Design of MobileNet algorithm to optimize image classification in Convolutional Neural Network (CNN) and implementation on FPGA

Akbar Payandan, S. Hossein Hosseini Nejad
Faculty of Engineering, Ahar Branch, Islamic Azad University, Ahar, Iran,
Email: payandan7393@gmail.com, S.hosseininejad@gmail.com

Abstract

Deep learning has developed rapidly in recent years and has been applied in many areas that are major areas of artificial intelligence. The combination of deep learning and embedded systems has created good dimensions in the technical field. In this paper, a deep learning neural network algorithm can be designed that can be implemented on FPGA hardware. The PyTorch and CUDA were used as assistant methods. Convolution neural network (CNN) was also used for image classification. Three good CNN models such as ResNet, ResNeXt and MobileNet were reviewed in this article. Using these models in the design, an algorithm was eventually designed with the MobileNet model. Models were selected from different aspects such as floating operation point (FLOP), number of parameters and classification accuracy. In fact, the MobileNet-based algorithm was selected with a top-1 error of 5.5% in software with a 6-class data set. In addition, hardware simulation in MobileNet-based algorithms was presented. The parameters were converted from floating numbers to 8-bit integers. The output numbers of each layer were cut into integer fixed bits to fit the hardware constraint. A method based on working with numbers was designed to simulate number changes in hardware. The results of simulation show that, the top-1 error increased to 12.3%, which is acceptable.

Keywords: Artificial Intelligence, Deep Learning, Image Classification, Convolution Neural Network, Deep Learning Algorithm.

1- Introduction

Deep learning has always been used in areas such as image classification and voice recognition with raw data. Compared to desktops, embedded systems (such as FPGA) have lower power consumption, smaller size, and lower unit cost [1]. The FPGAs are used in several fields such as robots and smartphones [2].

The FPGA board is using the Xilinx Zynq UltraScale + MPSoC ZCU104 evaluation kit with 38 MB of memory [3]. This memory is small enough to run a CNN model. Parameters and most workloads are stored in memory. It therefore needs to design a CNN model small enough to overcome the limitations of hardware memory. In this paper, by combining CNN and FPGA, an embedded CNN system with image recognition capability was produced. It is assumed that it detects the information on the image in a reasonable period of time [4].

Previous studies [5] on the use of specialized devices such as FPGAs and GPUs in heterogeneous computations to accelerate deep learning computations with energy efficiency constraints to evaluate efficient DNN performance show that using FPGA for fully connected layer and GPU for floating point operation can be faster calculation and much lower power consumption.

Also, studies on the study of FPGA-based accelerators from deep learning networks for learning and classification, recent

techniques to accelerate deep learning networks in FPGA reflect recent trends in FPGA-based accelerators of deep learning networks [6].

The purpose of this paper is image recognition and for this purpose, image classification was performed in the embedded system. The camera was used to capture the image and the captured images were transferred to the FPGA board. FPGA board is a hardware platform that is the main part of the system. The CPU on the FPGA board prepares images with the given size and type of data and then sends the data to the programmable logic in which CNN implements and executes. CNN then on the board, the FPGA output contains several digits that indicate the previously distributed image classification. The result of the diagnosis is displayed on the screen.

As mentioned earlier in this paper, a combination of CNN and FPGA produced an embedded CNN system with image recognition capabilities. Therefore, provide a CNN model with an accuracy greater than 90% (top-1 error less than 10%) based on the target data set; evaluation based on top-1 error, spatial resource cost, and model complexity; comparing different models with the proposed model and select the appropriate model for hardware implementation; designing a method to convert the parameters of the selected model to format (mobile numeric numbers); providing a solution for simulating hardware performance based on specific features of the FPGA board and achieving simulation accuracy of more than 85% are important objectives of this paper.

2- Methodology

Quantitative and qualitative methods were used in this article. Also, in this article, the deductive method was used. The models used in this article are: ResNet, ResNeXt, MobileNet and ShuffNet. Software performance and cost are shown. The networks designed in this article are not the main networks in the articles. The most important features of these networks were studied and the basic structure was maintained. Most importantly, by reducing the number of layers or channels, space costs can be reduced to an acceptable level.

