Real-Time Video Object Recognition Using Convolutional Neural Network

Byungik Ahn
Neurocoms
Seoul, South Korea
jerryahn@neurocoms.com

***Abstract*—A convolutional neural network (CNN) is implemented on a field-programmable gate array (FPGA) and used for recognizing objects in real-time video streams. In this system, an image pyramid is constructed by successively down-scaling the input video stream. Image blocks are extracted from the image pyramid and classified by the CNN core. The detected parts are then marked on the output video frames. The CNN core is composed of six hardware neurons and two receptor units. The hardware neurons are designed as fully-pipelined digital circuits synchronized with the system clock, and are used to compute the model neurons in a time-sharing manner. The receptor units scan the input image for local receptive fields and continuously supply data to the hardware neurons as inputs. The CNN core module is controlled according to the contents of a table describing the sequence of computational stages and containing the system parameters required to control each stage. The use of this table makes the hardware system more flexible, and various CNN configurations can be accommodated without re-designing the system. The system implemented on a mid-range FPGA achieves a computational speed greater than 170,000 classifications per second, and performs scale-invariant object recognition from a 720x480 video stream at a speed of 60 fps. This work is a part of a commercial project, and the system is targeted for recognizing any pre-trained objects with a small physical volume and low power consumption.**

Keywords—Convolutional Neural Network; FPGA, Video Object Recognition

I. Introduction

Convolutional neural networks (CNNs) have been used for image recognition systems, and as of February 2012, they have achieved the lowest error rate on the MNIST database. CNNs are particularly good candidates for field-programmable gate array (FPGA) implementation for two reasons. First, unlike other neural models that require an additional feature extraction function to be computed through other means, a CNN includes both feature extraction and classification as its functions. This can enable an FPGA to compute the entire system in hardware without suffering from a speed degradation owing to the computations required for the extra software functions. Second, FPGAs typically provide hundreds of small individual memories that can be used for parallelizing computations. However, the total memory capacity is limited to under 100 Mb, and cannot store information for a large number of connection weights within a neural network. Therefore, FPGAs can compute only small neural networks, or with a limitation in parallelism owing to the use of centralized external memory. However, a CNN shares connection weights among the local receptive fields and requires only a small number of connection weights even for a substantially large network, which can be accommodated by an FPGA.

In this paper, we describe a hardware architecture that can be used to implement CNN systems, and a system is implemented on an FPGA based on the hardware architecture in which the CNN is used as the core function for recognizing objects in real-time video signals. Even though the system is implemented in hardware without the use of any processor, it is self-contained and includes all components required to recognize a video stream. One of the design goals of this system is to maximize the performance resource ratio and achieve the same performance using a cheaper chip. This goal may also lead to smaller physical volume and less power consumption. In this system, a total of 150 standard multipliers are harnessed with an average idle time of approximately 6%, and the proportion of hardware resources used for components other than the multipliers is under 40%. As a result, the entire system is implemented using approximately one-third of the hardware resources provided by a mid-range FPGA.

In this paper, novel schemes for building hardware-based CNN systems are also presented. In addition, a multi-category recognition method is introduced in which different weight sets are used in alternating video frames and thus objects within different categories can be classified for the same video stream.

It is noted that only the feed-forward part of the CNN with pre-trained parameter sets would be sufficient for most image recognition applications. Even when the training is required, the training data are often collected from the environment periodically rather than pre-existing in a batch, and the training performance is less critical. Therefore, only the feed-forward part is used in our work.

Section II of this paper describes the base CNN algorithm that our work is based upon, and reviews a CNN with regard to a network of neurons. Section III describes the hardware architecture, implemented system, and schemes used to improve the efficiency. The implementation results, a discussion, and some concluding remarks are presented in Sections IV, V and VI, respectively.

II. Convolutional Neural Network

A. Base Algorithm

The CNN implemented in this work is based on the MATLAB code provided in [1] and [2]. It has a five-layer architecture with 28 × 28 gray-scale image blocks as input. The first layer has six feature maps connected to a single input layer through six 5 × 5 kernels. The second layer is a 2 × 2 max-pooling layer. The third layer has 12 feature maps each of which is connected to six max-pooling outputs in the second layer through 72 5 × 5 kernels. The fourth layer is a 2 × 2 max-pooling layer. The feature maps of this fourth layer are fed into the final fully connected classification layer, which consists of ten output neurons corresponding to the ten class labels. The original code is modified from a batch size of 50 to one (no-batch). After training 100 epochs on the full MNIST training set, a mean error rate of 1.01% was shown. The computation implied in this code is retained in our hardware implementation so as to verify the computation results of hardware system using the sequential code.

This specific CNN configuration is assumed throughout the remainder of this paper for convenience. However, the ideas described herein can be easily generalized to other CNN configurations.

B. Network of Neurons in CNN

Figure 1 illustrates the computation of layers 1 and 2 in our CNN.

