Squeeze-and-Excitation SqueezeNext: An Efficient DNN for Hardware Deployment

Ravi Teja N.V.S Chappa
Electrical and Computer Engineering
Purdue School of Engineering and Technology
Indianapolis, USA
nagchapp@iupui.edu

Mohamed El-Sharkawy
Electrical and Computer Engineering
Purdue School of Engineering and Technology
Indianapolis, USA
melshark@iupui.edu

***Abstract*—Convolution neural network is being used in field of autonomous driving vehicles or driver assistance systems (ADAS), and has achieved great success. Before the convolution neural network, traditional machine learning algorithms helped the driver assistance systems. Currently, there is a great exploration being done in architectures like MobileNet, SqueezeNext & SqueezeNet. It improved the CNN architectures and made it more suitable to implement on real-time embedded systems. This paper proposes an efficient and a compact CNN to ameliorate the performance of existing CNN architectures. The intuition behind this proposed architecture is to supplant convolution layers with a more sophisticated block module and to develop a compact architecture with a competitive accuracy. Further, explores the bottleneck module and squeezenext basic block structure. The state-of-the-art squeezenext baseline architecture is used as a foundation to recreate and propose a high performance squeezenext architecture. The proposed architecture is further trained on the CIFAR-10 dataset from scratch. All the training and testing results are visualized with live loss and accuracy graphs. Focus of this paper is to make an adaptable and a flexible model for efficient CNN performance which can perform better with the minimum tradeoff between model accuracy, size, and speed. Having a model size of 0.595MB along with accuracy of 92.60% and with a satisfactory training and validating speed of 9 seconds, this model can be deployed on real-time autonomous system platform such as Bluebox 2.0 by NXP.**

Index Terms*—Squeeze-and-Excitation SqueezeNext architecture(SE-SqueezeNext), Convolution Neural Networks (CNN), Deep Neural Networks (DNN), SqueezeNext, SqueezeNet, CIFAR-10, Pytorch.

I. Introduction

Most of the applications in real-time such as computer vision, robotics, image recognition and classification [10], autonomous vehicles and ADAS have been transformed with help of Deep Neural Networks. This has been made possible by undergoing deep research in this field over the past decade with the availability of more training data, and for training and validation having faster hardware. But not great amount of work is done in aspects of model size and speed. There is a down side to DNNs that it require more budget of resources that refers to more computation and memory resources. Most recently, DNN attained a bewildering benchmark of accuracy at 99% with GPipe [19]. With the emergence of macro architectures such as SqueezeNet, SqueezeNext and MobileNet, DNNs can be implemented on embedded systems [16]. SqueezeNet uses the fire module's squeeze and expand layer approach to design a smaller and shallow CNN architecture but it comes at a cost of model accuracy which is about 78%. Though, SqueezeNext achieves better results but it still can be improved as suggested by the author of this architecture with further hyperparameter tuning and modifications. This paper proposes an efficient network architecture in order to a build very small, efficient DNN model that is the proposed Squeeze-and-Excitation SqueezeNext architecture. Structure of the paper is as follows. In section II, we discuss about the foundation and existing architectures i.e., SqueezeNet and SqueezeNext. Followed by section III describes the proposed Squeeze-and-Excitation architecture. Section IV explains the hardware and software used to train and test the proposed architecture. Section V shows the experimental results obtained after training and validating the proposed neural network using CIFAR-10 dataset. Finally, the paper conclusion is made in section VI that demonstrates the wholesome overview of the paper.

