Using Arm Development Tools to Enable Machine-Learning on Device (blog)

Odin Shen | Staff FAE | Arm

GEC
May 2018

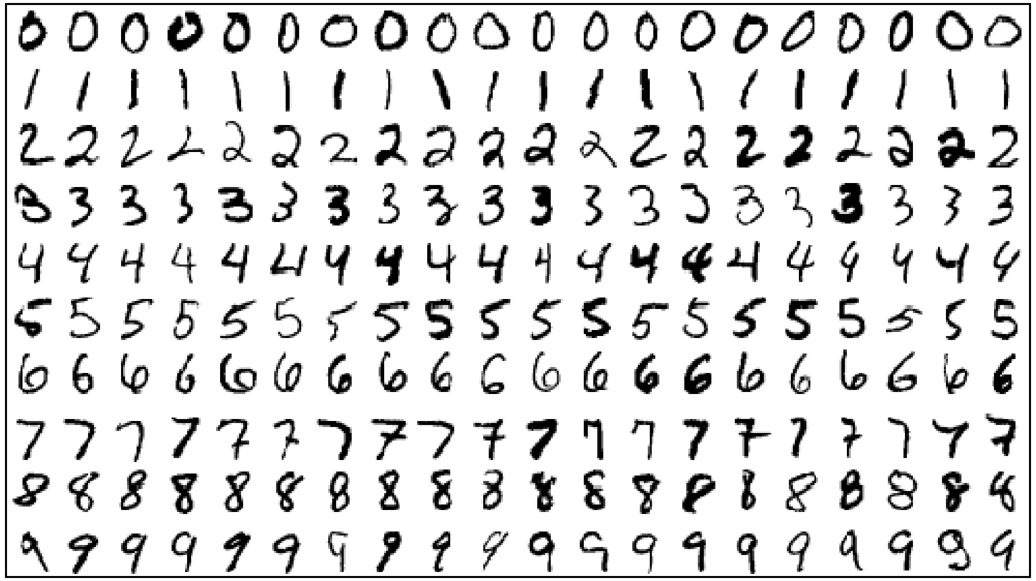
NN on Arm?

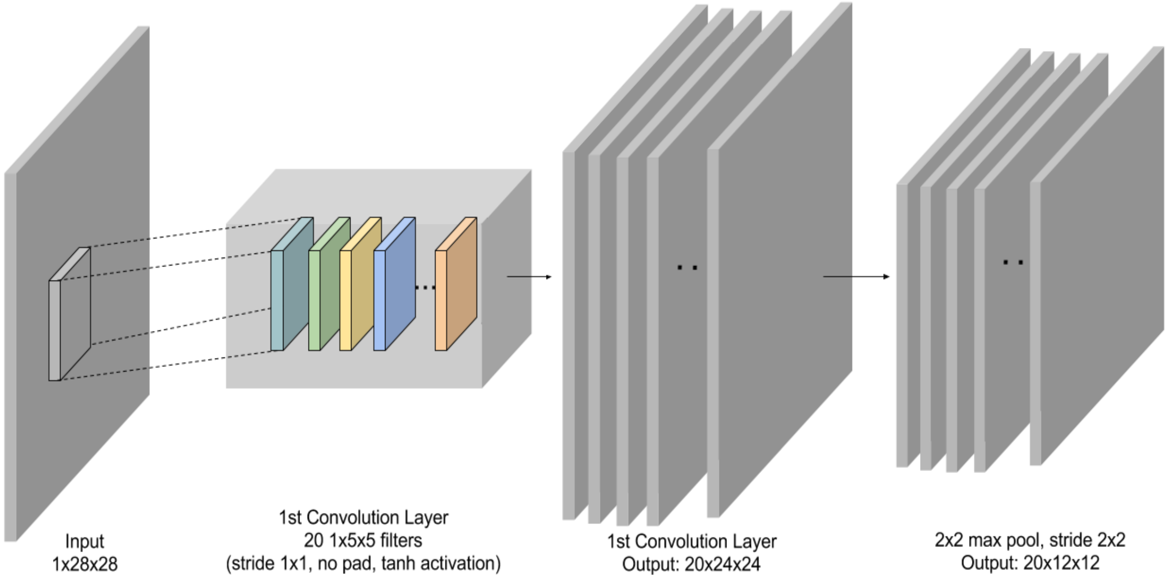
- Deploy NN inference on Cortex v8.2 platform before real hardware?
- How to select solution?

