Squeeze U-Net: A Memory and Energy Efficient Image Segmentation Network

Nazanin Beheshti
Department of Computer Science
University of Houston
Nsbeheshti70@gmail.com

Lennart Johnsson
Department of Computer Science
University of Houston
johnsson@cs.uh.edu

Abstract

To facilitate implementation of deep neural networks on embedded systems keeping memory and computation requirements low is critical, particularly for real-time mobile use. In this work, we propose a SqueezeNet inspired version of U-Net for image segmentation that achieves a 12X reduction in model size to 32MB, and 3.2X reduction in Multiplication Accumulation operations (MACs) from 287 billion ops to 88 billion ops for inference on the CamVid data set while preserving accuracy. Our proposed Squeeze U-Net is efficient in both low MACs and memory use. Our performance results using Tensorflow 1.14 with Python 3.6 and CUDA 10.1.243 on an NVIDIA K40 GPU shows that Squeeze U-Net is 17% faster for inference and 52% faster for training than U-Net for the same accuracy.

1. Introduction

Until recently, most deep convolutional neural network (CNN) research focused on increasing accuracy on computer vision datasets. The resulting large and powerful neural networks consume considerable energy, memory bandwidth and computational resources. For embedded mobile applications, not only accuracy matters but also energy consumption and model size are of top concerns, and in many applications also inference time for real-time use. A small CNN architecture may enable on-chip storage of the model thereby significantly reducing the energy consumption for retrieving model parameters from DRAM during inference. It also reduces energy requirements for computation. References to off-chip memory may incur a latency of hundreds of compute cycles and dissipate more than hundred times as much energy as arithmetic operations [1]. Our goal is to reduce energy and memory consumption of neural networks so they can be deployed on devices with limited resources while preserving accuracy. To achieve this goal, we devised Squeeze U-Net for image segmentation. The design of this architecture is inspired by SqueezeNet [2] and U-Net [3] . We choose the U-Net architecture [3] as a starting point because it can be successfully trained on small data sets which is beneficial for hardware with limited memory and for applications for which large training data sets may not be available. We replace the down and up sampling layers in U-Net with modules similar to the fire modules in SqueezeNet [2]. Our fire modules use point-wise convolutions followed by an inception stage [4] in which pointwise and 3×3 convolutions are performed independently then concatenated to form the output. It results in a small model with only 2.6 million parameters. The total number of parameters in our Squeeze U-Net is 1.68× , 2.59×, 11.58×, 3.65×, , 16.84×, 27.4× smaller than Mobile Net [5], Deep Lab [6], U-Net [3] , SegNet [7], FCN [8], DeconvNet [9] architectures. To analyze the merit of the Squeeze U-Net architecture, we have implemented both it and U-Net using Tensorflow 1.14 and Python 3.6 with CUDA 10.1.243, and measured execution time of every layer on an NVIDIA K40 GPU. We show that Squeeze U-Net for the contracting path requires 63% of the time for U-Net, for the expanding path 75% of the U-Net time and for the two together 69% of the U-Net time on the CamVid data set. For inference, Squeeze U-Net is 17% faster than U-Net. Next, we describe related work followed by a detailed description of the Squeeze U-Net architecture in Section 3. Section 4 discusses training using the Squeeze U-Net architecture followed by an evaluation in Section 5 and our conclusions in Section 6.

2. Related Work

It has become evident that Deep Neural Networks typically are over parametrized in that a variety of compression techniques have been applied on large parameter spaces with no or only minor loss of accuracy. Redundancy in deep learning models results in waste of computation, memory and energy. Shrinking, factorization or compressing pretrained networks are approaches for removing redundancy and obtaining smaller models [11]–[14]. One of the straightforward approaches in model compression is applying singular value decomposition (SVD) to a pretrained CNN model and finding low rank approximations of the parameters [15]. Other approaches are network pruning which takes a pretrained model and replaces parameters which are below a certain threshold with zeros to form sparse matrices. In sparse matrices relative encoding of indices can be used to compress indices to a few bits at the expense of indirection. To reduce the number of parameters and computational effort for CNNs several techniques based on factorizing the convolution kernel has been used. Depthwise separable convolution [16] separates convolution across channels from