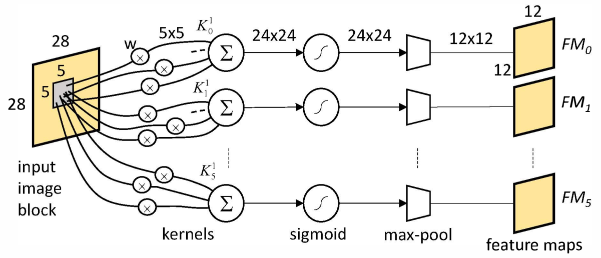

Fig. 1. Computation of layers 1 and 2. The input image block is scanned for a 5 × 5 local receptive field. From the view-point of a conventional neural network, the local receptive field becomes the input for six neurons (kernels) where the weighted input, net input, and sigmoid function are computed.

The inner area of the 28 × 28 input image block is scanned for a 5 × 5 local receptive field. A total of 24 × 24 local receptive fields are scanned. Each local receptive field becomes the common input for six kernels where the weighted input, net sum, and sigmoid function are computed in sequence. The 24 × 24 results of the sigmoid function are then 2 × 2 sub-sampled with a max-pooling function, and become a 12 × 12 feature map. The kernels, which are denoted by K^1_i, are in essence conventional neurons with an additional max-pooling function. Therefore, there are 24 × 24 × 6 neurons, each with 25 inputs (synapses) in layers 1 and 2. Note that 25 × 6 × 24 × 24 = 86,250 multiplications are required in computing the weighted inputs of neurons in layer 1, regardless of the computational system used.

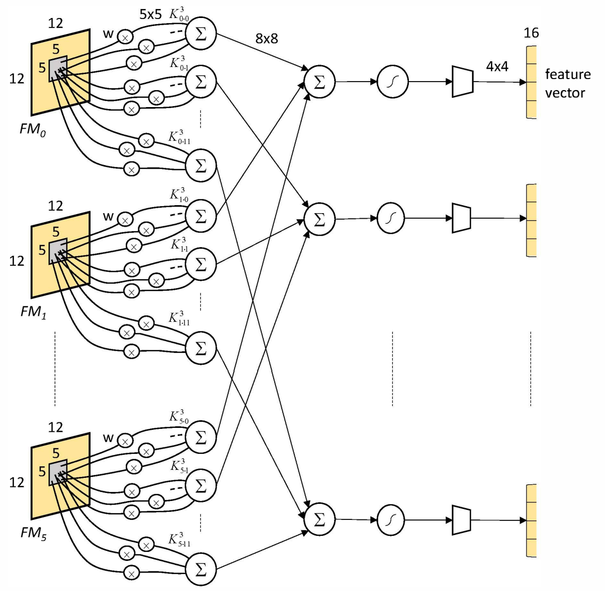

Fig. 2. Computation of layers 3 and 4. Six feature maps are scanned for a 5 × 5 local receptive field. Local receptive fields from the same position of the feature maps become the common inputs of 12 neurons, each with six kernels. In these layers, there are 8 × 8 × 12 neurons, each with 150 synapses.

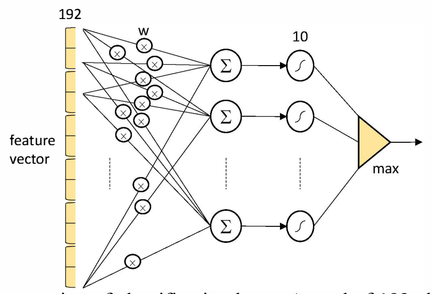

Fig. 3. Computation of classification layer. A total of 192 elements in the feature vector are fully connected to ten neurons. The neuron with highest output is picked for the classification result.

Figure 2 shows the computation of layers 3 and 4. The inner areas of six feature maps, which are the output of the previous layers, are scanned for 5 × 5 local receptive fields. The local receptive fields produced from the same position of the feature maps become the common inputs to 12 neurons, each with six kernels. The *i*th kernel of the *j*th neuron in layer 3 is denoted by K^3_{ij}. The 8 × 8 sigmoid outputs are sub-sampled with a max-pooling function resulting in a 4 × 4 feature map. In these layers, there are 8 × 8 × 12 neurons each with 150 synapses. Layer 3 requires at least 25 × 6 × 8 × 8 × 12 = 117,600 multiplications for a single classification.

Figure 3 shows the computation of the classification layer. The feature maps produced in the fourth layer are merged into a feature vector with 192 elements. The elements of the feature vector are fully connected to ten neurons. The neuron with the highest output value and over a given threshold is picked for the classification result. This layer has ten neurons, each with 192 synapses, and requires 1,920 multiplications.

