Embedded machine vision with the OpenMV Cam H7

Kwabena W. Agyeman
President
OpenMV, LLC
• What is the OpenMV Cam?
  • Arm® Cortex®-M7 Microcontroller with a camera
  • Open-source hardware/software product
  • Programmable in Python (or C)
    • Powered by MicroPython

• Specs:
  • 400 MHz Cortex-M7 Processor (~2000 coremark)
  • 1 MB of RAM (2 MB of flash)

#ArmDevDay19
Why does the OpenMV Cam exist?

- Provides a quick way to add machine vision to a robotic system.
  - Continuation of the CMUcam cameras.
- No complex software installation.
  - Just install our IDE and you’re ready to go.
- Get up and running in literally minutes.
- All in one platform now. You can do the processing and I/O actuation on it at the same time!
You program using OpenMV IDE!

- Cross Platform Python Editor
- Live Frame Buffer
- Serial Terminal
- Histogram Display

Based on QtCreator

#ArmDevDay19
What do people make with the OpenMV Cam?
Color tracking demo
April tags tracking demo
Thermal image demo
Arm CMSIS-NN

Neural Network Application Code

NNFunctions

Convolution  Pooling
Fully connected  Activations

NNSupportFunctions

Data type conversion  Activation tables
CNN output to code

1. Get training data
2. Train CNN using Caffe
3. Convert to CMSIS-NN C Code

https://github.com/ARM-software/ML-examples/tree/master/cmsisnn-cifar10

```c
void run_nn()
{
    q7_t* buffer1 = scratch_buffer;
    q7_t* buffer2 = buffer1 + 22768;
    arm_convolv_hwc_q7_RGB(input_data, CONV1_IN_DIM, CONV1_IN_CH, conv1_wt, CONV1_OUT_CH, CONV1_KER_DIM, CONV1_PAD, CONV1_STRIDE, conv1_bias,
                           arm_maxpool_q7_hwc(buffer1, POOL1_IN_DIM, POOL1_IN_CH, POOL1_KER_DIM, POOL1_PAD, POOL1_STRIDE, POOL1_OUT_DIM, col_buffer, buffer2);
    arm_convolv_hwc_q7(buffer2, RELU1_OUT_DIM, RELU1_OUT_DIM, RELU1_OUT_CH);
    arm_convolv_hwc_q7_fast(buffer2, CONV2_IN_DIM, CONV2_IN_CH, conv2_wt, CONV2_OUT_CH, CONV2_KER_DIM, CONV2_PAD, CONV2_STRIDE, conv2_bias,
                             arm_relu_q7(buffer1, RELU2_OUT_DIM, RELU2_OUT_DIM, RELU2_OUT_CH);
                             arm_maxpool_q7_hwc(buffer1, POOL2_IN_DIM, POOL2_IN_CH, POOL2_KER_DIM, POOL2_PAD, POOL2_STRIDE, POOL2_OUT_DIM, col_buffer, buffer2);
                             arm_convolv_hwc_q7_fast(buffer2, CONV3_IN_DIM, CONV3_IN_CH, conv3_wt, CONV3_OUT_CH, CONV3_KER_DIM, CONV3_PAD, CONV3_STRIDE, conv3_bias,
                             arm_relu_q7(buffer1, RELU3_OUT_DIM, RELU3_OUT_DIM, RELU3_OUT_CH);
                             arm_maxpool_q7_hwc(buffer1, POOL3_IN_DIM, POOL3_IN_CH, POOL3_KER_DIM, POOL3_PAD, POOL3_STRIDE, POOL3_OUT_DIM, col_buffer, buffer2);
                             arm_convolv_hwc_q7_fast(buffer2, CONV4_IN_DIM, CONV4_IN_CH, conv4_wt, CONV4_OUT_CH, CONV4_KER_DIM, CONV4_PAD, CONV4_STRIDE, conv4_bias,
                             arm_relu_q7(buffer1, RELU4_OUT_DIM, RELU4_OUT_DIM, RELU4_OUT_CH);
                             arm_maxpool_q7_hwc(buffer1, POOL4_IN_DIM, POOL4_IN_CH, POOL4_KER_DIM, POOL4_PAD, POOL4_STRIDE, POOL4_OUT_DIM, col_buffer, buffer2);
                             arm_convolv_hwc_q7_fast(buffer2, CONV5_IN_DIM, CONV5_IN_CH, conv5_wt, CONV5_OUT_CH, CONV5_KER_DIM, CONV5_PAD, CONV5_STRIDE, conv5_bias,
                             arm_relu_q7(buffer1, RELU5_OUT_DIM, RELU5_OUT_DIM, RELU5_OUT_CH);
                             arm_maxpool_q7_hwc(buffer1, POOL5_IN_DIM, POOL5_IN_CH, POOL5_KER_DIM, POOL5_PAD, POOL5_STRIDE, POOL5_OUT_DIM, col_buffer, buffer2);
                             arm_fully_connected_q7_out(buffer2, iout_wt, I1_OUT_DIM, I1_OUT_DIM, I1_BIAS_LSHIFT, I1_OUT_RSHIFT, I1_bias, output_data, (q15_t*)col
```
Re-loadable CNN architecture