II. Previous Work

A. SqueezeNet Architecture

This section reviews the SqueezeNet architecture [3] which achieves a better model performance than AlexNet with 50x fewer parameters when trained on ImageNet. A SGD optimizer is used with a small learning rate where there is no proper scheduling of learning rate included. When 1x1 kernels are used it lead to the model depth reduction as a result lead to computation reduction of the 3x3 filters. Deploying this model on a real time embedded system is easy as it has a model size less than 5MB. The parameter count is reduced greatly because of the lack of Fully Connected(FCC) layer. Instead, the class classification scores are calculated using softmax function and an average pooling layer. The foundation modules for this CNN architecture are fire modules. Due to these modules, it achieved a very small model size with better model accuracy. Further, SqueezeNet version 1.1 was introduced later which further reduced the parameter count by reducing the number

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

Chappa, R. T. N. V. S., & El-Sharkawy, M. (2020). Squeeze-and-Excitation SqueezeNext: An Efficient DNN for Hardware Deployment. 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), 0691–0697. https://doi.org/10.1109/CCWC47524.2020.9031119

of kernels and kernel sizes. The result of this reduction is 2.4 times better than the baseline SqueezeNet v1.0 with no affect on the accuracy of the model. Motivation from the incredibly small macro architecture of SqueezeNet helped in some modifications of the proposed architecture.

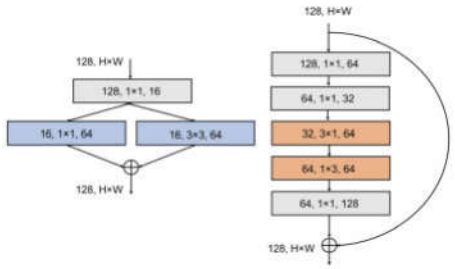


Fig. 1: Fire module of SqueezeNet and bottleneck module of SqueezeNext.

B. SqueezeNext Architecture

This architecture matches the model performance of AlexNet with 112x fewer parameters when trained on ImageNet. It is also able to match VGG-19 accuracy with 31x fewer parameters than VGG-19. The foundation module for this CNN architecture is the bottleneck module. Squeezenext baseline architecture [4] emerged after the advent of the squeezenet architecture. Squeezenext uses squeezenet baseline as its foundation and consists of the following strategies:

1) More aggressive channel reduction by introducing a 2-stage squeeze module, further reducing number of parameters used with the 3x3;
2) 3x3 separable convolutions, and remove the additional 1x1 branch after the squeeze module to reduce the model size;
3) An element-wise addition skip connection similar to ResNet architecture.

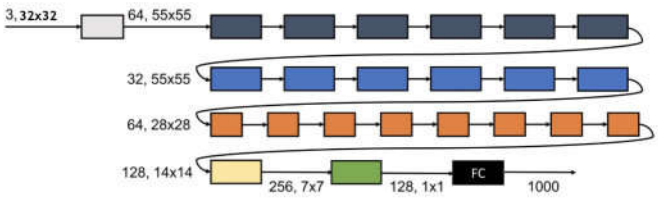


Fig. 2: Baseline architecture of SqueezeNext.

Squeezenext baseline architecture comprises of bottleneck modules with four stage implementation, batch normalization layers, Relu and Relu (in-place) nonlinear activations, max, and average pool layers, Xavier uniform initialization, a spatial resolution layer and a fully connected layer in the last with this [6,6,8,1] four stage block configuration. In fact, a better model accuracy and size is attained in comparison to squeezenet baseline architecture. The squeezenext baseline [6,6,8,1] architecture configuration shown in Fig. 2 that illustrates the squeezenext baseline architecture implemented on the CIFAR-10 dataset with input size 32x32 and 3 input channels. This is the input for the first convolution, the white block. Now, the output of the first convolution is the input for a max pooling layer after the first convolution, not shown in Figure 3, but shown in Figure 6. The consecutive different colored blocks that are dark blue, blue, orange and yellow blocks after the first convolution and max-pooling represents the four-stage configuration implementation followed by a green block, representing the spatial resolution layer and the average pooling layer. Finally, followed by one black block that is a fully connected layer. The color change in the four stage implementation configuration blocks that are dark blue, blue, orange and yellow blocks depict a change in the input feature maps resolution.

