Solution Approach in Integration of AI Engine into AGL

Japan Technical Jamboree 64

March 2nd, 2018

NTT DATA MSE Corporation
Hiroto Imamura
Tomonari Okuno

NTT DATA MSE Corporation
### Who are we?

#### NTT DATA MSE Corporation

<table>
<thead>
<tr>
<th>Name</th>
<th>Hiroto Imamura</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>Manager</td>
</tr>
</tbody>
</table>
| Carrier     | - Leader of OSS Collaboration related activities in NTT DATA MSE  
              - Linux-based embedded devices  
              - Architecture design / System debugging / Performance optimization / Security |

<table>
<thead>
<tr>
<th>Name</th>
<th>Tomonari Okuno</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>Deputy Manager</td>
</tr>
</tbody>
</table>
| Carrier     | - Lead Architect of R&D projects in NTT DATA MSE  
              - Linux-based embedded devices  
              - Performance optimization / OSS Licenses |
What is AGL?

AUTOMOTIVE GRADE LINUX

About

Automotive Grade Linux (AGL) is a collaborative open source project that is bringing together automakers, suppliers and technology companies to build a Linux-based, open software platform for automotive applications that can serve as the de facto industry standard. Adopting a shared platform across the industry reduces fragmentation and allows automakers and suppliers to reuse the same code base, leading to rapid innovation and faster time-to-market for new products.

As a “code first” organization, AGL’s goals are to:

- Build a single platform for the entire industry
- Develop 70-80% of the starting point for a production project
- Reduce fragmentation by combining the best of open source
- Develop an ecosystem of developers, suppliers, expertise all using a single platform

Although initially focused on infotainment, AGL is the only organization planning to address all software in the vehicle: infotainment, instrument cluster, heads-up-display (HUD), telematics/ connected car, advanced driver assistance systems (ADAS), functional safety and autonomous driving.

Automotive Grade Linux is a Project at The Linux Foundation.

Ref.
- https://www.automotivelinux.org/about
- https://www.automotivelinux.org
Why Edge AI?

So far

<table>
<thead>
<tr>
<th>Edge</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="car.png" alt="Car" /></td>
<td><img src="data.png" alt="Data Volume" /></td>
</tr>
<tr>
<td><img src="edge.png" alt="Edge AI Engine" /></td>
<td><img src="cloud.png" alt="Cloud AI Engine" /></td>
</tr>
</tbody>
</table>

Data Volume: LARGE
Response: SLOW

Near future

<table>
<thead>
<tr>
<th>Edge</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="car.png" alt="Car" /> + <img src="processor.png" alt="Processor" /></td>
<td><img src="data.png" alt="Data Volume" /></td>
</tr>
<tr>
<td><img src="edge.png" alt="Edge AI Engine" /> + <img src="cloud.png" alt="Cloud AI Engine" /></td>
<td></td>
</tr>
</tbody>
</table>

Data Volume: SMALL
Response: FAST
Edge AI is spotlighted

<table>
<thead>
<tr>
<th>Date</th>
<th>Solution</th>
<th>Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apr. 2017</td>
<td>e-AI</td>
<td>Renesas Electronics</td>
</tr>
<tr>
<td>Jun. 2017</td>
<td><strong>Neural Network Libraries</strong></td>
<td>Sony</td>
</tr>
<tr>
<td>Jun. 2017</td>
<td>Embedded Learning Library</td>
<td>Microsoft</td>
</tr>
<tr>
<td>Jul. 2017</td>
<td>TensorFlow Lite</td>
<td>Google</td>
</tr>
<tr>
<td>Jul. 2017</td>
<td>revision</td>
<td>Xilinx</td>
</tr>
<tr>
<td>Sep. 2017</td>
<td>Kirin 970</td>
<td>HUAWEI</td>
</tr>
<tr>
<td>Jan. 2018</td>
<td>Deep Learning Accelerator Card</td>
<td>PFU</td>
</tr>
</tbody>
</table>

**Our motivation**

1. OSS / Rich development environment  **Easy to try**
2. Few people addressed **Good chance to appeal our technical capabilities**
3. AGL Member  **Make collaboration**
Collaboration

supported by Sony members
Feedback at CES 2018

Valuable opinions, Valuable discussions
Technical Explanation
Our team have started AI related activities from October 2017

- There are AI related news almost everyday
- Our interest:
  - How is the performance of AI on edge devices?
  - What do we need to learn in order to realize AI on edge devices?
- First Step: Let’s use an AI engine on edge devices
- Implemented a demo system “Handwritten Digit Recognition App”

In this presentation

- Overview of Machine Learning on edge devices
- How we implemented the “Handwritten Digit Recognition App” on AGL
What is AI?
What are the relations of these terms?

