Ultra Power-Efficient CNN Domain Specific Accelerator with 9.3TOPS/Watt for Mobile and Embedded Applications

Baohua Sun, Lin Yang, Patrick Dong, Wenhan Zhang, Jason Dong, Charles Young
Gyrfalcon Technology Inc.
1900 McCarthy Blvd. Milpitas, CA 95035
{baohua.sun,lin.yang,patrick.dong,wenhan.zhang,jason.dong,charles.yang}@gyrfalcontech.com

Abstract

Computer vision performances have been significantly improved in recent years by Convolutional Neural Networks (CNN). Currently, applications using CNN algorithms are deployed mainly on general purpose hardwares, such as CPUs, GPUs or FPGAs. However, power consumption, speed, accuracy, memory footprint, and die size should all be taken into consideration for mobile and embedded applications. Domain Specific Architecture (DSA) for CNN is the efficient and practical solution for CNN deployment and implementation. We designed and produced a 28nm Two-Dimensional CNN-DSA accelerator with an ultra power-efficient performance of 9.3TOPS/Watt and with all processing done in the internal memory instead of external DRAM. It classifies 224x224 RGB image inputs at more than 140fps with peak power consumption at less than 300mW and an accuracy comparable to the VGG benchmark. The CNN-DSA accelerator is reconfigurable to support CNN model coefficients of various layer sizes and layer types, including convolution, depth-wise convolution, short-cut connections, max pooling, and ReLU. Furthermore, in order to better support real-world deployment for various application scenarios, especially with low-end mobile and embedded platforms and MCUs (Microcontroller Units), we also designed algorithms to fully utilize the CNN-DSA accelerator efficiently by reducing the dependency on external accelerator computation resources, including implementation of Fully-Connected (FC) layers within the accelerator and compression of extracted features from the CNN-DSA accelerator. Live demos with our CNN-DSA accelerator on mobile and embedded systems show its capabilities to be widely and practically applied in the real world.

1. Introduction

Computers have been divided into servers, desktop computers and embedded computers[10]. The basic building blocks for application domain-specific integrated computers include the input, output, data path, memory and control - as with an ordinary computer[5]. Typical performance metrics of the computer system include the execution time and power consumption[10]. Domain Specific Architectures (DSA) are the only path forward for improved performance and energy efficiency given the end of Moores Law and Dennard scaling[11].

Cellular Neural Networks or Cellular Nonlinear Networks have been applied to many different fields and problems including, but not limited to, image processing since 1988[4]. However, most of the prior art approaches are either based on software solutions (e.g., Convolutional Neural Networks (CNN)[17, 16, 32, 24, 9], Recurrent Neural Networks[21], etc.) or based on hardware that are designed for other purposes (e.g., graphic processing, general computation, etc.). As a result, prior CNN approaches are too slow in term of computational speed and/or too, expensive, thereby impractical for processing large amounts of imagery data. The imagery data can be from any two-dimensional signals (e.g., a still photo, a picture, a frame of a video stream, etc.)

There has been much interest in designing ASIC (Application-Specific Integrated Circuit) chips for computer vision tasks to efficiently accelerate CNN. Chen, Krishna, and et al. designed and made 65nm Eyeriss with the peak performance at 42GOPS at 200MHz core clock and 60MHz link clock, resulting in a frame rate of 34.7fps on the five convolutional layers in AlexNet and a measured power of 278mW at 1V[3]. The power efficiency of Eyeriss equals 151GOPS/Watt. Han, Liu, and et al. designed and made EIE with a processing power of 102 GOPS working directly on a compressed network, corresponding to 3 TOPS on an uncompressed network, and processes FC layers of AlexNet with a power dissipation of only 600mW[8]. The power efficiency of EIE equals 170GOPS/Watt, or corresponding to 5TOPS/Watt on an uncompressed network. Chen, Du, and et al. designed and made 65nm DianNao capable of performing 452 GOP/s at 485mW[1]. The power efficiency of

Diannao equals 932GOPS/Watt. Du, Fasthuber and et al. designed and made a 65 nm CMOS technology with a peak performance of 194 GOP/s at 320.10 mW[6]. The power efficiency of ShiDianNao equals 606GOPS/Watt. Chen, Luo and et al. designed and made 28nm 15.97Watt 5.58 TeraOps/s for 16-bit operation[2]. The power efficiency of DaDianNao equals 349.4GOPS/Watt. Jouppi, Young and et al. designed and made TPU with 92TOPS typically using 40 Watts[14]. The power efficiency of TPU equals 2.3TOPS/Watt.

We designed a Convolutional Neural Networks Domain Specific Architecture (CNN-DSA) accelerator for extracting features out of an input image. It processes 224x224RGB images at 140fps with ultra power-efficiency of 9.3TOPS/Watt and peak power less than 300mW. The die size of CNN-DSA accelerator is 7mm by 7mm. This architecture mainly focuses on inference, rather than training. The designed CNN-DSA contains a plurality of identical CNN processing engines operatively coupled to an input/output (I/O) data bus. A CNN processing engine controller is configured on the IC for controlling various operations of the CNN processing engines. Each CNN processing engine includes a CNN processing block, a first set of memory buffers for storing imagery data and a second set of memory buffers for storing filter coefficients.