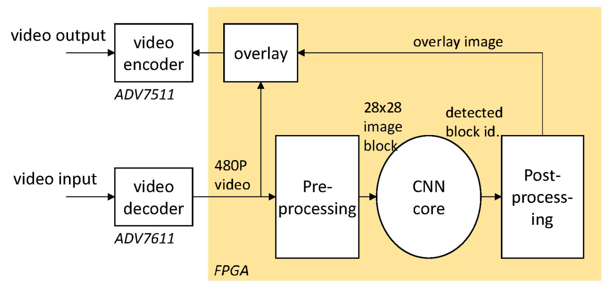

Fig. 4. Block diagram of the implemented system. All functions are synthesized in an FPGA chip except for the video decoder and encoder. The input video stream is sent to the output video stream with the recognition status overlaid on it.

Note that the computations of neurons in the feature extraction layers and the classification layer are essentially the same. The only differences are how the inputs and outputs of the neurons are prepared and processed.

III. Video Object Recognition System

Figure 4 shows a block diagram of the implemented system. The system is composed of a video decoder and encoder pair, pre-processing and post-processing modules, and the CNN core. The video decoder and encoder are on their own chips, and all other functions are synthesized in an FPGA chip. The format of the source video input is 480P with a 720 × 480 resolution and a frame rate of 60 fps.

A. *Preprocessing Module*

The preprocessing module consists of an image resizer, an image block sequencer, and a normalizer connected in a row, as shown in Figure 5.

The image resizer consists of seven image scalers. The video source is 1/2 down-scaled by the first scaler (bottom of the figure) and the scaled image is further 2/3 down-scaled by the second scaler, as shown in Figure 6. The outputs of the first and second scalers are then successively 1/2 down-scaled by the remaining scalers. Owing to the use of 1/2 and 2/3 scalers, which can be implemented using only adders and bit-shifters, the use of multipliers can be avoided. The scaled images constitute an image pyramid from which a scale-invariant image recognition is carried out. The sizes of the images are reduced to one-half in both the x and y dimensions for every two pyramid steps. Because the max-pooling layers of a CNN provide an additional scale invariance, these seven levels of the image pyramid are sufficient for catching objects at any scale.

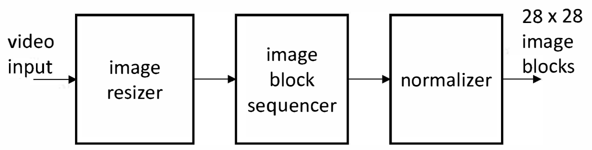

Fig. 5. Components of preprocessing module.

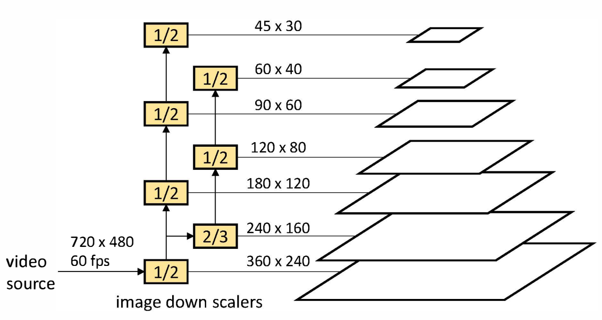

Fig. 6. Building of image pyramid by image scalers.

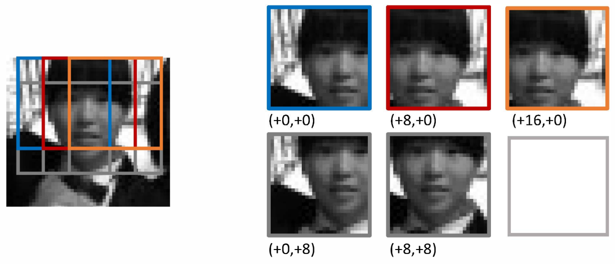

Fig. 7. Image blocks are shifted with 8-pixel step size. Positions of all image blocks are further shifted by four pixels over 4 video frames.

The image block sequencer extracts 28 × 28 image blocks from the image pyramid. The positions of the image blocks in each scaled image are deployed with an 8-pixel step size in both x and y dimensions, as shown in Figure 7. A total of 2,356 image blocks are extracted from a single video frame with a duration of approximately 7 μs. In addition, the positions of all image blocks are further shifted by four pixels in both the horizontal and vertical directions with a four video-frame cycle by which a finer image search space is provided.

The normalizer normalizes the image blocks on a per-image block basis. A linear normalization algorithm is used, the dynamic range of which is adjusted to be between 0 and 1.

B. *Post-processing module*

The identification number of the image block and the class label detected by the CNN core are sent to the post-processing module. This module identifies the position of the image block in the source image and marks a colored frame in an overlay memory. This memory is used to overlay the next video image frame to indicate that objects have been detected. Two such memories are used alternately: one for marking and the other as a display.

