LAPPED CONVOLUTIONAL NEURAL NETWORKS FOR EMBEDDED SYSTEMS

Xing Wang[1], *Him Wai Ng*[1,2], *Jie Liang*[1,2] *

[1]AltumView Systems Inc., Burnaby, BC, Canada
[2]School of Engineering Science, Simon Fraser University, Burnaby, BC, Canada

ABSTRACT

Convolutional neural network (CNN) has achieved numerous breakthroughs in many artificial intelligent applications. However, its complexity is quite high and usually requires expensive GPU or FPGA implementation, which is not cost-effective for many embedded systems. In this paper, we develop a novel lapped CNN (LCNN) architecture that is suitable for resource-limited embedded systems. Our architecture follows the divide-and-conquer principle. The CNN is designed such that it can be decomposed into two or more stages, each can be implemented by a hardware module with a low-resolution input and very low complexity. The original input image is divided into some subimages of the same size, with properly designed overlaps with each other. These subimages are sequentially processed by the hardware module that implements the first stage of the CNN. The outputs from different subimages are then merged and processed by the next stage low-cost hardware CNN module. The result is exactly identical to that of applying a larger-scale CNN to the entire image with higher resolution. Therefore, by reusing low-cost hardware CNN modules, a low-cost and larger-scale CNN system can be achieved. The performance of the proposed scheme is demonstrated by experimental results.

Index Terms— Convolutional neural network (CNN), CNN hardware implementation, Receptive field

1. INTRODUCTION

In the last decade, a new generation of artificial neural network called deep learning (DL) has been developed [1]. It has many layers of neurons and millions of parameters. The parameters can be trained by large amount of data on fast GPU-equipped computers [2], guided by novel training techniques that can work with many layers, such as regularized linear units (ReLU) [3], dropout [4], and stochastic gradient descent (SGD) [5].

Convolutional neural network (CNN) is the most popular DL architecture [6], and has achieved unprecedented performances in many computer vision and machine learning tasks, such as image classification [7], image caption generation, visual question answering, and automatic driving cars.

*Email: jiel@sfu.ca.

However, the complexity of modern CNNs is quite high. For example, the first large-scale CNN, AlexNet [7], has 60 millions of parameters, and the VGG network from Oxford University has 19 layers and 144 millions of parameters [8]. Therefore running these networks requires significant amount of hardware resources. This is especially challenging for embedded systems. Although there have been some embedded systems such as NVIDIA TK1 and TX1 that can run large-scale CNNs, they are quite expensive. These chips also require a lot more power to run than traditional embedded platforms, making them unsuitable for many applications where basic deep learning functionalities are required, but certain constraints on cost and power consumption should also be met.

Some low-cost CNN-enabled embedded systems have also started to emerge. One example is the HiSilicon Hi3519 chipset [9], which has a small built-in CNN module, whose architecture is very similar to LeNet [6], the first CNN architecture that was designed around 1990. Therefore it can only be used for simple applications. However, one benefit of the Hi3519 CNN module is that it is very fast, and only takes about $1ms$ to process an input image with 32×40 pixels.

In this paper, we develop a novel lapped CNN (LCNN) architecture that is suitable for resource-limited embedded systems such as Hi3519. Our architecture follows the divide-and-conquer principle. The CNN is designed such that it can be decomposed into two or more tiers or stages, each operates on a low-resolution input and can be implemented by low-cost hardware module. To use the proposed LCNN, the original input image is divided into subimages of the same size, with properly designed overlaps or shifts among each other. These subimages are fed to the first tier of the CNN sequentially. The outputs from different subimages are then merged, and the result is exactly identical to that of applying a large CNN to the entire image. The merged result is then processed by the next tier of the CNN, with a new set of network parameters. This framework essentially builds a large-scale CNN by reusing a small-scale CNN module, making it attractive for low-cost embedded systems.

The performance of the proposed scheme is demonstrated by experimental results on Hi3519, which shows that for applications such as image classification, the top-1 performance can be improved by 30% compared to the default built-in

CNN module on Hi3519.