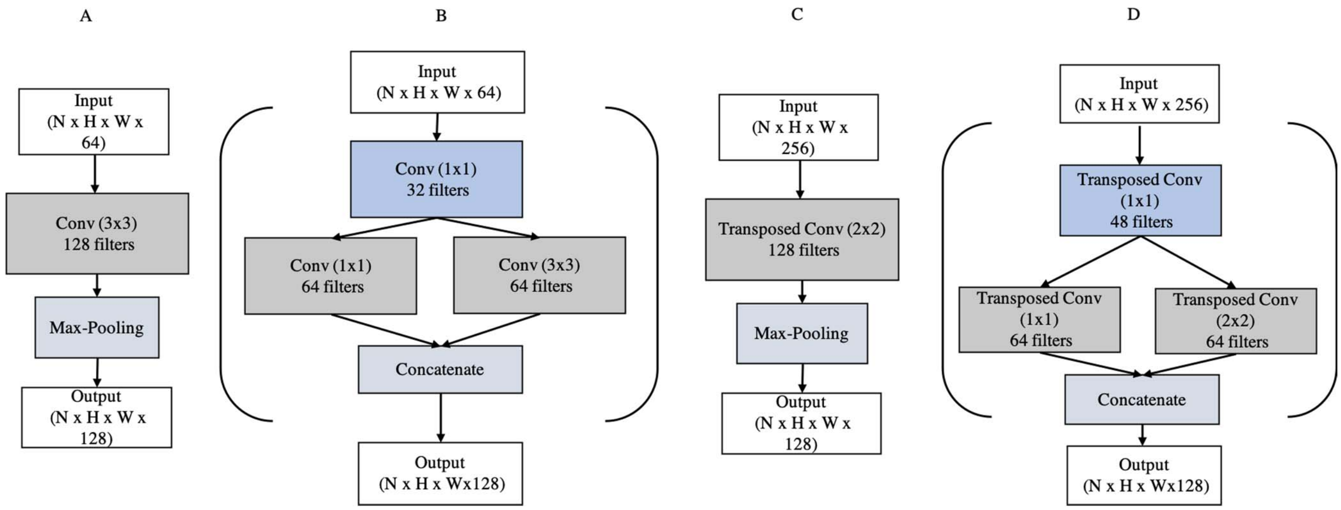

Figure 1: (A) shows convolution in the contracting path in U-Net (B) shows our corresponding Squeeze U-Net implementation using fire modules [2] instead of full convolution to reduce the reduce the number of parameters. (C) shows transposed convolution in the expansive path in U-Net (D) shows our Squeeze U-Net implementation corresponding to (C) . In (D) and (B), the fire modules first squeeze the number of output channels then apply two parallel convolutions with different kernel size to capture missing features from the previous layer and concatenate their outputs.

convolution within channels. Depthwise separable convolution is used in SqueezeNet and in the Squeeze U-Net and in e.g. [17][18]. Potential further reduction in model size by reducing the data type size from 32-bits to eight or 16 bits, commonly known as quantization, may be able to reduce the Squeeze U-Net model, but is not investigated here. Quantization in depthwise separable convolution networks using non-linear activation functions within layers may require special attention as shown in [19] for MobileNetV1. Other approaches towards reducing the computation time have focused on specialized hardware for CNNs, such as e.g. [20]–[22]. [22] focus on data reuse for dense uncompressed models thereby making the hardware architecture energy efficient.

Contributions: For image segmentation we show that Squeeze U-Net achieves the same accuracy as U-Net on the CamVid dataset [23] with 2.6 million parameters, a 12× reduction compared to U-Net [3]. In designing Squeeze U-Net we employ the SqueezeNet fire module [2] design in both the U-Net contracting and expansive paths. The fire modules initial depthwise convolution reduce the number of channels and compensates this reduction by an inception stage with two parallel convolutions each having half the number of output channels of the fire module's output channels. The two parallel convolutions help prevent feature loss and vanishing gradients which may be caused

Table 1: The number of parameters for a K×K size convolution kernel, C_i input channels and C_o output channels are a $K \times K \times C_i \times C_O$. and is given below for a few CNNs. The networks typically use 3×3 kernels or a combination of 3×3 and 1×1 kernels.

Model Architecture	#Params	Model Size (MB)	Kernel Size
Squeeze U-Net	2.59M	32	3×3 2×2 1×1
U-Net	30M	386	3×3 2×2 1×1
SegNet	30M	117	3×3
Deep Lab	20M	83	3×3 – 1×1
FCN -8s	132M	539	3×3–1×1– 4×4 7×7–16×16
DeconvNet	143M	877	1×1 – 3×3 7×7