**Artificial Intelligence**
Computer programs to simulate functions of human's brain such as "cognition", "judgement", etc.

**Machine Learning**
Technology to let computers learn rules and knowledge from vast amount of various data such as values, texts, pictures and voice.

**Deep Learning**
One of the method to perform machine learning by using neural network to realize high level of abstraction by extracting information from multiple layers one by one.
Types of AI

- **Rule-Based**
  - Decision rules are described by human.

- **Machine Learning**
  - Decision rules are derived from data analysis.

  - **Supervised Learning**
    - Systems which learn from given labeled training data (=Answers are given from a teacher).
    - (e.g. Voice/Image Recognition, Stock Price Prediction)

  - **Unsupervised Learning**
    - Systems which learn from given unlabeled training data (=Answers are not given).
    - (e.g. Automatic data classification, Anomaly detection)

  - **Reinforcement Learning**
    - Systems which learn in an environment where only the final result evaluations are given.
    - (e.g. AI-Shogi, AI-Go)
Machine Learning (Supervised learning)

Training

Input (Training Data)

Apple

Machine Learning System

Training

Trained Model

Unknown Data

Output

Apple 95%
Tomato 4%
Cherry 1%

Machine Learning System

Inference
Demo System:
Handwritten Digit Recognition App
System Overview

- Recognizes handwritten digits from an image captured with an USB Camera
- Deployed a pre-trained model to an edge device
Application GUI

USB Camera image

Input image to AI Engine
(28 x 28 pixels)

Prediction result from AI Engine
(Probability of each digit)

Recognized Area
(100 x 100 pixels)

Cropped image
(100 x 100 pixels)

Recognized digit

Time: 8.67ms
Neural Network Libraries (NNabla)

Neural Network Libraries
https://nnabla.org

• Deep learning framework.
• Intended to be used for research, development and production.
• Aim to have it running everywhere. Deployable to embedded devices.
• Apache License 2.0

Neural Network Console
https://dl.sony.com

• GUI tool for designing neural networks intuitively.
• Many useful functions to support research and development.
• Trained model can be embedded by using Neural Network Libraries.
Details of the Implementation

**Step 1**: Deep Learning on PC

**Step 2**: Building / Installation

**Step 3**: Enabling USB webcam

**Step 4**: Installation of the OpenCV

**Step 5**: App Implementation

Step 3 and 4 are not needed if AGL version is 5.0(EE)
Step 1: Deep Learning on PC

Used one of the examples available at the github
→ mnist-collection/classification.py

- Downloads and uses MNIST Dataset as training data
  (60,000 samples of handwritten digits image and label)

- Uses Convolutional Neural Network
  - Input: 28 x 28 pixels grayscale image
  - Output: Prediction of 10-way classification

<table>
<thead>
<tr>
<th>Input Value</th>
<th>Output Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image 28 x 28</td>
<td>Probability of each digit</td>
</tr>
<tr>
<td>Coordinate</td>
<td>Y₀</td>
</tr>
<tr>
<td>(0, 0) X₀</td>
<td>Y₁</td>
</tr>
<tr>
<td>(1, 0) X₁</td>
<td>Y₂</td>
</tr>
<tr>
<td>(2, 0) X₂</td>
<td>⋮</td>
</tr>
<tr>
<td>(27, 27) X₇₈₃</td>
<td>Y₉</td>
</tr>
</tbody>
</table>

Digit Probability
(10)
Step 1: Deep Learning on PC

Used one of the examples available at the github
→ mnist-collection/classification.py

Install the NNabla on a PC

$ pip install nnabla

Obtain the “mnist-collection”

$ cd ~/work/sony/
$ git clone https://github.com/sony/nnabla-examples

Start training

$ cd nnabla-examples/mnist-collection
$ python classification.py
2018-01-30 19:42:37,932 [nnabla][INFO]: Initializing CPU extension...
.....
2018-01-30 19:42:39,343 [nnabla][INFO]: iter=9 {Training loss}=2.30425691605
2018-01-30 19:42:39,343 [nnabla][INFO]: iter=9 {Training error}=0.8375
.....