Accuracy
Performance
Debug and Analysis
Availability
Capacity
Cost
Flexibility

Use MNIST as Example

- The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples.
- It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

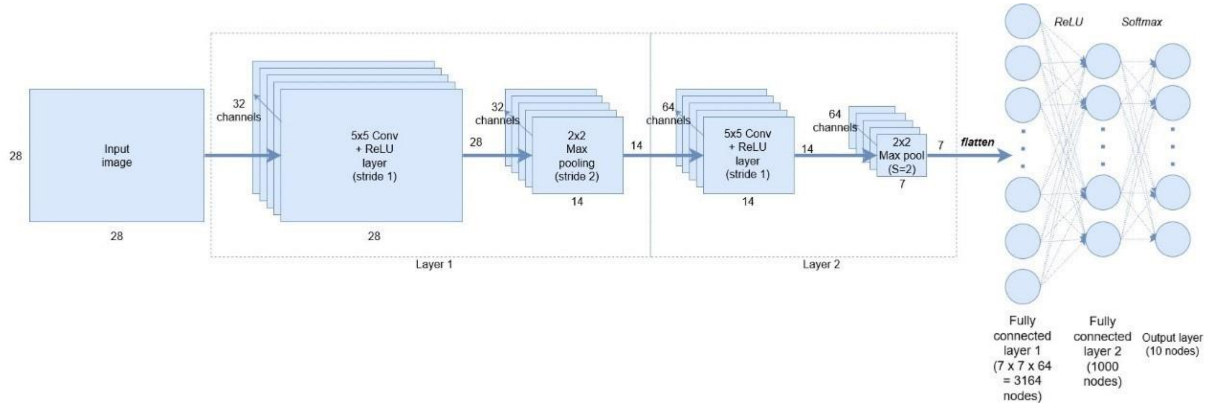


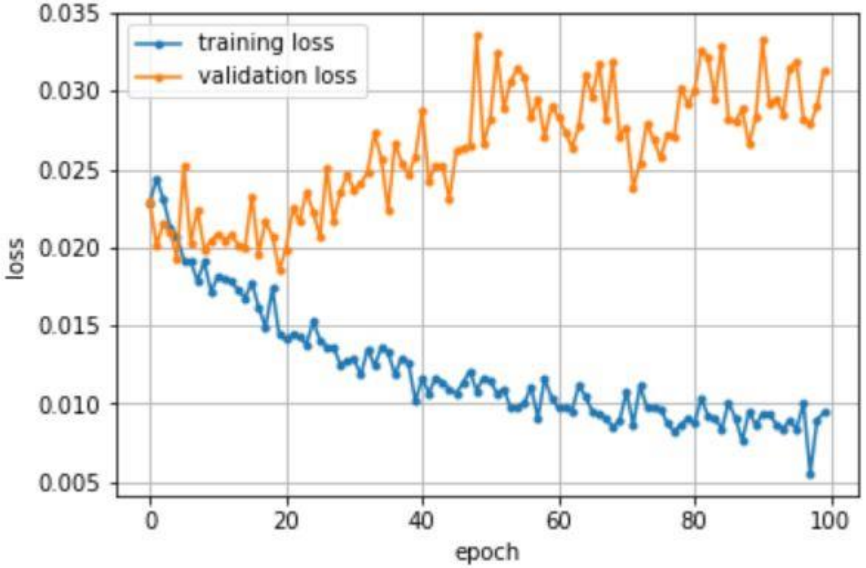


http://yann.lecun.com/exdb/mnist/
https://www.tensorflow.org/get_started/mnist/beginners