1. Get training data
2. Train CNN using Caffe
3. Convert to CMSIS-NN Loadable Network
4. Run on the OpenMV Cam

https://github.com/openmv/openmv/tree/master/ml/cmsisnn
TensorFlow on the OpenMV Cam?

- Whenever TensorFlow is supported on M4/M7 processors we’ll have TensorFlow too!
  - We’ve got a 100 Mb/s SD card interface to load models
  - We’ve got 1MB of RAM and 2MB of flash running at 400 MHz w/ a 64-bit bus (3.2 GB/s)
  - We’ve got a Cortex-M7 processor running at 400 MHz programmable in C/C++
    - The processor has the same performance as the RaspberryPi 0 ARM 9 running at 1 GHz
  - We’re ready!
Deep learning on OpenMV Cam H7

#ArmDevDay19

Copyright © 2019 Arm, All rights reserved.
Questions?
How to train a network
DEEP LEARNING!!!

WHY U NO WORK???
Remember this:

1. Deep learning is about data – not about networks

2. The quality of your training data determines your network performance!

3. Even a small CNN can do amazing things – they are extremely powerful
Small CNNs can do useful things:

- 64x64 Grayscale Pixel Input
- 3x3x32 Conv (50% dropout) + RELU + 2x2 Pool
- 3x3x32 Conv (25% dropout) + RELU + 2x2 Pool
- 3x3x32 Conv (30% dropout) + RELU + 2x2 Pool
- Fully Connected

• 23 KB CNN
• 2 Class Outputs

- Easily reaches 99% accuracy on 2 class (64x64 pixel) identification problems.
- Each new class outputs is 2.5KB of extra weights.
Good data set

- **Pros:**
  - Images are all the same size
  - Prevents re-scaling problems when training
  - Subject is in the center of the field
  - Only can learn from the character
Good data set

• **Pros:**
  • Images are all the same size
    • Prevents re-scaling problems when training
  • Subject is in the center of the field
  • Only can learn from the character

• **Cons:**
  • The CNN will assume the letter must be black and the background white.
    • Need to threshold input images.
  • The CNN will assume there’s nothing around each letter
    • I.e. might not handle close text.
Bad data set

- Images are different sizes resulting in scaling issues when training.
- Lots of objects in the image.
  - Not clear people are the thing you want the net to learn.
- Network may score high on test and training but not work at all in the real world.
Better dataset

- Images are all the same size at the res of the net.
  - Person is about the same size and in the same position in each image.
  - Rescaled images have no artifacts in them.

- The CNN is forced to learn what a person kinda looks like as they are the main salient feature.
Key takeaways:

1. Small CNNs can get really good training, test, and validation numbers on easy classification problems (like 99% accuracy)

   - Re-using our SMILE net arch for most classification problems is sufficient – no need to explore arch types

<table>
<thead>
<tr>
<th>Layer Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>64x64 Grayscale/RGB565 Pixel Input</td>
</tr>
<tr>
<td>3x3x32 Conv (50% dropout)</td>
</tr>
<tr>
<td>RELU + 2x2 Pool</td>
</tr>
<tr>
<td>3x3x32 Conv (25% dropout)</td>
</tr>
<tr>
<td>RELU + 2x2 Pool</td>
</tr>
<tr>
<td>3x3x32 Conv (30% dropout)</td>
</tr>
<tr>
<td>RELU + 2x2 Pool</td>
</tr>
<tr>
<td>Fully Connected</td>
</tr>
</tbody>
</table>
Key takeaways:

2. Focus on assembling an excellent dataset for your problem – this is most of the job.
   - And keep in mind that your CNN assumes the world will be what it learned.