3- Data set and analysis method

In this article, the data set containing fingerprints (representing numbers 0 to 5) in the Kaggle database was used.

The simulation was performed in MATLAB V.2015b and was performed on a system with 7-core processor specifications with 6 MB of cache and 3.6 MHz and 6 GB of memory in Windows 8.

4- Network training

Samples of datasets used in this paper are shown in Figure (1). Some methods have been used to complicate the data set, such as adding noise and background. Images include a gray scale channel measuring 128 x 128 x 128.

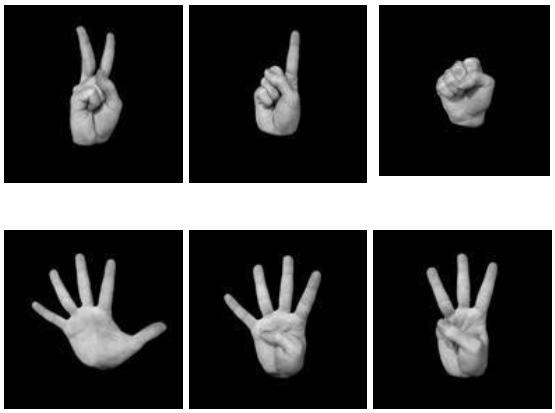

Fig (1): The main examples of data sets taken from the kaggle database.

The rest of the data set are all similar images with finger movements from

number 0 to 5. To show more datasets, the black background of the original dataset must be changed. These is to add random backgrounds to the pure black and white part of the images, which diversifies the data set and increases the complexity of the classification.

Fig (2): samples of data sets after processing.

After applying the background to the main data set, the images of the data set appear as Figure (2). According to this figure, the images of the other hand are not in the center and the same size. Images of hand movements are rotated. They are enlarged or reduced and then placed in different corners of the image. The background is real-world images to bring training and testing data set closer to the real uses. After using images, it is important to categorize this data into a file, including their File names and which class they belong to.

5- CNN models

5-1- ResNet model

To achieve residual deep learning, the open source ResNet model is used in accordance with [7] in this paper. This model defines the Bottleneck and ResNet classes. The bottleneck structure is made up of two point rotations and one deep rotation in the middle. After starting the convolution layer, there are four layers. Each layer starts with a max pool layer with stride 2 so that each layer starts with an input size divided by 2 in width and height. In this paper, each layer contains two bottleneck structures. At the end of each bottleneck structure, the input futures map is added to the calculated futures map. If the input and output channel numbers do not match, there is an additional layer of convolution that is also trained to match the channels. Details and FLOPs of the ResNet model are shown in Table (1).

Table (1): ResNet structure and FLOPs.

Layer name		Output size	Output channel	Kernel size	
Initial convolution		128×128	8	3×3	padding = 1
Layer 1	Max pool	max pool 2×2, stride 2			
	Pointwise*	64×64	4	1×1	Repeat once
	Depthwise	64×64	4	3×3	
	Pointwise	64×64	16	1×1	
Layer 2	Max pool	max pool 2×2, stride 2			
	Pointwise	32×32	64	1×1	Repeat once
	Depthwise	32×32	64	3×3	
	Pointwise	32×32	256	1×1	
Layer 3	Max pool	max pool 2×2, stride 2			
	Pointwise	16×16	64	1×1	Repeat once
	Depthwise	16×16	64	3×3	

	Pointwise	16×16	256	1×1	
Layer 4	Max pool	max pool 2×2, stride 2			
	Pointwise	8×8	4	1×1	Repeat once
	Depthwise	8×8	4	3×3	
	Pointwise	8×8	16	1×1	
Full connected layer		16-output fc1، 8-output fc2 and 6-output fc3			
FLOPs		1.93×10^8			

*padding = 1

In this model, the test result of the 1800 image dataset is shown in Table (2). According to this table, ResNet performance is not good at this size and data set.

Table (2): ResNet verification result.

