RMNv2: Reduced Mobilenet V2 for CIFAR10

Maneesh Ayi
Department of ECE
Purdue School of Engineering and Tech.
Indianapolis, USA
maayi@iu.edu

Mohamed El-Sharkawy
Department of ECE
Purdue School of Engineering and Tech.
Indianapolis, USA
melshark@iupui.edu

Abstract—In this paper, we developed a new architecture called Reduced Mobilenet V2 (RMNv2) for CIFAR10 dataset. The baseline architecture of our network is Mobilenet V2. RMNv2 is architecturally modified version of Mobilenet V2. The proposed model has a total number of parameters of 1.06M which is 52.2% lesser than the baseline model. The overall accuracy of RMNv2 for CIFAR10 dataset is 92.4% which is 1.9% lesser than the baseline model. The architectural modifications involve heterogeneous kernel-based convolutions, mish activation, etc. Also, we include a data augmentationtechnique called AutoAugment that contributes to increasing accuracy of our model. This architectural modification makes the model suitable for resource-constrained devices like embedded devices, mobile devices deployment for real-time applications like autonomous vehicles, object recognition, etc.

Index Terms—*MobilenetV2 architecture, Convolution Neural Networks (CNN), Deep Neural Networks (DNN), CIFAR-10, Pytorch.*

I. Introduction

Convolution neural networks (CNN) come into the picture in the field of Computer Vision with the introduction of AlexNet [6]. It proves to be a trendsetter in Computer vision platform by winning the Imagenet challenge: ILSVRC 2012 [7]. Then over the period of time, there have been many new architectures that are much bigger and complicatedintroduced to achieve higher accuracy. One of that kind is VGG net [11].[15][16] also follow the same trend with increased accuracies. However, implementing these networks in resource-constrained environments like embedded devicesis very difficult. Due to insufficient memory, limited computational capacity real-time implementation of bigger networks is not possible in these devices. So, there is a needto develop smaller networks that require less computation, less memory while maintaining competitive accuracy. This paper aims to introduce a new CNN architecture that gives out better accuracy with low model size.

Section-2 describes prior works in building small models. Section-3 describes the developed network RMNv2 architecture and Section-4 talks about the experiments carried out and results of that experiment. Finally, Section-5 concludes the paper.

II. Prior Work

There has been active research going from the past several years in designing small models. This includes either compressing a large neural network or designing small networks directly. There are certain techniques to compress a trained network like quantization[8], hashing[9], pruning, vector quantization, and Huffman Encoding[10]. Another method called distillation [12] which uses large network to teach small network. And also there are several techniques [13][14] that are used to compress the pre-trained neural network. Another approach is to design small networks directly for training. SqueezeNet[17] and SqueezeNext[18]are some of the examples of this approach. Most of these small networks mainly focus on small size but they do not focus on speed. Mobilenets[1] are introduced which are designed using depthwise separable convolutions and pointwise convolutions. They focus on both speed and accuracy. Our network is designed from Mobilenet V2[2].These networks try to reduce the number of computations required making it suitable for deploying in mobile and embedded devices.

TABLE I: Mobilenet V2 Baseline

Input	operator	t	c	n	s
$224^2 \times 3$	Conv2D	-	32	1	2
$112^2 \times 32$	Bottleneck	1	16	1	1
$112^2 \times 16$	Bottleneck	6	24	2	2
$56^2 \times 24$	Bottleneck	6	32	3	2
$28^2 \times 32$	Bottleneck	6	64	4	2
$14^2 \times 64$	Bottleneck	6	96	3	1
$14^2 \times 96$	Bottleneck	6	160	3	2
$7^2 \times 160$	Bottleneck	6	320	1	1
$7^2 \times 320$	Conv2D	-	1280	1	1
$7^2 \times 1280$	AvgPool	-	-	1	-
$1^2 \times 1280$	Conv2D	-	k	-	

III. RMNv2 Architecture

The proposed RMNv2 architecture is a CNN architecture inspired by Mobilenet V2, Heterogenous kernel-based convolution and mish activation function. Table-1 shows the baseline Mobilenet V2 architecture. In this section, we describe the detailed architecture of the proposed model.