2. THE PROPOSED ARCHITECTURE

In this paper, we use the built-in CNN module on Hi3519 as an example to illustrate how to design and implement the proposed lapped CNN architecture. It should be noted that the proposed framework can also be used on other embedded platforms to reuse their built-in CNN modules and improve their performances. Moreover, the proposed framework can be implemented as a low-cost FPGA module, and integrated with many low-cost embedded platforms.

The simple CNN module on Hi3519 only accepts input images of up to 1280 pixels. It can have $1 \sim 8$ convolutional (CONV) layers and $3 \sim 8$ fully-connected (FC) layers. Each CONV or FC layer is followed by a ReLU layer. The ReLU layer after FC layer is also followed by a dropout layer. The number of filters in each CONV layer is at most 50, and only 3×3 convolution filters are allowed. The convolution stride is fixed to be 1. The pooling window is fixed to be 2×2, and the stride can only be 2. The maximum input dimension for the FC part is 1024, and the number of neurons in the middle FC layers is at most 256. The dimension of the output layer is at most 256. The CNN on Hi3519 allows users to re-configure the network architecture and parameters, which can be pre-trained for different applications.

Due to its limited architecture, the CNN on Hi3519 is typically only suitable for simple tasks such as handwritten digit recognition and license plate recognition. For more challenging tasks such as face recognition, the performance will be constrained by the low resolution of the input image and the limited number of layers and filters in the network.

In this paper, our goal is to reuse the built-in CNN module on embedded systems such as Hi3519 to build a larger-scale CNN. This is motivated by the various matrix-decomposition-based fast structures for DCT, lapped transform, and wavelet transform, which have found wide applications in speech, image, and video coding [10–12]. In these applications, a filter bank with long filters is usually implemented via the concatenation of several stages, each has much shorter filters. In many cases, such decompositions do not loss any optimality, *i.e.*, the fast structure can still achieve the optimal performance among all possible filter banks.

Our general approach is to divide a large input image into some small subimages, and each of them is processed by a small hardware CNN module. The outputs from all subimages are then combined for further processing. In this paper, we focus on two-tier structure, and we denote the two tiers as CNN1 and CNN2, as shown in Fig. 1, but the scheme can be generalized to more than two tiers.

We want the combined outputs are exactly identical to running a large CNN to the input image directly. However, if all subimages are processed independently, there will be no convolution operator across the subimage boundaries; hence

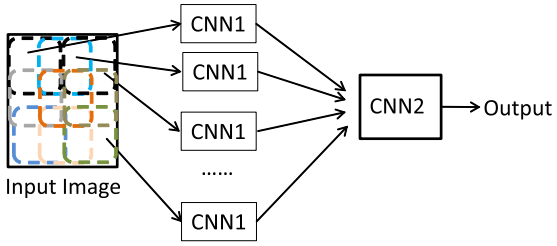

Fig. 1. The architecture of the proposed CNN scheme that can reuse simple hardware CNN module.

the features at subimage boundaries are not captured properly. This is similar to the blocking artifact in DCT-based image coding, which can be resolved by applying filters across the image block boundaries, leading to the lapped transform [11]. The same idea can be applied here as well. That is, we can apply CNN1 to the regions across subimage boundaries. After that, we can replace the boundary-affected outputs from each subimage by the correct ones from the boundary images.

Note that if there are only convolution operators in CNN1, we can use the classic overlap-and-add method to subimages and still avoid the boundary artifacts. However, due to the pooling operators in CNN1, the results of the overlap-and-add method are not equivalent to applying convolutions across the subimage boundary.

In order to achieve equivalent convolution and maxpooling results between the global approach and the subimage approach, we need to ensure two conditions: 1. The boundary effect of convolution should be avoided. 2. The input image size to each maxpooling operator in both cases is even.

In the following analyses, we focus on three stages of convolutions and maxpooling operators, 3×3 convolution filters, and 2×2 maxpooling. The ReLU unit is ignored since it does not affect the image size. The theory can be easily extended to more stages.