Class	0	1	2	3	4	5
Accuracy (%)	94	93	90	94	89	97
Top-1 error(%)	7.2					

5-2- ResNeXt model

The ResNeXt model is similar to ResNet. ResNet uses fully deep grouped convolution, and different channels do not communicate during deep convolution operations.

As shown in Table (3), the total number of layers (counting the deep layer as a reference) is 8, which is ResNet. The initial convolution is the max pool and fully connected layers of ResNet.

Table (3): ResNeXt structure and FLOPs.

Layer name	Output size	Output channel	Kernel size	
Initial convolution	128×128	8	3×3	padding = 1
Max pool 1	max pool 2×2, stride 2			
Pointwise*	64×64	64	1×1	Repeat once
Depthwise**	64×64	64	3×3	
Pointwise	64×64	32	1×1	
Max pool 2	max pool 2×2, stride 2			
Pointwise	32×32	128	1×1	Repeat 4 times
Depthwise	32×32	128	3×3	
Pointwise	32×32	64	1×1	
Max pool 3	max pool 2×2, stride 2			
Pointwise	16×16	32	1×1	Repeat once
Depthwise	16×16	32	3×3	
Pointwise	16×16	16	1×1	
Max pool 4	max pool 2×2, stride 2			
Full connected layer	6-output fc3، 8-output fc2 and 16-output fc1			
FLOPs	2.52×10^8			

*padding = 0

**group = 8, padding = 0

The training and validation process uses the same dataset. The validation result is shown in Table (4). The method of calculating FLOPs is presented in accordance with [8].

Table (4): ResNeXt verification result.

Class	0	1	2	3	4	5
Accuracy (%)	95	92	89	93	95	98
Top-1 error(%)	6.3					

5-3- MobileNet model

The structure of the MobileNet model is shown in Table (5) in this paper. In addition to the initial convolution layer, there are eight pairs of depth-points. The 2×2 max pool functions with stride 2 are placed after the second, fourth, fifth and eighth pairs. After the initial convolution layer, a ReLU layer is placed. In addition, each depthwise or pointwise convolution layer is followed by a batch-norm layer and a ReLU layer.
In this model, the test result on a data set of 1800 images is shown in Table (6).

Table (5): MobileNet structure and FLOPs.

Layer name	Output size	Output channel	Kernel size	Padding
Initial convolution	128×128	8	3×3	1
Depthwise 1	128×128	8	3×3	1
Pointwise 1	128×128	32	1×1	0
Depthwise 2	128×128	32	3×3	1
Pointwise 2	128×128	64	1×1	0
Max pool 1	max pool 2×2, stride 2			
Depthwise 3	64×64	64	3×3	1
Pointwise 3	64×64	64	1×1	0
Depthwise 4	64×64	64	3×3	1
Pointwise 4	64×64	64	1×1	0
Max pool 2	max pool 2×2, stride 2			
Depthwise 5	32×32	64	3×3	1
Pointwise 5	32×32	64	1×1	0
Max pool 3	max pool 2×2, stride 2			
Depthwise 6	16×16	64	3×3	1
Pointwise 6	16×16	64	1×1	0
Depthwise 7	16×16	64	3×3	1
Pointwise 7	16×16	32	1×1	0
Depthwise 8	16×16	32	3×3	1
Pointwise 8	16×16	16	1×1	0
Max pool 4	max pool 2×2, stride 2			
Full connected layer	6-output fc3، 8-output fc2 and 16-output fc1			
FLOPs	1.02×10^{8}			

Table (6): MobileNet verification result.

Class	0	1	2	3	4	5
Accuracy (%)	93	91	93	90	94	98
Top-1 error(%)	6.8					

6- Comparison and model selection

As shown in Figure (3), the dissipation curves of the three models are close to zero. This means that training processes perform almost at their best in these models. Comparisons in FLOPs, parameter values, and top-1 error are presented in Table (7). The method of calculating FLOPs and the number of parameters are presented in [8].

