided by Nvidia and it can detected objects of 4 types: Car, Person, Bicycle, Traffic Sign

To track detected objects - KLT Tracker was selected, this tracker comes with Deepstream SDK and provides better accuracy and performance than NvDCF and IoU.

To detect license plates - detection model provided by Nvidia was used.

To recognize license plates (image to text) OpenALPR Python library was used.

Deepstream application has built-in modules to perform RTSP input/output.

Detection data can be saved to the file using Python/C++ code.

Evaluation metrics

Following evaluation metrics were used to estimate accuracy of used neural networks and modules:

For license plate recognition:

Accuracy: Ratio of correctly recognized license plates in a sample data to the total number of ground truth license plates. $A = (t_p) / (N_{GT})$

Precision: Fraction of correctly recognized license plates to the sum of correct recognitions and incorrect ones; $p = t_p / (t_p + f_p)$

Recall: Fraction of correctly recognized license plates to the sum of correct recognitions and not recognized license plates; $r = t_p / (t_p + f_n)$

Discrete Accuracy: Ration of correctly recognized license plates to the sum of correctly recognized license plates, not recognized license plates and incorrect recognitions: $DA = (t_p) / (t_p + f_n + f_p)$

For traffic flow calculation:

Precision: Fraction of correctly detected vehicles to the sum of correctly detected vehicles and incorrect ones; $p = t_p / (t_p + f_p)$

Recall: Fraction of correctly detected vehicles to the sum of correct detections and not detected vehicles; $r = t_p / (t_p + f_n)$

METHODS AND TOOLS

In order to build multi-feature computer vision solution a research was made to pick a best framework which would meet the following criterias:

- Be able to smoothly run on GPU, especially on Nvidia GPU
- Be cross-platform
- Support of different types of neural network models
- Support of custom modules with custom logic
- Be 'flexible' to fit multiple networks

Deepstream SDK 5.0 was selected as a framework to build a solution. Deepstream is built by Nvidia as a framework which is designed to fully utilize Nvidia GPU capabilities, it can run on any Jetson device and Nvidia GPU, it can run Tnesorflow, Keras, Pytorch and Caffee models, framework has C++ and Python API so custom modules can be added and external libraries imported, and 'pipeline' structure allows to fit multiple networks inside of one application so each network would get the metadata from previous one.

In order to implement features:

Vehicle detection:

TrafficCamNet model was picked as a base as it was already trained on significant dataset (3 mil images), it provides good accuracy (0.88) and it;s optimized to run on Jetson Nano (17.7 fps). For such common task as vehicle detection it's best pick with a selected framework.

Object Tracking:

KLT Tracker was selected as it provided best precision, recall, accuracy and number of false detection on a test datasample. In addition the flexibility of Deepstream SDK allows to switch between trackers if needed, comparison of trackers can be seen here (page 29)

Object Classification:

VehicleTypeNet model was selected to define vehicle class. The list of classes supported by model are: Coupe, Sedan, SUV, Van, Large Vehicle, Truck. Model is also provided by Nvidia.

License plate detection:

Fd_lpd - Faces and License Plates detection model was selected as license plate detector due to it's fast inference and good accuracy.

License plate recognition:

OpenALPR - following framework was selected for a pilot for multiple reasons:

1. It provides a state-of-the-art market-level quality of license plate detection of any country or region
2. It's free to use
3. It has Python library extension

Traffic Statistics Calculation

DBSCAN - following clustering algorithm was selected for Traffic Statistics Calculations because it's one of the clustering algorithms which doesn't require a number of clusters as an input, clusters are detected as the result of algorithm execution. This makes this method universal for some task like ones when you don't know how many 'exit points' for vehicles exist.

With all models and algorithms listed above the application pipeline can be listed:

1. Receive RTSP Input stream
2. TrafficCamNet vehicle detector
3. KLT Tracker
4. Object Classifier
5. Fd_lpd network for license plate detection and anonymization
6. OpenALPR for license plate recognition (runs every N frames)
7. DBSCAN for traffic statistics calculation (runs every M frames)
8. Save data to text file
9. Return anonymized RTSP output stream

Modules described above can work with default properties but general accuracy is low, so additional Python logic was added and some settings changed to reach best performance.

Project implementation time estimation:

Edge device setup - System setup and software installation - 1-2 days with an instruction, depending on OpenCV and OpenALPR compilation time.

Raw Deepstream application development (Python) - 1 week to build and debug a pipeline with pretrained networks

Deepsteam application customization - 2-3 weeks - additional logic coding, additional features implementation, networks accuracy improvement.

RESULTS

Result of the project is Deepstream 5.0 application - Python script with the set of neural network modules which implements Deepsteram pipeline and RTSP stream processing.

Source code can be found in Github repository.