Use MNIST as Example

- Training on Tensor/Keras





In [7]:
```
model.summary()
```

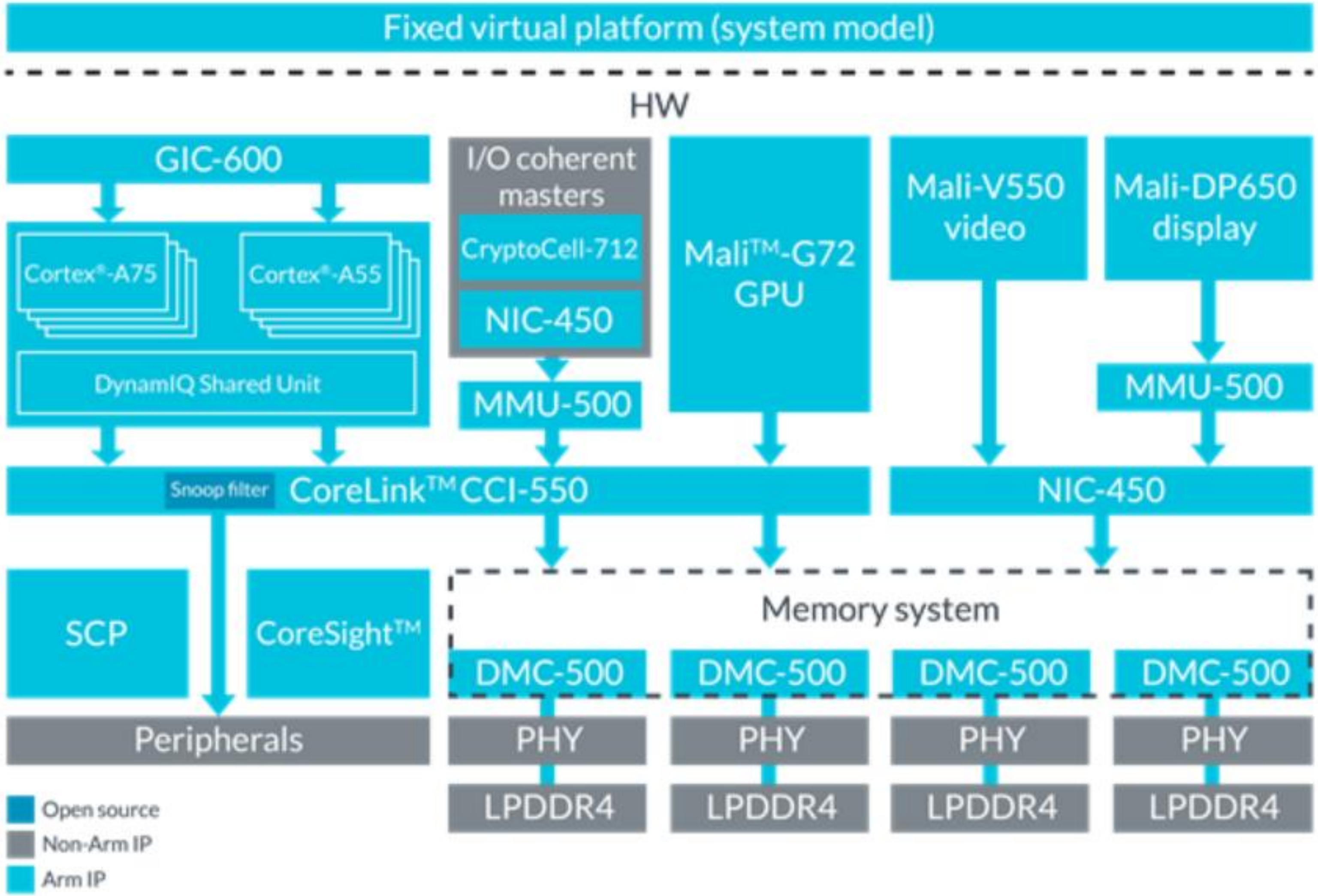
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 16)        416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 32)          12832
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 32)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 4, 4, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               65664
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 80,202
Trainable params: 80,202
Non-trainable params: 0
_________________________________________________________________
```


arm

Arm Fixed Virtual Platform (FVP)

arm

System Guidance for Mobile

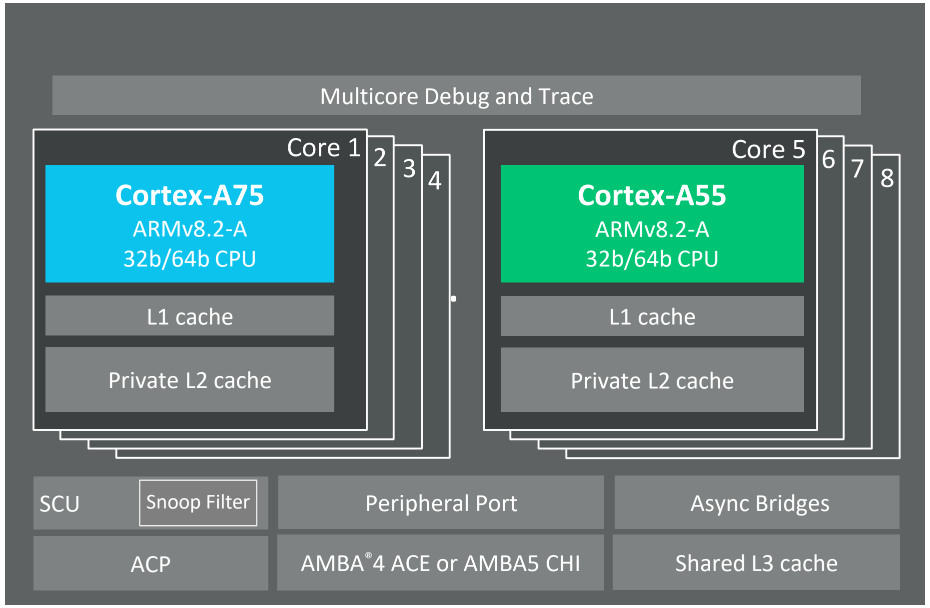


Early Deploy on Arm

- Purpose:
  - Prototyping on Arm platform
  - Early exploration NN engine design
  - SW framework profiling
- Script
  - Load image
  - Load parameter
- Automation
- Profiling

Debug
Trace
Streamline

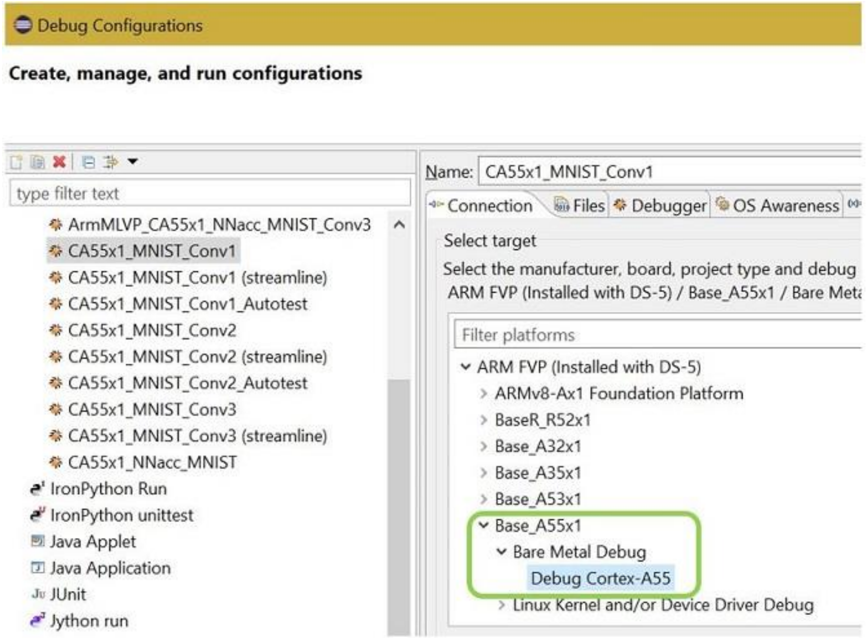
Trace and Debug



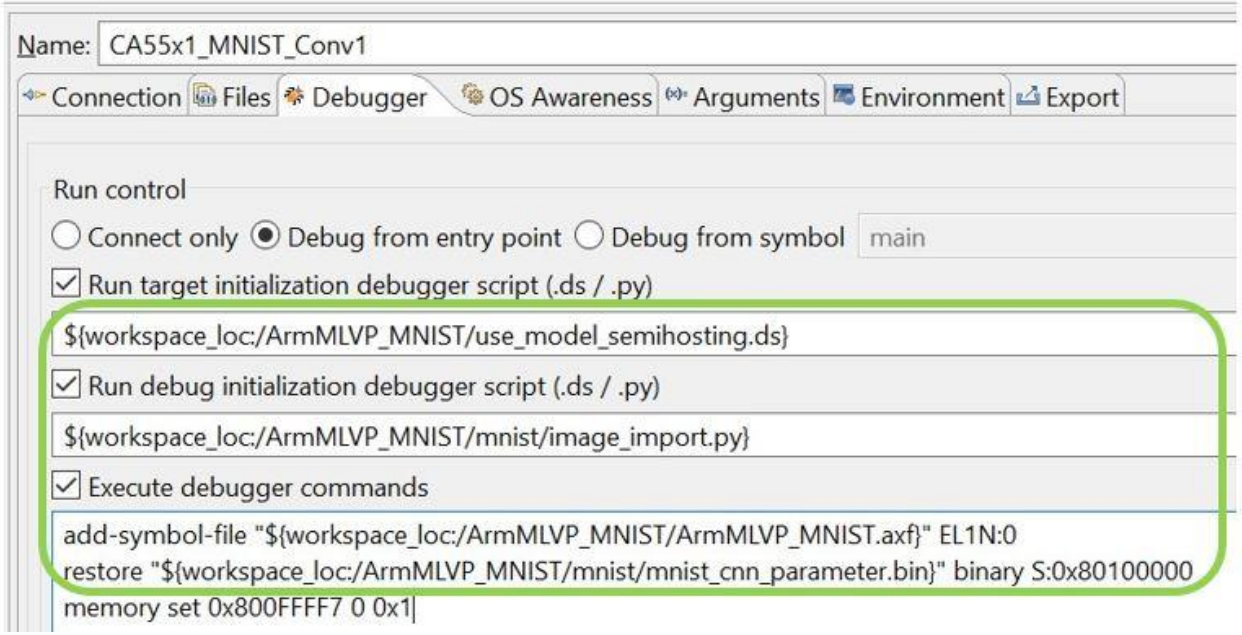
Construct Platform