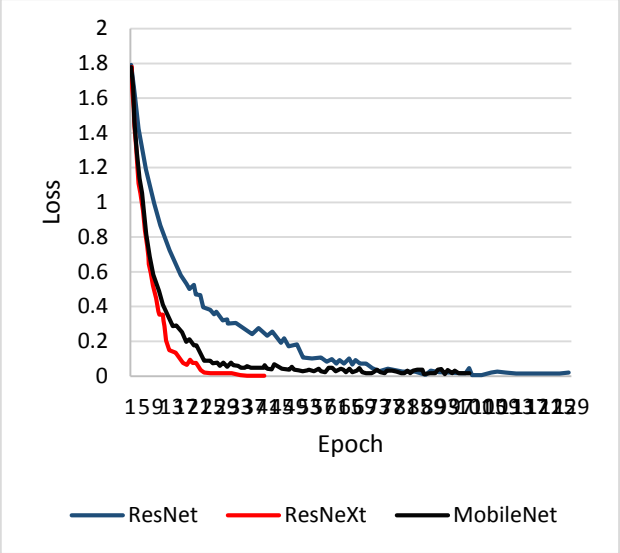

Fig (3): The loss curves of the three models.

Table (7): Comparison between three models.

	ResNet	ResNeXt	MobileNet
FLOPs (10^8%)	1.93	2.52	1.02
#parameters	360070	213800	43820
Top-1 error(%)	7.2	6.3	6.8

According to Table (7), the choice is clear. For FLOPs, ResNeXt is the worst case scenario because in some layers of convolution it uses the maximum number of channels and gives more communication to the channels than ResNet. ResNet has the highest number of parameters, while MobileNet has the lowest number of parameters with 1.5. ResNet is the worst in the top-1 error, however, the top-1 error is acceptable.

These three models all use a large 8-layer structure (a complete combination as a large layer, like a depthwise-pointwise layer combination in MobileNet). However, in the large layers of each model, MobileNet uses only two convulsions, while ResNet and ResNeXt use another layer of point-wise convulsions.

MobileNet stores resources in the hardware with fewer layers of convolution. On the other hand, ResNet and ResNeXt do not have much advantage over MobileNet. Therefore, the MobileNet model is selected for the next steps.

7- Building Integr-Net model

Integer-Net is built on MobileNet and maintains the same MobileNet structures as number of layers, layer types and parameters. The trend can be shown in Figure (4). This figure shows a layer, also called a module. The layer change is detachable.

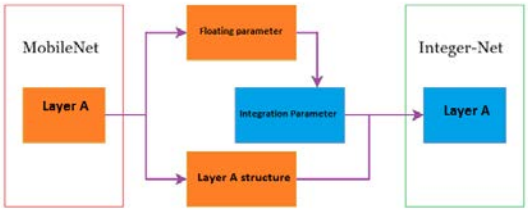

Fig (4): Copy process from MobileNet to Integer-Net.

For each layer, the parameters are tensor and all parameters are less than 1 in this particular model. They change in the same way. The change method is shown in Figure (5), which places these parameters in the

range of -127 to +127 and prevents the parameters from being overdone.

The classification of parameters is presented in Table (8). For layers that have only one type of parameter, such as convolution layers, they only have a 3 × 3 kernel weight as the parameter. Other layers, such as batch-norm layers and fully connected layers, have two types of parameters: weight and bias. The method of dealing with these parameters is to change these two types of parameters in the same way (multiplied by 2 at a time).

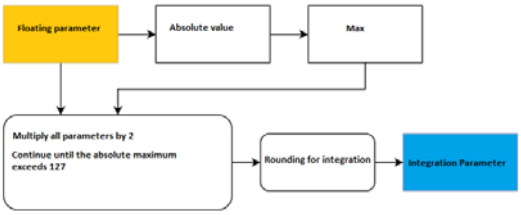

Fig (5): The process of changing the parameter from the floating point to integers.

Table (8): MobileNet model parameter categories and sizes.

Layer name	Parameters
Initial convolution	Width [1 × Out Channel × 3 × 3]
Depthwise convolution	Width [1 × Out Channel × 3 × 3]
Pointwise convolution	Width [Out Channel × In Channel × 3 × 3]
Batch-norm	Width [1]; Bias [1]
Full connected	Width [out channel × in channel] Bias [Out Channel × In Channel]
ReLU	No parameters

The reason that three convolution layers do not have bias as a parameter is the bias = False argument, which is set for convolution layers during network training.