A. Disabling downsampling layers

We have disabled some downsampling layers in our architecture. Its because the original Mobilenet V2 architecture is proposed for Imagenet dataset with an input image size of $224 \times 224 \times 3$. In order to make it compatible

This is the author's manuscript of the article published in final edited form as:

Ayi, M., & El-Sharkawy, M. (2020). RMNv2: Reduced Mobilenet V2 for CIFAR10. 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), 0287–0292. https://doi.org/10.1109/CCWC47524.2020.9031131

TABLE II: Strides changed from 2 to 1

Input	operator	t	c	n	s
$224^2 \times 3$	Conv2D	-	32	1	1
$112^c \times 32$	Bottleneck	1	16	1	1
$112^c \times 16$	Bottleneck	6	24	2	1
$56^c \times 24$	Bottleneck	6	32	3	1
$28^c \times 32$	Bottleneck	6	64	4	2
$14^c \times 64$	Bottleneck	6	96	3	1
$14^c \times 96$	Bottleneck	6	160	3	2
$7^c \times 160$	Bottleneck	6	320	1	1
$7^c \times 320$	Conv2D	-	1280	1	1
$7^c \times 1280$	AvgPool	-	-	1	-
$1^c \times 1280$	Conv2D	-	k	-	

to CIFAR10 dataset we have disabled some downsample layers by simply replacing strides 2 with stride 1. Table-2 shows us the disabled downsampled Mobilenet V2 architecture. The values highlighted in yellow color are the changes done with respect to base architecture.

B. Replacing Bottlenecks with HetConv Blocks:

Bottlenecks play a crucial role in Mobilenet V2. It helps in preventing from non-linearity destroying a lot of useful information. We can see the bottlenecks used in Mobilenet V2 in the following figure,

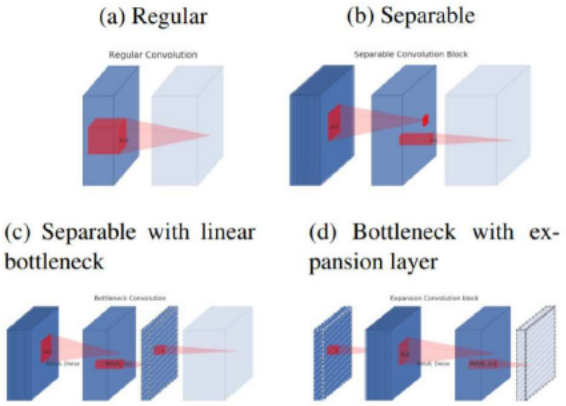

Fig. 1: Bottlenecks used in Mobilenet V2

These bottlenecks shown uses a homogeneous kernel of size 3 3. In our experiment we replace this homogeneous kernel with Heterogeneous Kernel[3] of different sizes. The main idea of replacing this bottleneck with hetconv block is explained below,

Let us assume an input feature map of size $A_i \quad A_i \times C_i$ where A_i is input feature map spatial height and width. Let C_i is the number of input channels. Also consider output feature map of size $A_o \quad A_o \quad C_o$ where A_o is output feature map spatial height and width. C_o is the number of output channels. An output feature map is obtained by multiplying C_o filters with size $k \quad k \quad C_i$ where K is kernel Size. For depthwise and pointwise convolutions applied in Mobilenet V2, the total computational cost will be,

$$cost = A_i \times A_i \times C_i \times k \times k + C_i \times C_o \times A_o \times A_o \quad (1)$$

There is a need to optimise parameter 'k'. Mobilenet V2 uses homogeneous kernel of size 3 3. In heterogeneous kernel, we use variable kernel sizes to reduce the number of parameters without much compromise in accuracy. In HetConv, we set a variable P which controls the number of different types of kernels. For part P, a fraction of $1 \div p$ willbe of kernel $k \times k$ size and remaining $(1 - (1 \div p))$ will be of 1×1 kernel size. So, the computational cost for kernel $k \times k$ at Layer L is,

$$cost1 = (A_i \times A_i \times C_i \times C_o \times k \times k) \div p \quad (2)$$