C. *CNN core*

The CNN core takes 28 × 28 image blocks as input and classifies them based on the pre-trained connection weights (or parameters). The entire portion of the CNN core is designed as a register-transfer level digital hardware system using only elementary components such as combinational and sequential logic, arithmetic operators, and memories. Neither embedded processor nor main memory is used. The design is based on a hardware architecture called Neuron Machine, which was introduced in [3], where a deep belief network was implemented.

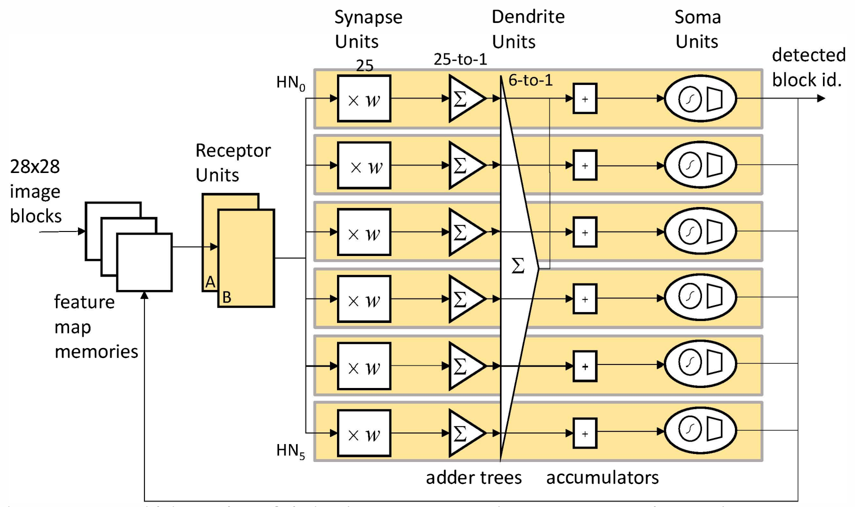

Fig. 8. Block diagram of the CNN core, which consists of six hardware neurons and two Receptor Units. Hardware neurons are designed as a fully pipelined circuit synchronized with the system clock.

TABLE I. STAGE OPERATION TABLE

	Computational Stages														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Clocks	576	64	64	64	64	64	64	64	64	64	64	64	64	10	10
Input	IB	FM_0	FM_1	FM_2	FM_3	FM_4	FM_5	FM_0	FM_1	FM_2	FM_3	FM_4	FM_5	FV_l	FV_h
RU	A	B	A	B	A	B	A	B	A	B	A	B	A	-	-
HN$_0$	K^1_0	$K^3_{0\cdot 0}$	$K^3_{1\cdot 0}$	$K^3_{2\cdot 0}$	$K^3_{3\cdot 0}$	$K^3_{4\cdot 0}$	$K^3_{5\cdot 0}$	$K^3_{0\cdot 6}$	$K^3_{1\cdot 6}$	$K^3_{2\cdot 6}$	$K^3_{3\cdot 6}$	$K^3_{4\cdot 6}$	$K^3_{5\cdot 6}$	1'st half of FV	2'nd half of FV
HN$_1$	K^1_1	$K^3_{0\cdot 1}$	$K^3_{1\cdot 1}$	$K^3_{2\cdot 1}$	$K^3_{3\cdot 1}$	$K^3_{4\cdot 1}$	$K^3_{5\cdot 1}$	$K^3_{0\cdot 7}$	$K^3_{1\cdot 7}$	$K^3_{2\cdot 7}$	$K^3_{3\cdot 7}$	$K^3_{4\cdot 7}$	$K^3_{5\cdot 7}$		
HN$_2$	K^1_2	$K^3_{0\cdot 2}$	$K^3_{1\cdot 2}$	$K^3_{2\cdot 2}$	$K^3_{3\cdot 2}$	$K^3_{4\cdot 2}$	$K^3_{5\cdot 2}$	$K^3_{0\cdot 8}$	$K^3_{1\cdot 8}$	$K^3_{2\cdot 8}$	$K^3_{3\cdot 8}$	$K^3_{4\cdot 8}$	$K^3_{5\cdot 8}$		
HN$_3$	K^1_3	$K^3_{0\cdot 3}$	$K^3_{1\cdot 3}$	$K^3_{2\cdot 3}$	$K^3_{3\cdot 3}$	$K^3_{4\cdot 3}$	$K^3_{5\cdot 3}$	$K^3_{0\cdot 9}$	$K^3_{1\cdot 9}$	$K^3_{2\cdot 9}$	$K^3_{3\cdot 9}$	$K^3_{4\cdot 9}$	$K^3_{5\cdot 9}$		
HN$_4$	K^1_4	$K^3_{0\cdot 4}$	$K^3_{1\cdot 4}$	$K^3_{2\cdot 4}$	$K^3_{3\cdot 4}$	$K^3_{4\cdot 4}$	$K^3_{5\cdot 4}$	$K^3_{0\cdot 10}$	$K^3_{1\cdot 10}$	$K^3_{2\cdot 10}$	$K^3_{3\cdot 10}$	$K^3_{4\cdot 10}$	$K^3_{5\cdot 10}$		
HN$_5$	K^1_5	$K^3_{0\cdot 5}$	$K^3_{1\cdot 5}$	$K^3_{2\cdot 5}$	$K^3_{3\cdot 5}$	$K^3_{4\cdot 5}$	$K^3_{5\cdot 5}$	$K^3_{0\cdot 11}$	$K^3_{1\cdot 11}$	$K^3_{2\cdot 11}$	$K^3_{3\cdot 11}$	$K^3_{4\cdot 11}$	$K^3_{5\cdot 11}$		
Acc.	pass	acc.	acc.	acc.	acc.	acc.	result	acc.	acc.	acc.	acc.	acc.	result	acc.	result