In Equation (1), all parameters change according to Integer-Net requirements, so the X^{BN} changes, which is related to the change in the output of the convolution layers. However, the X_{middle} feature will not change. The μ and σ are the mean and the deviation from the X^{BN} input, so we can consider X_{middle} unchanged. Accordingly, γ and α must be multiplied by the same rate to ensure that the output relationship remains unchanged.

$$Y^{BN} = \frac{X^{BN} - \mu}{\sqrt{\sigma^2 + \in}}\gamma + \alpha = X_{middle}\gamma + \alpha \quad (1)$$

8- Accuracy modification of the output futures map

Rounding the contents of the output futures map to fixed bit numbers is essential. This is done for two main reasons. The first reason is that the hardware must have numbers in the process with the same format selected for the integer signed with a fixed bit.

8-1- Test 1

The first method proposed is to multiply and divide the complete futures map after each layer by 2 continuously before the maximum absolute tensor is greater than 127, which is clearer according to the following codes.

```
while ( torch . max( torch .abs(x)) > 127):
  x = torch . round (x / 2)
```

This method ensures that each layer makes full use of 8-bit space.

Table (9): Accuracy result with number range [127-, 127].

Class	0	1	2	3	4	5
Accuracy (%)	34	78	77	66	47	20
Top-1 error(%)	46.3					

However, the result is not acceptable. When each intermediate futures map is in the range of -127 to +127, the accuracy is very poor, as shown in Table (9). This top-1 error means using 8-bit integers because the pixel size of the resulting content is bad. The small size may swipe the small numbers that are affected by the resulting pixel map variation. So that for the simulation function, a slightly larger integer must be used. 9-bit integers are chosen to improve this experiment. Hence the code that controls the content range of the output futures map changes as follows.

```
while ( torch . max( torch .abs(x)) > 255):
x = torch . round (x / 2)
```

This code only changes the number 127 to 255, however, the accuracy result is quite different. As shown in Table (10), the top-1 error is acceptable.

Table (10): The result of accuracy with number range [255-, 255].

Class	0	1	2	3	4	5
Accuracy (%)	92	88	88	81	87	87
Top-1 error(%)	12.8					

8-2- Test 2

Test 2 is based on each futures map output from each layer. By calculating and estimating the size of the number of each layer, the output number range of each layer is estimated. The output range must be an 8-bit signed integer, and the bit-cutting method of a result calculated in Table (11) is presented. The first bit is the sign bit, so the cutting method deals with 7 bits.

Table (11): Test bit control 2.

Layer name	Keep bits [big bit, small bit]
Depth-wise	[11,5]
64- channel Point-wise	[14,8]
32- channel Point-wise	[13,7]
16- channel Point-wise	[12,6]
Batch-norm	[11,5]
FC 1 (1024×16)	[22,16]
FC 2 (16×8)	[13,7]
FC 3 (8×6)	[12,6]

Unfortunately, this method gives bad results as shown in Table (12).

Table (12): Accuracy results in test 2.

Class	0	1	2	3	4	5
Accuracy (%)	0	12	18	27	51	25
Top-1 error(%)	77.8					

8-3- Test 3

This test is a combination of tests 1 and 2. The first step is to repeat the process in test 1 with the range [255-, 255] to maintain good performance. During this process, a change is recorded in each layer.

The second step is to adjust the amplitude for each layer, as in test 2. In the simulation process, the output of each layer changes to have numbers in the range [255, 255]. Different layers have different parameters and bit handling is different in layers.

After the two steps mentioned above, each layer produces a futures map with a 9-bit signed integer. Smaller bits are swiped in the torch.round () function. However, there is still no talk of overflow. According to the data set, the result will not go beyond the range.

When numbers are fixed in bit size, we still need to prevent overflow. The main idea of preventing overflow is to prevent switching between positive and negative numbers, which can lead to major failure.

