ApesNet: A Pixel-wise Efficient Segmentation Network

Chunpeng Wu, Hsin-Pai Cheng, Sicheng Li, Hai Li, Yiran Chen
Department of Electrical and Computer Engineering
University of Pittsburgh, Pittsburgh, PA 15232
{chw127, hsc38, sil27, hal66, yiran.chen}@pitt.edu

(Invited Special Session Paper)

ABSTRACT

Autonomous driving can effectively reduce traffic congestion and road accidents. Therefore, it is necessary to implement an efficient high-level, scene understanding model in an embedded device with limited power and sources. Toward this goal, we propose ApesNet, an efficient pixel-wise segmentation network, which understands road scenes in real-time, and has achieved promising accuracy. The key findings in our experiments are significantly lower the classification time and achieve a high accuracy compared to other conventional segmentation methods. The model is characterized by an efficient training and a sufficient fast testing. Experimentally, we use the well-known CamVid road scene dataset to show the advantages provided by our contributions. We compare our proposed architecture's accuracy and time performance with SegNet. In CamVid dataset training and testing, our network, ApesNet outperform SegNet in eight classes accuracy. Additionally, our model size is 37% smaller than SegNet. With this advantage, the combining encoding and decoding time for each image is 1.45 to 2.47 times faster than SegNet.

1. INTRODUCTION

In the last four years, deep learning and Convolutional neural networks (CNN) [1] has gained significant success in visual and audio recognition problems such as handwritten digit recognition, large-scale image and audio classification [2]. Motivated by this huge success, deep CNN were adopted to solved semantic segmentation, action recognition [3] and so on. Recent studies show that the network depth and width have crucial influence on classification accuracy. For example, many state-of art results on ImageNet Challenge contribute to very deep neural network models [4, 5, 6]. Whole-image classification has been done successfully. The next step is heading to pixel-wise semantic segmentation which means making prediction for every pixel; we are moving from image-label to pixel-wise label. However, the network will be inefficient if we directly use conventional deep learning architecture. The main obstacle is that there are too much information for training. The recent studies show that there is still room for improvement on accuracy and segmentation time. It is still a challenging problem for learning more complex information from pixel-wise labeling.

ESTIMedia'16, October 01-07, 2016, Pittsburgh, PA, USA

DOI: http://dx.doi.org/10.1145/2993452.2994306

Until now, semantic segmentation is an ongoing hot topic. There are many image segmentation methods that has been done. However, due to the model size, it is not feasible to be implemented on an embedded device for real-time segmentation. For example, Alex Kenall et al. proposed SegNet, a deep convolutional encoder-decoder architecture [7]. Although SegNet has competitive performance on class accuracy, it does not achieve real-time performance. Our motivation for segmentation comes from the remaining issue, processing time. In this paper, we proposed a novel approach of pixel-wise object class segmentation for use in area of road scene understanding and autonomous driving. Our ultimate goal is to implement a segmentation neural network in an inexpensive embedded system with minimum hardware and software supply.

To achieve this, our model should be small, computational efficient and real-time. We experimentally analyze the relationship between processing time and each layer. We find that, it is not necessary to symmetrically align encoder and decoder. Inspired by ResNet [8], we insert shortcut connections to encoder part. We present comprehensive experiments on CamVid dataset [9] to show the processing time problem and evaluate our method. Additionally, for the consideration of safety, the network has a better distinguish accuracy on smaller objects,such as sign, pedestrian, bicyclist. For example, SegNet-basic can only achieve 16.4% and 36.2% accuracy on the segmentation of bicyclist and sign. ApesNet can achieve 46.1% and 52.3% respectively.

The remainder of the paper is structured as follows. In Section 2, the related work is given. In section 3, we describe the proposed method. In section 4, we explain the experimental setup, results, discussion and evaluation. In Section 5, we discuss the conclusion and future work.

2. RELATED WORK

The semantic segmentation has been widely studied with a variety of methods. There were many breakthrough results that were done. Most of them combined other algorithms with convolution neural network (CNNs) which outperform many conventional algorithms. For example, Conditional Random Fields (CRF) [10] were adopted for image segmentation. CRF effectively assigned appropriate class labels and constructed semantic-level regions. The hierarchical features were extracted to explain the meaning in the image.