IB: input image block, FM: feature map, FV: feature vector, RU: Receptor Unit, HN$_i$: ith hardware neuron, K^l_{ij}: ith kernel of jth neuron in lth layer,
Acc: accumulator operation in Dendrite Units, pass: bypass data, acc: accumulate, result: return accumulated data

Figure 8 shows a block diagram of the CNN core. It mainly consists of six hardware neurons, labeled HN_0 through HN_5, and two Receptor Units, *A* and *B*. Each hardware neuron is composed of a Synapse Unit, a Dendrite Unit, and a Soma Unit. Each Synapse Unit contains 25 multipliers and 25 weight memories, and computes the weighted inputs. The Dendrite Unit contains a 25-to-1 adder tree that sums the weighted inputs in parallel, and an accumulator that accumulates the output of the adder tree sequentially. Each accumulator contains a memory to store temporary data. The Soma Unit computes the sigmoid and max-pooling functions. Receptor Units are circuits of shift registers and scan the feature map memories for 5 × 5 sized local receptive fields. Their job is to continuously supply 25-value outputs to the hardware neurons without stopping such that the hardware neurons are fully utilized.

The operation of the system consists of a sequence of computational stages, each of which computes CNN layers 1 and 2, 3 and 4, or layer 5. The six hardware neurons in the CNN core are shared during all stages. In the first stage, CNN layers 1 and 2 are computed using an input image block as their common input. Each hardware neuron computes one of the kernels in Figure 1. In the Soma Units, six feature maps are generated simultaneously and stored in the feature map memories. In the following six stages, the first six feature maps of layers 3 and 4 are computed using the feature maps generated in the first stage as inputs. Each hardware neuron computes one of the six kernels for each logical neuron. The partial sums of the kernels are stored temporarily in the memories of the accumulators in the Dendrite Units in stages 2 through 6, and the accumulated results are sent to the Soma Unit in stage 7. In the next six stages, stages 8 through 13, the second six feature maps in layers 3 and 4 are computed in the same way. In stages 14 and 15, the computation of the classification layer is carried out based on the feature maps computed in the previous stages. In this stage, an additional 6-to-1 adder tree is used to sum the results of six 25-to-1 adder trees in the Dendrite Units. Therefore, the six hardware neurons are combined together to compute the larger neurons simultaneously. The same sequences of the computational stages are repeated for the next input image blocks.

The overall system is controlled by Control Unit, which is not shown in Figure 8. The Control Unit references a table called the Stage Operation Table to generate control signals for each stage. Table I shows some of the data stored in the Stage Operation Table. The third row in the table indicates the number of clock cycles required to run each stage. The fourth and fifth rows refer to the feature memories and the Receptor Units used in each stage, respectively. The next six rows designate the CNN kernels that the hardware neurons compute. The last row is for the operation of the accumulators in the Dendrite Units. The use of the Stage Operation Table makes the hardware system more flexible, and accommodates various CNN configurations without a re-design of the system.

With a fully pipelined design, the hardware neurons can collectively compute 150 multiplications (in the Synapse Units), 149 additions (in the Dendrite Units), six sigmoid functions (in the Soma Units), all in a single system clock cycle. At this computational speed, an image block can be classified within 1380 clock cycles. However, there are a few aspects that might delay the computation. First, the system pipeline may have to be flushed out before the next stage begins if the same control signal is referenced by all components in the system. Consider Figure 9(a) in which a control signal called *current_stage* is shared by all pipelined stages in the circuit. Because two stages cannot co-exist, the pipeline has to be flushed out before the next computational stage begins. As our system has a pipeline delay of 43, a total of 645 clock cycles (43 × 15 stages) may have to be wasted for the delay. In our system, however, the control signals are also pipelined, as shown in Figure 9(b), and each pipeline stage refers to the respective control signal in the control pipeline.

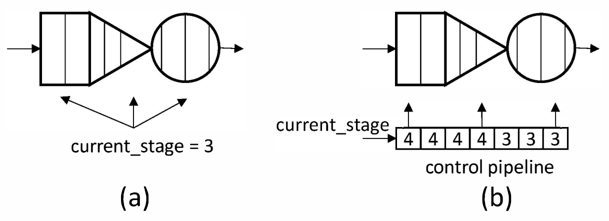

Fig. 9. (a) Global control signal and (b) use of the control pipeline. By pipelining the control signals, the computational stages can be executed without delays between them.