III. Squeeze-and-Excitation SqueezeNext Architecture

The proposed Squeeze-and-Excitation SqueezeNext is a CNN model. Motivation for this architecture is through SqueezeNet, Mobilenet [5] and SqueezeNext architectures. This contains basic blocks organized in 4-stage configuration called bottleneck modules, a squeeze-and-excitation(SE) block [6], average pooling layer, fully connected layer and a spatial resolution layer. Nestrov [9], decay and momentum are implemented with SGD optimizer. We implemented learning rate schedule which is exponentially decaying by updating learning rate in four stages: 1st after 60 epochs, 2nd after 120 epochs, 3rd after 150 epochs and last after 180 epochs. As shown in Fig. 6, the bottleneck module contains a basic block with 1x1 convolution, another basic block with 1x1 convolution, basic block with 3x1 convolution, basic block with 1x3 convolution, last basic block with 1x1 convolution and finally a se block. From Fig.3, basic block contains convolution layer followed by BN layer [8] & ReLu-in-place. Here basic blocks line up convolutions enclosed by bottleneck modules which are grouped and organized in the 4-stage implementation configuration along with a spatial layer, dropout layer [11], se block, average pool layer and a fully connected layer are shown in Fig. 5. To reduce the parameter count, spatial layer can be eliminated in the small sized models of the proposed architecture.

The descriptions of two parameters which helped in shrinking the model are given as follows along with the squeeze-and-excitation(SE) block. The [1,2,4,1] 4-stage configuration of the SE-SqueezeNext architecture is illustrated in Fig. 7. The Squeeze-and-Excitation SqueezeNext bottleneck module containing basic blocks with SE blocks included are illustrated in Fig. 6. Table V presents the SE-SqueezeNext table with [1,2,4,1] 4-stage configuration, spatial resolution, dropout layer, SE block and FC convolution. In comparison with SqueezeNext baseline architecture, different basic block is being used by the

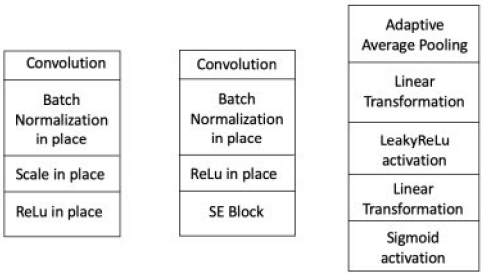


Fig. 3: Basic block of SqueezeNext baseline, Basic block of SE-SqueezeNext and SE block.

bottleneck module in this architecture which is shown in Fig. 3.

A. Squeeze-and-Excitation block

The structure of the SE block is depicted in Fig. 4. For any given transformation $\mathbf{F}_{tr}$ mapping the input $\mathbf{X}$ to the feature maps $\mathbf{U}$ where $\mathbf{U} \in \Re^{H \times W \times C}$, e.g. a convolution, we can construct a corresponding SE block to perform feature recalibration. A squeeze operation is done through which the features U are first passed, then by aggregating feature maps across their spatial dimensions H×W a channel descriptor is produced. This descriptor primarily generates an embedding of the global distribution of channel-wise feature responses which allows all its layers to use the information from the global receptive field of the network. After the aggregation is performed, a simple self-gating mechanism where a collection of per-channel modulation weights are produced by taking the embedding as an input. This is the excitation operation. The above obtained weights are used to produce the output of the SE block by applying to the feature maps. This can be loaded into subsequent layers of the network.

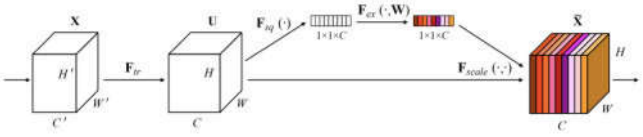


Fig. 4: Structure of SE block

B. Dropout layer

This layer is used to improvise the overfitting problem existing in CNNs or neural networks. This is a regularization method for approximation and dropping some random weight from large set of weights in a CNN or neural network. A DNN or a deep CNN trained on a small dataset (lack of data) can result in overfitting problem which results in poor performance and increase generalization problem or errors due to the problem of over fitting. This is a simple approach to reduce overfitting in a CNN and improve the performance of DNN/CNN.