The Computational cost for 1×1 kernel will be,

$$cost2 = (A_o \times A_o \times C_o) \times (C_i - (C_i \div p)) \quad (3)$$

Total cost will be,

$$cost = cost1 + cost2 \quad (4)$$

The total reduction when compared to standard convolution is,

$$R_{Hetconv} = (1 \div p) + ((1 - (1 \div p)) \div k^2) \quad (5)$$

In the above equation, if we put p=1 it becomes a standard convolution filter The architectural representation of our network is shown in figure-2. In the architecture, we can see that we have replaced bottleneck layers with hetconv blocks. The mathematical explanation for this replacement is shown above. We haven't replaced the bottleneck layers where we require downsampling in the network.

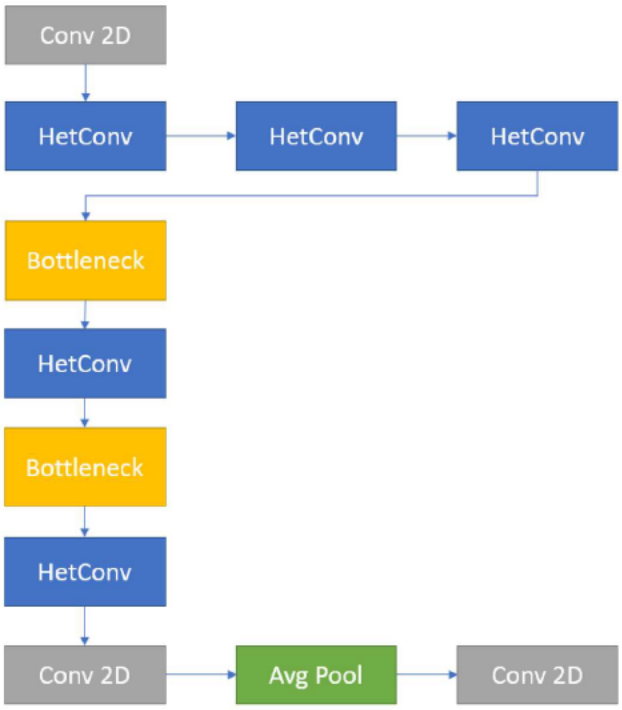

Fig. 2: RMNv2 Architecture

C. Mish activation function

Non Linearity is an important concept in a neural network which is introduced through an activation function. Some of the widely used activation functions are ReLU (Rectified Linear Units), TanH (tan Hyperbolic), Swish activationfunction, etc. In our network, we used another activation function called Mish activation function[4]. It is defined as,

$$f(x) = x \times (\tanh(1 + e^{x})) \quad (6)$$

where $\tanh(1 + e^{x})$ is a softplus activation. The graph of the mish activation function is shown below,