Using this scheme, the next computational stage is started immediately after the previous one is finished, thereby avoiding a delay. Second aspect that could delay the computational is the fact that the Receptor Unit has to be prepared before it produces an output. Because the Receptor Unit scans the inner area of the input image, at least five lines of the first part of the input image has to be read into the buffer of the Receptor Unit in advance in order to scan 5 × 5 sized local receptive fields. This may delay the stages if only a single Receptor Unit is used. Therefore two Receptor Units are used alternately in our system, i.e., when one receptor is used for reading, the other is being prepared by reading the feature map memory for the next stage.

In spite of using delay-avoidance schemes, some delays are unavoidable owing to the data dependencies between two consecutive stages. There is a total delays of 83 clock cycles per classification in our system.

D. *Multi-Category Recognition*

In this paper, multi-category object recognition refers to the ability to recognize objects in different categories at the same time. For example, with the multi-category recognition capability, a human face and hand gestures can be recognized simultaneously, as shown in Figure 10.

We implement multi-category recognition through a simple solution: using different weight sets in alternating video frames. Multiple weight sets that are trained independently of each other are stored in the weight memories in a different memory space, and the MSB of the address port of each weight memory is switched by the Control Unit. For example, weight sets for categories 1 and 2 are selected in the odd and even frames, respectively. For a multi-category recognition with *n* categories, the recognition frame rate has to be reduced to 1/*n* of a single-category recognition.

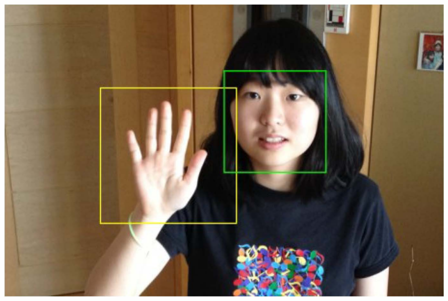

Fig. 10. Multi-category object recognition. Using different weight sets in alternating video frames, objects of different categories can be recognized.

IV. Implementation Results

The system is implemented on a Xilinx KC705 evaluation board with an add-on IMAGEON card from Avnet, Inc., as shown in Figure 11. The major components used in our system are a Kintex 7 FPGA on the KC705 board, and an ADV7611 video decoder chip and an ADV7511 video encoder chip on the IMAGEON card. Our system operates at a clock frequency of 250 MHz.

A. *Precision and Computational Errors*

Fixed point arithmetic operations are used throughout the system. Connection weights are stored in memory in a 25 + 2 bit representation in which a 25-bit signed integer is coupled with 2 additional exponent bits. The exponent bits are used to bit-shift the multiplication results for large connection weight values. The use of the exponent bits helps increase the dynamic ranges of the connection-weights without using computationally expensive floating-point operators. Multipliers

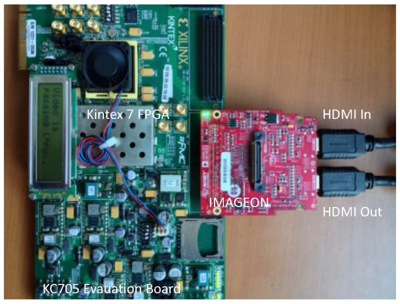

Fig. 10. Photograph of the developed system.

with 25 × 25 inputs are used in the Synapse Units and 33-bit adders are used in the Dendrite Units. A piecewise second order approximation scheme [4] is used for computing the sigmoid functions in the Soma Units. For this scheme, two multipliers and three 128 × 25 bit look-up tables are used in each Soma Unit. The average computational error of the sigmoid function is 2.4×10^{-7}, and the total error accumulated from the input (image block) to the output (classification result) is 1.1×10^{-6} on average compared to the results of double-precision arithmetic. Therefore, the impact of using fixed-point arithmetic on the classification results is negligible.

The results of the original source code that our system is based upon are shown in Table II. These results are comparable to other recently reported state-of-the-art results [5].

TABLE II. CLASSIFICATION ERRORS OF ORIGINAL CODE

Data Set	Mean Error (%)
MNIST Handwritten digits	1.01
Face detection on LFW face database [6]	1.13
Gender classification on LFW database	14.2

B. *Resource Utilization*

Table III shows the resource utilization of the FPGA in our system.

TABLE III. FPGA RESOURCE UTILIZATION

Resource	Utilization	Available	Utilization (%)
LUT	42,616	203,800	20.1
Block RAM	31.5	445	7.1
DSP48	326	840	38.8

The resources in the table include video pre-processing and post-processing modules and the CNN core. All memories in the CNN core are synthesized as distributed memories which consume the LUTs. Block memories are used only for buffers in the image resizers. Approximately one-third of the total resources provided by a Kintex 7 FPGA are used.