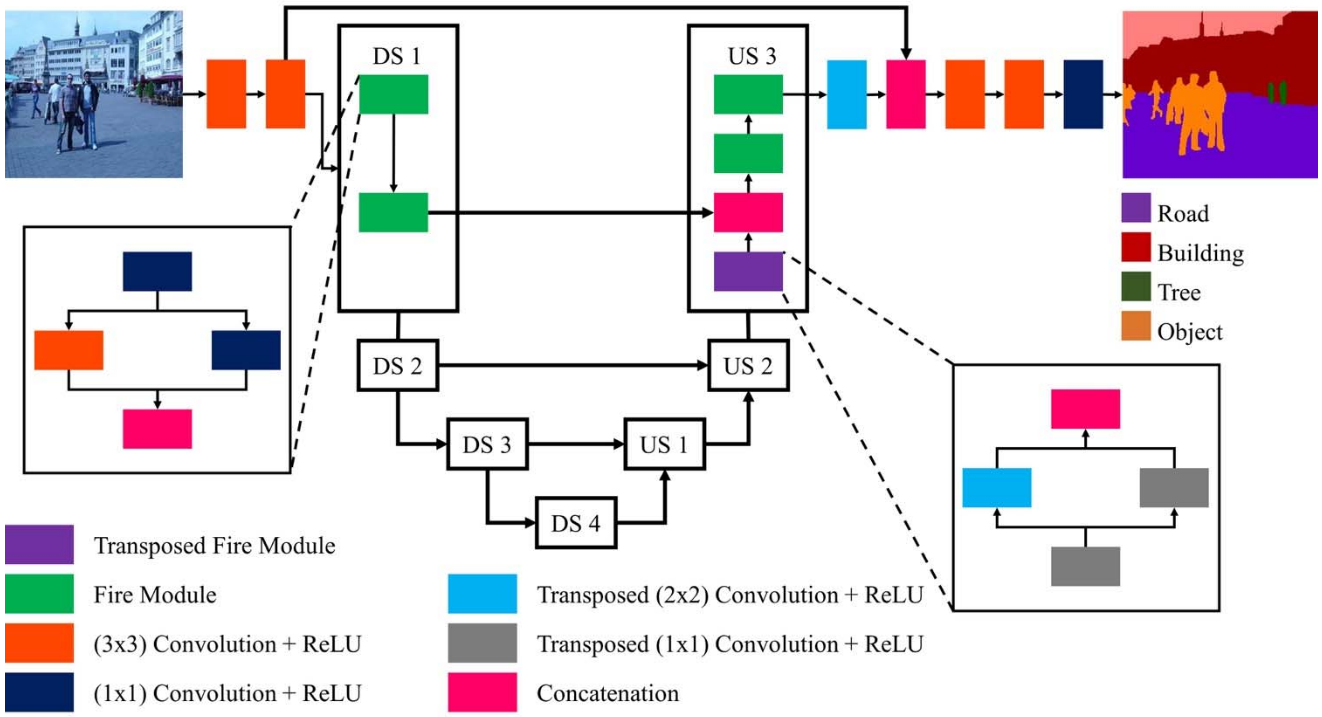

Figure 2: The Squeeze U-Net architecture consists of down sampling units in the contracting U-Net path, and up sampling units in the expansive U-Net path. Every down sampling (DS) unit consists of two fire modules which extract features. The extracted features are passed down to the next down sampling unit and the corresponding up sampling unit (US). Every up-sampling unit consist of a transposed fire module, a concatenation unit and two fire modules which in order up samples their input, extract features, and concatenate features to construct the output.

by reducing the number of channels [24]. Furthermore, we show that, although the Squeeze U-Net has more layers than U-Net, the inference time for Squeeze U-Net implemented in Tensorflow 1.14 using Python 3.6 and CUDA 10.1.243 on an NVIDIA K40 GPU is 17 % faster than U-Net for the CamVid data set and only requires 66% of the U-Net training time

3. Architecture

3.1 Contracting Path

Inspired by SqueezeNet [2], we adopt fire modules for the down sampling (DS) units in the contracting path of Squeeze U-Net. Each fire module in the contracting path consists of one 1×1 convolution layer with C'_o output channels, $C'_o < C_i$ and an inception layer with two parallel convolutions with 3×3 and 1×1 kernel size and $C_o/2$ output channels each. Concatenation of the parallel convolution output channels form the fire module output. It is passed to the next contracting layer and also to the corresponding layer in the expansive path of Squeeze U-Net with long skip connections[25], [26]. For down sampling we use stride 2 convolutions instead of max or average pooling. Striding increase the expressiveness of our network [27].

3.2 Expansive path

For the expansive path in the Squeeze U-Net we also use the SqueezeNet fire modules to reduce the total number of parameters.

Table 2: Quantitate comparison of Squeeze U-Net and U-Net regarding model size, number of convolution and multiplication operations. These numbers are obtained using the TensorFlow 1.14 profiler on a saved model. The number of convolutions and multiplication are gathered during inference time.

Model	Size (MB)	#CONV (CamVid) Billion	#Mult (CamVid) Million
Squeeze U-Net	32	432.71	33.03
U-Net	386.6	1315.60	61.44
Factor of reduction	12.08×	3.04×	1.86×

Table 3: Comparison of the number of Squeeze U-Net and U-Net parameters and MACs in the contracting path. Squeeze U-Net achieves a 12.18× reduction in the number of parameters and a 3.7× reduction in MACs for the contracting path.

Layer Name	Squeeze U-Net			U-Net			Feature Size (× HW)	Reduction Factor (#Params)	Reduction Factor (#MACs)
	Layer	#Params	#MACs (× HW)	Layer	#Params	#MACs (× HW)			
Convolution	$[3\times3\times64]\times2$	38592	77184	$[3\times3\times64]\times2$	38592	77184	64	1×	1×
DS1	$\begin{bmatrix}1\times1\times32\\3\times3\times64\\1\times1\times64\end{bmatrix}\times2$	47104	23552	$[3\times3\times128]\times2$	221184	110592	$\frac{128}{2\times2}$	4.69×	4.69×
DS 2	$\begin{bmatrix}1\times1\times48\\3\times3\times128\\1\times1\times128\end{bmatrix}\times2$	141312	17664	$[3\times3\times256]\times2$	884736	110592	$\frac{256}{4\times4}$	6.2×	6.2×
DS 3	$\begin{bmatrix}1\times1\times64\\3\times3\times256\\1\times1\times256\end{bmatrix}\times2$	376832	11776	$[3\times3\times512]\times2$	3538944	110592	$\frac{512}{8\times8}$	9.39×	9.39×
DS 4	$\begin{bmatrix}1\times1\times80\\3\times3\times512\\1\times1\times512\end{bmatrix}\times2$	942080	7360	$[3\times3\times1024]\times2$	14155776	110592	$\frac{1024}{16\times16}$	15.2×	15.2×
Total		**1545920**	**137456**		**18,839,232**	**519552**		**12.18×**	**3.7×**