Obtain examples

Start

Complete
Step 2: Building / Installation

Build NNabla for R-Car H3(ARMv8(64bit))

Install cross SDK on a PC (AGL R-Car ARMv8 toolchain)

$ wget https://download.automotivelinux.org/AGL/release/eel/5.0.0/m3ulcb-nogfx/deploy/sdk/poky-agl-glibc-x86_64-agl-image-ivi-crosssdk-aarch64-toolchain-5.0.0.sh
$ chmod a+x poky-agl-glibc-x86_64-agl-image-ivi-crosssdk-aarch64-toolchain-5.0.0.sh
$ ./poky-agl-glibc-x86_64-agl-image-ivi-crosssdk-aarch64-toolchain-5.0.0.sh

See also:
http://docs.automotivelinux.org/docs/getting_started/en/dev/reference/source-code.html

Build NNabla for R-Car

$ git clone https://github.com/sony/nnabla
$ source /opt/poky-agl/5.0.0/environment-setup-aarch64-agl-linux
$ mkdir -p nnabla/build && cd nnabla/build
$ cmake .. -DBUILD_CPP_UTILS=ON -DBUILD_PYTHON_PACKAGE=OFF
$ make
$ ls -l lib/
-rwxrwxr-x 1 nttdmse nttdmse 191170296 12月 20 06:21 libnnabla.so
-rwxrwxr-x 1 nttdmse nttdmse 25392344 12月 20 06:21 libnnabla_utils.so

See also:
Step 2: Building / Installation

Install the built shared libraries of the NNabla and the pre-trained model to the target filesystem.

Shared libraries of the NNabla

```bash
$ export SDCARD=/tmp/agl
$ sudo mount /dev/sdc1 $SDCARD
$ sudo cp libnnabla.so $SDCARD/usr/lib/
$ sudo cp libnnabla_utils.so $SDCARD/usr/lib/
```

Pre-trained model

```bash
$ cd ~/work/sony/nnabla/examples/cpp/mnist_runtime
$ NNABLA_EXAMPLES_ROOT=~/work/sony/nnabla-examples python save_nnp_classification.py
$ ls -l lenet_010000.nnp
-rw-rw-r-- 1 nttdmse nttdmse 86920 1月 29 19:16 lenet_010000.nnp
$ sudo cp lenet_010000.nnp $SDCARD/home/data/
$ sync
$ sudo umount $SDCARD
```

Convert to NNabla file format (NNP)
Step 3: Enabling USB webcam

Enable the USB Video Class in order to use USB webcam on R-Car H3.

Enable the UVC (USB Video class) of kernel config

```
$ export MACHINE=h3ulcb
$ export MACHINE
$ source meta-agl/scripts/aglsetup.sh -m $MACHINE -b build-renesas-kernel agl-devel agl-demo agl-netboot agl-appfw-smack agl-localdev

$ bitbake virtual/kernel
$ bitbake linux-renesas -c menuconfig

Device Drivers --> Multimedia support ---> [*] Media USB Adapters
[*] USB Video Class (UVC)
[*] UVC input events device support (NEW)

$ bitbake virtual/kernel

Setup environment
Build Kernel
Change kernel configuration
Rebuild kernel

Update kernel image

$ sudo cp tmp/deploy/images/m3ulcb/Image--4.9.0+git0+098ccf1c9b-r1-m3ulcb-20171116044641.bin SDCARD/boot/Image-4.9.0-yocto-standard
```
Step 4: Installation of the OpenCV

Install the OpenCV to handle camera images
- OpenCV = Open Source Computer Vision Library
- Used for obtaining image from the webcam and pre-process the acquired images before passing them to the NNabla.

Build the OpenCV

```bash
$ export MACHINE=h3ulcb
$ source meta-agl/scripts/aglsetup.sh -m $MACHINE -b build-opencv agl-devel agl-demo agl-netboot agl-appfw-smack agl-localde
$ bitbake opencv
```

Install built binaries to target filesystem

```bash
$ sudo cp -a ./tmp/work/aarch64-agl-linux/opencv/3.2+gitAUTOINC+70bbf17b13-r0/image/* $SDCARD/
$ sudo cp -a ./tmp/work/aarch64-agl-linux/v4l-utils/1.12.3-r0/image/* $SDCARD/
$ sudo cp -a ./tmp/work/aarch64-agl-linux/libwebp/0.6.0-r0/image/* $SDCARD/
$ sync; sudo umount $SDCARD
```
Step 5: App Implementation