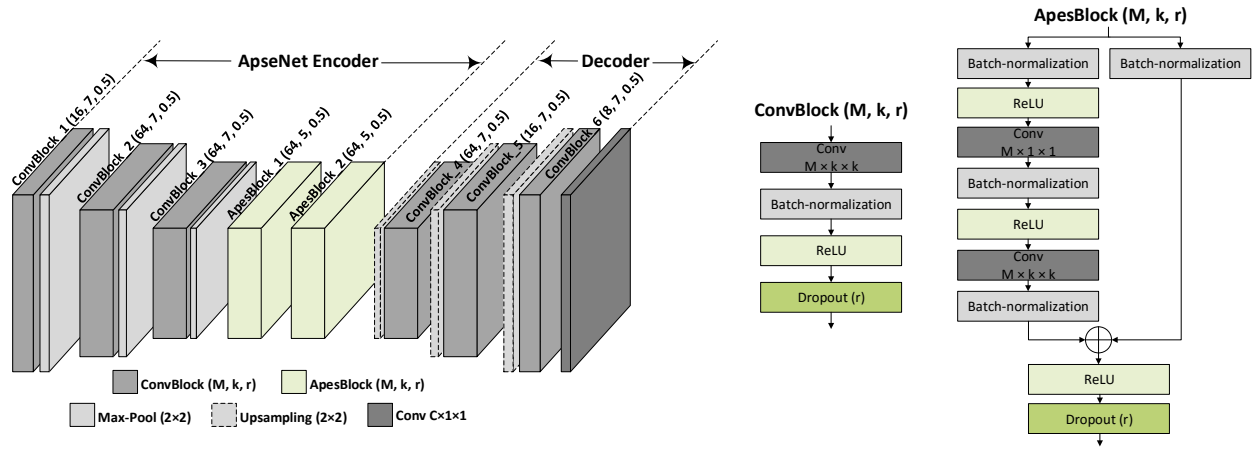

Figure 1: The proposed encoder-decoder architecture of ApesNet and the inner structure of ConvBlock and ApesBlock.

It is worth mentioning that Farabet et al. used multiscale convolutional network for pixel-wise labelling, by combining CNNs with Conditional Random Fields (CRF) model on many datasets [11]. However, an unavoidable disadvantage was that the information of mid-level features was lost by using spatial pooling. Recent approach uses encoder-decoder network architecture such as SegNet. Since that SegNet is based on VGG16 architecture, it is not applicable for embedded devices. Even though the result is fascinating, the class accuracy and segmentation time is not satisfactory.

3. DESIGN ANALYSIS

As our goal is to accelerate the network testing without losing accuracy, our basic idea is to reduce the operations which are relatively time consuming but with less contribution to the segmentation accuracy, and then improve the accuracy by adding efficient operations. We ran a time profiling for a popular network, AlexNet [1] which includes convolution, max-pooling, ReLU, local contrast normalization (LRN), dropout and full-connection. We added batch-normalization [12] between convolution and dropout. The recent proposed module of shortcut [8] is also studied, and we find its time consumption is mainly dependent on the convolution branch. Therefore, we will discuss the convolution instead of the shortcut module in this section. Our GPU device is NVIDIA TITAN X and cuDNN v5 [13] is adopted. The time spent on ReLU, LRN, batch-normalization and dropout is trivial which is shorter than 2ms for each operation in our experiment. The running time of max-pooling and full-connection is increased with the size and number of feature maps, but still less than 4ms for each operation. As more recent CNNs [14, 15] goes deeper, the ratio of layer number of these two types to total number becomes lower, we do not focus on these layers. Actually, the feature maps can be rescaled by setting the stride size of a convolutional layer bigger than 1, instead of using the pooling layers in a CNN, as suggested by [16]. We tried to remove the pooling layers in this manner, however, the final segmentation accuracy is significantly degraded in our experiment. A possible explanation is that average-pooling can be replaced by the convolution with stride bigger than 1 as in [16], but the biologically inspired max-pooling cannot. For AlexNet, the convolution occupies most of the running time. Specifically, the first convolution takes about 10ms, while the last takes only 4ms, *i.e.*, convolution layers with larger feature maps are the main time consumers.