The main component of the expansive path in Squeeze U-Net is up sampling units (US). In every up-sampling unit, the transposed fire module consists of a 1×1 transposed convolution with C_o' output channels, $C_o' < C_i$ as in the DS fire modules. For the inception stage of the transposed fire module, the output from the 1×1 transposed convolution is fed into two parallel transposed convolutions with 2×2 and 1×1 kernel size, each with $C_o/2$ output channels that are concatenated to form the transpose fire module output, as for the DS units. Further, up sampling units also have a stage for concatenating the bypass connections from the corresponding down sampling unit with the output of the transposed fire module thereby merging higher-resolution features from fire modules in the contracting path with lower resolution features in the expansive path. The concatenating unit is followed by two successive fire modules. As shown in Figure 2 and Table 4, there are three up sampling units in the expansive path followed by one 2×2 transposed convolution. The features from its corresponding layer, the 3×3 convolution layer before the contracting path, is concatenated with features from the 2×2 transposed convolution and passed to two 3×3 convolution layers and one 1×1 convolution layer to generate a H×W×classes tensor for pixel wise segmentation.

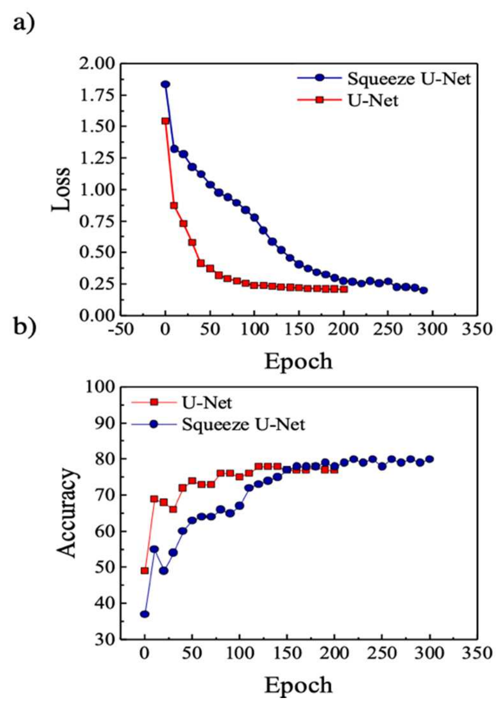

Figure 3: Training loss and accuracy per epoch for Squeeze U-Ne and U Net. (a) shows loss per epoch. (b) shows accuracy per epoch We train Squeeze U-Net until it converges to the same accuracy as U-Net.

4.Training

To evaluate Squeeze U-Net, we use the CamVid road scenes dataset [23]. This dataset is small, consisting of 701

Table 4: Comparison of Squeeze U-Net and U-Net parameter and MAC requirements for the expansive path. Squeeze U-Net achieves a 11.4× parameter reduction and a 3.05× MACs reduction in the expansive path

Layer Name	Squeeze U-Net			U-Net			Feature Size (× HW)	Reduction Factor (#Params)	Reduction Factor (#MACs)
	Layer	#Params	#MACs (× HW)	Layer	#Params	#MACs (× HW)			
US1	$\begin{bmatrix} 1\times 1\times 80 \\ 2\times 2\times 256 \\ 1\times 1\times 256 \end{bmatrix}$	184320	5760	$[2\times 2\times 512]$	2097152	65536	$\frac{512}{8\times 8}$	11.37×	11.37 ×
	$\begin{bmatrix} 1\times 1\times 64 \\ 3\times 3\times 256 \\ 1\times 1\times 256 \end{bmatrix}\times 2$	425984	13312	$[3\times 3\times 512]\times 2$	7077888	221184	$\frac{512}{8\times 8}$	16.6×	16.6×
US2	$\begin{bmatrix} 1\times 1\times 64 \\ 2\times 2\times 128 \\ 1\times 1\times 128 \end{bmatrix}$	73728	9216	$[2\times 2\times 256]$	524288	65536	$\frac{256}{4\times 4}$	7.1×	7.1×
	$\begin{bmatrix} 1\times 1\times 48 \\ 3\times 3\times 128 \\ 1\times 1\times 128 \end{bmatrix}\times 2$	159744	19968	$[3\times 3\times 256]\times 2$	1769472	221184	$\frac{256}{4\times 4}$	11×	11×
US3	$\begin{bmatrix} 1\times 1\times 48 \\ 2\times 2\times 64 \\ 1\times 1\times 64 \end{bmatrix}$	27684	13824	$[2\times 2\times 128]$	131072	65536	$\frac{128}{2\times 2}$	4.73×	4.73×
	$\begin{bmatrix} 1\times 1\times 32 \\ 3\times 3\times 64 \\ 1\times 1\times 64 \end{bmatrix}\times 2$	53248	26624	$[3\times 3\times 128]\times 2$	442368	221184	$\frac{128}{2\times 2}$	8.3×	8.3×
Transposed Convolution	$[2\times 2\times 64]$	32768	65536	$[2\times 2\times 64]$	32768	65536	64	1×	1×
Convolution	$[3\times 3\times 64]\times 2$	110592	221184	$[3\times 3\times 64]\times 2$	110592	221184	64	1×	1×
Total		**1,068,068**	**375,424**		**12,185,600**	**1,146,880**		**11.4×**	**3.05×**