In the sections below, we will first provide our hardware design for CNN-DSA accelerator, and then the hardware performance metrics measured on test bench, followed by its implementation on mobile and embedded platforms with live demos. After that, we will show that our CNN-DSA can support the majority of CNN layer types through algorithm designs on the accelerator, and also proved that ResNet[9], MobileNet[12] and ShiftNet[27] are special cases of VGG[24]-Type model. For the convenience of this paper, we define the CNN model as VGG-Type CNN models if the layer types only consist of ReLU, 3x3 convolution, and max pooling. Lastly, we design algorithms and show the experimental results on the CNN-DSA accelerator for real-world applications on mobile and embeded platforms, and even very low-end MCUs (Microcontroller Unit). The target is to reduce the computation outside the CNN-DSA accelerator by fully utilizing our low power and ultra efficient CNN-DSA accelerator, including implementation of FC (Fully Connected) layers inside CNN-DSA accelerator, and compression of features extracted from CNN-DSA accelerator.

2. CNN-DSA hardware design

For convolution on 2-D image, both the convolution filters and input data in each channel are two dimensional. We design a matrix architecture of CNN-DSA Engine (CE)[30] as shown in Figure 1. CNN processing engines extract features out of an input image by performing multiple layers

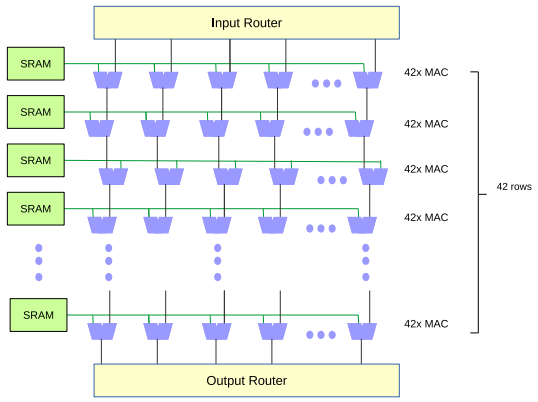

Figure 1. CNN-DSA processing engine

of 3x3 convolutions with rectifications and 2x2 pooling operations. It includes a CNN processing block, a first set of memory buffers for storing imagery data and a second set of memory buffers for storing filter coefficients. The CNN processing block is configured to simultaneously perform 3x3 convolutions at MxM pixel locations using received imagery data and corresponding filter coefficients, In Figure 1 we show our implementation of $M = 14$. Imagery data represents a $(M+2)$-pixel by $(M+2)$-pixel region of the input image. The CNN processing block further performs rectification and/or 2x2 pooling operations as directed. When two or more CNN processing engines are configured on the integrated circuit (IC), the CNN processing engines connect to one another as a loop via a clock-skew circuit for cyclic data access.

In Figure 2 we show the block diagram of our CNN-DSA which contains a plurality of identical processing engines operatively coupled to the input/output (I/O) data bus. The CNN-DSA contains an NE number of CNN processing engines connected in a loop via a clock-skew circuit. In Figure 2 we show our implementation of $NE = 16$. The CNN processing engines are connected to one another to form a cyclic data access loop via a clock-skew circuit. The clock-skew circuit enables the CNN processing engine to receive imagery data from a first neighbor CNN processing engine while sending its own imagery data to a second neighbor CNN processing engine. A CNN processing engine controller is also configured on the IC for controlling operations of the CNN processing engine. The imagery data and filter coefficients are arranged in a specific scheme to fit the data access pattern that the CNN based digital integrated circuit requires to operate. The specific scheme is determined based on the number of imagery data, the number of filters and the characteristics of the CNN based digital IC, such as the number of CNN processing engines, the connection direction of clock-skew circuit and the number of the

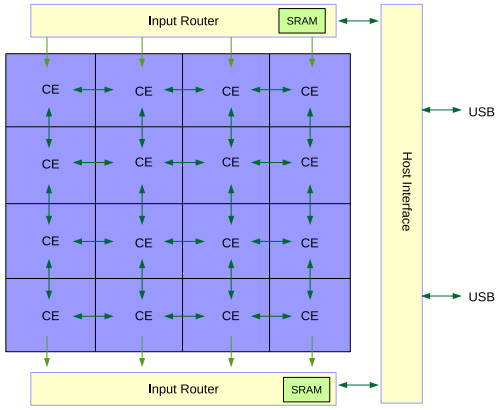

Figure 2. CNN-DSA block diagram

I/O data bus. The method for arranging imagery data and filter coefficients includes the following steps and actions: (a) determining the number of imagery data groups required for storing NIM sets of imagery data in the CNN processing engines, with each imagery data group containing NE sets of the NIM sets of imagery data, where NE is the number of CNN processing engines connected in a loop via a clock-skew circuit, and NIM is a positive integer; (b) circularly storing the NE sets of the imagery data of each imagery data group in the respective CNN processing engines; (c) repeating (b) for the remaining imagery data groups; (d) determining the number of filter groups required for storing all filter coefficients for NF number of filters in the CNN processing engines, with each filter group containing NE sets of filter coefficients and said each filter group being further divided into one or more subgroups with each subgroup containing a portion of the NE sets that correlates to a corresponding group of the imagery data groups, where NF is a positive integer; (e) storing the portion of the NE sets of filter coefficients in a corresponding one of the CNN processing engines, with the portion of filter coefficients being arranged in a cyclic order for accommodating convolution operations with imagery data received from an upstream neighbor CNN processing engine; and (f) repeating (e) for the remaining subgroups; and (g) repeating (e) and (f) for the remaining filter groups.