Another design issue of networks used for pixel-wise applications (*e.g.*, image segmentation) is how to make the resolution of the output prediction map the same as that of the input image. Inspired by the auto-encoder architecture [17], the popular CNNs for segmentation, e.g., FCN [18], SegNet-Basic [7, 19] and EDeconvNet [20], adopt symmetric encode and decoder to gradually restore the shrunk feature maps to the original size of the input, and their differences lie in the implementation of up-sampling. However, more recent methods [21, 22] keeps the encoder (*e.g.*, VGG) unchanged while use a shallow decoder or a full-connected layer to get the output prediction vector corresponding to each pixel in the input image. We tried to replace the decoder of SegNet-Basic and EDeconvNet with that of [21, 22]. The testing time decreased by significantly around 11ms at cost of only averagely 1% accuracy drop in segmentation accuracy. Note that the total testing time of SegNet-Basic is 63ms using TITAN X with cuDNN v5. We further tried to gradually shrink the encoder by removing convolutional layers while keep the decoder unchanged. The obtained results showed more than 10% decreased accuracy, indicating a deep encoder is probably necessary for extracting expressive visual features to distinguish semantic classes of objects.

4. ARCHITECTURE

Based on our above analysis of time profiling and encoder-decoder architecture, we adopt the following two strategies. First, the number of large feature maps in convolutional layers are decreased for acceleration compared to previous methods [18, 19, 20, 21, 22]. Here, "large feature maps" indicates the maps in the first *ConvBlock* and the last *ConvBlock* in our network as shown in Figure 1. Respectively 16 and 8 feature maps are used for these two *ConvBlock*. For simplicity, we do not tune the feature map numbers of

Table 1: Segmentation results on CamVid testing set.

Method	Building	Tree	Sky	Car	Sign	Road	Pedestrian	Fence	Pole	Sidewalk	Bicyclist	Class Avg.	Mean IoU
SegNet-Basic	75.1%	**83.1%**	88.3%	80.2%	36.2%	91.4%	56.2%	**46.1%**	**44.1%**	74.8%	16.4%	62.9%	46.2%
ApesNet	**76.0%**	80.2%	**95.7%**	**84.2%**	**52.3%**	**93.9%**	**59.9%**	43.8%	42.6%	**87.6%**	**46.1%**	**69.3%**	**48.0%**

Table 2: Comparison on testing speed with 360×480 input.

	SegNet-Basic	ApesNet
Model Size	5.40MB	**3.40MB**
GTX 760M	181ms	**73ms**
GTX 860M	170ms	**70ms**
TITAN X	63ms	**40ms**
Tesla K40	58ms	**39ms**
GTX 1080	48ms	**33ms**

other convolutional layers. Second, we will use an asymmetric network structure with a deep encoder and a relatively shallow decoder. The deep encoder lies in adding convolutional layers with relatively small feature maps, *i.e.*, two *ApesBlock* as shown in Figure 1, in order to improve the accuracy. The convolution kernel size of these two *ApesBlock* is 5×5 to extract features at a finer scale, compared to the kernel size 7 of all other modules in our network. Ablation studies in Section 5.2 will show the contributions of above two strategies in details. Figure 1 shows the architecture of ApesNet, and two basic modules in it (*ConvBlock* and *ApesBlock*). Our network consists of an encoder and a decoder. The *ConvBlock (M, k, r)* is adopted for both the encoder and decoder, which includes a convolutional layer with $k \times k$ kernel and M feature maps, batch-normalization, ReLU activation, and drop-out with ratio r. The *ApesBlock (M,k,r)* is used only for the encoder and inspired by the ResNet [8]. Two branches, *i.e.*, the shortcut connections and two convolutional layers with batch-normalization and ReLU, are combined by element-wise add, which is followed by ReLU and dropout with ratio r. The 1×1 convolution in the *ApesBlock* serves as an identity mapping. As shown in Figure 1, an input image will be passed through three pairs of *ConvBlock* and max-pooling, and then two repeated *ApesBlock* in the encoder. The kernel size of all the first three *ConvBlock* is 7×7, while that of the latter two *ApesBlock* is smaller, 5×5. The max operator is over a 2×2 region with stride 2. In addition, all the modules in the encoder has 64 feature maps except the first *ConvBlock* with fewer (*i.e.*, 16) maps. The max-pooing indices in the encoder are used to up-sample the feature maps of the corresponding un-pooling layer in the decoder, therefore the upsampling operator is also over a 2×2 region with stride 2. The decoder is comprised of three pairs of un-pooling and *ConvBlock*, and a final 1×1 convolutional layer as a classifier. The kernel size of all three *ConvBlock* in the decoder is 7×7, however, their feature map numbers vary: 64, 16 and 8. The output map size is the same as input, and each pixel in the input image corresponds to a vector of length C where C is the number of semantic classes. The cross-entropy is used as the objective function for training. As the pixel number of each semantic class is not balanced, the loss of each class is weighted according to median frequency balancing [23].