Solution is running at:

5.9 - 20 FPS on Jetson Nano

X FPS on Jetson AGX Xavier

Feature metrics:

License plate recognition - scenario when some vehicles are partially visible (see Feature Sheet)

Accuracy	Precision	Recall
52.63%	0.76	0.53

License plate recognition - scenario when all the vehicles are fully visible

Accuracy	Precision	Recall
78.95%	0.68	0.79

Traffic Statistics (see Feature Sheet)

Precision	Recall

0.87	1.00

TESTING INSTRUCTIONS

Application description

Project Solution can be found under:

/opt/nvidia/deepstream/deepstream-5.0/sources/python/apps/deepstream-amtraffic

It consists of several python script files, txt-settings files and folders with test data and support scripts.

Main module of the application is located in 'deepstream_amtraffic_msq.py' file, it's settings located in 'deepstream_config.py'.

Application run syntax looks like:

python3 deepstream_amtraffic_msq.py [file:///<absolute path to a file>] [rtsp:///<address of RTSP stream>]

Examples:

Python3 deepstream_amtraffic_msq.py
file:///opt/nvidia/deepstream/deepstream-5.0/samples/streams/18_LPs_1280_Trim.mp4

python3 deepstream_amtraffic_msq.py rtsp://192.168.100.2:8554

In the current testing scenario it is advised to test on videos from 'streams' folder (/opt/nvidia/deepstream/deepstream-5.0/samples/streams) or video streams from production cameras.

Features testing

Features below can be tested only after you execute deepstream_amtraffic_msq.py with 18_LPs_1280_Trim.mp4 file. In other cases, numbers would differ from those provided in the accuracy section or scripts would output errors.

Table below provides feature description and ways to test them:

Feature	Testing Scenario
RTSP feed input	Run application with rtsp-stream input parameter
RTSP feed output	Use VLC player to read from rtsp stream with device IP address, 8555 port and ds-test app Example: rtsp://10.8.0.34:**8555**/ds-test

License plate recognition Traffic Statistics calculation Save data (license plate detections and traffic statistics) to a file	**License plate recognition:** ● License plate data is saved into '/opt/nvidia/deepstream/deepstream-5.0/sources/python/apps/deepstream-amtraffic/file_licensePlatesDetections.txt'. ● Make sure that deepstream_config.py file has following values: ● SAVE_LICENSE_PLATES_TO_FILE = True ● LICENSE_PLATES_FILENAME = 'file_licensePlatesDetections.txt' ● File contains a list of recognized license plates ● To check the accuracy run: python3 test/test_lpcomparison.py --gtLP=data/laneALL.txt --recLP=file_licensePlatesDetections.txt ● Script will compare algorithm execution results with laneALL.txt file containing all the readable license plates from 18_LPs_1280_Trim.mp4 video. **Traffic Statistics:** ● Traffic Statistics data is saved into '/opt/nvidia/deepstream/deepstream-5.0/sources/python/apps/deepstream-amtraffic/file_statisticsDetections.txt'. ● Make sure that deepstream_config.py file has following values: ● SAVE_STATISTICS_TO_FILE = True ● STATISTICS_FILENAME = 'file_statisticsDetections.txt' ● To check the accuracy run: ● python3 test/test_trafficstatistics.py --stats=file_statisticsDetections.txt --gtVehicles=102 ● Script will compare algorithm execution with actual number of vehicles on the video (102). ● Actual file contents is described below: ○ First line contains time period of vehicles calculation ○ Each following line contains statistics from each 'Exit Point'. Line structure looks like: ■ PointID:CoordinateX:CoordinateY:Number of vehicles detected:Vehicles statistics (Vehicle Type: Number of Objects)
License plate anonymization	Make sure that deepstream_config.py file has following value: ANONYMIZE_LICENSE_PLATES = True While application is running view it's RTSP stream, see that license plates are hidden with a rectangle

Test Environments

There are 3 test environments available:

Env name	Connection instructions

Xavier	**Linux:**
	$ ssh -A -t -L 8554:localhost:8554 -i ~/AI_private_key.pem developer@62.75.155.217 ssh -p 60022 nvidia@localhost
	Windows(putty):
	$ putty.exe -v -ssh -2 -C -L 8554:localhost:8554 developer@62.75.155.217 -i C:\AI_private_key.ppk
	$ ssh -p 60022 -A -t -L 8554:localhost:8554 nvidia@localhost

Nano	**Linux:**
	$ ssh -L 9999:localhost:44422 developer@62.75.155.217
	$ ssh -L localhost:10080:192.168.2.251:80 -p 9999 nvidia@localhost
	Windows (putty):
	$ putty.exe -v -ssh -2 -C -L 9999:localhost:44422 developer@62.75.155.217 -i C:\AI_private_key.ppk
	$ putty.exe -v -ssh -2 -C -L 10080:192.168.2.251:80 nvidia@localhost -P 9999 -i C:\AI_private_key.ppk

Perm Nano	Application testing instruction on Perm Nano device are available here