RGB images at 360×480 resolution. The challenge is to segment images that may have 11 classes such as building tree, car, road, pedestrian etc. We use 468 images as training set and 233 images as test set, without data augmentation. We use the AdamOptimizer [28] with a fixed learning rate of 0.0001. On every epoch, we pick one batch of 8 images with batches picked in order, thus ensuring that each image is used only once. We use cross-entropy loss as the objective function for training the network. Figure 3a shows the improvement in the loss function with the number of epochs. The reductions in Squeeze U-net compared to U-Net clearly impacts the speed at which the loss decreases. We train until the loss converges to the same accuracy as U-Net. Though the loss function decreases more slowly for Squeeze U-net a segmentation accuracy of 78% are achieved by both architectures at 155 epochs for the CamVid data set, as seen in Figure 3b. Because the reduced number of operations in Squeeze U-Net the accuracy of 78% is reached in 69% of the time required by U-Net. We give an assessment of the Squeeze U-Net and U-Net architectures

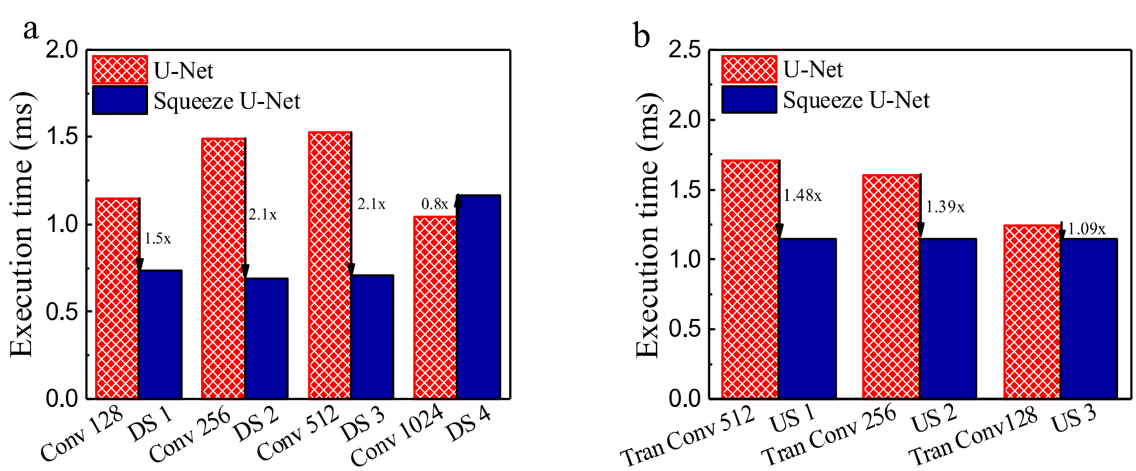

Figure 4: Execution time per layer in Squeeze U-Net and U-Net. (a) execution times for the layers in the contracting path (b) execution times for the layers in the expansive path.

5.Evaluation

5.1 Memory Consumption and Floating-Point Operations

Based on Tables 3 and 4, the total number of parameters in Squeeze U-Net is about 2.6 Million which is a 12× reduction compared to the 30.5 Million parameters in U-Net. The total number of MACs in Squeeze U-Net in one forward pass is 88 Billion which is a 3.2× reduction compared to the 287 Billion MACs in U-Net. To evaluate our models for inference, we use the Tensorflow 1.14 profiler [29] on our saved models. As shown in Table 2, our model size is 32MB which is consistent with our total parameter calculations in Tables 3 and 4 and, as expected, is a 12× reduction compared to U-Nets 384MB. The number of Squeeze U-Net convolutions (CONV) for inference is reduced by 3× which is expected from our total MAC calculations in Tables 3 and 4.

5.2 Execution Time Per Layer

We measure execution time of Squeeze U-Net and U-Net per layer on an NVIDIA K40 GPU by using Tensorflow 1.14 and report them in Table 5 and 6. As an example, for showing the effect of parameter reduction using fire modules, a U-Net with 64 input channels, 128 output channels using 3×3 convolution kernels require 288kB (formula given in Table 1 legend). A corresponding 64 input channels, 128 output channels fire module as specified in Table 3 would have a pointwise convolution kernel with 64 input channels and 32 output channels requiring 8kB and an inception stage with a pointwise convolution also requiring 8kB and a 3×3 convolution kernel with 32 input channels and 64 output channels requiring 72 kB in single precision. Thus, in single precision fire module as defined in Table 3 requires 88KB for filter parameters compared to 288kB for a full convolution, or about 30% of the U-Net parameter memory the data transfers from DRAM is correspondingly reduced. The execution times for the different layers in our Tensorflow implementation are shown in Table 5 and 6. For the contracting path Squeeze U-Net requires 63% of the time of U-Net and for the expanding path 75% of U-Net and for the two combined 69%. From the profiling of the Tensorflow implementation shown in Figure 5 we estimate that the execution time for Squeeze U-Net for the contracting and expanding path can be reduced by about 20% of the two convolutions in the inception stage of the fire module and transpose fire module are executed in