5. EXPERIMENTS

We will evaluate the accuracy and testing speed of our method using a popular image segmentation dataset. Ablation studies of our improvements on accuracy and speed as stated in Section 3 will be shown. With consideration of physical constraints of embedded hardware, a fixed-point version of our network and the corresponding performance will be provided.

5.1 ApesNet on CamVid

The CamVid is a road scene segmentation dataset, consisting of 367 training and 233 testing RGB images (day and dusk scenes) with 11 semantic classes of manually labeled objects, *e.g.*, car, tree, road, fence, pole, building *etc.* [9]. The original image resolution is 720×960, while we downsample all images to 360×480. Two popular measures of segmentation performance are used: Class Average Accuracy (Class Avg.) which is the mean of the predictive accuracy over all classes, and Mean Intersection over Union (Mean IoU) as used in the Pascal VOC12 Challenge [24].

Our method will be compared with SegNet-Basic [7] as its model scale is similar with ours. Both of SegNet-Basic and our model are initiated using He *et al.* [25], and trained with stochastic gradient descent with a fixed learning rate of 0.1 and momentum of 0.9. The quantitative comparison on accuracy and testing speed is respectively shown in Table 1 and Table 2. Our method achieves better accuracy (Class Avg. and Mean IoU) and faster speed with smaller model size. As shown in Table 1, our method obtains higher per-class accuracy on eight of all eleven classes of object. In addition, the traditional SegNet-Basic severely biases towards certain classes of small-scale objects, *e.g.*, bicyclist, while our method keeps a better balance of accuracy among classes of object. An explanation is that two *ApesBlock* with smaller kernel size (5×5) are used in our method, therefore small objects will benefit from our convolutional feature extractor at finer scale. Six examples of visualized segmentation results are shown in Figure 2. Images from the first to the fourth column are respectively input image, SegNet-Basic, our method and ground-truth, which further validates our advantage. On the testing speed as listed in Table 2, the performance gap between mobile-based GTX 760M and PC-based GTX 1080 for SegNet-Basic is 133ms, however, the gap is only 40ms for our method. This indicates that our acceleration strategy does not heavily rely on the advance of GPU hardware itself compared with SegNet-Basic. Therefore, our method can be further speed up using NVIDIA's techniques such as Dynamic

