

# Fast-SCNN: Fast Semantic Segmentation Network

Rudra PK Poudel  
Toshiba Research Europe, UK  
rudra.poudel@crl.toshiba.co.uk

Stephan Liwicki  
Toshiba Research Europe, UK  
stephan.liwicki@crl.toshiba.co.uk

Roberto Cipolla  
Cambridge University, UK  
rc10001@cam.ac.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 ( $1024 \times 2048px$ ) suited to efficient computation on embedded devices with low memory. Building on existing two-branch methods for fast segmentation, we introduce our ‘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 Cityscapes. We also show that large scale pre-training is unnecessary. We thoroughly validate our metric in 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 real-time applications, where input is to be parsed quickly to facilitate responsive interactivity with the environment. Due to the increasing interest in autonomous systems and robotics, it is therefore evident that the research into real-time semantic segmentation has recently enjoyed significant gain in popularity [21, 34, 17, 25, 36, 20]. We emphasize, faster than real-time performance is in fact often necessary, since semantic labeling is usually employed only as preprocessing step of other time-critical tasks. Furthermore, real-time semantic segmentation on embedded devices (without access to powerful GPUs) may enable many additional ap-

plications, such as augmented reality for wearables.

We observe, in literature semantic segmentation is typically addressed by a deep convolutional neural network (DCNN) with an encoder-decoder framework [29, 2], while many runtime efficient implementations employ a two- or multi-branch architecture [21, 34, 17]. It is often the case that

- a larger receptive field is important to learn complex correlations among object classes (*i.e.* global context),
- spatial detail in images is necessary to preserve object boundaries, and
- specific designs are needed to balance speed and accuracy (rather than re-targeting classification DCNNs).

Specifically in the two-branch networks, a deeper branch is employed at low resolution to capture global context, while a shallow branch is setup to learn spatial details at full input resolution. The final semantic segmentation result is then provided by merging the two. Importantly since the computational cost of deeper networks is overcome with a small input size, and execution on full resolution is only employed for few layers, real-time performance is possible on modern GPUs. In contrast to the encoder-decoder framework, the initial convolutions at different resolutions are not shared in the two-branch approach. Here it is worth noting, the guided upsampling network (GUN) [17] and the image cascade network (ICNet) [36] only share the weights among the first few layers, but not the computation.

In this work we propose *fast segmentation convolutional neural network* Fast-SCNN, an above real-time semantic segmentation algorithm merging the two-branch setup of prior art [21, 34, 17, 36], with the classical encoder-decoder framework [29, 2] (Figure 1). Building on the observation that initial DCNN layers extract low-level features [35, 19], 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

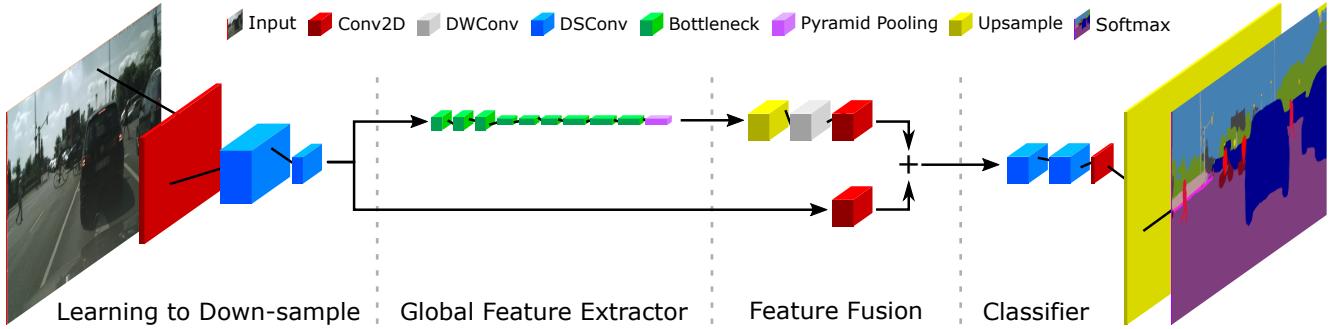


Figure 1. Fast-SCNN shares the computations between two branches (encoder) to build a above real-time semantic segmentation network.

encoder-decoder model, but the skip is only employed once to retain runtime efficiency, and the module is kept shallow to ensure validity of feature sharing. Finally, our Fast-SCNN adopts efficient depthwise separable convolutions [30, 10], and inverse residual blocks [28].

