Animal Recognition and Identification with Convolutional Neural Networks for Farm Monitoring

Ryunosuke Murakami, Abderazek Ben Abdallah
Graduate School of Computer Science and Engineering,
Adaptive Systems Laboratory
The University of Aizu, Japan

Outline

- Background
- Motivation
- Research goal
- Implementation
- Evaluation
- Conclusion
- Future work

Background

- Due to the increasing demand in the agricultural industry, the need to effectively grow and protect a plant and increase its yield is necessary
- It is important to monitor the plant during its growth period and protect it from animals (pig, etc.) at the time of harvest

Fig 1. Farm and enemy(pig)

Fig 2. Olive and plant disease

Motivation

- Monitoring the plants from plantation to harvesting is necessary for better productivity
- Smart farming needs right decision and monitoring tools for better productivity, quality and profit
- Artificial neural network concept is efficient for image processing

Convolutional Neural Network

- Effective to object recognition
  - Process data while keeping the shape of image
- Behavior is similar to *visual cortex*
  - Performance is close to human-level
- Learn feature vector automatically from data

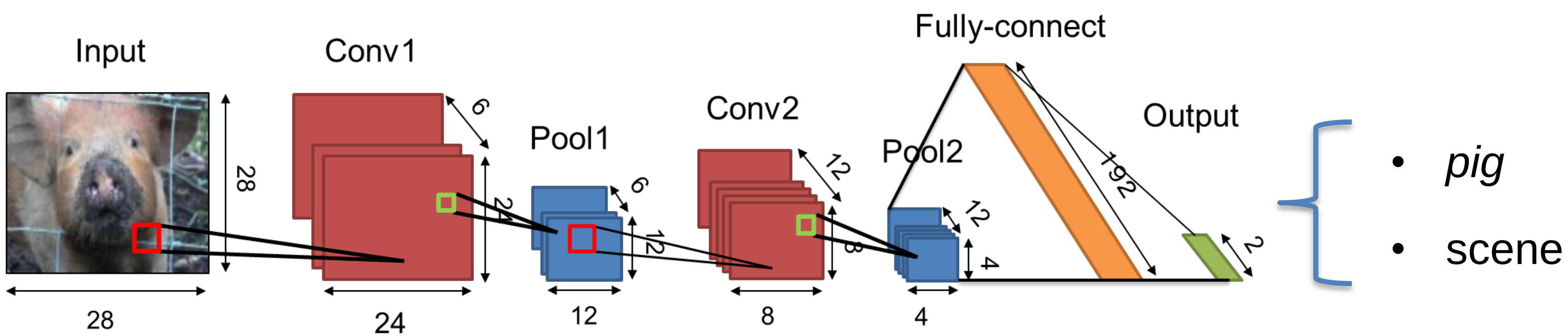

Fig 3. CNN example

Research goal

- Develop a power efficient object classification system for FARM monitoring system based on artificial neural network concept:
  - Hardware implementation of Convolutional Neural Network on FPGA
  - Evaluation of real hardware complexity (power and area) and performance (recognition accuracy, time)
- The purpose is to monitor strange animals (pig, etc)