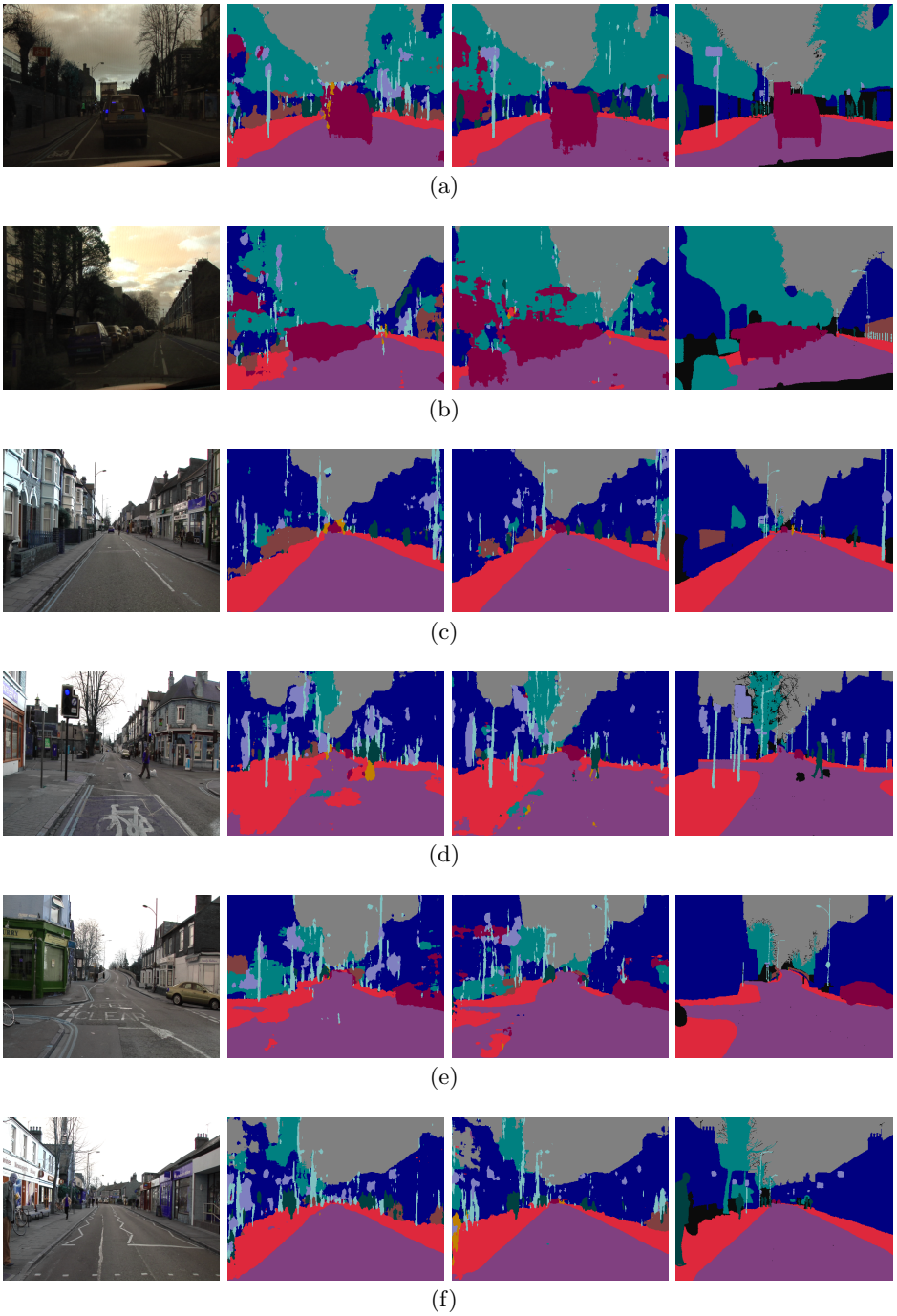

Figure 2: Examples of segmentation results. From the first to the fourth column are: Input image, SegNet-Basic, ApesNet and ground-truth.

Parallelism and Hyper-Q [26].

5.2 Ablation Studies

The contribution of two components in our method will be discussed: *ApesBlock* and decreased number of large feature maps. As shown in Figure 1, our asymmetric encoder-decoder architecture lies in that there are two *ApesBlock* at end of the encoder (before the first up-sampling layer). The large feature maps we will discuss here are those in the *ConvBlock* at start of the encoder (*ConvBlock_1*) and the *ConvBlock* at end of the decoder (*ConvBlock_6*) in Figure 1. As stated in Section 3, adopting *ApesBlock* will improve the segmentation accuracy without significantly increasing the testing time, while decreased large feature maps will improve the testing speed without significantly degrading the segmentation accuracy.