Applied on Cityscapes [6], 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 ( $1024 \times 2048px$ ) resolution, which is twice as fast as prior art i.e. BiSeNet (71.4% mIoU)[34].

While we use 1.11 million parameters, most offline segmentation methods (*e.g.* DeepLab [4] and PSPNet [37]), and some real-time algorithms (*e.g.* GUN [17] and ICNet [36]) 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 [27] is frequently advised to boost accuracy and generality [37]. 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 [6]). In summary our contributions are:

1. We propose Fast-SCNN, a competitive (68.0%) and above real-time semantic segmentation algorithm (123.5 fps) for high resolution images ( $1024 \times 2048px$ ).
2. We adapt the skip connection, popular in offline DCNNs, and propose a shallow *learning to downsample* module for fast and 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.

## 2. Related Work

We discuss and compare semantic image segmentation frameworks with a particular focus on real-time execution with low energy and memory requirements [2, 20, 21, 36, 34, 17, 25, 18].

### 2.1. Foundation of Semantic Segmentation

State-of-the-art semantic segmentation 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) [29, 2]. Most commonly, the encoder is adapted from a simple classification DCNN method, such as VGG [31] or ResNet [9]. In semantic segmentation, the fully connected layers are removed.

The seminal fully convolution network (FCN) [29] laid the foundation for most modern segmentation architectures. Specifically, FCN employs VGG [31] as encoder, and bilinear upsampling in combination with skip-connection from lower layers to recover spatial detail. U-Net [26] further exploited the spatial details using dense skip connections.

Later, inspired by global image-level context prior to DCNNs [13, 16], the pyramid pooling module of PSPNet [37] and atrous spatial pyramid pooling (ASPP) of DeepLab [4] are employed to encode and utilize global context.

Other competitive fundamental segmentation architectures use conditional random fields (CRF) [38, 3] or recurrent neural networks [32, 38]. However, none of them run in real-time.

Similar to the object detection [23, 24, 15], speed became one important factor in image segmentation system design [21, 34, 17, 25, 36, 20]. Building on FCN, SegNet [2] introduced a joint encoder-decoder model and became

one of the earliest efficient segmentation models. Following SegNet, ENet [20] also design an encoder-decoder with few layers to reduce the computational cost.

More recently, two-branch and multi-branch systems were introduced. ICNet [36], ContextNet [21], BiSeNet [34] and GUN [17] learned global context with reduced-resolution input in a deep branch, while boundaries are learned in a shallow branch at full resolution.

However, state-of-the-art real-time semantic segmentation remains challenging, and typically requires high-end GPUs. Inspired by two-branch methods, Fast-SCNN incorporates a shared shallow network path to encode detail, while context is efficiently learned at low resolution (Figure 2).

## 2.2. Efficiency in DCNNs

The common techniques of efficient DCNNs can be divided into four categories:

**Depthwise Separable Convolutions:** MobileNet [10] 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 convolutional parameters, hence the computational cost and memory requirement of the model is reduced.

**Efficient Redesign of DCNNs:** Chollet [5] designed the Xception network using efficient depthwise separable convolution. MobleNet-V2 proposed inverted bottleneck residual blocks [28] to build an efficient DCNN for the classification task. ContextNet [21] used inverted bottleneck residual blocks to design a two-branch network for efficient real-time semantic segmentation. Similarly, [34, 17, 36] propose multi-branch segmentation networks to achieve real-time performance.

**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 [11, 22, 33].

**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 [21, 8, 14].

Fast-SCNN relies heavily on depthwise separable convolutions and residual bottleneck blocks [28]. Furthermore we introduce a two-branch model that incorporates our learning to downsample module, allowing for shared feature extraction at multiple resolution levels (Figure 2). Note, even though the initial layers of the multiple branches extract similar features [35, 19], common two-branch approaches do not leverage this. Network quantization and network compression can be applied orthogonally, and is left to future work.