C. Resolution Multiplier

To reduce the cost of computation of a neural network, this multiplier is used. Resolution multiplier ultimately reduces the internal representation of each layer and is applied to input image. Generally, value of the input resolution is set absolutely. Reduced DNN models/ architectures are generated if the value is less than 1. The cost of computation is reduced by the square of this parameter. In this paper, we used the values of 6, 7, 8, 10, 11, 12, 14, 16, 21 & 23.

D. Width Multiplier

To design smaller and less computationally expensive models, width multiplier is used. At each layer, this also helps for making a unvarying thin CNN. Default or general values for width multiplier are 0.25, 0.5, 0.75 and 1. It is the key element for decreasing the cost of computation and parameter count. Mathematically, it is quadratically reduces it by two times the power of width multiplier.

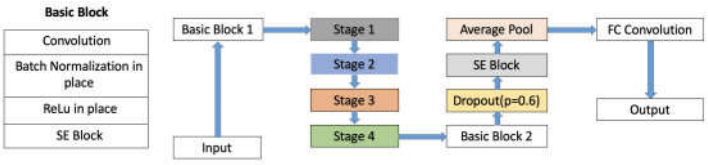


Fig. 5: Illustration of Basic Block (left) and Squeeze-and-Excite SqueezeNext architecture.

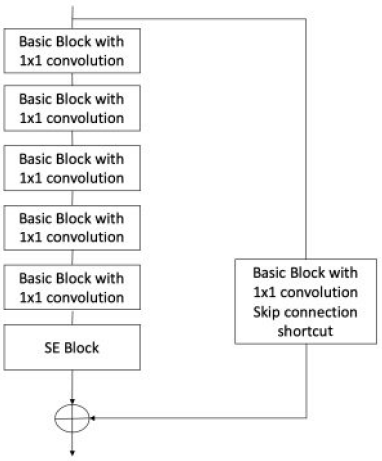


Fig. 6: Illustration of Squeeze-and-Excitation SqueezeNext'sbottleneck module

IV. Hardware and Software Requirements

- Aorus Geforce RTX 2080Ti GPU.
- Nvidia Geforce GTX 1080Ti GPU.
- Python version 3.6.7.
- Spyder version 3.6.
- Pytorch version 1.0.
- Livelossplot (Loss and accuracy visualization).
- Netscope (SqueezeNext baseline visualization).

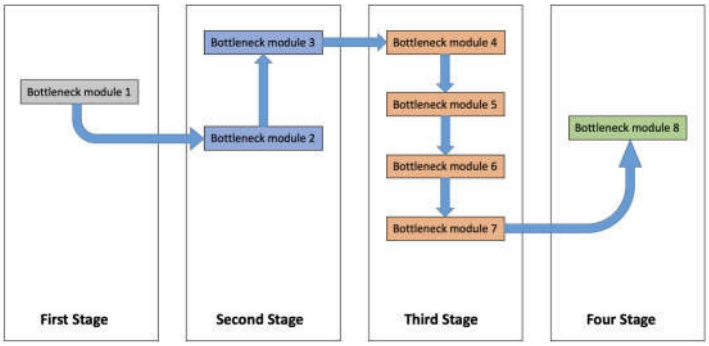


Fig. 7: 4-stage [1,2,4,1] configuration of the Squeeze-and-Excitation SqueezeNext architecture.

V.R ESULTS

A. *Squeeze-and-Excitation SqueezeNext Results*

Efficient CNN architectures are generated due to the modifications in the proposed architecture, which is tangible as SE-SqueezeNext having a model sizes ranging from 0.595MB to 6.59MB as shown in Table III. When compared to baseline Squeezenext's model size i.e., 9.531MB, the proposed SE-Squeezenext architecture has a reduced model size of 0.595MB. Few major factors leading to this reduction of model size are various resolution and width multipliers. From the observations, dropout layer performance is finer than BN layer. The detailed description of squeeze and excitation operators in SE block are as follows:

1) Squeeze Operator:: The significance of using global average pooling over global max pooling as our squeeze operator is examined. Average pooling [21] achieves better performance than max pooling, even though both are effective. The basis of squeeze operation is justified by selecting average pooling. However, we note that the performance of SE blocks is fairly robust to the choice of specific aggregation operator.