DSP slices are synthesized for all multipliers, which account for 38.8% of DSP48 shown in Table III. Because this number is not comparable to other resources, we recompiled the system using all multipliers synthesized to the LUTs, and compared the resources used for the multipliers with the rest of the resources in the system, as shown in Table IV.

TABLE IV. MULTIPLIER COMPOSITION RATE

	LUT Count[1]	Rate (%)
Resources used for 150 multipliers	101,100	66.3
Resources used for other components	51,307	33.7
Total resources used for system	152,407	100.0

1. Synthesized without using DSP slices.

C. *Computational Speed*

The CNN core in our system classifies one image block in 1,463 clock cycles or in approximately 5.9 μs. Therefore the speed of the system is 170,881 classifications per second. This speed is compared with the speed of a PC running the original CNN code in Table V.

TABLE V. SPEED COMPARISON

	Batch size	Speed (classifications/s)	Speedup
Our system	1	170,881	94x
PC[1]	50	1,806	1x

1. Intel i5-3337U CPU 1.8GHz.
Similar speed was measured in [1] for this open source code.
PC computes with double-precision floating-point arithmetic.

D. *Power Consumption*

The actual power consumption of the KC705 board can be calculated by measuring the voltages across the voltage sense registers on the board using a digital multimeter. The total power consumption of the board measured in this manner was 3.1 W. This can be compared with a GPU, which consumes more than 100 W even during an idle state.

V. DISCUSSION

A. *Efficiency Estimation*

It would not be surprising if a supercomputer can compute a CNN faster than any other type of computer. In evaluating the system efficiency, the computational resources used in the system have to be considered together with the computational speed. From this point of view, a computational efficiency can be defined as

$$E = \text{Computational\_speed} / \text{Resource\_used},$$

where E denotes the system efficiency.

Multipliers are the most resource-intensive components in a fixed-point computational system. For example, a 16-bit multiplier requires 9,000 transistors, whereas a 16-bit adder and a 16-bit multiplexer need only 448 and 96 transistors respectively [7]. Because the CNN algorithm inevitably requires a given amount of multiplications as stated in Section II, the utilization of the multipliers can be an approximate measure for the system efficiency. Consider an ideal system composed only of the most efficient multipliers, which are fully utilized without an idle time, thereby removing the computational workloads as quickly as possible. As this ideal system can be described by three independent factors, the following measure can be formulated:

$$E \approx R_c \times R_u \times E_{mul}$$

where R_c is the compositional rate of the resources used for the multipliers over the total resource used, R_u is the average

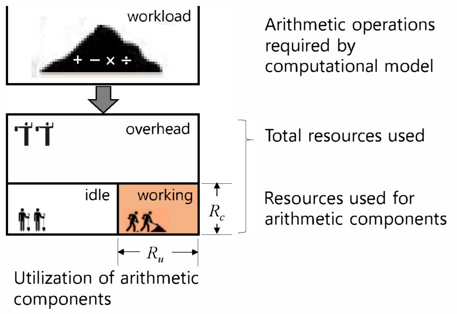
workload
+ − × ÷
Arithmetic operations required by computational model
overhead
Total resources used
idle
working
R_c
Resources used for arithmetic components
R_u
Utilization of arithmetic components

Fig. 11. Picture illustrating the evaluation of system efficiency. The efficiency depends on the proportion (R_c), utilization (R_u), and the cost (E_{mul}) of workers.

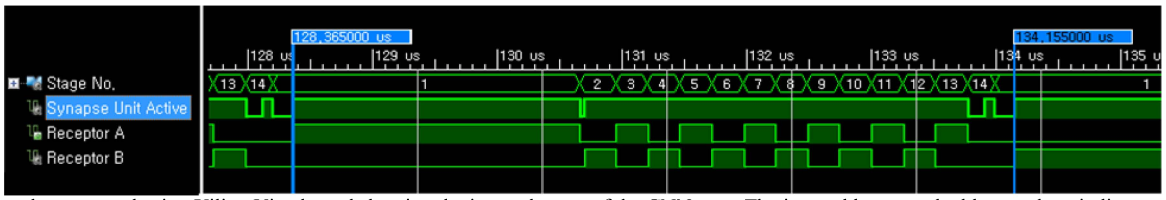

Fig. 12. Snapshot captured using Xilinx Vivado tool showing the internal states of the CNN core. The interval between the blue markers indicates one classification cycle (1463 clock cycles). The gaps between the active states in the Synapse Units indicate delays incurred by the data dependencies between stages. The inputs are supplied by using two receptors alternately. All 150 multipliers in the Synapse Units produce their outputs for approximately 94% of the time (clock cycles). The same rate can be calculated by comparing the number of multiplications minimally required for each classification, 86250 + 117600 + 1920 = 205770 as explained in Section II, with the peak performance of 150 multipliers in 1463 clock cycles, 150 × 1463 = 219000.

utilization rate of the multipliers, and E_{mul} is the efficiency of the multipliers compared to the most efficient multiplier, as illustrated by Figure 11. In an ideal system, all three factors are 1. Note that such an ideal system cannot exist in practice as the multiplications are not the only computational requirements and there should be other components such as adders required for weighted-sum. In addition, no system can achieve better efficiency than this ideal system.