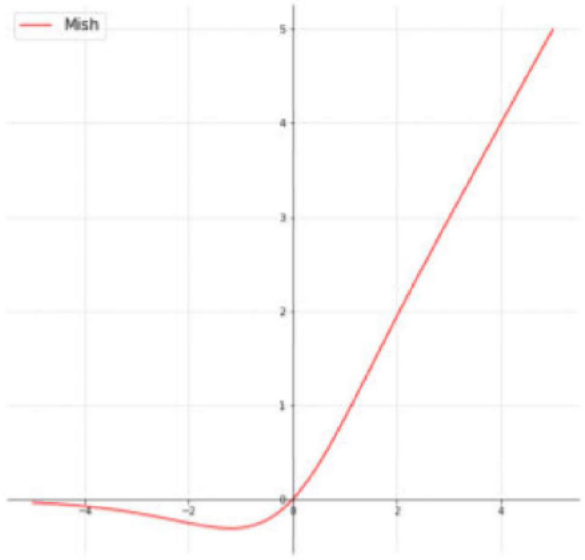

Fig. 3: Mish Activation Function

D. AutoAugmentation

For modern image classifiers, data augmentation plays a crucial role in altering the performance of a classifier. So, effective data augmentation techniques are required to boost up the accuracy. AutoAugmentation[5] is one of the implementations which automatically searches for improved data augmentation policies. In a policy, there are many sub-policies designed. Sub policy consists of two functions, one function is an image processing function like rotate, translate, etc. and other is probability or magnitude of that image processing function applied. This advancement in data augmentation bolstered our proposed network by increasing its accuracy.

IV. Hardware and Software Used

- Intel i7-8700 processor with 32 GB RAM.
- Nvidia Geforce GTX 1080Ti GPU.
- Python version 3.6.7.
- Spyder version 3.6.
- Pytorch version 1.0.
- Livelossplot (Loss and accuracy visualization).

TABLE III: Comparison of Various Results between Baseline and Proposed Network

Comparision of Various Results			
Model	Model Accuracy	#parameters	Model Size(in MB)
Baseline	94.3%	2 2378M	9.1
RMNv2 (Ours)	92.4%	1.0691M	4.3

V. Results

We have trained our network using PyTorch framework using Nvidia Geforce GTX 1080Ti GPU. The network is trained with Stochastic Gradient descent (SGD) optimizer with a variable learning rate of 0.1, 0.01, 0.001 keeping a total number of epochs up to 200. For training the network, we have used a batch size of 128 and for test epoch we useda batch size of 64. The baseline Mobilenet V2 after disabling the downsampling layers gives us the following results shown in Figure,

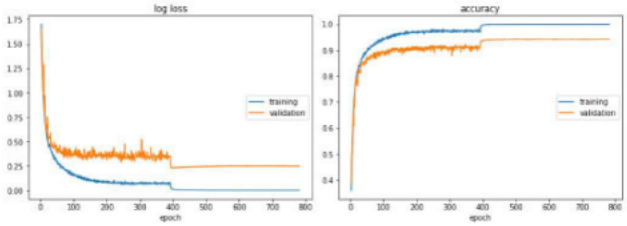

Fig. 4: Baseline Results

we have used the same optimiser, learning rates and batch sizes that are used to train baseline network. The accuracy and loss curves of our network (RMNv2) are shown in Figure-6. For our network we assigned the number of groups, p value to 4 diagrammatically it is shown as,

Fig. 5: Heterogeneous Kernel for P=4

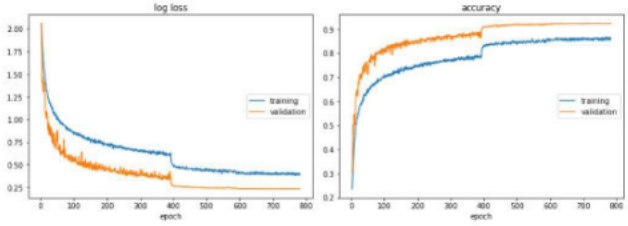

Fig. 6: RMNv2 Results

These curves are plotted using LiveLossPlot Visualization library. Table-3 compares various results like model accuracy, number of parameters, model size, etc.

Table-4 shows us the time taken by each model to complete one epoch as well as the whole training for 200 epochs.

TABLE IV: Comparison of Time between Baseline and Proposed Network

Comparision of Time		
Model	For one epoch(in min)	For Complete training(in hr)
Baseline	1.9097	6.7
RMNv2 (Ours)	0.7621	2.7

In the appendix section, we have shown model summary for both baseline and RMNv2. In that table, we can clearly see how many parameters produced by each layer as well as the output shape produced by the input tensor multiplying with different filters.