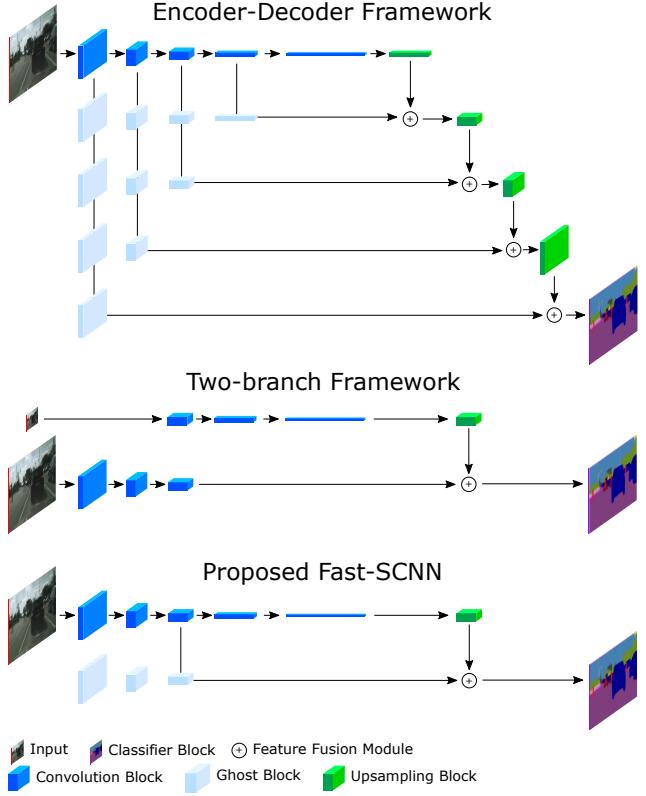


Figure 2. Schematic comparison of Fast-SCNN with encoder-decoder and two-branch architectures. Encoder-decoder employs multiple skip connections at many resolutions, often resulting from deep convolution blocks. Two-branch methods employ global features from low resolution with shallow spatial detail. Fast-SCNN encodes spatial detail and initial layers of global context in our learning to downsample module simultaneously.

## 2.3. Pre-training on Auxiliary Tasks

It is a common belief that pre-training on auxiliary tasks boosts system accuracy. Earlier works on object detection [7] and semantic segmentation [4, 37] have shown this with pre-training on ImageNet [27]. Following this trend, other real-time efficient semantic segmentation methods are also pre-trained on ImageNet [36, 34, 17]. However, it is not known whether pre-training is necessary on low-capacity networks. Fast-SCNN is specifically designed with low capacity. In our experiments we show that small networks do not get significant benefit from pre-training. Instead, aggressive data augmentation and more number of epochs provide similar results.

## 3. Proposed Fast-SCNN

Fast-SCNN is inspired by the two-branch architectures [21, 34, 17] and encoder-decoder networks with skip connections [29, 26]. Noting that early layers commonly ex-

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).

tract low-level features. We reinterpret skip connections as a learning to downsample module, enabling us to merge the key ideas of both frameworks, and allowing us to build a fast semantic segmentation model. 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.

### 3.1. 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 [21, 34, 17]. 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 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 [35, 19]. 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.

### 3.2. Network Architecture

Our Fast-SCNN uses a learning to downsample module, a coarse global feature extractor, a feature fusion module and a standard classifier. All modules are built using depthwise separable convolution, which has become a key building block for many efficient DCNN architectures [5, 10, 21].

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 and stride of the layer.

#### 3.2.1 Learning to Downsample

In our learning to downsample module, we employ three layers. Only three layers are employed 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 [12] and ReLU. The spatial kernel size of the convolutional and depthwise layers is  $3 \times 3$ . Following [5, 28, 21], we omit the nonlinearity between depthwise and pointwise convolutions.

#### 3.2.2 Global Feature Extractor

The global feature extractor module is aimed at capturing the global context for image segmentation. In contrast to common two-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 efficient bottleneck residual block introduced by MobileNet-V2 [28] (Table 2). In particular, we employ residual connection for the bottleneck residual blocks when the input and output are of the same size. Our bottleneck block uses an efficient depthwise separable convolution, resulting in less number of parameters and floating point operations. Also, a pyramid pooling module (PPM) [37] is added at the end to aggregate the different-region-based context information.

#### 3.2.3 Feature Fusion Module

Similar to ICNet [36] and ContextNet [21] we prefer simple addition of the features to ensure efficiency. Alternatively, more sophisticated feature fusion modules (e.g. [34]) could be employed at the cost of runtime performance, to reach

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.

better accuracy. The detail of the feature fusion module is shown in Table 3.

### 3.2.4 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 the 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 (classification). On the other hand, if a standard DCNN based probabilistic model is desired, softmax is used, denoted as Fast-SCNN prob (probability).

## 3.3 Comparison with Prior Art

Our model is inspired by the two-branch framework, and incorporates ideas of encoder-decoder methods (Figure 2).