2) Excitation Operator:: The option for non-linearity in the excitation mechanism is assessed here. Two further options: LeakyReLU and Tanh are considered, and experiment with replacing the sigmoid with these alterantive non-linearities. By interchanging the sigmoid with Tanh slightly worsens the performance, where LeakyRelu is emphatically worse and causes the performance of SE-SqueezeNext to drop below the baseline of SqueezeNext. This propounds that for the SE block to be efficient, careful selection of excitation operator is important.

Useful modifications to produce small DNNs and deployable on real time embedded devices are resolution and width multipliers. Therefore, SE-SqueezeNext-10-0.5 is 16X compact than SqueezeNext-23-1x-v1. This architecture is made more efficient, flexible and compact by implementing in-place operations such as Relu-in-place and eliminating the extra max pooling layers using a SE block, width and resolution multipliers. Without using the transfer learning method, each model is validated on CIFAR-10 from the scratch. Deployment on a real-time system having memory constraints is the major advantage of this architecture. The accuracy of this architecture is enhanced by dropout layer. The format for SE-SqueezeNext in all the tables illustrates SE-SqueezeNext with resolution multiplier along with width multiplier.

From the results presented in Table I, it is observed that SE-SqueezeNext with dropout layer along with appropriate use of multipliers and a large difference is made by bottleneck module. Different dropout layer probabilities results of SE-SqueezeNext are presented in Table IV. Different accuracies for various models are shown in Table V.

TABLE I: Results comparison with SqueezeNet & SqueezeNext

Name of Model	Accuracy%	Size of model(MB)	Speed of model(seconds)
SqueezeNet-v1.0	79.59	3.01	4
SqueezeNet-v1.1	77.55	2.96	4
SqueezeNext-23-1x-v1	87.15	2.57	19
SqueezeNext-23-1x-v5	87.95	2.57	19
SqueezeNext-23-2x-v1	90.51	9.53	22
SqueezeNext-23-2x-v5	90.50	9.53	28
SE-SqueezeNext-10-1.0x-v1	90.48	1.81	13
SE-SqueezeNext-10-2.0x-v1	92.60	6.59	21

*All results are 3 average runs with SGD, LR is 0.1

TABLE II: Squeeze-and-Excitation SqueezeNext Results withdifferent resolution multipliers

Name of Model	Resolution	Accuracy%	Size of Model(MB)	Speed of Model(seconds)
SE-SqueezeNext-06-1x-v1	1111	87.84	1.22	9
SE-SqueezeNext-10-1x-v1	1241	90.48	1.81	13
SE-SqueezeNext-12-1x-v1	1261	90.50	2.16	15
SE-SqueezeNext-14-1x-v1	1281	89.93	2.52	16
SE-SqueezeNext-22-1x-v1	12161	80.69	3.94	25
SE-SqueezeNext-23-1x-v1	22161	81.41	3.97	26

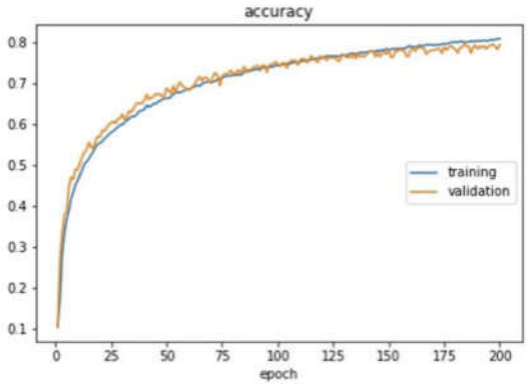
*Results obtained are 3 average runs with LR, SGD as 0.1

TABLE III: Different width multipliers results of Squeeze-and-Excitation SqueezeNext