- Add a debug connection to hock on the bare-metal Cortex-A55 [Fixed Virtual Platforms](#) (FVP). You can treat virtual platform as a real silicon, all my following works are running on this.

- Select platform



- Scripting for image loading and H/W initialization



arm

NN Performance Index

arm

How to Analysis Your NN Device?

- Analysis your NN from four angles:
  - Execution time
  - NN code size,
  - CPU loading and
  - Memory access usage
- Memory access usage is another crucial factor of AI performance at the edge.

	Execution Time (instr)	NN code size (byte)	Convolution usage (%)	CPU Load/Store
#1				
#2				

Initial Version (O1)

Initial Version NN API Definition & Implementation

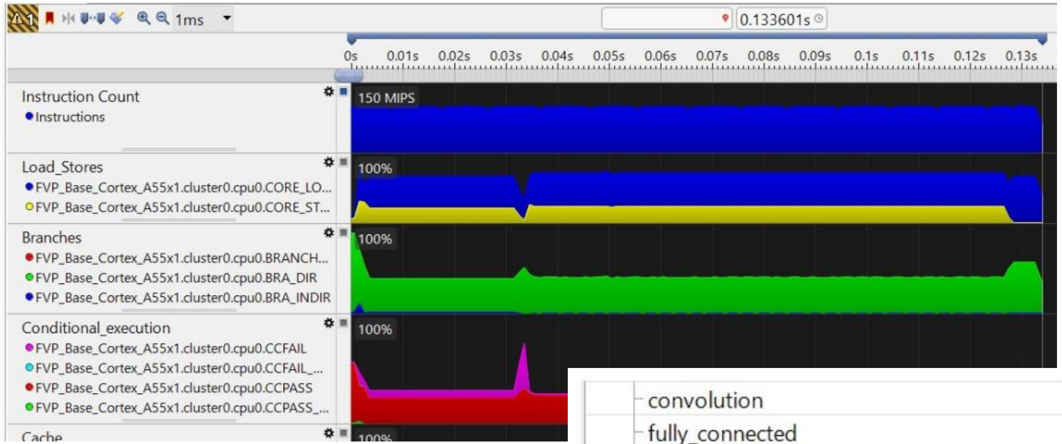
API	Description
convolution	Creates a convolution kernel that is convoluted with the layer input to produce a tensor of outputs
max_pooling	Reduce the number of parameters and amount of computation in the network
fully_connected	Connections to all activations in the previous layer, computed with a matrix multiplication followed by a bias offset