The CNN-DSA contains 16x42x42=28224 MAC units, and 9MB SRAM to hold the coefficients. All processings are in memory by CNN Domain Specific Floating Point (CNN-DSFP), enabling it to operate without external DRAM. CNN-DSFP has totally 9-bits total for image data and activations, where 5 bits for mantissa and 4 bits for exponents. Filter coefficients are allocated 15-bits total, the mantissa 12 bits, exponents 2 bits, and 1 bit sign.

Our design of the CNN-DSA accelerator is optimized for VGG-Type CNN models. Specifically, it only consists of convolution kernels of size 3x3. In the next section, we will prove that our design can implement the majority of CNN models, including but not limited to, Resnet, MobileNet and ShiftNet.

2.1. Hardware system implementation with CNN-DSA as coprocessor

When building a hardware system, the CNN-DSA accelerator serves as a coprocessor. The host processor could be a mobile or embeded processor, or other host CPU, which serves as the controller. The interface between the host processor and the CNN-DSA accelorator is USB, EMMC or PCIe. To run inference with the CNN-DSA accelerator, the host processor first loads weights and instructions of the CNN model configurations to the on-chip SRAM. After that, the host processor sends image data to the CNN-DSA accelerator, and the data goes through the convolution processed by the coefficients loaded in the CNN-DSA accelerator. When the CNN-DSA accelerator finishes convolution, it signals the host processor and sends back the convolution output to the host processor for next step processing.

3. CNN-DSA hardware performance measurments

3.1. Speed and power consumption measurement

The CNN-DSA accelerator speed and power consumption measurements are shown in Figure 3. The equipments used include: A test board with CNN-DSA accelerator, a Fluke 179 DMM core voltage monitor, a Protek 506 DMM current monitor in Amps, an Agilent E3630A DC Power Supply which supplies core voltage power at 0.90 Volts normal, a Tektronix TDS 3054 Digital Phosphor Oscilloscope, a FTDI USB bridge UMFT601x, and Dell PowerEdge T30 PC with Ubuntu.

Figure 3 shows that, we process the RGB image of size 224x224 at a speed of 142.86fps when the CNN-DSA accelerator is set at 66MHz. Based on the readings from meters, the voltage is 0.904 volts, and the current is 0.15 Amps. Thus the power consumption is 0.904x0.15=0.1356W. Considering the different temperature conditions, we estimate that the power consumption in worst temprature conditions should be under 400mW, so the corresponding power efficiency is 28224x2x66M/0.4 = 9.3 TOPS/Watt. We also measured with frequency set to 50MHz, and the power consumption is less than 300mW.

3.2. Accuracy on ImageNet[23] dataset

Table 1 compares the accuracy and model size of the CNN models in our accelerator with other published models. For fair comparison, we only compared the convolutional layers sizes, without FC layers. Here Gnet-1 is our compressed model using CNN-DSFP with same 13 layer

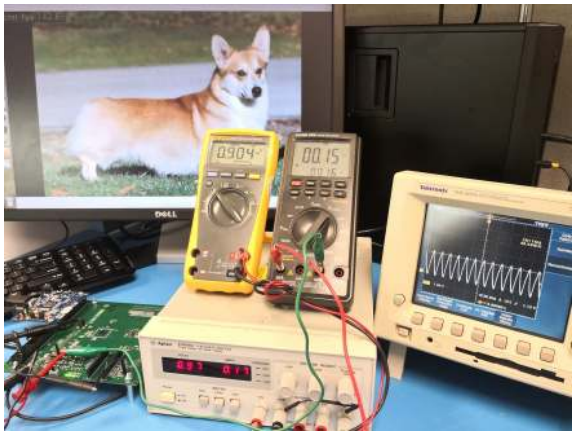

Figure 3. CNN-DSA accelerator speed and power consumption measurement. It is recommanded to zoom in this figure to see processing speed in fps in the top-left portion of the photo.