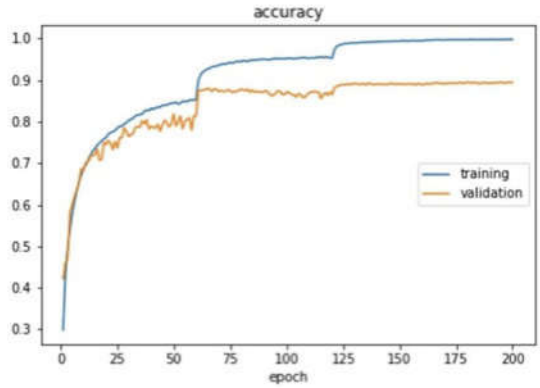
Name of Model	Width	Accuracy%	Size of Model(MB)	Speed of Model(seconds)
SE-SqueezeNext-10-0.5x-v1	0.5x	86.71	0.595	10
SE-SqueezeNext-10-0.6x-v1	0.6x	88.18	0.760	11
SE-SqueezeNext-10-0.7x-v1	0.7x	89.39	0.968	12
SE-SqueezeNext-10-0.8x-v1	0.8x	90.33	1.21	12
SE-SqueezeNext-10-0.9x-v1	0.9x	89.79	1.48	13
SE-SqueezeNext-10-1.0x-v1	1.0x	90.48	1.81	13
SE-SqueezeNext-10-1.2x-v1	1.2x	91.04	2.48	15
SE-SqueezeNext-10-1.5x-v1	1.5x	92.07	3.81	16
SE-SqueezeNext-10-1.7x-v1	1.7x	92.10	4.78	18
SE-SqueezeNext-10-2.0x-v1	2.0x	92.60	6.59	21

TABLE IV: Different dropout layer probabilities results of Squeeze-and-Excitation SqueezeNext

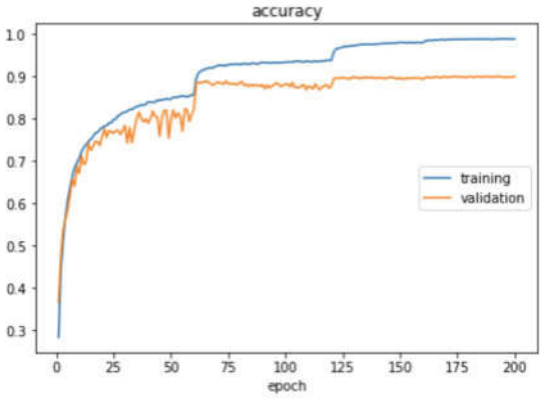
Name of Model	dropout (p)	Accuracy%	Size of Model(MB)	Speed of Model(seconds)
SE-SqueezeNext-10-1x-v1	0.1	91.06	2.16	15
SE-SqueezeNext-10-1x-v1	0.2	90.33	2.16	15
SE-SqueezeNext-10-1x-v1	0.3	90.46	2.16	14
SE-SqueezeNext-10-1x-v1	0.4	90.06	2.16	14
SE-SqueezeNext-10-1x-v1	0.5	90.22	2.16	13
SE-SqueezeNext-10-1x-v1	0.6	90.53	2.16	13



(a) SqueezeNet accuracy



(b) SqueezeNext accuracy



(c) Squeeze-and-Excitation SqueezeNext accuracy

Fig. 8: Accuracy plots comparison with baseline architectures

VI. Conclusion

It is evident from the results that there is a tradeoff between model's size, accuracy and speed for different resolution and width multipliers. We can also observe that there is no change in the accuracy of model upon reducing the depth of model. Selection of hyperparmeters such as width and resolution multipliers are the primary factors in loss minimization, obtaining a good size and an accurate model. From the results, the performance of SGD optimizer along with nestrov, decay and momentum are observed to be better than other terms. Proposed architecture was trained and validated on CIFAR-10 with a better model size of 0.595 MB which is 6x better than SqueezeNet Baseline & 16x better than SqueezeNext baseline. A best model speed of 9 sec which is 10 sec better than SqueezeNext baseline and at the same time similar to SqueezeNet baseline. As an extension to this work, for increasing the performance of this architecture transfer learning and data augmentation [12], [22] can be used.