```
mnist_cnn_eval() {
    // Pre process
    convolution(&lay, layer0, layer1, layer0_paramter);
    max_pooling(&lay, layer1, layer2);
    convolution (&lay, layer2, layer3, layer2_paramter);
    max_pooling (&lay, layer3, layer4);
    fully_connected (&lay, layer4, layer5, layer5_paramter);
    fully_connected (&lay, layer5, layer6, layer6_paramter);
    // Post process
}
```

NN#1 Result

- Test 

```
DS5 debug console
---------------------------------------
Inf selected image [8] from CPU: 0
    prob [ -16, -30, -6, -7, -18, -8, -21, -24, 17, -20,], avg/std: -13/10
    Conv_mode: 1    selected image [8] from CPU: 0, inference: 8,       [Pass]
        Instr count is 13146336
             Cnt 0 is 0
             Cnt 1 is 109812
             Cnt 2 is 3416674
             Cnt 3 is 13146336
             Cnt 4 is 13146336
```



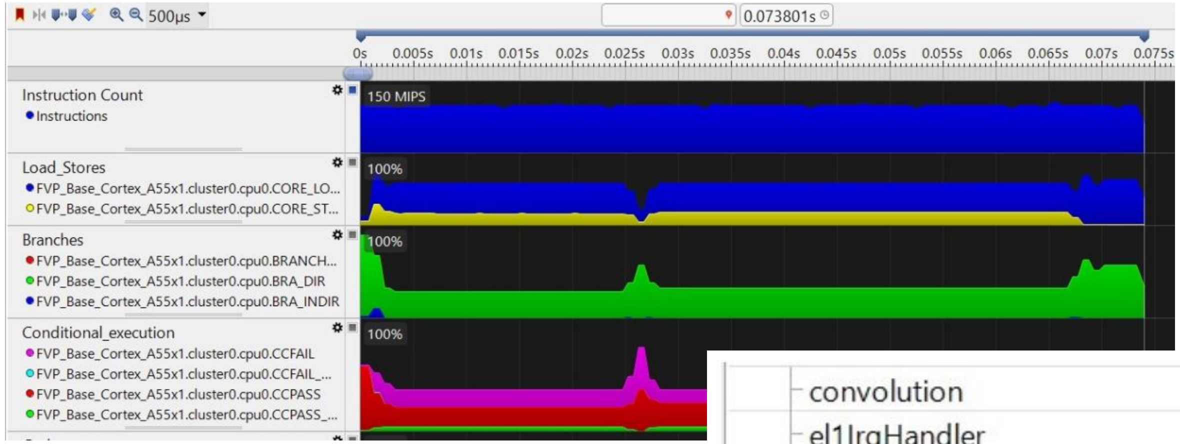
convolution	96.71%		1,292	96.71%		1,292	96.71%	
fully_connected	1.35%		18	1.35%		18	1.35%	
__flsbuf	0.67%		9	0.67%		9	0.67%	
InvalidateUDCaches	0.60%		8	0.60%		8	0.60%	
pmu_counter_get_event_type	0.22%		3	0.22%		3	0.22%	
btod_internal_mul	0.07%		1	0.07%		1	0.07%	
initTimerInterrupt	0.07%		1	0.07%		1	0.07%	
_printf_fp_dec_real	0.07%		1	0.07%		1	0.07%	
_printf_int_common	0.07%		1	0.07%		1	0.07%	
_printf_int_dec	0.07%		1	0.07%		1	0.07%	
_writebuf	0.07%		1	0.07%		1	0.07%	

- Nested loop consume 96.7% CPU loading

Compiler Optimization Version

arm

Compiler Optimization NN (O3)

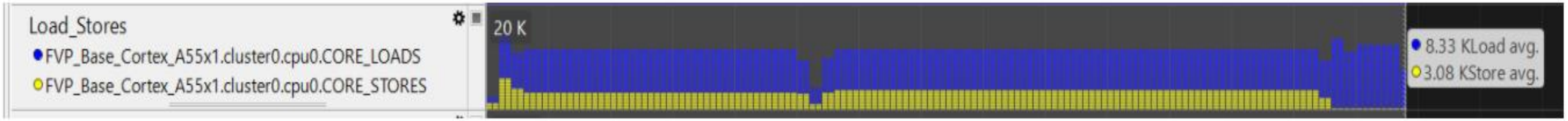


convolution	86.99%	642	86.99%	642	86.99%
el1IrqHandler	7.32%	54	7.32%	54	7.32%
max_pooling	2.30%	17	2.30%	17	2.30%
__flsbuf	1.36%	10	1.36%	10	1.36%
InvalidateUDCaches	1.08%	8	1.08%	8	1.08%

	Execution Time (instr)	NN code size (byte)	Convolution function usage	CPU Load/Store
O1	13146K	49K	96.7%	1187 / 547
O3	7166K	31K	86.99%	608 / 224

Deep Look CPU Load/Store Utilization

- Streamline provide the average CPU load/store count during the each of sampling to help the user has roughly number on any selected period. Use that we can roughly calculate the total load/store on whole NN operation.



Load: 8.33 K/s * 73ms = 608

Store: 3.08 K/s * 73ms = 224

- By tracing the assemble code on DS5,

  I can find the better coding to …

Disassembly