Model	Accuracy Top-1	Size(MB)
AlexNet[16]	57.1	15.0
VGG-16[24]	**71.5**	58.9
Googlenet[25]	69.8	23.2
ShuffleNet-2x[33]	70.9	19.2
1.0 MobileNet-224[12]	70.6	15.9
Compact DNN[28]	68.9	13.6
Gnet-1(ours)	67.0	5.5
Gnet-2(ours)	58.1	**2.8**

Table 1. Comparison of model size and accuracy on ImageNet[23].

convolutional architecture as VGG, and the FC layers are deployed outside the accelerator. In Section 6, we will show that FC can be implemented within our CNN-DSA accelerator as well. Gnet-2 is almost the same architecture as Gnet-1 except that we halved the number of channels for layers 1 through 10, and kept the same number of 512 channels for layer 11 through 13 . We can see that Gnet-1 compressed the original VGG model by more than 10x with the cost of only a 4.5% decrease of Top-1 accuracy. And Gnet-2 further compressed the VGG model by more than 20x and the accuracy is still 1% better than AlexNet.

4. CNN-DSA accelerator deployed on mobile and embedded platforms

Figure 4 shows an image classification demo on a mobile platform. We use a Qualcomm 820 development board with an Android system. It has 8 cores, with 4 at 1.6GHz and 4 at 2.1GHz. The memory is 2.8GB and storage is 19GB. The interface to the CNN-DSA accelerator is USB 3.0. Figure 5 shows our Chinese handwritten OCR (Optical Character Recognition) demo on the same embedded platform.

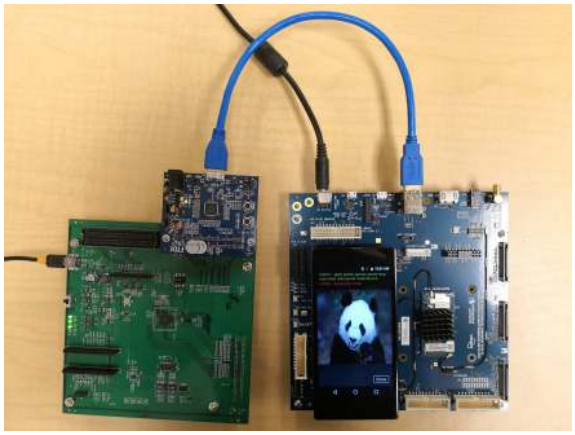

Figure 4. Image recognition on CNN-DSA accelerator connected to mobile platform.

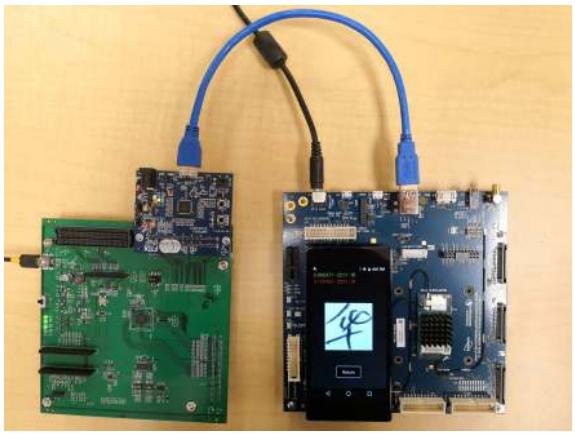

Figure 5. OCR on CNN-DSA accelerator connected to mobile platform.

5. Algorithm design on CNN-DSA to support various CNN models

In this section, we design algorithms for CNN-DSA to implement various CNN models. And we prove that the majority of CNN layer types can be replaced and implemented by VGG-Type layers with 3x3 convolutional filters.

5.1. ResNet is a special case of VGG-Type model[20]

In addition to convolutional layers, activation layers and pooling layers, ResNet[9] requires shortcut layers. It typically contains an original convolutional layer W1 along with a second original convolutional layer W2 followed by element-wise add operations, as shown in Figure 6. Since 3x3 convolutional operations are conducted with very fast speed in the CNN-DSA accelerator, it would therefore be desirable to implement a deep neural network using 3x3 convolutional filter kernels to replace such operations in a

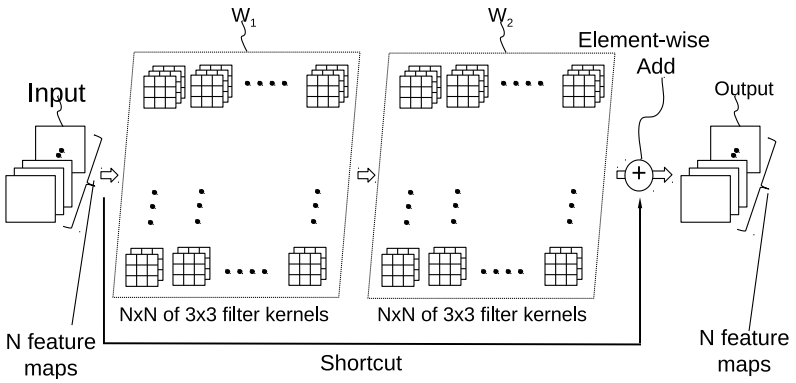

Figure 6. Example of a shortcut layer.