Table 5: Execution time per layer in the Expansive path

Layer name		Exe time (microseconds)		Conv Exe time (microseconds)		Fraction of convolution time to total time	
Squeeze U-Net	U-Net	Squeeze U-Net	U-Net	Squeeze U-Net	U-Net	Squeeze U-Net	U-Net
US1	Transposed convolution 512	1149	1709	651	1524	56%	89%
US2	Transposed convolution 256	1149	1604	651	1417	56%	88%
US3	Transposed convolution 128	1142	1245	653	1056	57%	84%
Total		3440	4558	1955	3997	56%	87%

Table 6: Execution time per layer in the contracting path

Layer name		Exe time (microseconds)		Conv Exe time (microseconds)		Fraction of convolution time to total time	
Squeeze U-Net	U-Net	Squeeze U-Net	U-Net	Squeeze U-Net	U-Net	Squeeze U-Net	U-Net
DS1	Conv128 (x2)	736.084	1147	394	1005	53%	87%
DS2	Conv256 (x2)	687.828	1492	377	1359	54%	91%
DS3	Conv512 (x2)	706.012	1526	398	1393	56%	91%
DS4	Conv1024 (x2)	1170	1040	860	947	73%	91%
Total		3301.68	5205	2029	4704	61%	90%

Table 7: Comparison of Squeeze U-Net (1) and U-Net (2) regarding intersection over union (IoU) and false positive to true positive for test set

	Building		Tree		Sky		Car		Road		Average	
	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
Percent True positive pixels												
Average	78.5%	83.3%	51.1%	72.6%	93.7%	96.9%	71.1%	84.5%	95.6%	97.4%	78.0%	86.9%
Max	86.3%	90.1%	73.8%	92.4%	99.4%	99.4%	99.3%	97.8%	98.2%	99.2%	91.4%	95.8%
Min	58.8%	73.3%	18.0%	51.7%	76.4%	91.1%	46.0%	49.0%	90.0%	93.0%	57.8%	71.6%
Intersection over Union (IoU)												
Average	0.689	0.639	0.370	0.403	0.842	0.879	0.427	0.628	0.751	0.800	0.616	0.670
Max	0.808	0.734	0.575	0.650	0.928	0.945	0.667	0.812	0.913	0.937	0.778	0.816
Min	0.395	0.443	0.113	0.220	0.676	0.753	0.011	0.020	0.575	0.686	0.354	0.424
False Positive pixels relative to true positive pixels												
Average	22.1%	39.3%	106.3%	130.1%	12.5%	11.0%	688.2%	354.1%	31.3%	23.4%	172.1%	111.6%
Max	83.1%	89.4%	331.0%	280.4%	24.8%	23.1%	8870.8%	4752.1%	62.8%	39.4%	1874.5%	1036.9%
Min	6.0%	23.4%	18.6%	20.1%	6.3%	4.5%	26.2%	10.7%	7.7%	5.6%	13.0%	12.8%

parallel. That would reduce the execution time for the contracting and expanding path of Squeeze U-Net to 55% of U-Net.

5.3 Inference Accuracy

We assessed the quality of the Squeeze U-Net model relative to the U-Net model on a set of 120 CamVid images not part of the training set. For the assessment the 120 images were divided into 15 batches of eight images each and for each batch the average of true positive, false positive, false negative pixel classification was recorded as well as the average pixel count for each class in each batch. In the evaluation set of 120 images segmentation and classification was successful for Squeeze U-Net and U-Net for the five classes labeled building, tree sky, car and road. Squeeze U-Net in addition successfully segmented and classified the class sidewalk. The results for the five common classes are shown in Table 7.

For true positive pixels the average accuracy for the building, tree, sky, car, and road classes, for U-Net was 86.9% vs Squeeze U-Net's 78%. For the road and sky

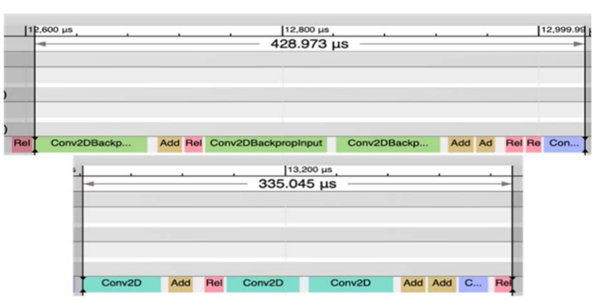

Figure 5. The execution times of fire modules in our Tensorflow implementation. By parallelizing the execution of the 2nd and third convolution we estimate that Squeeze U-Net should be able to achieve a reduction of about 20%.

classes Squeeze U-net is 2 – 3 % less accurate than U-Net. Sky represent on average about 14% of the pixels and the road about 25% of the pixels in the test set. For the building class Squeeze U-Net is on average about 5% less accurate for the test set than U-Net (78.5% vs 83.3%). The building class represent about 41% of the pixels in the test set. For the tree class with on average 6% of the pixels in the test set Squeeze U-net is about 20% less accurate than U-Net (51.1 % vs 72.6%) and for the car class having about 1% of the pixels Squeeze U-Net is about 13% less accurate than U-Net (71.1% vs 84.5%). Squeeze U-net appears more sensitive to the number of pixels representing the class than U-Net, and to also have difficulties with less well-defined structures like trees. The range of true positive pixel classification accuracy across images in the evaluation set is generally higher for Squeeze U-Net than U-Net for each class. The Min and Max values in Table 7 are based on averages for batches of eight images due to our implementation of the test cases, and hence do not cover extreme cases.