References

[1] J. K. Duggal and M. El-Sharkawy, "Shallow SqueezeNext: An Efficient & Shallow DNN," 2019 IEEE International Conference of Vehicular Electronics and Safety (ICVES), Cairo, Egypt, 2019, pp. 1-6. doi: 10.1109/ICVES.2019.8906416

[2] Duggal, Jayan Kant, 2019. Design Space Exploration of DNNs for Autonomous Systems (MSECE Thesis, Purdue University, Indianapolis).

[3] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J. and Keutzer, K., (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡ 0.5MB model size.arXiv preprint arXiv:1602.07360.

[4] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai,Xiangyu Yue, Peter Jin, Sicheng Zhao, Kurt Keutzer, (2018). SqueezeNext: Hardware-Aware Neural Network Design. arXiv preprint arXiv: 1803.10615

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

[6] J. Hu, L. Shen, S. Albanie, G. Sun and E. Wu, "Squeeze-and-Excitation Networks," in IEEE Transactions on Pattern Analysis and Machine Intelligence. doi: 10.1109/TPAMI.2019.2913372

[7] Ashraf, Khalid, et al. "Shallow networks for high-accuracy road object-detection." arXiv preprint arXiv:1606.01561 (2016).

[8] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).

[9] Aleksandar Botev, Guy Lever, David Barber (2016). Nesterovs Accelerated Gradient and Momentum as approximations to Regularised Update Descent. arXiv preprint arXiv :1607.01981

[10] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[11] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

[12] Stanley, Kenneth O., and Risto Miikkulainen. "Evolving neural networks through augmenting topologies." Evolutionary computation 10.2 (2002): 99-127.

[13] B. Wu, A. Wan, X. Yue, and K. Keutzer. Squeezeseg: Convolutional

neural nets with recurrent crf for real-time road-object segmentation from 3d lidar p
[14] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. arXiv preprint arXiv:1612.01051, 2016.
[15] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. arXiv preprint arXiv:1611.10012, 2016. preprint arXiv:1609.07061, 2016.
[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093, 2014.
[17] V. Sindhwani, T. Sainath, and S. Kumar. Structured transforms for small-footprint deep learning. In Advances in Neural Information Processing Systems, pages 30883096, 2015.
[18] Krizhevsky, Alex, Vinod Nair, and Geoffrey Hinton. "Cifar-10 (canadian institute for advanced research)." URL http://www. cs. toronto. edu/kriz/cifar. html (2010).
[19] Huang, Yanping, et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." arXiv preprint arXiv:1811.06965 (2018).
[20] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems, pages 9199, 2015.
[21] T. Hsiao, Y. Chang and C. Chiu, "Filter-based Deep-Compression with Global Average Pooling for Convolutional Networks," 2018 IEEE International Workshop on Signal Processing Systems (SiPS), Cape Town, 2018, pp. 247-251.
[22] Barret Zoph, Ekin D. Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, Quoc V (2019). Le. Learning Data Augmentation Strategies for Object Detection. arXiv preprint arXiv : 1906.11172
[23] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. arXiv preprint arXiv:1512.00567, 2015.
[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE.
[25] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015
[26] Ludermir, Teresa B., Akio Yamazaki, and Cleber Zanchettin. "An optimization methodology for neural network weights and architectures." IEEE Transactions on Neural Networks 17.6 (2006): 1452-1459.
[27] Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European conference on computer vision. Springer, Cham, 2014.
[28] Guo, Yiwen, Anbang Yao, and Yurong Chen. "Dynamic network surgery for efficient dnns." Advances In Neural Information Processing Systems. 2016.
[29] Denton, Emily L., et al. "Exploiting linear structure within convolutional networks for efficient evaluation." Advances in neural information processing systems. 2014.
[30] Chetlur, Sharan, et al. "cudnn: Efficient primitives for deep learning." arXiv preprint arXiv:1410.0759 (2014).
[31] T.-J. Yang, Y.-H. Chen, and V. Sze. Designing energyefficient convolutional neural networks using energyaware pruning. arXiv preprint, 2017.