FARM monitoring system overview

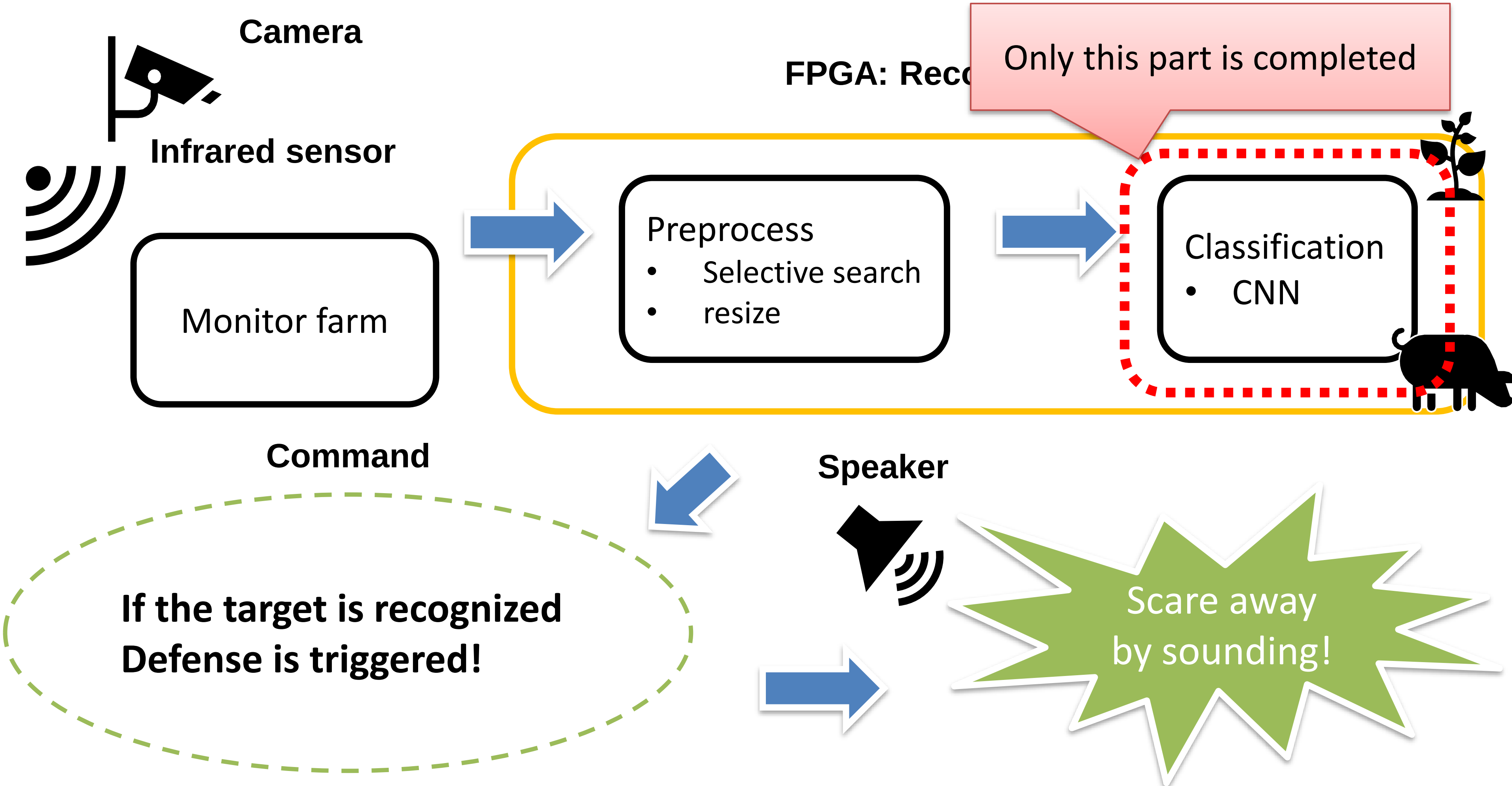

Fig 4. system overview

Dataset

- Collected from *ImageNet*
- Image: 60x60 pixel, RGB
- Target class and distribution of original data
  - Pig class: 685
  - Scene class: 685

Table 1. Data distribution

Class	Train	Valid	Test
Pig	411	137	137
Scene	411	137	137

Data samples

Pig class data

Scene class data

Fig 5. Data samples

Data augmentation

- Augmented training data twice
  - Original data: 822
  - Augmented data(x2): 1,644

[Applied image conversion]

- Random combination of following conversions
  - Slide the pixels randomly within range [-4,4]
  - Flip horizontal randomly

Network structure

Table 2. Network structure

CNN Layer	filterHxW	stride	output ch	output HxW
data	-	-	3	60x60
conv1	5x5	1	4	56x56
pool1	2x2	2	4	28x28
conv2	5x5	1	4	24x24
pool2	2x2	2	4	12x12
conv3	5x5	1	8	8x8
pool3	2x2	2	8	4x4
FC Layer	input	output	-	-
linear1	128	80	-	-
linear2	80	2	-	-

Learning environment

Table 3. Learning environment

Name	Details
OS	Ubuntu 16.04.3 LTS
CPU	Intel(R) Core(TM)i7-4770CPU@ 3.40GHz
GPU	GeForce GTX 1060
Language	Python (ver:2.7.13)
Framework	Chainer (ver: 6.0.0)