VI. Conclusion

The results show that our network Reduced Mobilenet V2 (RMNv2) requires fewer computations, lesser time than the original model with not much decrease in accuracy, 1.9%. It is clearly shown that replacing bottlenecks with HetConv layer helped us in decreasing the model size by 52.2%. In order to boost up the accuracy, we replaced ReLU6 activation with Mish activation and we also used effective data augmentation technique like Autoaugmentation whichhelped us in increasing accuracy to 92.4%. Altogether it forms up a new architecture called Reduced Mobilenet V2 (RMNv2). The optimised model size with competing accuracy makes the RMNv2 model suitable for deploying in resource-constrained devices like Embedded devices, Mobile devices, etc. For future work, one can try further model reduction techniques like model pruning, quantization and huffman encoding etc. And also we can try to improve the accuracy by implementing better augmentation techniques, transfer learning and better architectural changes like changing convolutions etc. We can also check the real-time inference of RMNv2 in embedded devices to demonstrate its performance in real-world applications.

References

[1] Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).

[2] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks." arXiv preprint arXiv:1801.04381v4 (2019)

[3] Pravendra Singh, Vinay Kumar Verma, Piyush Rai, Vinay P. Namboodiri. "HetConv: Heterogeneous Kernel-Based Convolutions for Deep CNNs" arXiv preprint arXiv:1903.04120v2 (2019)

[4] Diganta Misra, "Mish: A Self Regularized Non-Monotonic Neural Activation Function" arXiv preprint arxiv:1908.08681 (2019)

[5] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, Quoc V. Le Google Brain, "AutoAugment: Learning Augmentation Strategies from Data" arXiv preprint arXiv:1805.09501v3 (2019)

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, " Imagenet classification with deep convolutional neural networks" In Advances in neural information processing systems, pages 10971105, 2012.

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge, International Journal of Computer Vision, 2015

[8] J Wu, C Leng, Y Wang, Q Hu, and J Cheng "Quantized convolutional neural networks for mobile devices". arXiv preprint arXiv:1512.06473, 2015.

[9] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. "Compressing neural networks with the hashing trick". CoRR, abs/1504.04788, 2015

[10] S. Han, H. Mao, and W. J. Dally."Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding". CoRR, abs/1510.00149, 2, 2015.

[11] Karen Simonyan, Andrew Zisserman. "VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION" arXiv preprint arXiv:1409.1556v6 (2015)

[12] G. Hinton, O. Vinyals, and J. Dean. "Distilling the knowledge in a neural network". arXiv preprint arXiv:1503.02531, 2015.

[13] M. Jaderberg, A. Vedaldi, and A. Zisserman. "Speeding up convolutional neural networks with low rank expansions." arXiv preprint arXiv:1405.3866, 2014.

[14] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. "Speeding-up convolutional neural networks using fine-tuned cp-decomposition." arXiv preprint arXiv:1412.6553, 2014.

[15] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the inception architecture for computer vision." arXiv preprint arXiv:1512.00567, 2015.

[16] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. arXiv preprint arXiv:1602.07261, 2016

[17] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 1mb model size. arXiv preprint arXiv:1602.07360, 2016.

[18] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, Kurt Keutzer SqueezeNext: Hardware-Aware Neural Network Design arXiv preprint arXiv:1803.10615v2