CNN-DSA. By performing a network surgery and modifying the architecture of the trained ResNet, we can replace the shortcut layer in Figure 6 by a set of three particular convolutional layers of multiple 3x3 filters as shown in Figure 7. In Figure 7, first particular convolutional layer contains 2NxN of 3x3 filter kernels formed by placing the original NxN of 3x3 filter kernels of W1 in the left side and NxN of 3x3 filter kernels of an impulse response (or, identity-value) convolutional layer P1 in the right side. Each of the 3x3 kernels of P1 contains numerical value "0" except for the kernels located on the diagonal of the NxN kernels. Each of the diagonal kernels contains the numerical value "0" in each of the eight perimeter positions and "1" in the center position. All off-diagonal kernels contains nine "0". Because of this, the first particular convolutional layer is configured for N-channels or an N-'feature maps' input with 2N-channels output. The second particular convolutional layer contains 2Nx2N of 3x3 filter kernels formed by placing NxN of 3x3 filter kernels of the second original convolutional layer W2 in the upper left corner and NxN of 3x3 filter kernels of impulse response convolutional layer P1 in the lower right corner, and two zero-value convolutional layers P0 in either off diagonal corner. The zero-value convolutional layers P0 contains NxN of 3x3 filter kernels with all zero numerical values in each of the 3x3 kernels. As a result, the second particular convolutional layer is configured for a 2N-channel input and 2N-channel output. The third replacement convolutional layer contains Nx2N of 3x3 filter kernels formed by two impulse response convolutional layer P1 each containing NxN of 3x3 filter kernels in a vertical stack. As a result, the third particular convolutional layer is configured for a 2N-channel input and N-channel output. It is trivial to see that the output from Figure 7 is equal to that from Figure 6. By repeating the above mentioned method several times for all the shortcut in Resnet, we can convert all the shortcut into VGG type layers. Thus, ResNet is a special case of VGG-Type network. One point to mention here: we can only deploy shallow or narrow ResNet model due to the memory limitation of the accelerator.

5.2. MobileNet is also special case of VGG-Type model[29]

In additional to convolutional layers, activation layers and pooling layers, MobileNet[12] requires operations of depthwise separable layers. It typically contains a combination of a depthwise convolutional layer followed by a pointwise convolutional layer as shown in Figure 8. Input and depthwise convolution contain P feature maps or channels, while the output contains Q feature maps. P and Q are positive integers. Pointwise convolutional layer contains QxP of 1x1 filter coefficients. Similarly, we change the architecture of the trained MobileNet by replacing the depthwise convolution with the combination of multiple 3x3 filters, and set the weight to a specific value as shown in Figure 9, such that these filters are equal to the functionality of a depthwise convolution. Figure 9 shows an example of the first replacement convolutional layer, which contains PxP number of 3x3 filter kernels formed by placing Px1 number of 3x3 filter kernels (W1, W2, ..., WP) of the depthwise convolutional layer on respective diagonal locations. The remaining off-diagonal locations of the PxP number of 3x3 filter kernels are filled by zero-value 3x3 filter kernels. As a result, the first replacement convolutional layer is configured for replacing the depthwise convolutional layer. The second replacement convolutional layer shown in Figure 9 contains QxP number of 3x3 filter kernels formed by placing the QxP number of 1x1 filter coefficients Y11, Y21, Y31, ..., YQ1, Y12, Y22, ..., YQ2, ..., Y1P, Y2P, ..., YQP of the pointwise convolutional layer in the center position of the respective QxP number of 3x3 filter kernels, and numerical value zero in eight perimeter positions. As a result, the second replacement convolutional layer is configured for replacing the pointwise convolutional layer. Similarly, we utilize many 3x3 filters to implement the equivalent depth wise convolution in the deep neural network, and finally replace the depthwise convolution with the VGG-Type layers. Us-

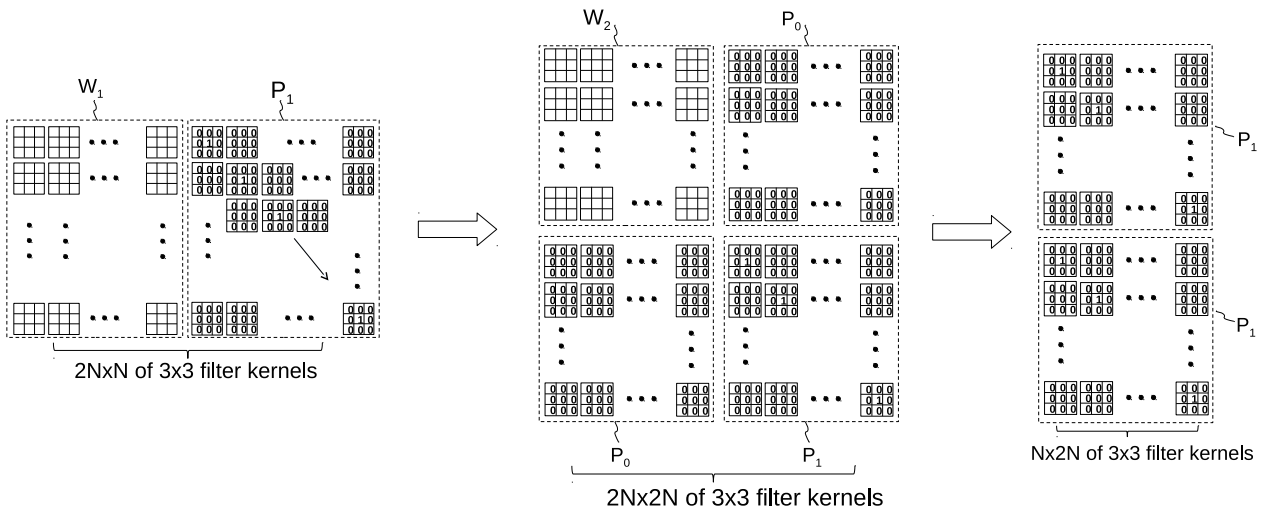

Figure 7. Implement Resnet in VGG-Type 3x3 convolutional filters.