Suppose K is the number of rows or columns of an image. After the first convolution, there will be $K+2$ output coefficients, but only $K-2$ of them are not affected by the boundary effect. These coefficients will be sent to the first maxpooling operator. Therefore to avoid boundary effect, we need $K-2$ to be even. That is,

$$K - 2 = 2x_1 \quad \Rightarrow \quad K = 2x_1 + 2, \tag{1}$$

where x_1 is a positive integer.

After the first maxpooling, the output size is $(K-2)/2$. In the second layer, the size after convolution is $(K-2)/2-2$, which should also be even, i.e.,

$$(K-2)/2 - 2 = 2x_2 \quad \Rightarrow \quad K = 4x_2 + 6, \tag{2}$$

where x_2 is another positive integer.

After the second maxpooling, the sizes reduces to $(K-2)/4-1$. In the third layer, the size after convolution is $(K-$

$2)/4-3$, which should still be even. This means

$$(K-2)/4-3=2x_3 \quad \Rightarrow \quad K=8x_3+14, \qquad (3)$$

where x_3 is also a positive integer.

It is easy to verify that the solutions for K given by Eq. (3) form a subset of the solutions for Eq. (2), and the latter are a subset of that of Eq. (1). Therefore solutions for Eq. (3) can meet all three constraints in Eq. (1) to Eq. (3). If there are more than three layers, it can be shown that the solutions are also given by the constraint from the last layer, which is also a subset of the solutions of previous layers. Therefore feasible solutions still exist.

The first few solutions given by Eq. (3) are 22, 30, 38, and 46. Since the maximum number of input pixels for Hi3519 CNN is 1280, the closest subimage dimension is thus 38×30 (1140 pixels). For this choice of subimage, the size after the first convolution is 36×28. After the first maxpooling, the size becomes 18×14. After the second convolution and maxpooling, we get 16×12 and 8×6 respectively. Finally, the size reduces to 6×4 after the third convolution, and the final output size is 3×2 after the third maxpooling.

On the other hand, the size of the entire input image should also meets Eq. (3). This means that simply putting some subimages side by side cannot guarantee the equivalence, even if the subimage size meets Eq. (3). For example, an image of size 76×60 cannot achieve the same results between the global CNN and subimage-based CNN, because the numbers 70 and 60 do not satisfy Eq. (3), although it can be divided into four subimages of 38×30.

To achieve both feasible subimage size and entire image size, we can introduce some extra pixels between neighboring subimages. For the example above, the closest feasible solution is 78×62, *i.e.*, there are two extra pixels between two neighboring subimages both horizontally and vertically. In fact, the size of the entire image can be other values that satisfy Eq. (3). These extra pixels (or gaps) can be filtered by some special subimages that straddle the boundaries of neighboring subimages. This approach is similar to the pre/post-filter in the lapped transform [11]. The results of these lapped subimages can be merged with the outputs of other subimages.

Another constraint of many CNN modules is that they can only support one input image size. Therefore all subimages, including those that straddle neighbouring subimages, should have the same size, for example, 38×30. With this additional constraint, the next question is what should be the optimal amount of distances or shifts between neighboring subimages. Since the output of each 38×30 subimage is 3×2, we need to choose the distance such that the outputs of two overlapped subimages are shifted by 3 vertically and by 2 horizontally, otherwise we either have gaps in the outputs and therefore cannot ensure equivalence, or there are duplicated outputs, which waste the computing resource.

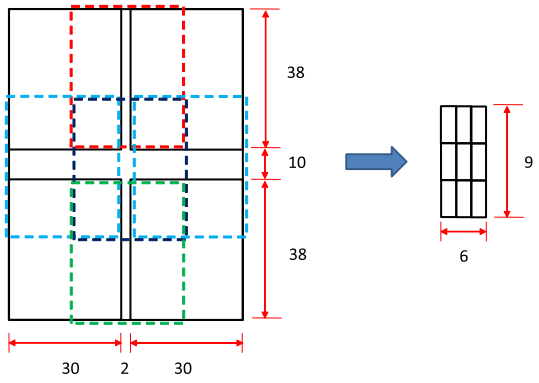

Fig. 2. An example of image and subimage configuration to use the proposed lapped CNN.