Learning accuracy

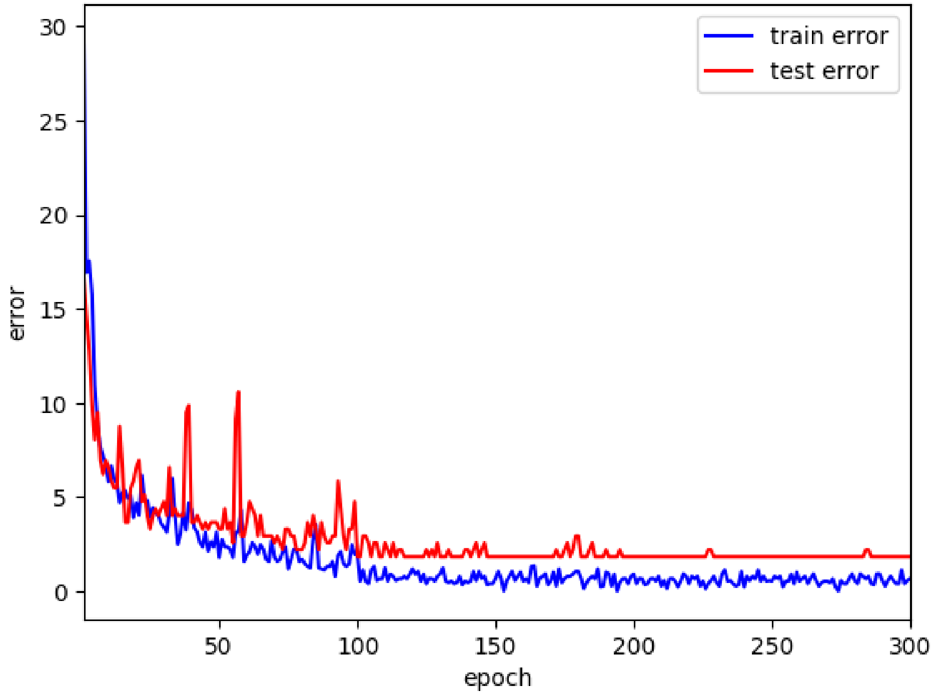

Fig 6. Leaning curve

- Iteration number: 300
- Learning decay: each 100 iteration
- Batch size: 128
- Optimizer: Adam
- Accuracy: 98.18%

FPGA design platform

- Used SDSoC 2018.1 for CPU/FPGA codesign
  - High-level synthesis for hardware design
- Board: Zynq UltraScale+ MPSoC zcu102
  - 64-bit quad core ARM Cortex™-A53

Table 5. FPGA resource

Name	Available
BRAM	32 Mbits
LUTs	274,080
Flip Flop	548,160
DSP48E	2,520

Hardware implementation

- Convolution part is in FPGA, and all other parts are in CPU
- Applied two optimization on convolution part
  - Pipeline execution
    - Hardware block design on kernel row, column level
    - Pipeline execution on kernel depth level
  - Fixed point implementation
    - Original: 64 bit floating point
    - Fixed point: 16 and 8 bit