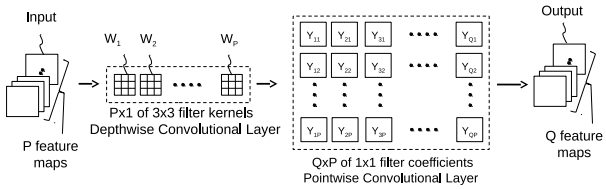

Figure 8. Depth-wise convolution followed by point-wise convolution.

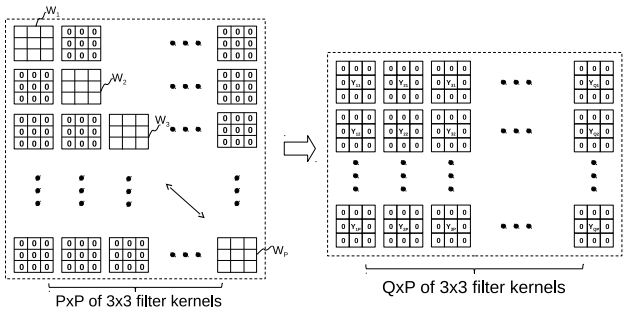

Figure 9. Implement MobileNet with VGG-Type 3x3 convolutional filters.

ing the same scheme, Inception[25] and ShiftNet[27] can also be proved as special cases of the VGG-Type model.

6. Algorithm design and experiments on CNN-DSA for various application scenarios

Considering CNN-DSA accelerator applications in real-world scenarios, we design algorithms that best utilize the fast speed and low power consumption of CNN-DSA accelerator to best fit mobile and embedded platforms.

6.1. Basic model applications with CNN-DSA accelerator and experiment result on CASIA-HWDB[31]

Traditional deep neural network architectures for 2-D inputs generally have two parts: convolution layers and FC layers. Notably, convolution layers require less storage for holding coefficients but require significant amounts of computation (for example, VGG16 [24] requires 15 GFLOPs) for Mult-Adds due to the repeated applications of filters. On the contrary, FC layers require less computation for Mult-Adds but necessitate a significant amount of storage (for example, VGG16 requires storage for 123M FC coefficients) for storing coefficients due to inner-products.

VGG-Type models are optimally accelerated in the CNN-DSA. The basic application mode is to use the CNN-DSA accelerator as the feature extractor, so most of the heavy-lifting convolutional computations are taken care of by the power efficient accelerator. The output features from the CNN-DSA accelerator connect to the external devices for the FC computation. The FC which is typically handled by either cpu or gpu, consumes more storage and memory but little computation and power. We show one experiment for this basic application mode on the CNN-DSA accelerator. The data set we used here is the offline handwritten isolated character recognition CASIA-HWDB[31] released by CASIA (the Institute of Automation of Chinese Academy of Sciences (CASIA)). We used the designed training and testing data sets of unconstrained offline Chinese handwriting database level-1 set of GB2312-80 with 3755 Chinese characters to train our model and evaluate accuracy performance. We used the same architecture as VGG-16[24] and trained the CNN-DSFP fixed point model to load it into our CNN-DSA accelerator, and the fully connected is processed

by mobile device with Android system, as shown in Figure 5. We obtained 94.95% accuracy on the testing data running low power CNN-DSA accelerator connected to a mobile device. The winning model of the ICDAR[31] competition on this data set attained a 94.77% accuracy in 2013 [31] and a 92.18% accuracy in 2011 [26].

6.2. Mobile friendly: compress extracted features directly from CNN-DSA accelerator[19]

For tasks like object classification and face authentication, feature extraction from input is useful. Traditional methods of face feature extraction use one or several layers of the FC layers in the CNN network. Through inner product computation, the output of FC layer dimension would be reduced from convolution layers of very high dimensional data. For example, the convolution output after pooling in VGG-16 gives a 512x7x7=25088 high dimensional feature which the FC layer will project into a relatively low dimensional space, like 4096, or 1024, or 128, as done by[22]. But the disadvantage of this feature extraction is that the model size remains very large, especially for mobile devices. For example, the FC layer connecting convolution layers in VGG-16 will require 25k*4k=100M parameters. At the same time, the runtime performance will be low due to the high computation complexity.