In our system, all 150 multipliers in the Synapse Units produce their outputs approximately 94% of the time (clock cycles), as shown in Figure 12. Therefore, R_c and R_u in our system are 0.66 (Table IV) and 0.94, respectively. The third factor in our system, E_{mul}, refers to the efficiency of the multipliers in the FPGA as compared to the most efficient multipliers. A 16-bit fixed-point multiplier can be implemented using 9,000 transistors (A) in a state-of-the-art technology [7] and 280 LUTs (B) in Xilinx FPGA [8], and a six-input LUT can be implemented using approximately 2,100 transistors (C) [9]. Therefore, E_{mul} of FPGA systems can be estimated as

$$E_{mul} \approx A \,/\, (B \times C) = 0.015.$$

Note that this low rate of E_{mul} originates from using an FPGA, which is a programmable general-purpose hardware. Suppose our system is implemented on a custom chip where the state-of-the-art multipliers can be directly implemented. In this case, E_{mul} may approach to 1. The efficiencies of an ideal system, our FPGA system, and our system implemented on a custom chip are compared in Table VI.

TABLE VI. ESTIMATED EFFICIENCY COMPARISON

	R_c	R_u	E_{mul}	E
Ideal System	1.0	1.0	1.0	1.0
Our FPGA system	0.66	0.94	≈ 0.015	≈ 0.01
Our system on custom chip	0.66[1]	0.94	≈ 1.0	≈ 0.62

1. Same composition rate as our FPGA system is assumed.

An efficiency comparable to the ideal system could be achieved when our system is implemented on a custom chip.

B. Scalability

In principle, each video frame can be computed separately with multiple systems because the computations of the video frames are independent of each other. Therefore the computational speed can be linearly increased with more hardware resources.

Figure 13 compares the resources used in our system with the resources provided by modern FPGAs. This shows that a considerable speedup might be achieved using a single high-end FPGA chip. The increased speed can be used to recognize a higher-definition video sequence or in running CNNs with larger configurations.

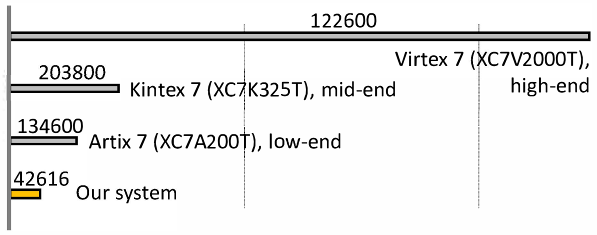

Fig. 13. Comparison of LUT resources used in our system and provided by modern FPGAs

VI. Conclusion

In this paper, a CNN system for recognizing objects in a real-time video stream was implemented, and various schemes to improve the efficiency of the system were presented. This system can be used in various applications such as video surveillance, automotive control, or digital signage.

References

[1] Palm, Rasmus Berg. "*Prediction as a candidate for learning deep hierarchical models of data*." Technical University of Denmark, Palm 25 (2012).

[2] Palm, Rasmus Berg. *A Matlab toolbox for Deep Learning*. Available: https://github.com/rasmusbergpalm/DeepLearnToolbox

[3] Ahn, Byungik. "*Computation of deep belief networks using special-purpose hardware architecture*." Neural Networks (IJCNN), 2014 International Joint Conference on. IEEE, 2014.

[4] Tommiska, M. T. "*Efficient digital implementation of the sigmoid function for reprogrammable logic*." Computers and Digital Techniques, IEE Proceedings-. Vol. 150. No. 6. IET, 2003.

[5] Dago-Casas, Pablo, et al. "*Single-and cross-database benchmarks for gender classification under unconstrained settings*." Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on. IEEE, 2011.

[6] Huang, Gary B., et al. *Labeled faces in the wild: A database for studying face recognition in unconstrained environments*. Vol. 1. No. 2. Technical Report 07-49, University of Massachusetts, Amherst, 2007.

[7] Ghasemizadeh, Habib, Amir Fathi, and Akbar Ghasemizadeh. "*High Speed 16× 16-bit Low-Latency Pipelined Booth Multiplier*." Electrical and Electronic Engineering 2.3 (2012): 121-127.

[8] Xilinx Inc., *LogiCORE IP Multiplier v12.0*, 2014.

[9] Koch, Dirk. *Partial Reconfiguration on FPGAs: Architectures, Tools and Applications*. Vol. 153. Springer Science & Business Media, 2012.