APPENDIX A
MODEL SUMMARIES

Layer	output Shape	Param #
Conv2d-1	[-1, 32, 32, 32]	864
BatchNorm2d-2	[-1, 32, 32, 32]	64
Conv2d-3	[-1, 32, 32, 32]	1,024
BatchNorm2d-4	[-1, 32, 32, 32]	64
Conv2d-5	[-1, 32, 32, 32]	288
BatchNorm2d-6	[-1, 32, 32, 32]	64
Conv2d-7	[-1, 16, 32, 32]	512
BatchNorm2d-8	[-1, 16, 32, 32]	32
BaseBlock-9	[-1, 16, 32, 32]	0
Conv2d-10	[-1, 96, 32, 32]	1,536
BatchNorm2d-11	[-1, 96, 32, 32]	192
Conv2d-12	[-1, 96, 32, 32]	864
BatchNorm2d-13	[-1, 96, 32, 32]	192
Conv2d-14	[-1, 24, 32, 32]	2,304
BatchNorm2d-15	[-1, 24, 32, 32]	48
BaseBlock-16	[-1, 24, 32, 32]	0
Conv2d-17	[-1, 144, 32, 32]	3,456
BatchNorm2d-18	[-1, 144, 32, 32]	288
Conv2d-19	[-1, 144, 32, 32]	1,296
BatchNorm2d-20	[-1, 144, 32, 32]	288
Conv2d-21	[-1, 24, 32, 32]	3,456
BatchNorm2d-22	[-1, 24, 32, 32]	48
BaseBlock-23	[-1, 24, 32, 32]	0
Conv2d-24	[-1, 144, 32, 32]	3,456
BatchNorm2d-25	[-1, 144, 32, 32]	288
Conv2d-26	[-1, 144, 32, 32]	1,296
BatchNorm2d-27	[-1, 144, 32, 32]	288
Conv2d-28	[-1, 32, 32, 32]	4,608
BatchNorm2d-29	[-1, 32, 32, 32]	64
BaseBlock-30	[-1, 32, 32, 32]	0
Conv2d-31	[-1, 192, 32, 32]	6,144
BatchNorm2d-32	[-1, 192, 32, 32]	384
Conv2d-33	[-1, 192, 32, 32]	1,728
BatchNorm2d-34	[-1, 192, 32, 32]	384
Conv2d-35	[-1, 32, 32, 32]	6,144
BatchNorm2d-36	[-1, 32, 32, 32]	64
BaseBlock-37	[-1, 32, 32, 32]	0
Conv2d-38	[-1, 192, 32, 32]	6,144
BatchNorm2d-39	[-1, 192, 32, 32]	384
Conv2d-40	[-1, 192, 32, 32]	1,728
BatchNorm2d-41	[-1, 192, 32, 32]	384
Conv2d-42	[-1, 32, 32, 32]	6,144
BatchNorm2d-43	[-1, 32, 32, 32]	64
BaseBlock-44	[-1, 32, 32, 32]	0
Conv2d-45	[-1, 192, 32, 32]	6,144
BatchNorm2d-46	[-1, 192, 32, 32]	384
Conv2d-47	[-1, 192, 16, 16]	1,728
BatchNorm2d-48	[-1, 192, 16, 16]	384
Conv2d-49	[-1, 64, 16, 16]	12,288
BatchNorm2d-50	[-1, 64, 16, 16]	128
BaseBlock-51	[-1, 64, 16, 16]	0
Conv2d-52	[-1, 384, 16, 16]	24,576
BatchNorm2d-53	[-1, 384, 16, 16]	768
Conv2d-54	[-1, 384, 16, 16]	3,456
BatchNorm2d-55	[-1, 384, 16, 16]	768
Conv2d-56	[-1, 64, 16, 16]	24,576
BatchNorm2d-57	[-1, 64, 16, 16]	128
BaseBlock-58	[-1, 64, 16, 16]	0
Conv2d-59	[-1, 384, 16, 16]	24,576
BatchNorm2d-60	[-1, 384, 16, 16]	768
Conv2d-61	[-1, 384, 16, 16]	3,456
BatchNorm2d-62	[-1, 384, 16, 16]	768
Conv2d-63	[-1, 64, 16, 16]	24,576
BatchNorm2d-64	[-1, 64, 16, 16]	128
BaseBlock-65	[-1, 64, 16, 16]	0
Conv2d-66	[-1, 384, 16, 16]	24,576
BatchNorm2d-67	[-1, 384, 16, 16]	768
Conv2d-68	[-1, 384, 16, 16]	3,456
BatchNorm2d-69	[-1, 384, 16, 16]	768
Conv2d-70	[-1, 64, 16, 16]	24,576
BatchNorm2d-71	[-1, 64, 16, 16]	128
BaseBlock-72	[-1, 64, 16, 16]	0
Conv2d-73	[-1, 384, 16, 16]	24,576
BatchNorm2d-74	[-1, 384, 16, 16]	768
Conv2d-75	[-1, 384, 16, 16]	3,456
BatchNorm2d-76	[-1, 384, 16, 16]	768
Conv2d-77	[-1, 96, 16, 16]	36,864
BatchNorm2d-78	[-1, 96, 16, 16]	192
BaseBlock-79	[-1, 96, 16, 16]	0
Conv2d-80	[-1, 576, 16, 16]	55,296
BatchNorm2d-81	[-1, 576, 16, 16]	1,152
Conv2d-82	[-1, 576, 16, 16]	5,184
BatchNorm2d-83	[-1, 576, 16, 16]	1,152
Conv2d-84	[-1, 96, 16, 16]	55,296
BatchNorm2d-85	[-1, 96, 16, 16]	192
BaseBlock-86	[-1, 96, 16, 16]	0
Conv2d-87	[-1, 576, 16, 16]	55,296
BatchNorm2d-88	[-1, 576, 16, 16]	1,152
Conv2d-89	[-1, 576, 16, 16]	5,184
BatchNorm2d-90	[-1, 576, 16, 16]	1,152
Conv2d-91	[-1, 96, 16, 16]	55,296
BatchNorm2d-92	[-1, 96, 16, 16]	192
BaseBlock-93	[-1, 96, 16, 16]	0
Conv2d-94	[-1, 576, 16, 16]	55,296
BatchNorm2d-95	[-1, 576, 16, 16]	1,152
Conv2d-96	[-1, 576, 8, 8]	5,184
BatchNorm2d-97	[-1, 576, 8, 8]	1,152