For false positives Squeeze U-Net tend to have more false positive pixels relative to the true pixels. For the sky class Squeeze U-net only has on average 1.5 % more false positives, but for the road class it has about 8% (31.3% vs 23.4%) more false positives. For the building class Squeeze U-Net however has about 17% less false positives than U-Net (22.1% vs 39.3%) and for the tree class Squeeze U-Net also has less false positives though still high (106% vs 130%). For the car class both Squeeze U-Net and U-Net had a very large number of false positives with Squeeze U-Net performing worse than U-Net. As for true positives the variability in the number of false positives across images

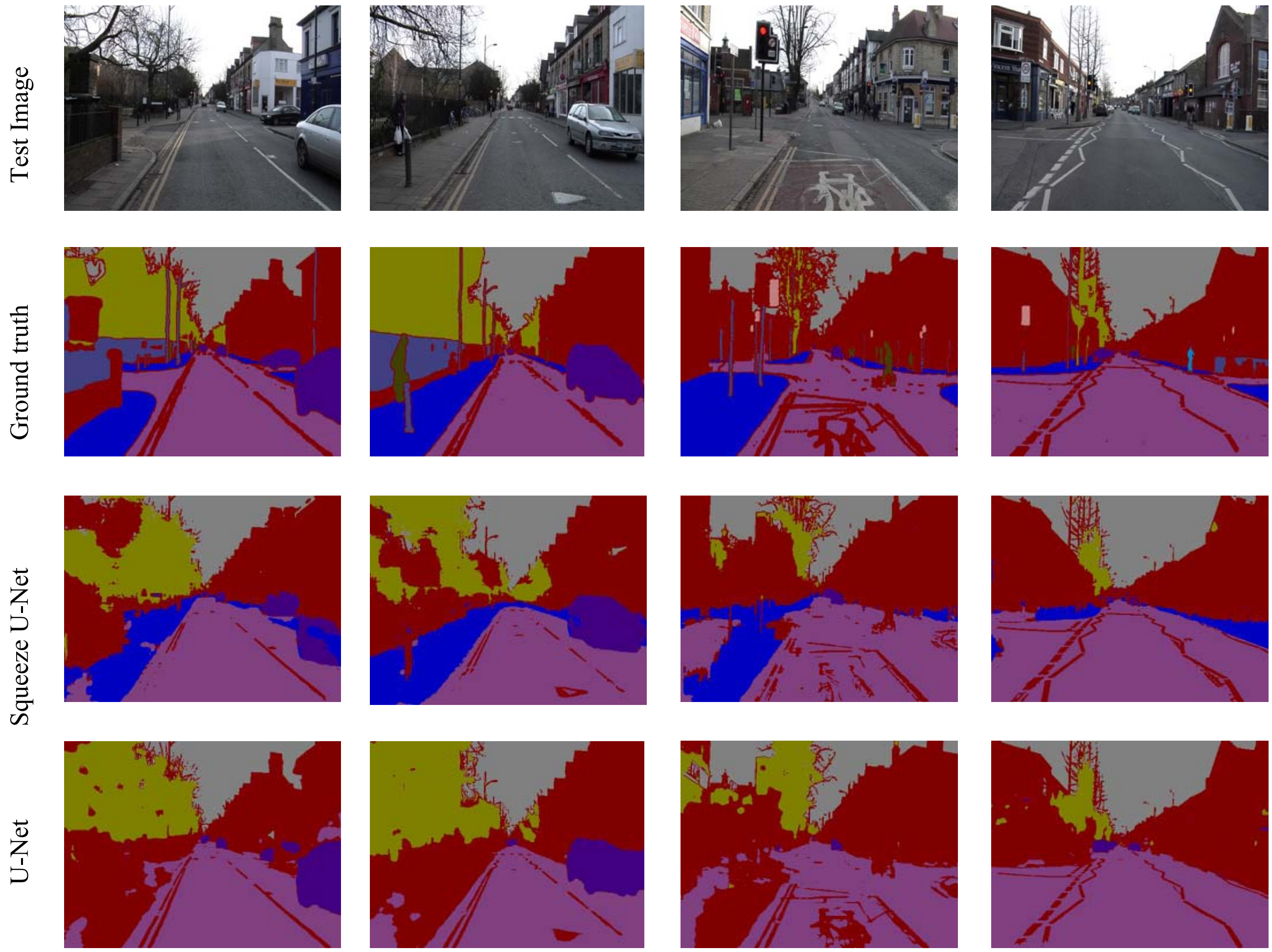

Figure 6: Qualitative assessment of Squeeze U-Net and U-Net segmentation on the CamVid road scenes dataset

tend to be higher for Squeeze U-Net than U-Net. For the Intersection over Union measure (IoU) (the ratio of true positive pixels to true pixels plus false positives) on average the Squeeze U-Net measure is less than about 3 - 5%, lower than the U-Net measure for the tree, sky and road classes. For the building class the Squeeze U-Net measure is about 5% higher than that of U-Net. For the *car* class the Squeeze U-Net measure is considerably lower than that for U-Net. The variability of the IoU measure across the test images is higher than that for U-Net. Figure 6 shows the classification generated by Squeeze U-Net and U-Net for four CamVid images.