TABLE V: Squeeze-and-Excitation SqueezeNext architecture [1,2,4,1] four stage configuration

Layer Name	**Input Size** Wi × Hi × Ci	**Padding** Pw × Ph	**Stride**	**Filter size** Kw × Kh	**Output size** W0 × H0 × C0	**Repeat**	**Parameters**
Convolution 1	32x32x3	0x0	1	3x3	30x30x64	1	1792
Convolution 2	30x30x64	0x0	1	1x1	30x30x16	1	1040
Convolution 3	30x30x16	0x0	1	1x1	30x30x8	1	136
Convolution 4	30x30x8	0x1	1	1x3	30x30x16	1	400
Convolution 5	30x30x16	1x0	1	3x1	30x30x16	1	784
Convolution 6	30x30x16	0x0	1	1x1	30x30x32	1	544
SE Layer 27	30x30x32	-	-	-	30x30x32	1	-
Convolution 32	30x30x32	0x0	2	1x1	30x30x32	1	1056
Convolution 33	15x15x32	0x0	1	1x1	15x15x16	1	528
Convolution 34	15x15x16	0x1	1	1x3	15x15x32	1	1568
Convolution 35	15x15x32	1x0	1	3x1	15x15x32	1	3104
Convolution 36	15x15x32	0x0	1	1x1	15x15x64	1	2112
Convolution 37	15x15x64	0x0	1	1x1	15x15x32	1	2080
Convolution 38	15x15x32	0x0	1	1x1	15x15x16	1	528
Convolution 39	15x15x16	1x0	1	3x1	15x15x32	1	1568
Convolution 40	15x15x32	0x1	1	1x3	15x15x32	1	3104
Convolution 41	15x15x32	0x0	1	1x1	15x15x64	1	2112
SE Layer 52	15x15x64	-	-	-	15x15x64	1	-
Convolution 62	15x15x64	0x0	2	1x1	15x15x64	1	4160
Convolution 63	8x8x64	0x0	1	1x1	8x8x32	1	2080
Convolution 64	8x8x32	1x0	1	3x1	8x8x64	1	6208
Convolution 65	8x8x64	0x1	1	1x3	8x8x64	1	12352
Convolution 66	8x8x64	0x0	1	1x1	8x8x128	1	8320
Convolution 67	8x8x128	0x0	1	1x1	8x8x64	7	57792
Convolution 68	8x8x64	0x0	1	1x1	8x8x32	7	14560
Convolution 69	8x8x32	1x0	1	3x1	8x8x64	7	43456
Convolution 70	8x8x64	0x1	1	1x3	8x8x64	7	86464
Convolution 71	8x8x64	0x0	1	1x1	8x8x128	7	58240
SE Layer 75	8x8x64	-	-	-	8x8x64	1	-
Convolution 102	8x8x128	0x0	2	1x1	8x8x128	1	16512
Convolution 103	4x4x128	0x0	1	1x1	4x4x64	1	8256
Convolution 104	4x4x64	0x1	1	1x3	4x4x128	1	24704
Convolution 105	4x4x128	1x0	1	3x1	4x4x128	1	49280
Convolution 106	4x4x256	0x0	1	1x1	4x4x256	1	65792
Convolution 107	4x4x256	0x0	1	1x1	4x4x128	1	32896
SE Layer 123	8x8x256	-	-	-	8x8x256	1	-
Dropout (p=0.6)	4x4x256	-	-	-	4x4x256	1	-
Average Pool	4x4x256	-	-	-	4x4x256	1	-
Fully Connected Convolution	1x1x128	0x0	1	1x1	1x1x10	1	1290

* Wi, Hi, Ci refer to input width, height and number of channels, Pw, Ph refer to padding width and height, Kw, Kh refer to filter or kernel width and height, W0, H0, C0 refer to output width, output height and output number of channels