Table 3 lists the ablation study on *ApesBlock*. Two *ApesBlock* improves the Class Avg. and Mean IoU respectively by

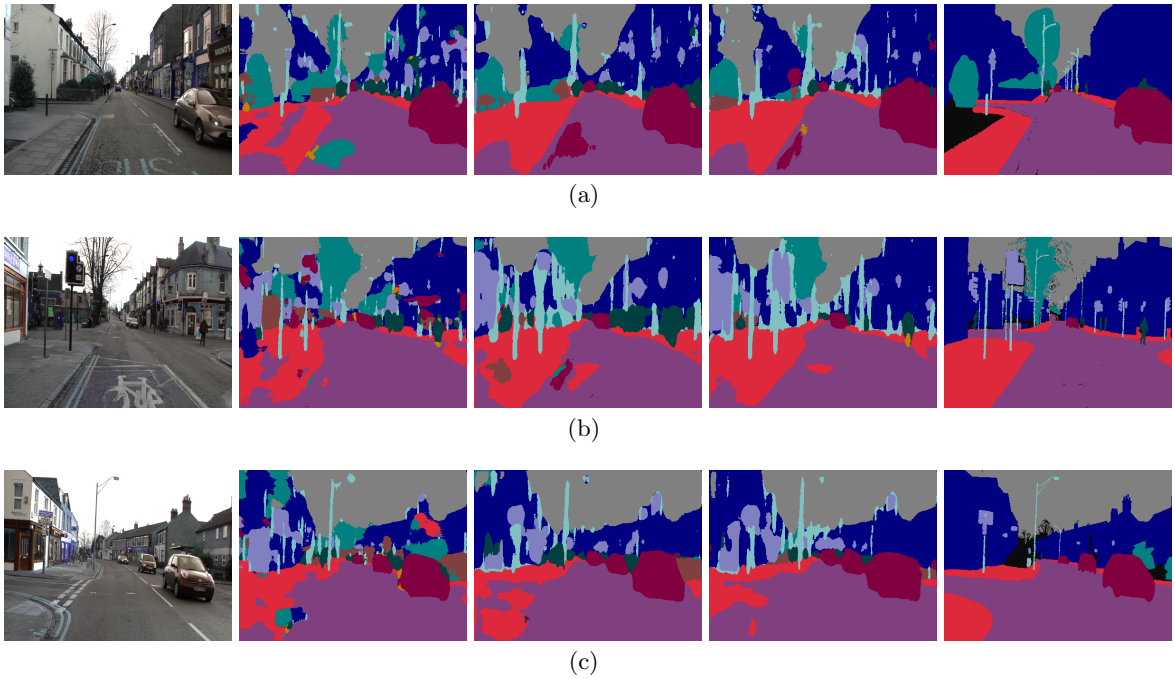

Figure 3: Comparison of segmentation results with no *ApesBlock* (2nd column), one *ApesBlock* (3rd column) and two *ApesBlock* (4th column). The first column is the original images, and the 5th column is the ground-truth.

7.2% and 4.2% compared to the first row where no *ApesBlock* is used while the running time only increases 4ms. Three examples of visualized segmentation results are shown in Figure 3. Images from the first to the fifth column are: Input image, No *ApesBlock*, One *ApesBlock*, Two *ApesBlock* and Ground-Truth. The segmentation results in the fourth column (Two *ApesBlock*) are more smooth than those in the second column (No *ApesBlock*).

Table 4 lists the ablation study on decreased number of large feature maps. The "*ConvBlock_1*: 64", for instance, indicates that there are 64 feature maps in *ConvBlock_1*. The comparison of the first and the fourth row show that running time significantly reduces by 12 ms while the Class Avg. and Mean IoU only decrease respectively by 0.8% and 0.7%.

5.3 Fixed Point

Typical embedded devices prefer algorithms with limited numerical precision. Therefore, we truncate our 32-bit floating-point network to 16-bit and 8-bit fixed-point networks. The quantization process is executed after a floating-point network is trained. Table 5 lists the segmentation accuracy based on 16-bit and 8-bit fixed-point network. Compare to our floating-point network as listed in Table 1, the 16-bit network only loses Class Avg. and Mean IoU respectively by 3.1% and 1.1%. Six examples of visualized segmentation results are shown in Figure 4. Images from the first to the fourth column are: Input image, Our floating-point network, Our 16-bit fixed-point network and Ground-Truth. The second column (floating-point) is slightly better than the third column (fixed-point).