Layer	output shape	Param #
Conv2d-98	[-1, 160, 8, 8]	92,160
BatchNorm2d-99	[-1, 160, 8, 8]	320
BaseBlock-100	[-1, 160, 8, 8]	0
Conv2d-101	[-1, 960, 8, 8]	153,600
BatchNorm2d-102	[-1, 960, 8, 8]	1,920
Conv2d-103	[-1, 960, 8, 8]	8,640
BatchNorm2d-104	[-1, 960, 8, 8]	1,920
Conv2d-105	[-1, 160, 8, 8]	153,600
BatchNorm2d-106	[-1, 160, 8, 8]	320
BaseBlock-107	[-1, 160, 8, 8]	0
Conv2d-108	[-1, 960, 8, 8]	153,600
BatchNorm2d-109	[-1, 960, 8, 8]	1,920
Conv2d-110	[-1, 960, 8, 8]	8,640
BatchNorm2d-111	[-1, 960, 8, 8]	1,920
Conv2d-112	[-1, 160, 8, 8]	153,600
BatchNorm2d-113	[-1, 160, 8, 8]	320
BaseBlock-114	[-1, 160, 8, 8]	0
Conv2d-115	[-1, 960, 8, 8]	153,600
BatchNorm2d-116	[-1, 960, 8, 8]	1,920
Conv2d-117	[-1, 960, 8, 8]	8,640
BatchNorm2d-118	[-1, 960, 8, 8]	1,920
Conv2d-119	[-1, 320, 8, 8]	307,200
BatchNorm2d-120	[-1, 320, 8, 8]	640
BaseBlock-121	[-1, 320, 8, 8]	0
Conv2d-122	[-1, 1280, 8, 8]	409,600
BatchNorm2d-123	[-1, 1280, 8, 8]	2,560
Linear-124	[-1, 10]	12,810