App main logic

1. Capture image
2. Crop image (100 x 100 pix)
3. Convert to gray scale image
4. Convert to binary image
5. Resize image (100 x 100 → 28 x 28 pix)
6. Perform inference
7. Update HMI
Step5: App Implementation(Source Code)

```
// (0) Open Device
cv::VideoCapture cap("/dev/video22", cv::CAP_V4L2);    // videoN: Set according to the environment. This is R-Car H3 with AGL5.0.0.

// (1) Capture image
cv::Mat frame;
cap >> frame;

// Pre-process image
// (2) Crop video image [100x100pix]
cv::Rect rect(GET_VIEW_SIZE_LEFT, GET_VIEW_SIZE_TOP, GET_VIEW_SIZE_WIDTH, GET_VIEW_SIZE_HEIGHT);
cv::Mat rectImg(frame, rect);

// (3) Convert to gray scale image
cv::Mat grayImg;
cv::cvtColor(rectImg, grayImg, CV_RGB2GRAY);

// (4) Convert to binary image (Invert | Threshold = 127)
cv::Mat binImg;
cv::threshold(grayImg, binImg, 127, 255, cv::THRESH_BINARY_INV);

// (5) Resize image [100x100pix -> 28x28pix]
cv::Mat resizeImg;
cv::resize(binImg, resizeImg, cv::Size(), PGM_WIDTH/grayImg.cols,PGM_HEIGHT/grayImg.rows);

// Add pgm header
pgmformat((char *)"cap.pgm",resizeImg);

// (6) Perform inference
int prediction = 0;
float score[10] = {};
double elapsed = 0;
const std::string npn_filepath[] = "/home/data/lenet_010000.nnp";
const std::string pgmImageName[] = "cap.pgm";
prediction = EdgeAIINNblaMnistRuntime(pgmImageName, npn_filepath, score, &elapsed);  // nnabla wrapper API
```
Experiment Result
Recognition Result (First Try)

Not possible to recognize digits for the following patterns

**Recognized**

- 

**Not Recognized**

- Shifted
- Rotated
- Scaled
Recognition Result (After improvement)

- **Used Data Augmentation to improve pre-trained model**
  - Randomly alters the MNIST digit images when perform training.
    - scaling, rotation, aspect ratio, distorting, brightness, contrast, add noise
  - Updated the pre-trained model on the device.
  - Recognition rate has been improved.
    - Tested with 1,000 cases. (Handwritten digits by our colleague)
    - Counted as “Recognized” if the probability is larger than 50%

- **Improvement of predictions are possible by updating pre-trained models**
  - Delivering up-to-date pre-trained models to devices can provide more accurate prediction results for users.

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>86.0%</td>
<td>94.8%</td>
</tr>
<tr>
<td>(860/1,000)</td>
<td>(948/1,000)</td>
</tr>
</tbody>
</table>
Performance

Processing Time

8 ms
Image capture → Pre-processing

2 ms
Inference

*Measured on R-Car H3

Resource Usage

<table>
<thead>
<tr>
<th>#</th>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CPU usage rate</td>
<td>55 %</td>
</tr>
<tr>
<td>2</td>
<td>Memory [RSS] usage</td>
<td>80 MB</td>
</tr>
<tr>
<td></td>
<td>Size of the Pre-trained Model</td>
<td>85 KB</td>
</tr>
</tbody>
</table>

*Measured on R-Car H3
Conclusion
Summary

- **Explained how we implemented the Handwritten Digit Recognition App**
  - Performed deep learning on a PC
  - Deployed pre-trained model to R-Car H3 board
  - Variation of training data set affects inference results
  - Updating of pre-trained model can improve inference results

- **It wasn’t difficult than expected to take the first step toward using AI engine on AGL**
  - Implemented the app in 3 weeks
  - Please try it!
Next Step

1. Use other types of data for deep learning
   - Create own neural network architecture
   - Use information other than image available in vehicle

2. Performance improvement
   - More complicated neural network architecture causes performance issues
   - Utilize GPU to accelerate calculations

Future use-cases

- Use various information of embedded devices
- Detect driver's drowsiness
- Improvement of autonomous driving technology
Drowsiness Detection

Drowsiness Detection Algorithm Based on Heart Rate Variability

Trial on-going: Replace the algorithm with AI

Prepare vital data
- Electrocardiac information

Create data set
- Data set
  - Training data
  - Validation data

Training
- Trained Model

NTT DATA MSE Corporation
Questions?
Thank you very much!!

Smart Life Community®