6. CONCLUSIONS

We proposed a semantic segmentation network model, ApesNet, which can process images in real-time. We expect this network will have a wide range application on embedded GPU or any sources limited embedded devices. ApesNet has been test on 6 different GPUs, all the result proves that our network has significant advantages on class accuracy and

Table 3: Ablation Study on ApesBlock.

	Class Avg.	Mean IoU	Speed
ApesNet without *ApesBlock*	62.1%	43.8%	**29ms**
ApesNet with one *ApesBlock*	66.4%	46.3%	31ms
ApesNet with two *ApesBlock*	**69.3%**	**48.0%**	33ms

Table 4: Ablation Study on Decreased Number of Large Feature Maps.

	Class Avg.	Mean IoU	Speed
ApesNet (*ConvBlock_1*: 64, *ConvBlock_6*: 64)	**70.1%**	**48.7%**	45ms
ApesNet (*ConvBlock_1*: 16, *ConvBlock_6*: 64)	69.8%	48.5%	41ms
ApesNet (*ConvBlock_1*: 64, *ConvBlock_6*: 8)	69.5%	48.1%	37ms
ApesNet (*ConvBlock_1*: 16, *ConvBlock_6*: 8)	69.3%	48.0%	**33ms**

Table 5: Segmentation results of our fixed-point networks.

	Class Avg.	Mean IoU
16-bit ApesNet	66.2%	46.9%
8-bit ApesNet	15.2%	5.06%

time consuming, compared to other semantic segmentation method such as SegNet. For road scene understanding and autonomous driving, ApesNet provides a feasible way to efficiently distinguish different objects, especially smaller objects, *i.e.*, sign, bicyclist, pedestrian, which is essential to overall safety. By testing on an automotive data and measuring run time, our network is a possible solution for real-time embedded systems.

7. REFERENCES

[1] A. Krizhevsky, I. Sutshever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[2] H. Lee, P. Pham, Y. Largman, and A. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," 2009.