Our CNN-DSA accelerator can load the entirety of the VGG net into the chip, and completes the heavy-lifting convolution computations locally. We design a method of extracting features directly from convolution layers in the CNN-DSA with a very small number of channels. For a given image input, the output from the CNN-DSA accelerator completes feature extraction without using FC layers. After our invention method is implemented in our CNN-DSA accelerator, it requires minimum hardware support for feature extraction. Feature vectors are obtained directly from the convolution layer output, instead of from the FC layers. We also use very small number of channels from the last convolution layer. Thus, the output from our CNN-DSA accelerator requires minimum hardware requirement from outside the accelerator.

We show our method implemented in the CNN-DSA in a face authentication experiment. We have a collected data set with 21278 different people, each with at least 10 pictures. First, we trained our model using the same model architecture as VGG using our CNN-DSFP fixed point. Then we changed the number of channels for 5_3 from 512 to 8,4,2,1, but still connected to the same FC layers of fc6 of size 4096, fc7 of size 4096, and fc8 of size 21278. After training, the convolutional layers of the model are loaded into CNN-DSA, and can be used directly for face feature extraction from the output of convolutional layer 5_3. The FC layers are no longer used. Input face image will be feature-extracted directly from the output of the CNN-DSA. Thus, the extracted feature will only be a vector of 49x1 (7x7x1) for the model with 5_3 set to 1 channel. This low dimensional data is feasible for computation in very low end devices like MCUs (Microcontroller Unit).

#Channels	LFW Verification Accuracy
512	**95.57**
8	94.92
4	94.76
2	94.42
1	94.25

Table 2. Compress the number of channels of VGG 5-3 layer

We use the LFW [13] data set for face authentication performance evaluation. For preprocessing before performance evaluation, we only made a cropping using dlib[15] face detection on the original LFW data set. The same preprocessing used in VGG Face[22] gives an accuracy of 92.83% with no alignment and no embedding. Table 2 shows the result on LFW unrestricted setting with a compressed number of channels from CNN-DSA. We have decreased the number of channels all the way down from 512 to 1 but accuracy performance degradation is negligible. In this way we compressed the size of extracted features. The trade off of using this method is that the robustness for face verification degrades a little bit. However, the computaion needed in addition to the accelerator is minimized.

6.3. Implement FC inside CNN-DSA accelerator[18]

FC computations are mainly composed of inner-products, which can easily be replaced by convolution layers if the input data size of the convolution layers is the same as that of the inner-product. However, our CNN-DSA accelerator only supports 3x3 convolution kernel, because large kernel size convolutional layers are not efficient in matrix products. As pointed out by the VGG paper[24], two connected 3x3 convolution kernels have the same reception field as that of a 5x5 kernel, and three connected 3x3 kernels can have the same reception field as that of 7x7 kernel. With operations of convolutional layers performed very fast in our CNN-DSA accelerator, the computation bottleneck in deep learning networks is in FC which are processed by CPU or GPU. We provide a solution for utilizing CNN-DSA accelerator in classification tasks without FC. We use multiple layers of 3x3 kernels to approximate larger-sized kernels. By using L layers of connected 3x3 kernels, we can approximate the inner-product for an input feature map of (2*L+1)*(2*L+1) size. For example, for a feature map input of size 7x7, we can use L=3 layers of connected 3x3 convolution kernels to approximate the inner-product. For a feature map input of size 7x7, applying a 3x3 convolution without padding will output a 5x5 feature map. Applying

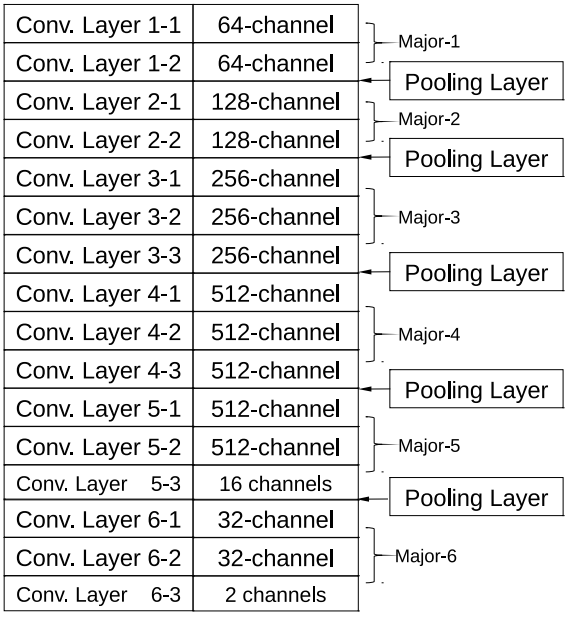

Figure 10. CNN model architecture of implementation of Fully Convolutional Classification using only 3x3 convolution kernels.

a second layer of size 3x3 without padding, connected to this 5x5 feature map, will result in an output of a 3x3 feature map. Finally, applying a third layer of size 3x3 without padding, connected to this 3x3 feature map, will output a 1x1 map, which is a scalar. This scalar can be treated as equal to the result given by an inner-product of the original 7x7 feature map. We can still use $L' >$ (F-1) /2 (L' is another possible number of layers of connected 3x3 kernels) to arrive at the ($2L'$+1) layers approximation for the inner-product by simply increasing the redundancy. As a result, FC is implemented within the CNN-DSA accelerator.