Note that each maxpooling reduces the image size by half. Therefore each output after three maxpooling stages corresponds to 8 input pixels. As a result, in order for the output to shift by 2 and 3 respectively, we need to shift the subimage by 16 and 24 pixels respectively. Therefore, to avoid any gap or duplication in the output, the aforementioned image size 78×62 should be adjusted to 86×62, where $86 = 38 + 24 \times 2$, and $62 = 30 + 16 \times 2$. That is, the gap has 2 pixels horizontally, but 10 pixels vertically. This example is illustrated in Fig. 2. There are 9 overlapped subimages, with rows starting at 1, 25, and 49, and columns starting at 1, 17, and 33. The design is actually closed related to the receptive field concept in CNN [13].

Once we apply CNN1 to each subimage sequentially, the outputs from different CNN1-processed subimages are combined to form the input to CNN2. CNN1 only includes some CONV layers and pooling layers. Its output feature maps still preserve the spatial information. CNN2 only includes FC layers. It merges the features from different subimages and makes the final classification decision.

In fact, since the dimension of the output of CNN1 is much smaller than that of the input, the FC layers in CNN2 do not have to be implemented by hardware. Carefully optimized software implementation can be very fast.

3. EXPERIMENTAL RESULTS

In this part, we test the impact of different input image sizes on the performance of the CNN models using two tasks, image classification and age estimation, and verify the efficiency of our proposed scheme. The baseline method uses input image size of 36×35, wth 1260 pixels, which is close to the upper limit of 1280 of Hi3519. Therefore it can be processed by the Hi3519 CNN directly. Our method uses an input size of 86×78, which is divided into 12 overlapped subimages of size 38×30, similar to the example in Fig. 2. The total running time for the large image size is only about $18ms$.

Network	N1 (86×78)	N1 (36×35)	N2 (86×78)	N2 (36×35)
Top-1	36.25	28.36	38.12	29.86
Top-5	62.75	54.37	64.24	55.58

Table 1. Image classification accuracy [%] with different networks and different input sizes.

Network	N1 (86×78)	N1 (36×35)	N2 (86×78)	N2 (36×35)
Accuracy	52.27	49.34	53.34	48.43

Table 2. Age group estimation accuracy [%] with different networks and different input sizes

This is enough to meet the real-time requirements of many applications.

For image classification, Tiny ImageNet, a subset of ILSVRC-2012 dataset [14], is used. For age group estimation, we use the Adiencedb dataset [15].

For each task, two network architectures are tested. The first is denoted as **N1**, and the architecture is CONV(24)-CONV(48)-CONV(48)-FC(256)-SOFTMAX, where the numbers in bracket denote the number of filters in CONV layers or the size of FC vectors. The other is denoted as **N2**, with the architecture CONV(32)-CONV(48)-CONV(48)-FC(512)-FC(512)-SOFTMAX. The configuration of N1 strictly follows th Hi3519 hardware requirements and can be implemented entirely by reusing the built-in hardware CNN module. In N2, only the CONV layers follow the Hi3519 hardware requirements, and the FC layers are implemented by software since their sizes are over the hardware limit, but since the complexity of the FC layers is much lower than convolution layers, software implementation does not slow down the overall speed too much.

For the Tiny ImageNet dataset, we randomly select 200 classes from the ILSVRC-2012 dataset. In each class, we randomly choose 500 images as training samples and another 50 images as validation samples. All images are resized to 90×90 after center crop. Together, there are 200 classes, 100K training samples and 10K validation samples. We use the Adam [16] optimizer in TensorFlow [17]. To get input size 86×78, we randomly crop image of size 90×90. To get input size 36×35, we first randomly crop 90×90 to 80×80, and then resize to 36×35.