6. Conclusion

In this work we presented Squeeze U-Net for image segmentation that has 12× fewer parameters, and 3× fewer MACs. Squeeze U-Net training time for the CamVid data set is 69% of that of U-Net and for inference the Squeeze U-Net architecture is 17% faster. We also estimate, that by using parallel convolutions, training and inference times of Squeeze U-Net may achieve 2× reduction in execution time compared to U-Net. Squeeze U-Net use fire modules [2] and transposed fire modules in the contracting and expansive paths of U-Net to reduce model size and generate a memory and power efficient segmentation model. The 12× reduction in memory requirements should result in a significant reduction in energy requirement compared to U-Net and the 3× reduction in MACs similarly should reduce energy dissipation for computation. Energy measurements is part of our future work as is a concurrent implementation of the inception stage in the fire modules. A further understanding of the Squeeze U-Net accuracy and sensitivity to class characteristics and training needs are also part of future work.

Acknowledgement

The University of Houston Data Science Institute's Research Computing Center made available NVIDIA K40 GPU nodes free of charge which is gratefully acknowledged. The many helpful discussions with members of the Advanced Computing Research Lab were also very helpful in achieving our results. Especially Suyash Bakshi's insights on GPU programming and tools were very helpful.

References

[1] A. Pedram, S. Richardson, M. Horowitz, S. Galal, and S. Kvatinsky, “Dark memory and accelerator-rich system optimization in the dark silicon era,” *IEEE Des. Test*, vol. 34, no. 2, pp. 39–50, 2016.

[2] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size,” *arXiv Prepr. arXiv1602.07360*, 2016.

[3] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation BT - Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015,” 2015, pp. 234–241.

[4] C. Szegedy *et al.*, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[5] A. G. Howard *et al.*, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv Prepr. arXiv1704.04861*, 2017.

[6] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, 2017.

[7] V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, 2017.

[8] J. Long, E. Shelhamer, and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[9] H. Noh, S. Hong, and B. Han, “Learning deconvolution network for semantic segmentation,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1520–1528.

[10] S. Chetlur *et al.*, “cudnn: Efficient primitives for deep learning,” *arXiv Prepr. arXiv1410.0759*, 2014.

[11] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv Prepr. arXiv1510.00149*, 2015.

[12] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural networks,” *Adv. Neural Inf. Process. Syst.*, vol. 2015-Janua, pp. 1135–1143, 2015.

[13] S. Han *et al.*, “Dsd: Dense-sparse-dense training for deep neural networks,” *arXiv Prepr. arXiv1607.04381*, 2016.

[14] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, “A survey of model compression and acceleration for deep neural networks,” *arXiv Prepr. arXiv1710.09282*, 2017.

[15] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, “Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation,” in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 1269–1277.

[16] L. Sifre and S. Mallat, “Rigid-motion scattering for image classification,” *Ph. D. thesis*, 2014.

[17] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[18] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, “Encoder-decoder with atrous separable convolution for semantic image segmentation,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 801–818.

[19] T. Sheng, C. Feng, S. Zhuo, X. Zhang, L. Shen, and M. Aleksic, “A quantization-friendly separable convolution for mobilenets,” in *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, 2018, pp. 14–18.

[20] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, “Neuflow: A runtime reconfigurable dataflow processor for vision,” in *Cvpr 2011 Workshops*, 2011, pp. 109–116.

[21] J.-Y. Kim, M. Kim, S. Lee, J. Oh, K. Kim, and H.-J. Yoo, “A 201.4 GOPS 496 mW real-time multi-object recognition processor with bio-inspired neural perception engine,” *IEEE J. Solid-State Circuits*, vol. 45, no. 1, pp. 32–45, 2009.

[22] T. Chen *et al.*, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 269–284, 2014.

[23] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognit. Lett.*, vol. 30, no. 2, pp. 88–97, 2009.

[24] L. Saadatifard, A. Mobiny, P. Govyadinov, H. Nguyen, and D. Mayerich, "DVNet: A Memory-Efficient Three-Dimensional CNN for Large-Scale Neurovascular Reconstruction," *arXiv Prepr. arXiv2002.01568*, 2020.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[26] S. Sun, J. Pang, J. Shi, S. Yi, and W. Ouyang, "Fishnet: A versatile backbone for image, region, and pixel level prediction," in *Advances in neural information processing systems*, 2018, pp. 754–764.

[27] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv Prepr. arXiv1412.6806*, 2014.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv Prepr. arXiv1412.6980*, 2014.

[29] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th ${$USENIX$}$ Symposium on Operating Systems Design and Implementation (${$OSDI$}$ 16)*, 2016, pp. 265–283.