Design architecture

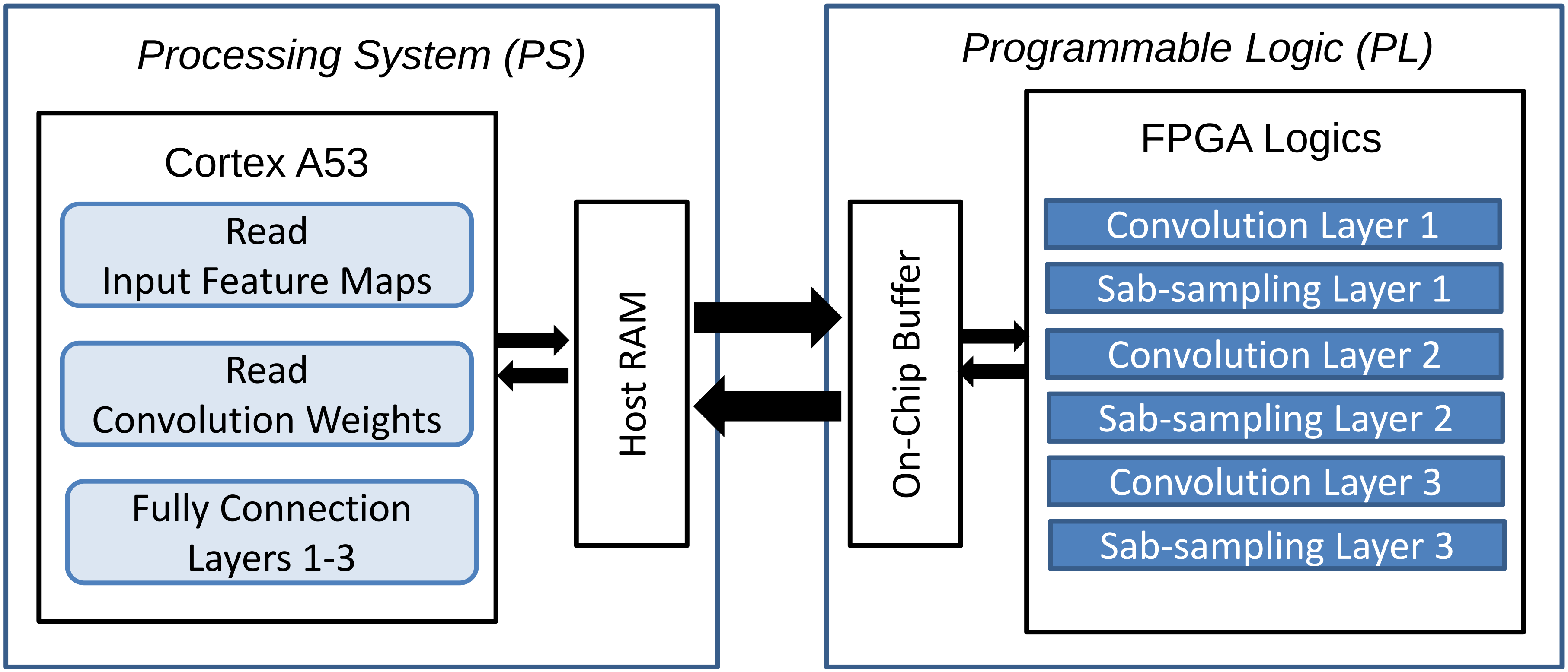

Fig 7. Design architecture

Pipeline design

Pipeline execution

Parallel execution

```
//Load input feature map, convolution kernel to buffer
for(b=0; b<B; b++){// Input image
 for(o=0; o<O; o++){// Output feature map depth
  for(ir=0,ic=0;ir<IR,ic<IL;ir++,ic++){// Input feature map position
   sum = 0;
   for(kd = 0; kd < KD; ++kd){// Kernel depth
    #pragma HLS PIPELINE
    for(kr = 0; kr < KR; ++kr){// Kernel row
     #pragma HLS UNROLL
     for(kc = 0; kc < KC; ++kc){// Kernel column
      #pragma HLS UNROLL
      sum += input_data[b][kd][ir+kr][ic+kc] * kernel[o][kd][kr][kc];
     }
    }
   }
    output[b][o][ir][ic] = Activation(sum + bias[o]);
}}}
```

Fig 8. Pseudo code of convolution part

Fixed-point implementation

- Fixed-point operation requires fewer clock cycles compared with floating-point one
- Used VIVADO HLS library for fixed-point precision

Value range regulation

- If $a \in [-1,1]$ and $b \in [-1,1], then\ a \cdot b \ \in [-1,1]$
- Regulate the computation value range [-1, 1]
  - only 1-bit is for integer part, and the position of decimal point does not change
    - Can avoid a large rounding error from decimal point adjustment

Estimate maximum value for [-1,1] regulation

- Input all training data, and get the absolute maximum value on each layers' output feature map
  - conv1: 2.91
  - conv2: 2.54
  - conv3: 3.80
  - fc1: 3.15
  - fc2: 17.71
- Divide parameters by 15% scaled up absolute maximum value
  - Overflow is processed to round down to 0

Evaluation configuration

- Accuracy was evaluated with test data
- Power consumption was analyzed by Vivado power analyzer
- 1,000 images were used for time, power, and energy estimation

Table 6. Design metrics

Metrics	SW_{PS}	HW_{O}
Core	Cortex A53	Cortex A53 / FPGA
Clock (GHz)	1.334	1.334 (PS) / 0.1 (FPGA)
Num of Core	4	4 (PS)

Evaluation result

Table 7. Performance comparison

Metrics	SW_{PS}	HWfloat64	HWfixed16	HWfixed8
Accuracy (%)	98.18	98.18	98.18	94.53
Time (s)	43.53	23.14	11.87	9.35
Power (W)	3.49	4.21	4.01	4.00
Energy (J)	151.92	97.42	47.60	37.4

Table 8. FPGA resource utilization

Metrics	DSP (%)	BRAM (%)	FF (%)	LUT (%)
HWfloat64	242 (9)	270 (14)	33285 (6)	31764 (11)
HWfixed16	233 (9)	50 (2)	15906 (2)	24931 (9)
HWfixed8	230 (9)	34 (1)	15187 (2)	24511 (8)

Conclusion

- Classifier used for recognition system was implemented in CNN and successfully optimized
- Computation speed and energy consumption in 16 bit fixed point implementation were improved 3.67 times and 3.19 times without accuracy degrade
- Computation speed and energy consumption in 8 bit fixed point implementation were improved 4.66 times and 4.06 times with slight accuracy degrade

Future work

- Implement object detection part so that target image is captured for classification
- Experiment on how much accuracy / speed is necessary for this farm monitoring system