The results are reported in Table 1, which shows that large input size can improve the Top-1 and Top-5 accuracies by about 30% and 15% respectively with the same architecture.

For the age estimation using the Adiencedb dataset, there are 8 age groups and 5 folders of images. We randomly select 1 of 5 folders as validation set and use the remaining 4 folders as training set. There are 11K images in the training set and 3K in the validation set.

The results are summarized in Table 2. In this example, the relative improvement of our method is between $6-10\%$. The impact of image size is not as much as image classification. Note that the best result achieved on this dataset is 55.6±6.1 in [18], which is based on the much more complicated 16-layer VGG16 network. The second best result is 50.7±5.1 in [19], whose network has 3 CONV layers and 2 FC layers (same as ours). However, their image size is 227×227 and their network has many more CONV filters: 96 in CONV1, 256 in CONV2 and 256 in CONV3. Therefore our implementation on Hi3519 is very competitive if the cost is considered.

4. CONCLUSION

In this paper, we develop an efficient CNN framework for embedded systems, which repeatedly applies a small-scale CNN module to different subimages of a larger input image. By judiciously selecting the architecture of the network, the sizes of the input image, the sizes and overlaps of the subimages, the result can be exactly equivalent to applying a larger CNN to the large input image directly. This approach enables more cost-effective CNN solutions for some embedded systems and is very suitable for applications where basic deep learning functionalities are required, but certain constraints on cost and power consumption should also be met.

5. REFERENCES

[1] G. E. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, pp. 1527–1554, 2006.

[2] L. Deng, M. Seltzer, D. Yu, A. Acero, A.-R. Mohamed, and G. Hinton, "Binary coding of speech spectrograms using a deep auto-encoder," *International Speech Communication Association*, pp. 1692–1695, 2010.

[3] V. Nair and G. Hinton, "Rectified linear units improve restricted boltzmann machines," *Proceedings of the 27th Internation Conference on Machine Learning*, Haifa, Israel, 2010.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[5] L. Bottou, "Stochastic gradient learning in neural networks," *Proceedings of Neuro-Nimes*, 1991.

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of IEEE*, Nov. 1998.

[7] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems (NIPS)*, pp. 1097–1105, 2012.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *International Conference on Learning Representations*, 2015.

[9] HiSilicon, "Hi3519 advanced industry ip camera soc," *http://www.hisilicon.com/en/Products*.

[10] J. Liang and T. D. Tran, "Fast multiplierless approximation of the DCT with the lifting scheme," *IEEE Trans. Signal Processing*, vol. 49, no. 12, pp. 3032–3044, Dec. 2001.

[11] T. D. Tran, J. Liang, and C. Tu, "Lapped transform via time-domain pre- and post-processing," *IEEE Trans. Signal Processing*, vol. 51, no. 6, pp. 1557–1571, Jun. 2003.

[12] D. S. Taubman and M. W. Marcellin, *JPEG 2000: Image compression fundamentals, standards, and practices*, Kluwer, Norwell, MA, 2002.

[13] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the effective receptive field in deep convolutional neural networks," *In Neural Information Processing Systems (NIPS)*, 2016.

[14] 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)*, vol. 115, no. 3, pp. 211–252, 2015.

[15] E. Eidinger, R. Enbar, and T. Hassner, "Age and gender estimation of unfiltered faces," *Trans. Info. For. Sec.*, vol. 9, no. 12, pp. 2170–2179, Dec. 2014.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization.," *CoRR*, vol. abs/1412.6980, 2014.

[17] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, Software available from tensorflow.org.

[18] R. Rothe, R. Timofte, and L. Van Gool, "Deep expectation of real and apparent age from a single image without facial landmarks," *International Journal of Computer Vision (IJCV)*, July 2016.

[19] G. Levi and T. Hassner, "Age and gender classification using convolutional neural networks," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) workshops*, June 2015.