TABLE V: Summary of Baseline

Layer	output Shape	Param #
Conv2d-1	[-1, 32, 32, 32]	864
BatchNorm2d-2	[-1, 32, 32, 32]	64
Conv2d-3	[-1, 32, 32, 32]	1,024
BatchNorm2d-4	[-1, 32, 32, 32]	64
Conv2d-5	[-1, 32, 32, 32]	288
BatchNorm2d-6	[-1, 32, 32, 32]	64
Conv2d-7	[-1, 16, 32, 32]	512
BatchNorm2d-8	[-1, 16, 32, 32]	32
BaseBlock-9	[-1, 16, 32, 32]	0
Conv2d-10	[-1, 24, 32, 32]	864
Conv2d-11	[-1, 24, 32, 32]	384
HetConv-12	[-1, 24, 32, 32]	0
Conv2d-13	[-1, 24, 32, 32]	1,296
Conv2d-14	[-1, 24, 32, 32]	576
HetConv-15	[-1, 24, 32, 32]	0
Conv2d-16	[-1, 32, 32, 32]	1,728
Conv2d-17	[-1, 32, 32, 32]	768
HetConv-18	[-1, 32, 32, 32]	0
Conv2d-19	[-1, 32, 32, 32]	2,304
Conv2d-20	[-1, 32, 32, 32]	1,024
HetConv-21	[-1, 32, 32, 32]	0
Conv2d-22	[-1, 32, 32, 32]	2,304
Conv2d-23	[-1, 32, 32, 32]	1,024
HetConv-24	[-1, 32, 32, 32]	0
Conv2d-25	[-1, 192, 32, 32]	6,144
BatchNorm2d-26	[-1, 192, 32, 32]	384
Conv2d-27	[-1, 192, 16, 16]	1,728
BatchNorm2d-28	[-1, 192, 16, 16]	384
Conv2d-29	[-1, 64, 16, 16]	12,288
BatchNorm2d-30	[-1, 64, 16, 16]	128
BaseBlock-31	[-1, 64, 16, 16]	0
Conv2d-32	[-1, 64, 16, 16]	9,216
Conv2d-33	[-1, 64, 16, 16]	4,096
HetConv-34	[-1, 64, 16, 16]	0
Conv2d-35	[-1, 64, 16, 16]	9,216
Conv2d-36	[-1, 64, 16, 16]	4,096
HetConv-37	[-1, 64, 16, 16]	0
Conv2d-38	[-1, 64, 16, 16]	9,216
Conv2d-39	[-1, 64, 16, 16]	4,096
HetConv-40	[-1, 64, 16, 16]	0
Conv2d-41	[-1, 96, 16, 16]	13,824
Conv2d-42	[-1, 96, 16, 16]	6,144
HetConv-43	[-1, 96, 16, 16]	0
Conv2d-44	[-1, 96, 16, 16]	20,736
Conv2d-45	[-1, 96, 16, 16]	9,216
HetConv-46	[-1, 96, 16, 16]	0
Conv2d-47	[-1, 96, 16, 16]	20,736
Conv2d-48	[-1, 96, 16, 16]	9,216
HetConv-49	[-1, 96, 16, 16]	0
Conv2d-50	[-1, 576, 16, 16]	55,296
BatchNorm2d-51	[-1, 576, 16, 16]	1,152
Conv2d-52	[-1, 576, 8, 8]	5,184
BatchNorm2d-53	[-1, 576, 8, 8]	1,152
Conv2d-54	[-1, 160, 8, 8]	92,160
BatchNorm2d-55	[-1, 160, 8, 8]	320
BaseBlock-56	[-1, 160, 8, 8]	0
Conv2d-57	[-1, 160, 8, 8]	57,600
Conv2d-58	[-1, 160, 8, 8]	25,600
HetConv-59	[-1, 160, 8, 8]	0
Conv2d-60	[-1, 160, 8, 8]	57,600
Conv2d-61	[-1, 160, 8, 8]	25,600
HetConv-62	[-1, 160, 8, 8]	0
Conv2d-63	[-1, 320, 8, 8]	115,200
Conv2d-64	[-1, 320, 8, 8]	51,200
HetConv-65	[-1, 320, 8, 8]	0
Conv2d-66	[-1, 1280, 8, 8]	409,600
BatchNorm2d-67	[-1, 1280, 8, 8]	2,560
Linear-68	[-1, 10]	12,810

TABLE VI: Summary of RMNv2