[3] S. J, M. Yang, K. Yu, and W. Xu, "3D convolutional

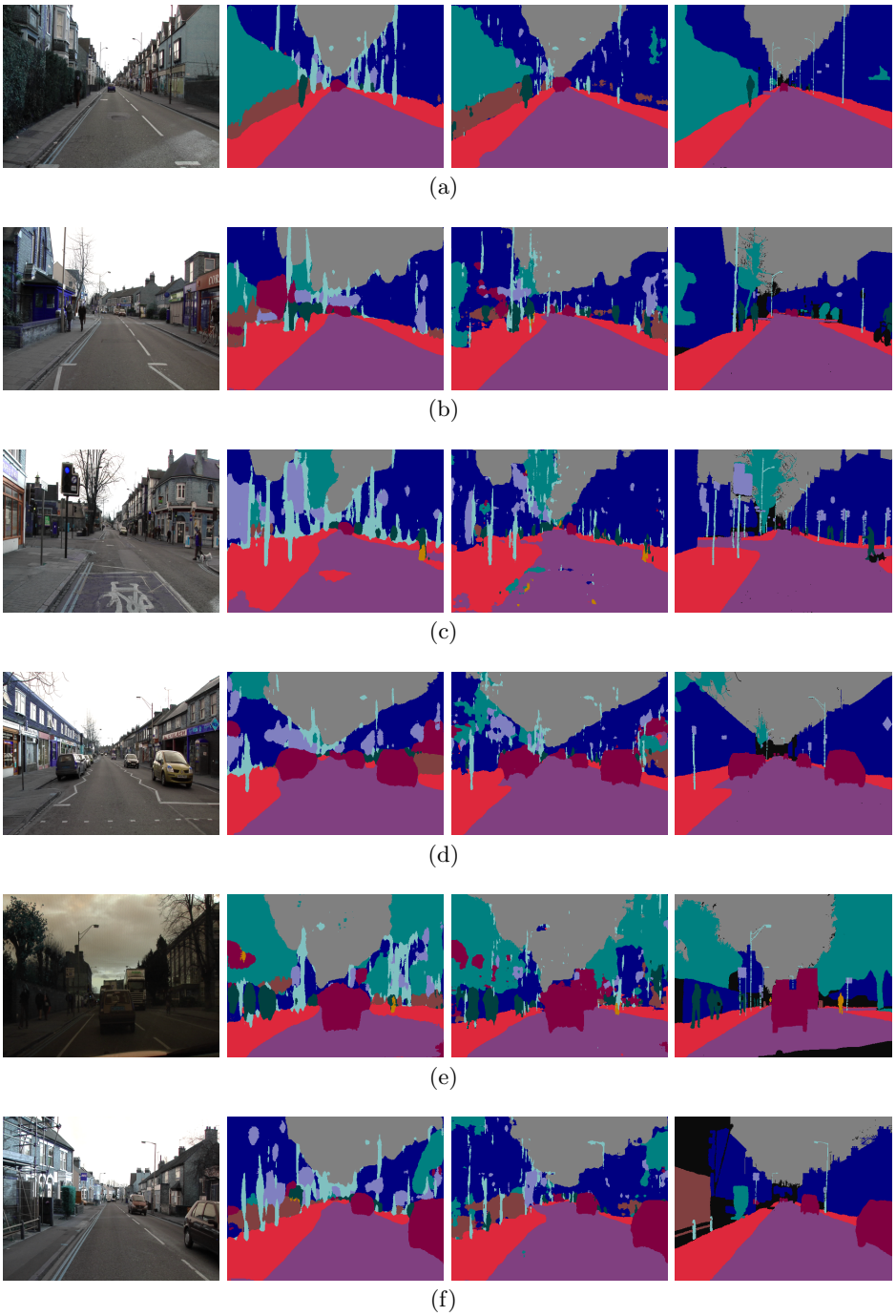

Figure 4: Comparison between our networks of floating-point and fixed-point. From the first to the fourth column are: Input image, our floating-point network, our fixed-point network (16-bit) and ground-truth.

neural networks for human action recognition," *IEEE Trans*, 2013.

[4] C. Szegedy, W. Liu, Y. Jia, and P. Sermanet, "Going deeper with convolutions," *ImageNet Chall*, 2015.

[5] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *ImageNet Chall*, 2013.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *CoRR*, 2015.

[7] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labeling," *arxiv*, 2015.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arxiv*, 2015.

[9] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, "Segmentation and Recognition using Structure form Motion Point Clouds," *European Conference on Computer Vision (ECCV)*, 2008.

[10] J. Verbeek and W. Triggs, "Scene segmentation with crfs learned from partially labeled images," *NIPS*, 2007.

[11] C. Couprie, L. Najman, and Y. Lecun, "Learning hierarchical features for scene labeling," *IEEE Trans*, August 2013.

[12] S. Ioffe and C.Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Convariate Shift," *Journal of Machine Learning Research (JMLR)*, 2015.

[13] https://developer.nvidia.com/cudnn

[14] K. Simonyan and A. Zisserman, "Veyr Deep Convolutional Networks for Large-Scale Image Recognition," *International Conference on Learning Representations (ICLR)*, 2015.

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[16] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis," *International Conference on Document Analysis and Recognition (ICDAR)*, 2003.

[17] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," *Journal of Machine Learning Research (JMLR)*, 2010.

[18] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[19] V. Badrinarayanan, A. Kendall, and E. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *arxiv*, 2015.

[20] H. Noh, S. Hong, and B. Han, "Learning Deconvolution Network for Semantic Segmentation," *International Conference on Computer Vision (ICCV)*, 2015.

[21] F. Yu and B. Koltun, "Multi-Scale Context Aggregation by Dilated Convolutions," *International Conference on Learning Representations (ICLR)*, 2016.

[22] G. Lin, C. Shen, A. van den Hengel, and I. Reid, "Efficient Piecewise Training of Deep Structured Models for Semantic Segmentation," *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[23] D. Eigen and R. Fergus, "Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture," *arxiv*, 2014.

[24] M. Everingham, S. A. Eslami, L. V. Gool, C. K. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes Challenge: A Retrospective," *International Journal of Computer Vision (IJCV)*, 2014.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *arxiv*, 2015.

[26] https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf