# Fast-SCNN: Fast Semantic Segmentation Network

Rudra P K Poudel[1]
rudra.poudel@crl.toshiba.co.uk
Stephan Liwicki[1]
stephan.liwicki@crl.toshiba.co.uk
Roberto Cipolla[1,2]
rc10001@cam.ac.uk

[1] Toshiba Research Europe Cambridge, UK
[2] University of Cambridge UK

**Abstract**

The encoder-decoder framework is state-of-the-art for offline semantic image segmentation. Since the rise in autonomous systems, real-time computation is increasingly desirable. In this paper, we introduce fast segmentation convolutional neural network (Fast-SCNN), an above real-time semantic segmentation model on high resolution image data ($1024 \times 2048 px$) suited to efficient computation on embedded devices with low memory and power. We introduce a 'learning to downsample' module which computes low-level features for multiple resolution branches simultaneously. Our network combines spatial detail at high resolution with deep features extracted at lower resolution, yielding an accuracy of 68.0% mean intersection over union at 123.5 frames per second on full scale images of Cityscapes. We also show that large scale pre-training is unnecessary. We thoroughly validate our experiments with ImageNet pre-training and the coarse labeled data of Cityscapes. Finally, we show even faster computation with competitive results on subsampled inputs, without any network modifications.

## 1 Introduction

Fast semantic segmentation is particular important in autonomous systems, where input is to be parsed quickly to facilitate responsive interactivity with the environment. It is therefore evident that recently the research into real-time semantic segmentation has gained in popularity [15, 17, 18, 20, 27, 30]. We emphasize, faster than real-time performance is in fact often necessary, since semantic labeling is usually employed only as a preprocessing step of other time-critical tasks. Furthermore, real-time semantic segmentation on embedded devices may enable many additional applications, such as augmented reality for wearables.

State-of-the-art semantic segmentation *deep convolutional neural networks* (DCNNs) combine two separate modules: the encoder and the decoder. The encoder module uses a combination of convolution and pooling operations to extract DCNN features. The decoder module recovers the spatial details from the sub-resolution features, and predicts the object labels (*i.e.* the semantic segmentation) [24]. Later, inspired by global image-level context prior to DCNNs [11, 12], the pyramid pooling module of PSPNet [29] and atrous spatial

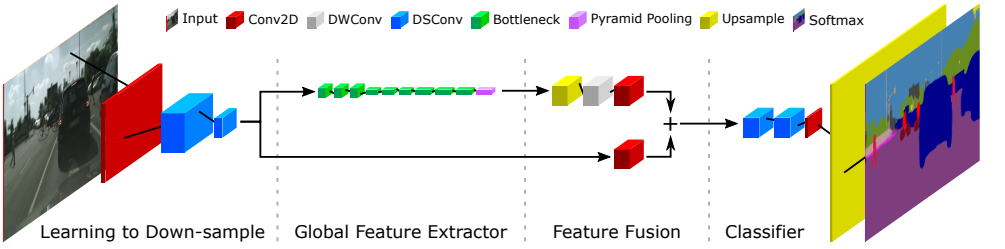


Figure 1: Fast-SCNN shares the computations between branches (encoders) to build faster real-time semantic segmentation network.

pyramid pooling (ASPP) of DeepLab [3] are employed to utilize global context. However, none of them run in real-time.

Recent interest in real-time application, speed and memory became important factors in image segmentation system design [18, 27]. Common techniques include network quantization, network compression, factorization of standard convolution and efficient redesign of DCNNs.

**Network Quantization:** Since floating point multiplications are costly compared to integer or binary operations, runtime can be further reduced using quantization techniques for DCNN filters and activation values [9, 19, 26].

**Network Compression:** Pruning is applied to reduce the size of a pre-trained network, resulting in faster runtime, a smaller parameter set, and smaller memory footprint [7, 12, 18]. Network quantization and compression are orthogonal to proposed model in this paper.

**Factorization of Convolution:** MobileNet [8] decomposes a standard convolution into a depthwise convolution and a $1 \times 1$ pointwise convolution, together known as depthwise separable convolution. Such a factorization reduces the floating point operations and parameters, hence the computational cost and memory requirement of the model is reduced.

**Efficient Redesign of DCNNs:** Building on fully convolutional network (FCN), SegNet [2] introduced a joint encoder-decoder model. Following SegNet, ENet [17] also design an encoder-decoder with few layers to reduce the computational cost. More recently, two-branch and multi-branch models were introduced to achieve real-time performance. ICNet [30], ContextNet [18], BiSeNet [27] and GUN [15] learned global context with reduced-resolution input in a deep branch, while boundaries are learned in a shallow branch at full resolution. Even though the initial layers of the multiple branches extract similar features [16, 28], they do not leverage this. In this paper we propose *learning to downsample* module to address the issue.

Further, state-of-the-art real-time semantic segmentation remains challenging, and typically requires high-end GPUs. Hence, in this paper we aim for an above real-time semantic segmentation model on high resolution image data.

## 1.1 Contributions

Currently, semantic segmentation is typically addressed by a DCNN [2, 13, 24, 30]. Further, we observe that (i) a larger receptive field is important to learn complex correlations among object classes (*i.e.* global context) [29], (ii) spatial detail in images is necessary to preserve object boundaries [21], and (iii) initial DCNN layers extract low-level features [16, 28].

Consequently, we propose *fast segmentation convolutional neural network* (Fast-SCNN), an above real-time semantic segmentation algorithm, merging the two-branch setup of prior art [15, 18, 27, 30]. Building on the observation that initial DCNN layers extract low-level features [16, 28], we share the computations of the initial layers in the two-branch approach. We call this technique *learning to downsample*. The effect is similar to a skip connection in the encoder-decoder model, but the skip is only employed once to retain runtime and memory efficiency, and the module is kept shallow to ensure validity of feature sharing. Finally, our Fast-SCNN adopts efficient depthwise separable convolutions [8, 25], and inverse residual blocks [23].

Applied on Cityscapes [5], Fast-SCNN yields a mean intersection over union (mIoU) of 68.0% at 123.5 frames per second (fps) on a modern GPU (Nvidia Titan Xp (Pascal)) using full ($1024 \times 2048 px$) resolution, which is twice as fast as prior art BiSeNet [27] and ContextNet [18]. While we use 1.11 million parameters, most offline segmentation methods (*e.g.* DeepLab [3] and PSPNet [29]), and some real-time algorithms (*e.g.* GUN [15] and ICNet [30]) require much more than this. The model capacity of Fast-SCNN is kept specifically low. The reason is two-fold: (i) lower memory enables execution on embedded devices, and (ii) better generalisation is expected. In particular, pre-training on ImageNet [22] is frequently advised to boost accuracy and generality [29]. In our work, we study the effect of pre-training on the low capacity Fast-SCNN. Contradicting the trend of high-capacity networks, we find that results only insignificantly improve with pre-training or additional coarsely labeled training data (+0.5% mIoU on Cityscapes [5]). Instead, aggressive data augmentation and more number of epochs provide similar results. In summary our contributions are:

1. We propose Fast-SCNN, a competitive (68.0%) and above real-time semantic segmentation algorithm (123.5 fps on $1024 \times 2048 px$), which is twice as fast as prior art.
2. We propose a *learning to downsample* module for efficient multi-branch low-level feature extraction.
3. We specifically design Fast-SCNN to be of low capacity, and we empirically validate that running training for more epochs is equivalently successful to pre-training with ImageNet or training with additional coarse data in our small capacity network.

Moreover, we employ Fast-SCNN to subsampled input data, achieving state-of-the-art performance without the need for redesigning our network. Additionally, network quantization and network compression can be applied orthogonally, and is left to future work.

## 2 Proposed Fast-SCNN

Figure 1 and Table 1 present the layout of Fast-SCNN. In the following we discuss our motivation and describe our building blocks in more detail.

### 2.1 Motivation

Current state-of-the-art semantic segmentation methods that run in real-time are based on networks with two branches, each operating on a different resolution level [15, 18, 27]. They learn global information from low-resolution versions of the input image, and shallow networks at full input resolution are employed to refine the precision of the segmentation

| Input | Block | t | c | n | s |
|---|---|---|---|---|---|
| $1024 \times 2048 \times 3$ | Conv2D | - | 32 | 1 | 2 |
| $512 \times 1024 \times 32$ | DSConv | - | 48 | 1 | 2 |
| $256 \times 512 \times 48$ | DSConv | - | 64 | 1 | 2 |
| $128 \times 256 \times 64$ | bottleneck | 6 | 64 | 3 | 2 |
| $64 \times 128 \times 64$ | bottleneck | 6 | 96 | 3 | 2 |
| $32 \times 64 \times 96$ | bottleneck | 6 | 128 | 3 | 1 |
| $32 \times 64 \times 128$ | PPM | - | 128 | - | - |
| $32 \times 64 \times 128$ | FFM | - | 128 | - | - |
| $128 \times 256 \times 128$ | DSConv | - | 128 | 2 | 1 |
| $128 \times 256 \times 128$ | Conv2D | - | 19 | 1 | 1 |

Table 1: Fast-SCNN uses standard convolution (Conv2D), depthwise separable convolution (DSConv), inverted residual bottleneck blocks (bottleneck), a pyramid pooling module (PPM) and a feature fusion module (FFM) block. Parameters t, c, n and s represent expansion factor of the bottleneck block, number of output channels, number of times block is repeated and stride parameter which is applied to first sequence of the repeating block. The horizontal lines separate the modules: learning to down-sample, global feature extractor, feature fusion and classifier (top to bottom).

results. Since input resolution and network depth are main factors for runtime, these two-branch approaches allow for real-time computation.

It is well known that the first few layers of DCNNs extract the low-level features, such as edges and corners [16, 28]. Therefore, rather than employing a two-branch approach with separate computation, we introduce learning to downsample, which shares feature computation between the low and high-level branch in a shallow network block.

## 2.2 Network Architecture

Our Fast-SCNN uses a learning to downsample module, a coarse global feature extractor, a feature fusion module and a standard classifier.

**Learning to Downsample:** In our learning to downsample module, we employ only three layers to ensure low-level feature sharing is valid, and efficiently implemented. The first layer is a standard convolutional layer (Conv2D) and the remaining two layers are depthwise separable convolutional layers (DSConv). Here we emphasize, although DSConv is computationally more efficient, we employ Conv2D since the input image only has three channels, making DSConv's computational benefit insignificant at this stage.

All three layers in our learning to downsample module use stride 2, followed by batch normalization [10] and ReLU. The spatial kernel size of the convolutional and depthwise layers is $3 \times 3$. Following [4, 18, 23], we omit the nonlinearity between depthwise and pointwise convolutions.

**Global Feature Extractor:** The global feature extractor module is aimed at capturing the global context for image segmentation. In contrast to common multi-branch methods which operate on low-resolution versions of the input image, our module directly takes the output of the learning to downsample module (which is at $\frac{1}{8}$-resolution of the original input). The detailed structure of the module is shown in Table 1. We use an efficient bottleneck residual block introduced by MobileNet-V2 [23] (Table 2). In particular, we employ residual

| Input | Operator | Output |
|---|---|---|
| $h \times w \times c$ | Conv2D $1/1, f$ | $h \times w \times tc$ |
| $h \times w \times tc$ | DWConv $3/s, f$ | $\frac{h}{s} \times \frac{w}{s} \times tc$ |
| $\frac{h}{s} \times \frac{w}{s} \times tc$ | Conv2D $1/1, -$ | $\frac{h}{s} \times \frac{w}{s} \times c'$ |

Table 2: The *bottleneck residual block* transfers the input from $c$ to $c'$ channels with expansion factor $t$. Note, the last pointwise convolution does not use non-linearity $f$. The input is of height $h$ and width $w$, and x/$s$ represents kernel-size/stride of the layer.

| Higher resolution | X times lower resolution |
|---|---|
| - | Upsample $\times$ X |
| - | DWConv (dilation X) $3/1, f$ |
| Conv2D $1/1, -$ | Conv2D $1/1, -$ |
| add, $f$ | |

Table 3: Features fusion module (FFM) of Fast-SCNN. Note, the pointwise convolutions are of desired output, and do not use non-linearity $f$. Non-linearity $f$ is employed after adding the features.

connection for the bottleneck residual blocks when the input and output are of the same size. Our bottleneck block uses an efficient depthwise convolution, resulting in less number of parameters and floating point operations. Also, a pyramid pooling module (PPM) [29] is added at the end to aggregate the different-region-based context information.

**Feature Fusion Module:** We prefer simple addition of the features to ensure efficiency. Alternatively, more sophisticated feature fusion modules (*e.g.* [27]) could be employed at the cost of runtime performance, to reach better accuracy. The detail of the feature fusion module is shown in Table 3.

**Classifier:** In the classifier we employ two depthwise separable convolutions (DSConv) and one pointwise convolution (Conv2D). We found that adding few layers after the feature fusion module boosts the accuracy. The details of the classifier module is shown in Table 1.

Softmax is used during training, since gradient decent is employed. During inference we may substitute costly softmax computations with argmax, since both functions are monotonically increasing. We denote this option as Fast-SCNN cls *i.e.* classification. On the other hand, if a standard DCNN based probabilistic model is desired, softmax is used, denoted as Fast-SCNN prob *i.e.* probability.

# 3 Experiments

We evaluated our proposed Fast-SCNN on the validation set of the Cityscapes dataset [5], and report its performance on the Cityscapes test set, *i.e.* the Cityscapes benchmark server.

## 3.1 Implementation Details

Implementation detail is as important as theory when it comes to efficient DCNNs. Hence, we carefully describe our setup here. We conduct experiments on the TensorFlow machine learning platform using Python. Our experiments are executed on a workstation with either Nvidia Titan X (Maxwell) or Nvidia Titan Xp (Pascal) GPU, with CUDA 9.0 and CuDNN

| Model | Class | Category | Params |
|---|---|---|---|
| DeepLab-v2 [3]* | 70.4 | 86.4 | 44.– |
| PSPNet [29]* | 78.4 | 90.6 | 65.7_ |
| SegNet [2] | 56.1 | 79.8 | 29.46 |
| ENet [17] | 58.3 | 80.4 | 00.37 |
| ICNet [30]* | 69.5 | - | 06.68 |
| ERFNet [20] | 68.0 | 86.5 | 02.1_ |
| BiSeNet [27] | 71.4 | - | 05.8_ |
| GUN [15] | 70.4 | - | - |
| ContextNet [18] | 66.1 | 82.7 | 00.85 |
| Fast-SCNN (Ours) | 68.0 | 84.7 | 01.11 |

Table 4: Class and category mIoU of the proposed Fast-SCNN compared to other state-of-the-art methods on the Cityscapes test set. Number of parameters is listed in millions.

| | road | sidewalk | building | wall | fence | pole | traffic light | traffic sign | vegetation | terrain | sky | person | rider | car | truck | bus | train | motorcycle | bicycle |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ContextNet | 97.6 | 79.2 | 88.7 | 43.8 | 42.8 | 37.9 | 52.0 | 58.8 | 90.0 | 66.8 | 91.9 | 72.1 | 53.9 | 91.6 | 54.0 | 66.4 | 58.3 | 48.9 | 61.0 |
| Fast-SCNN | 97.9 | 81.5 | 89.6 | 46.3 | 48.6 | 48.3 | 53.0 | 60.5 | 90.7 | 67.1 | 94.3 | 73.9 | 54.5 | 92.9 | 57.4 | 65.4 | 58.2 | 50.0 | 61.2 |

Table 5: Detailed class-level mIoU of the proposed Fast-SCNN compared with ContextNet.

v7. Runtime evaluation is performed in a single CPU thread and one GPU with batch-size one to measure the forward inference time. We use 100 frames for burn-in and report average of 100 frames for the frames per second (fps) measurement.

We use stochastic gradient decent (SGD) with momentum 0.9 and batch-size 12. Inspired by [3, 8, 29] we use 'poly' learning rate with the base one as 0.045 and power as 0.9. Similar to MobileNet-V2 we found that $\ell_2$ regularization is not necessary on depthwise convolutions, for other layers $\ell_2$ is 0.00004. Since training data for semantic segmentation is limited, we apply various data augmentation techniques: random resizing between 0.5 to 2, translation/crop, horizontal flip, color channels noise and brightness. Our model is trained with cross-entropy loss. We found that auxiliary losses at the end of learning to downsample and the global feature extraction modules with 0.4 weights are beneficial.

Batch normalization [10] is used before every non-linear function. Dropout is used only on the last layer, just before the softmax layer. Contrary to MobileNet [8] and ContextNet [18], we found that Fast-SCNN trains faster with ReLU and achieves slightly better accuracy than ReLU6, even with the depthwise separable convolutions that we use throughout our model.

We found that the performance of DCNNs can be improved by training for higher number of iterations, hence we train our model for 1,000 epochs unless otherwise stated, using the Cityescapes dataset [5]. It is worth noting here, Fast-SCNN's capacity is deliberately very low, as we employ 1.11 million parameters. Later we show that aggressive data augmentation techniques make overfitting unlikely.

## 3.2 Evaluation on Cityscapes

We evaluate our proposed Fast-SCNN on Cityscapes, the largest publicly available dataset of urban roads [5]. This dataset contains a diverse set of high resolution images ($1024 \times 2048px$) captured from 50 different cities. It has 5,000 images with high label quality: a

| | $1024 \times 2048$ | $512 \times 1024$ | $256 \times 512$ |
|---|---|---|---|
| SegNet [2] | 1.6 | - | - |
| ENet [17] | 20.4 | 76.9 | 142.9 |
| ICNet [30] | 30.3 | - | - |
| ERFNet [20] | 11.2 | 41.7 | 125.0 |
| ContextNet [13] | 41.9 | 136.2 | 299.5 |
| Fast-SCNN prob (Ours) | 62.1 | 197.6 | 372.8 |
| Fast-SCNN cls (Ours) | 75.3 | 218.8 | 403.1 |
| BiSeNet [27]* | 57.3 | - | - |
| GUN [15]* | 33.3 | - | - |
| Fast-SCNN prob (Ours)* | 106.2 | 266.3 | 432.9 |
| Fast-SCNN cls (Ours)* | 123.5 | 285.8 | 485.4 |

Table 6: Runtime (fps) on Nvidia Titan X (Maxwell, 3,072 CUDA cores) with TensorFlow [1]. '*' represent results on Nvidia Titan Xp (Pascal, 3,840 CUDA cores). Two versions of Fast-SCNN are shown: softmax output (our prob), and object label output (our cls).

training set of 2,975, validation set of 500 and test set of 1,525 images. The label for the training set and validation set are available and test results can be evaluated on the evaluation server. Additionally, 20,000 weakly annotated images (coarse labels) are available for training. We report results with both, fine only and fine with coarse labeled data. Cityscapes provides 30 class labels, while only 19 classes are used for evaluation. The mean of intersection over union (mIoU), and network inference time are reported in the following.

We evaluate overall performance on the withheld test set of Cityscapes [5]. The comparison between the proposed Fast-SCNN and other state-of-the-art real-time semantic segmentation methods and offline methods is shown in Table 4. Fast-SCNN achieves 68.0% mIoU, which is slightly lower than BiSeNet (71.5%) and GUN (70.4%). ContextNet only achieves 66.1% here. Further, Table 5 compares detailed class-level mIoU of the Fast-SCNN with ContextNet. The results of Fast-SCNN are displayed in Figure 2 for qualitative analysis.

Table 6 compares runtime at different resolutions. Here, BiSeNet (57.3 fps) and GUN (33.3 fps) are significantly slower than Fast-SCNN (123.5 fps). Compared to ContextNet (41.9 fps), Fast-SCNN is also significantly faster on Nvidia Titan X (Maxwell). Therefore we conclude, Fast-SCNN significantly improves upon state-of-the-art runtime with minor loss in accuracy. At this point we emphasize, our model is designed for low memory embedded devices. Fast-SCNN uses 1.11 million parameters, that is five times less than the competing BiSeNet at 5.8 million.

**Ablation Study** In our ablation study of learning to downsample module, we zero-out the contribution of the skip connection and measure Fast-SCNN's performance. The mIoU reduced from 69.22% to 64.30% on the validation set. The qualitative results are compared in Figure 3. As expected, Fast-SCNN benefits from the skip connection, especially around boundaries and objects of small size.

## 3.3 Lower Input Resolution

Since we are interested in embedded devices that may not have full resolution input, or access to powerful GPUs, we conclude our evaluation with the study of performance at half, and quarter input resolutions (Table 7).

At quarter resolution, Fast-SCNN achieves 51.9% accuracy at 485.4 fps, which signif-

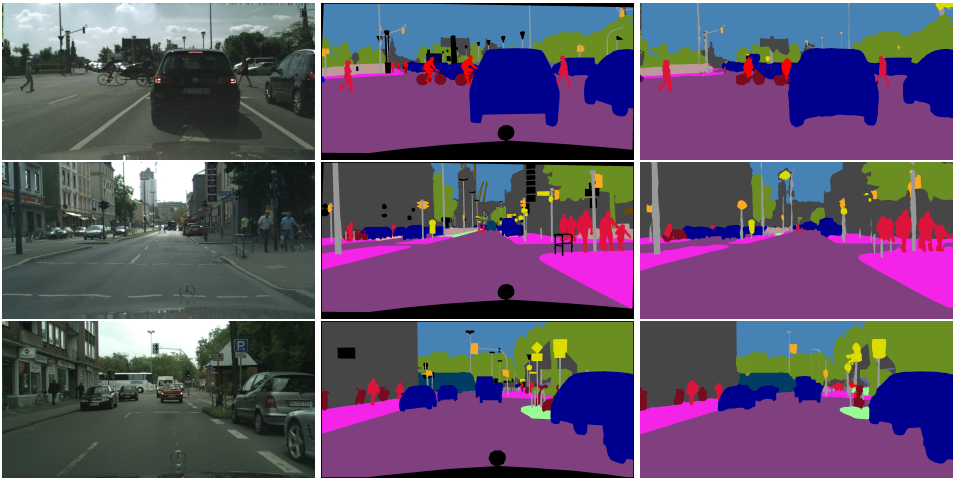

Figure 2: Qualitative results of Fast-SCNN on Cityscapes [5] validation set. First column: input RGB images; second column: ground truth labels; and last column: Fast-SCNN outputs. Fast-SCNN obtains 68.0% class level mIoU and 84.7% category level mIoU.

| Input Size | Class | FPS |
|---|---|---|
| $1024 \times 2048$ | 68.0 | 123.5 |
| $512 \times 1024$ | 62.8 | 285.8 |
| $256 \times 512$ | 51.9 | 485.4 |

Table 7: Evaluation of Fast-SCNN at different input resolutions on Cityscapes' test set.

| Model | Class |
|---|---|
| Fast-SCNN | 68.62 |
| Fast-SCNN + ImageNet | 69.15 |
| Fast-SCNN + Coarse | 69.22 |
| Fast-SCNN + Coarse + ImageNet | 69.19 |

Table 8: Class mIoU of different Fast-SCNN settings on the Cityscapes validation set.

icantly improves on (anonymous) MiniNet with 40.7% mIoU at 250 fps [5]. At half resolution, a competitive 62.8% mIoU at 285.8 fps is reached. We emphasize, without modification, Fast-SCNN is directly applicable to lower input resolution, making it suitable for embedded devices.

## 3.4 Pre-training and Weakly Labeled Data

High capacity DCNNs, such as R-CNN [6] and PSPNet [29], have shown that performance can be boosted with pre-training through different auxiliary tasks. As we specifically design Fast-SCNN to have low capacity, we now want to test performance with and without pre-training, and in connection with and without additional weakly labeled data. To the best of our knowledge, the significance of pre-training and additional weakly labeled data on low capacity DCNNs has not been studied before. Table 8 shows the results.

We pre-train Fast-SCNN on ImageNet [22] by replacing the feature fusion module with average pooling and the classification module now has a softmax layer only. Fast-SCNN achieves 60.71% top-1 and 83.0% top-5 accuracies on the ImageNet validation set. This result indicates that Fast-SCNN has insufficient capacity to reach comparable performance to most standard DCNNs on ImageNet (>70% top-1) [8, 23]. The accuracy of Fast-SCNN with ImageNet pre-training yields 69.15% mIoU on the validation set of Cityscapes, only 0.53%

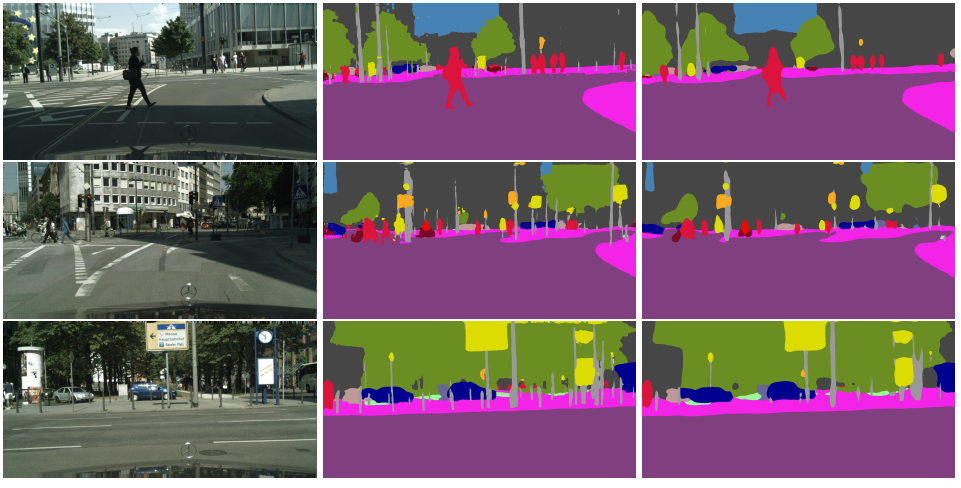

Figure 3: Visualization of Fast-SCNN's segmentation results. First column: input RGB images; second column: outputs of Fast-SCNN; and last column: outputs of Fast-SCNN after zeroing-out the contribution of the skip connection. In all results, Fast-SCNN benefits from skip connections especially at boundaries and objects of small size.

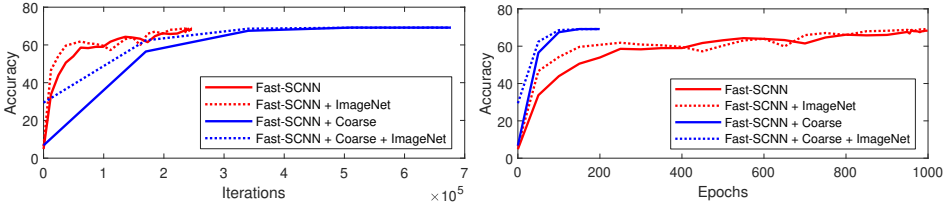


Figure 4: Training curves on Cityscapes. Accuracy over iterations (left), and accuracy over epochs are shown (right). Dash lines represent ImageNet pre-training of the Fast-SCNN.

improvement over Fast-SCNN without pre-training. Therefore we conclude, no significant boost can be achieved with ImageNet pre-training in Fast-SCNN.

Since the overlap between Cityscapes' urban roads and ImageNet's classification task is limited, it is reasonable to assume that Fast-SCNN may not benefit due to limited capacity for both domains. Therefore, we now incorporate the 20,000 coarsely labeled additional images provided by Cityscapes, as these are from a similar domain. Nevertheless, Fast-SCNN trained with coarse training data (with or without ImageNet) performs similar to each other, and only slightly improve upon the original Fast-SCNN without pre-training. Please note, small variations are insignificant and due to random initializations of the DCNNs.

It is worth noting here that working with auxiliary tasks is non-trivial as it requires architectural modifications in the network. Furthermore, licence restrictions and lack of resources further limit such setups. These costs can be saved, since we show that neither ImageNet pre-training nor weakly labeled data are significantly beneficial for our low capacity DCNN.

Figure 4 shows the training curves. Fast-SCNN with coarse data trains slow in terms of iterations because of the weak label quality. Both ImageNet pre-trained versions perform

better for early epochs (upto 400 epochs for training set alone, and 100 epochs when trained with the additional coarse labeled data). This means, we only need to train our model for longer to reach similar accuracy when we train our model from scratch on reasonable size dataset, confirming similar finding by [13].

# 4 Conclusions

In this work we proposed a fast segmentation network for above real-time scene understanding. Proposed 'learning to downsample' module computes low-level features for multiple resolution branches simultaneously. Sharing the computational cost of the multi-branch network yields run-time efficiency. In experiments our skip connection is shown beneficial for recovering the spatial details. We also demonstrate that if trained for long enough, large-scale pre-training of the model on an additional auxiliary task is not necessary for the low capacity network.

# References

[1] M. Abadi and et. al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/.

[2] V. Badrinarayanan, A. Kendall, and R. Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *TPAMI*, 2017.

[3] 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. *arXiv:1606.00915 [cs]*, 2016.

[4] F. Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. *arXiv:1610.02357 [cs]*, 2016.

[5] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013.

[7] S. Han, H. Mao, and W. J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR*, 2016.

[8] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]*, 2017.

[9] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized Neural Networks. In *NIPS*. 2016.

[10] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, 2015.

[11] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, volume 2, pages 2169–2178, 2006.

[12] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning Filters for Efficient ConvNets. In *ICLR*, 2017.

[13] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning, 2018.

[14] A. Lucchi, Y. Li, X. B. Bosch, K. Smith, and P. Fua. Are spatial and global constraints really necessary for segmentation? In *ICCV*, 2011.

[15] D. Mazzini. Guided Upsampling Network for Real-Time Semantic Segmentation. In *BMVC*, 2018.

[16] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017.

[17] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation. *arXiv:1606.02147 [cs]*, 2016.

[18] R.P.K. Poudel, U. Bonde, S. Liwicki, and C. Zach. ContextNet: Exploring context and detail for semantic segmentation in real-time. In *BMVC*, 2018.

[19] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *ECCV*, 2016.

[20] E. Romera, J. M. Álvarez, L. M. Bergasa, and R. Arroyo. ERFNet: Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 2018.

[21] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *MICCAI*, 2015.

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015.

[23] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation. *arXiv:1801.04381 [cs]*, 2018.

[24] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *PAMI*, 2016.

[25] L. Sifre. *Rigid-motion scattering for image classification*. PhD thesis, 2014.

[26] S. Wu, G. Li, F. Chen, and L. Shi. Training and Inference with Integers in Deep Neural Networks. In *ICLR*, 2018.

[27] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *ECCV*, 2018.

[28] M. D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. In *ECCV*, 2014.

[29] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid Scene Parsing Network. In *CVPR*, 2017.

[30] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia. ICNet for Real-Time Semantic Segmentation on High-Resolution Images. In *ECCV*, 2018.