```
0x0000000080004AFC:  SUBS   w4, w4, #0x8
0x0000000080004B00:  FMUL   v2.4s, v1.4s, v2.4s
0x0000000080004B04:  FMUL   v3.4s, v1.4s, v3.4s
0x0000000080004B08:  FADD   v2.4s, v4.4s, v2.4s
0x0000000080004B0C:  FADD   v3.4s, v5.4s, v3.4s
0x0000000080004B10:  ADD    w14, w14, #0x8
0x0000000080004B14:  STP    q2, q3, [x24]
0x0000000080004B18:  B.NE   0x80004ae8
0x0000000080004B1C:  LDR    w4, [sp, #0x5c]
0x0000000080004B20:  CMP    w25, w17
0x0000000080004B24:  B.NE   0x80004ba8
0x0000000080004B28:  B      0x80004bd4
0x0000000080004B2C:  ADD    w14, w5, w18
0x0000000080004B30:  ADD    w4, w16, w18
0x0000000080004B34:  LDR    s0, [x1, w14, uxtw #2]
0x0000000080004B38:  MUL    w14, w10, w4
0x0000000080004B3C:  LSL    x19, x15, #2
0x0000000080004B40:  LDR    s1, [x3, w14, uxtw #2]
0x0000000080004B44:  LDR    s2, [x2, x19]
0x0000000080004B48:  CMP    w22, #0x2
0x0000000080004B4C:  FMUL   s1, s0, s1
0x0000000080004B50:  FADD   s1, s2, s1
0x0000000080004B54:  STR    s1, [x2, x19]
0x0000000080004B58:  B.LO   0x80004bd4
0x0000000080004B5C:  CMP    w25, #0x7
0x0000000080004B60:  B.LS   0x80004ba4
0x0000000080004B64:  MUL    w14, w22, w18
0x0000000080004B68:  CMN    w27, w26
```

API Optimization Version

Re-design API

API	Description
convolution	Creates a convolution kernel that is convoluted with the layer input to produce a tensor of outputs
convolution_conv2()	Redesign convolution layer, dispatch each 2D input element calculation into convolution_filter2()
convolution_filter2()	Filter kernel convolution implementation
max_pooling	Reduce the number of parameters and amount of computation in the network
fully_connected	Connections to all activations in the previous layer, computed with a matrix multiplicatio followed by a bias offset

```
//pseudo code
float int convolution_filter2() {
    // Load parameter
    for (current_filter_row = 0; current_filter_row < filter_rows; current_filter_row++
        for (current_filter_col = 0; current_filter_col < filter_cols; current_filter_c
            for (in_ch = 0; in_ch < input_channel; in_ch++) {
                current_input = ((float*)inputs)[  ((stride_row + current_filter_row) *
                                                 + ((stride_col + current_filter_col) *
                                                 + in_ch];
                for (out_ch = 0; out_ch < output_channel; out_ch++) {
                    current_weight = ((float*)weights)[  (current_filter_row * filter_c
                                                       + (current_filter_col * input_ch
                                                       + (in_ch                * output_c
                                                       + out_ch];
                    current_result = current_input * current_weight;
                    ((float*)outputs)[  (stride_row * output_columns * output_channel)
                                      + (stride_col * output_channel)
                                      + out_ch]
                    += current_result;
                }