Based on the numbers generated in the second step, the system wants to perform the remaining calculation on these numbers to prevent overflow. The goal is to get the domain [128-, 127]. We need tricks to achieve this. The little trick is shown in the code below. This trick handles positive and negative numbers differently and simulates the hardware performance of cutting bits out of range.

```
relu_mod = torch .nn. modules . activation . ReLU ()
x = relu_mod . forward (x)%128 - torch . ceil (
( relu_mod . forward (0-x *2 -1))%256/2 )
```

Accuracy is not greatly affected by the number generated by the steps mentioned above. As Table (13) shows, the top-1 error is approximately similar to the result in test 1.

Table (13): Accuracy results in test 3.

Class	0	1	2	3	4	5
Accuracy (%)	91	89	83	84	88	91
Top-1 error(%)	12.3					

9- Conclusion

As shown in Table (7), ResNeXt is worst in FLOPs, ResNet has the highest number of parameters, and all three models have the same top-1 errors. Using the quantitative method, the MobileNet model is the best choice.

Both quantitative and qualitative methods have been used in hardware simulation. The result of Experiment 1 with a top-1 error of less than 15% is acceptable. However, due to the quality standard, Test 1 goes beyond the hardware capability. Therefore, test 1 cannot be the final decision.

Test 2 have more emphasis on hardware capability as it uses a fixed cutting rule but produces a bad result.

Test 3 combines the benefits of the previous 2 tests. The only quantitative standard is the top-1 error after simulation. This test has a top-1 error of 12.3%, which is less than 15%. Given the details mentioned, there are no other complex calculations. This does not add extra cost to the hardware. Hence the simulation reaches its goal.

The evaluation results of CNN algorithms based on the current CNN model (ResNet, ResNeXt and MobileNet) showed that it has a good accuracy of 93.7%. This article compared different models by limiting standards and selecting the MobileNet model for hardware implementation. Finally, this article successfully converts the selected model into a suitable format for hardware. In particular, the parameters are changed to an 8-bit integer without affecting the result.

This paper provides a solution for simulating network performance on hardware (especially FPGAs). The simulation considers the numbers as 8-bit integers and simulates the performance on the hardware well, which helps to reflect the performance on the hardware. The simulation performance on the hardware shows that it leads to an accuracy of 87.7%, which is an acceptable accuracy.

This project can be improved in the future. To test the performance of the model used in this paper, different datasets can be selected. Which can show whether this model works on public data as well.

Also, more CNN models can be selected and compared to the current model to determine the most efficient model. For this purpose, it is suggested to continue research on several other efficient models, such as ShuffleNet and BinaryNet.

Refrence

[1] "Andrew G. Howard et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: CoRR abs/1704.04861 (2017). arXiv: 1704.04861. url: http://arxiv.org/abs/1704.04861.".

[2] "A. Shawahna, S. M. Sait and A. El-Maleh, "FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review," in IEEE Access, vol. 7, pp. 7823-7859, 2019, doi: 10.1109/ACCESS.2018.2890150.".

[3] "Baris Kayalibay, Grady Jensen, and Patrick van der Smagt. “CNN-based Segmentation of Medical Imaging Data”. In: CoRR abs/1701.03056 (2017). arXiv: 1701.03056. url: http://arxiv.org/abs/1701.03056.".

[4] " Forrest, J. R. K. , Thorp, R. W. , Kremen, C. , & Williams, N. M. (2015). Contrasting patterns in species and functional-trait diversity of bees in an agricultural landscape. Journal of Applied Ecology, 52, 706–715. 10.1111/1365-2664.12433".

[5] "Y. Tu, S. Sadiq, Y. Tao, M. Shyu and S. Chen, "A Power Efficient Neural Network Implementation on Heterogeneous FPGA and GPU Devices," 2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI), Los Angeles, CA, US".

[6] "A. Shawahna, S. M. Sait and A. El-Maleh, "FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review," in IEEE Access, vol. 7, pp. 7823-7859, 2019, doi: 10.1109/ACCESS.2018.2890150.".

[7] "fmassa. Datasets, Transforms and Models specific to Computer Vision. vision/torchvision/models/resnet.py.2O19.".

[8] "Flops counter for convolutional networks in pytorch framework. URL: https://github.com/sovrasov/flops-counter.pytorch.".