We show an example of implementing FC in CNN-DSA on the Kaggle competition cats and dogs data set[7]. The CNN architecture is shown in Figure 10. The first five major layers are the same as VGG-16, with the exception that the 5_3 layer has 16 channels. We also add a sixth major layer consisted of 3 sub-layers, with respectively 32 channels, 32 channels and 2 channels in these 3 sub-layers. On this data set, we achieved 97.4% accuracy with the FC and convolution layers all inside the accelerator. This gives the same accuracy as the traditional (convolution + FC layered) solutions. This model also makes use of our CNN-DSFP fixed point model.

Although we only show a special case of binary classification, it indicates the potential of using our chip in cheap, low-end computation devices. Classification computation is done within the chip, while only a tiny portion of calculation is done outside the chip.

7. Conclusion and future work

In conclusion, we have designed a CNN-DSA accelerator which achieved a power consumption of less than 300mW and an ultra-power efficiency of 9.3TOPS/Watt. Demos on mobile and embedded systems show its applications in real-world scenarios. This 28nm Two-Dimensional CNN-DSA accelerator attains a 140fps at an accuracy comparable to that of the VGG architecture. The CNN-DSA accelerator has four main advantages: First, it is a CNN-specific matrix architecture and distributes memory blocks as the center of the processing unit. Second, it is data driven and enables full parallel processing. Third, CNN domain-specific floating point data structures efficiently use memory and completes all processing in internal memory rather than external DRAM. Fourth, it allows for a flexible process unit expansion. We also proved that shorcut layer types and depthwise separable convolution layer types are special cases of VGG-Type layers. The current CNN-DSA accelerator supports ResNet and MobileNet, but it is originally optimized for VGG-Type CNN models. Future work involves designing a CNN-DSA optimized for ResNet and MobileNet, among other architectures.

References

[1] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM Sigplan Notices*, 49(4):269–284, 2014. 1

[2] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, et al. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE Computer Society, 2014. 2

[3] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017. 1

[4] L. O. Chua and L. Yang. Cellular neural networks: Applications. *IEEE Transactions on circuits and systems*, 35(10):1273–1290, 1988. 1

[5] A. P. David and L. H. John. Computer organization and design: the hardware/software interface. *San mateo, CA: Morgan Kaufmann Publishers*, 1:998, 2005. 1

[6] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam. Shidiannao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 92–104. ACM, 2015. 2

[7] J. Elson, J. J. Douceur, J. Howell, and J. Saul. Asirra: A captcha that exploits interest-aligned manual image categorization. In *Proceedings of 14th ACM Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery, Inc., October 2007. 8

[8] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: efficient inference engine on com-

pressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 243–254. IEEE, 2016. 1

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 2, 4

[10] J. L. Hennessy and D. A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011. 1

[11] J. L. Hennessy and D. A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2017. 1

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 2, 4, 5

[13] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007. 7

[14] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12. ACM, 2017. 2

[15] D. E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009. 7

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1, 4

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[18] Y. Lin, S. Baohua, and et. al. Approximating fully-connected layers with multiple arrays of 3x3 convolutional filter kernels in a cnn based integrated circuit, March 2018. US Patent Application 15920842. 7

[19] Y. Lin, S. Baohua, and et. al. Convolutional layers used directly for feature extraction with a cnn based integrated circuit, Janurary 2018. US Patent Application 15880375. 7

[20] Y. Lin, S. Baohua, and et. al. Implementation of resnet in a cnn based digital integrated circuit, March 2018. US Patent Application 15897143. 4

[21] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010. 1

[22] O. M. Parkhi, A. Vedaldi, A. Zisserman, et al. Deep face recognition. In *BMVC*, volume 1, page 6, 2015. 7

[23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 3, 4

[24] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1, 2, 4, 6, 7

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015. 4, 6

[26] D. Wang, F. Yin, C. Liu, and Q. Wang. Icdar 2011 chinese handwriting recognition competition. In *2011 International Conference on Document Analysis and Recognition(ICDAR)*, volume 00, pages 1464–1469, 09 2011. 7

[27] B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. *arXiv preprint arXiv:1711.08141*, 2017. 2, 6

[28] C. Wu, W. Wen, T. Afzal, Y. Zhang, Y. Chen, and H. Li. A compact dnn: Approaching googlenet-level accuracy of classification and domain adaptation. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017. 4

[29] L. Yang, B. Sun, and et. al. Implementation of mobilenet in a cnn based digital integrated circuit, March 2018. US Patent Application 15910005. 5

[30] L. Yang and H. Yu. Digital integrated circuit for extracting features out of an input image based on cellular neural networks, April 2018. US Patent 9,940,534. 2

[31] F. Yin, Q.-F. Wang, X.-Y. Zhang, and C.-L. Liu. Icdar 2013 chinese handwriting recognition competition. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 1464–1470. IEEE, 2013. 6, 7

[32] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014. 1

[33] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017. 4