            }
        }
    }
    for (out_ch = 0; out_ch < output_channel; out_ch++) {
        current_biase = ((float*)biases)[out_ch];
        kernel_output_addr = (stride_row * output_columns * output_channel) + (stride_c
        kernel_result = ((float*)outputs)[kernel_output_addr];
        kernel_result += current_biase;
        if (relu_activation) {
            kernel_result = relu(kernel_result);
        }
        ((float*)outputs)[kernel_output_addr] = kernel_result;
    }
}

int convolution_conv2() {
    // Initial
    // Preload parameter
    // Pre-processing
    for (stride_row = 0; stride_row < lay->output_rows; stride_row++) {
        for (stride_col = 0; stride_col < lay->output_columns; stride_col++) {
            convolution_filter2(stride_row, stride_col, ...);
        }
    }
}
```

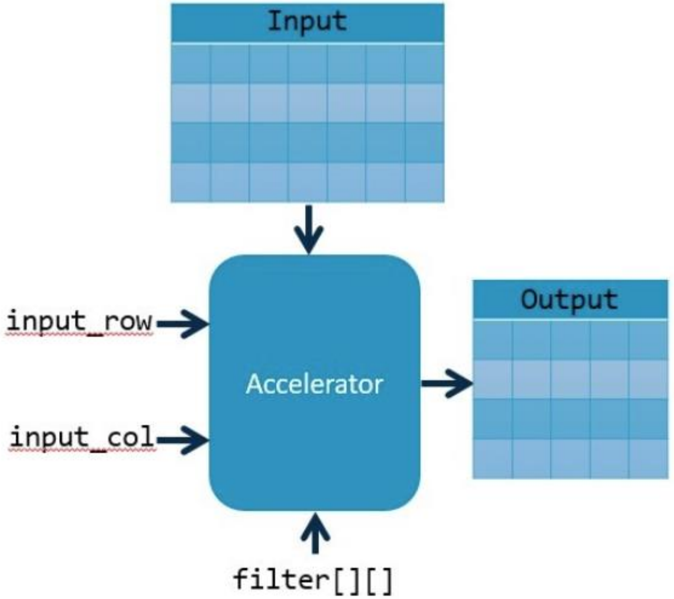
NN#3 Result

	Execution Time (instr)	NN code size (byte)	Convolution function usage	CPU Load/Store
O1	13146K	49K	96.7%	1187 / 547
O3	7166K	31K	86.99%	608 / 224
O3 w/conv2	3952K	28K	77.16%	250 / 76

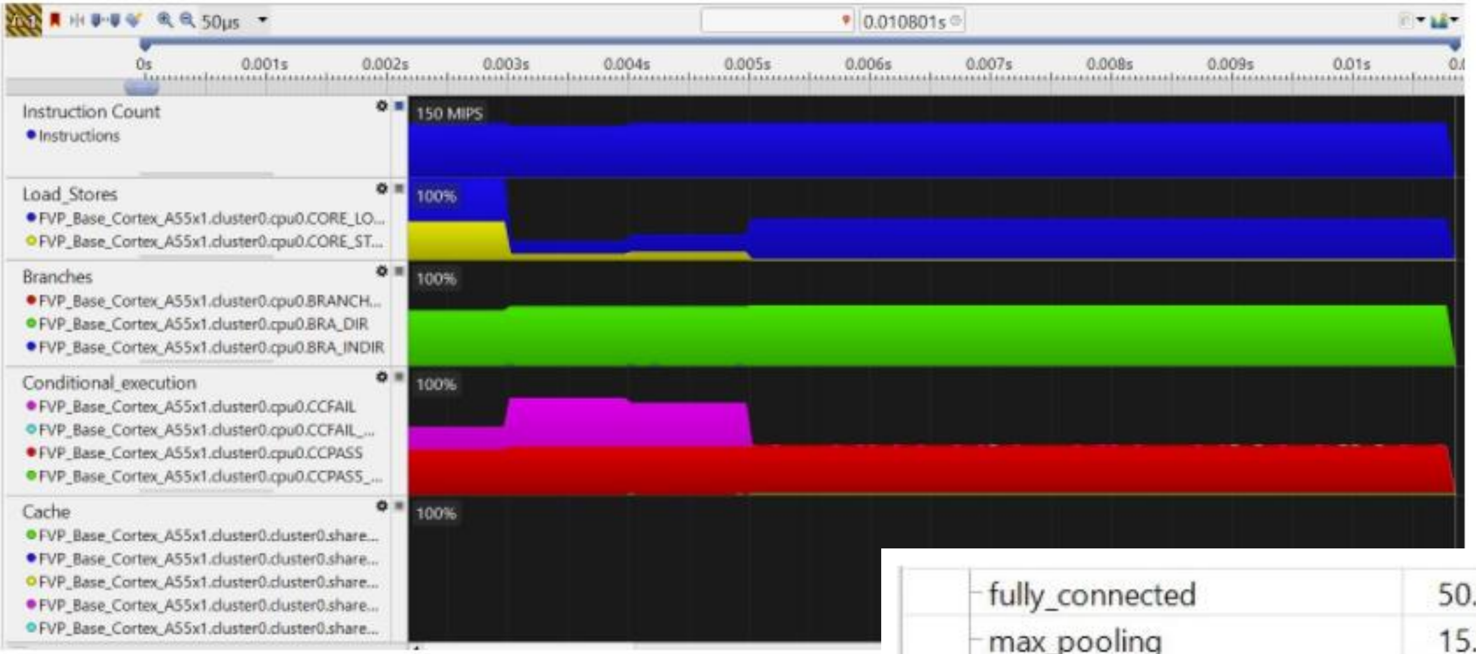
NN w/ Accelerator Model

Accelerator Concept

- One of straightforward concept is using NN accelerator to offload some of the work from the CPU, which has the capability to direct access input data, model parameter and write into the output buffer. Enable parallel 5x5 matrix multiplier.
- That's a straightforward idea, but the problem is I don't have RTL design to confirm with, does the same virtual platform environment can help with?
- Using SystemC/TLM to implement an idea of approximately timing behavior model and integrate into Fast Model.



NN#4 Result

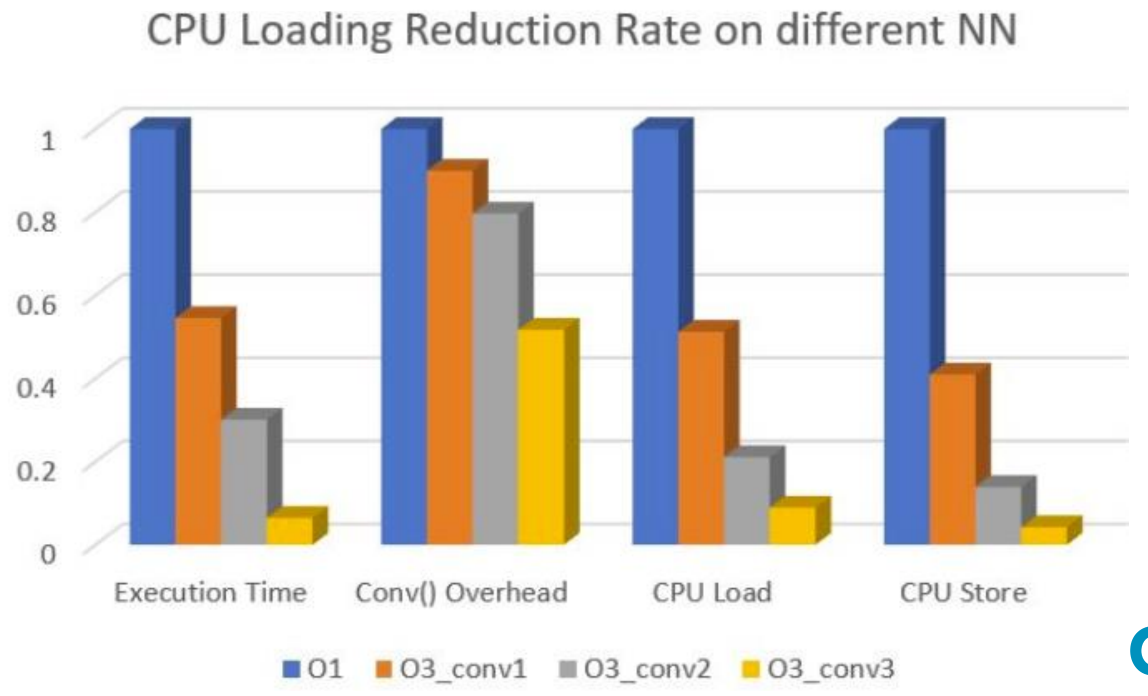


fully_connected	50.00%	54	50.00%	54	50.00%
max_pooling	15.74%	17	15.74%	17	15.74%
convolution_conv3	12.96%	14	12.96%	14	12.96%
InvalidateUDCaches	7.41%	8	7.41%	8	7.41%
__flsbuf	6.48%	7	6.48%	7	6.48%
_printf_int_common	1.85%	2	1.85%	2	1.85%
puts	0.93%	1	0.93%	1	0.93%
_fputc$unlocked	0.93%	1	0.93%	1	0.93%
_mutex_acquire	0.93%	1	0.93%	1	0.93%
_printf_f	0.93%	1	0.93%	1	0.93%
_printf_fp_dec_real	0.93%	1	0.93%	1	0.93%
__printf	0.93%	1	0.93%	1	0.93%

Summary

Final Result

	Execution Time (instr)	NN code size (byte)	Convolution function usage	CPU Load/Store
O1	13146K	49K	96.7%	1187 / 547
O3	7166K	31K	86.99%	608 / 224
O3 w/conv2	3952K	28K	77.16%	250 / 76
O3 w/NNacc	850k	24K	50%	106 / 23



arm

Conclusion

- Using a sample NN application to demonstrate how to bring machine learning inference to an Arm device. This use case shows DS-5 and Fast Models are excellent tools to help people develop and profile software algorithms on any Arm CPU.
- Welcome to visit blog and code for the detail.
- Next Step:
  - Adapt ML on Cortex-M by CMSIS-NN. (target MCU partner & ODM/OEM)
  - Compute Library profiling (target ISP partner)
  - ArmNN

Thank You!

Danke!

Merci!

谢谢!

ありがとう!

Gracias!

Kiitos!

감사합니다

धन्यवाद