### 3.3.1 Relation with Two-branch Models

The state-of-the-art real-time models (ContextNet [21], BiSeNet [34] and GUN [17]) use two-branch networks. Our learning to downsample module is equivalent to their spatial path, as it is shallow, learns from full resolution, and is used in the feature fusion module (Figure 1).

Our global feature extractor module is equivalent to the deeper low-resolution branch of such approaches. In contrast, our global feature extractor shares its computation of the first few layers with the learning to downsample module. By sharing the layers we not only reduce computational complexity of feature extraction, but we also reduce the required input size as Fast-SCNN uses  $\frac{1}{8}$ -resolution instead of  $\frac{1}{4}$ -resolution for global feature extraction.

### 3.3.2 Relation with Encoder-Decoder Models

Proposed Fast-SCNN can be viewed as a special case of an encoder-decoder framework, such as FCN [29] or U-Net [26]. However, unlike the multiple skip connections in FCN and the dense skip connections in U-Net, Fast-SCNN only

employs a single skip connection to reduce computations as well as memory.

In correspondence with [35], who advocate that features are shared only at early layers in DCNNs, we position our skip connection early in our network. In contrast, prior art typically employ deeper modules at each resolution, before skip connections are applied.

## 4 Experiments

We evaluated our proposed fast segmentation convolutional neural network (Fast-SCNN) on the validation set of the Cityscapes dataset [6], and report its performance on the Cityscapes test set, *i.e.* the Cityscapes benchmark server.

### 4.1 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 v7. Runtime evaluation is performed in a single CPU thread and one GPU 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 [4, 37, 10] 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 [12] is used before every non-linear function. Dropout is used only on the last layer, just before the softmax layer. Contrary to MobileNet [10] and ContextNet [21], 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 [6]. 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.

Model	Class	Category	Params
DeepLab-v2 [4]*	70.4	86.4	44.-
PSPNet [37]*	78.4	90.6	65.7_-
SegNet [2]	56.1	79.8	29.46
ENet [20]	58.3	80.4	00.37
ICNet [36]*	69.5	-	06.68
ERFNet [25]	68.0	86.5	02.1_-
BiSeNet [34]	71.4	-	05.8_-
GUN [17]	70.4	-	-
ContextNet [21]	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 semantic segmentation methods on the Cityscapes test set. Number of parameters is listed in millions.

	1024 × 2048	512 × 1024	256 × 512
SegNet [2]	1.6	-	-
ENet [20]	20.4	76.9	142.9
ICNet [36]	30.3	-	-
ERFNet [25]	11.2	41.7	125.0
ContextNet [21]	41.9	136.2	299.5
Our prob	62.1	197.6	372.8
Our cls	75.3	218.8	403.1
BiSeNet [34]*	57.3	-	-
GUN [17]*	33.3	-	-
Our prob*	106.2	266.3	432.9
Our cls*	123.5	285.8	485.4

Table 5. Runtime (fps) on Nvidia Titan X (Maxwell, 3,072 CUDA cores) with TensorFlow [1]. Methods with “\*” 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).

## 4.2. Evaluation on Cityscapes

We evaluate our proposed Fast-SCNN on Cityscapes, the largest publicly available dataset on urban roads [6]. This dataset contains a diverse set of high resolution images ( $1024 \times 2048px$ ) captured from 50 different cities in Europe. It has 5,000 images with high label quality: a 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 [6]. The comparison between the proposed Fast-SCNN and other state-of-the-art real-time semantic segmentation methods (ContextNet [21], BiSeNet [34], GUN [17], ENet [20] and ICNet [36]) and offline

Model	Class
ContextNet [21]	65.9_-
Fast-SCNN	68.62
Fast-SCNN + ImageNet	69.15
Fast-SCNN + Coarse	69.22
Fast-SCNN + Coarse + ImageNet	69.19

Table 6. Class mIoU of different Fast-SCNN settings on the Cityscapes validation set.

methods (PSPNet [37] and DeepLab-V2 [4]) 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.

Table 5 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.

Finally, 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.

## 4.3. Pre-training and Weakly Labeled Data

High capacity DCNNs, such as R-CNN [7] and PSPNet [37], 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 6 shows the results.

We pre-train Fast-SCNN on ImageNet [27] 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) [10, 28]. The accuracy of Fast-SCNN with ImageNet pre-training yields 69.15% mIoU on the validation set of Cityscapes, only 0.53% improvement over Fast-SCNN without pre-training. Therefore we conclude, no significant boost can be achieved with ImageNet pre-training

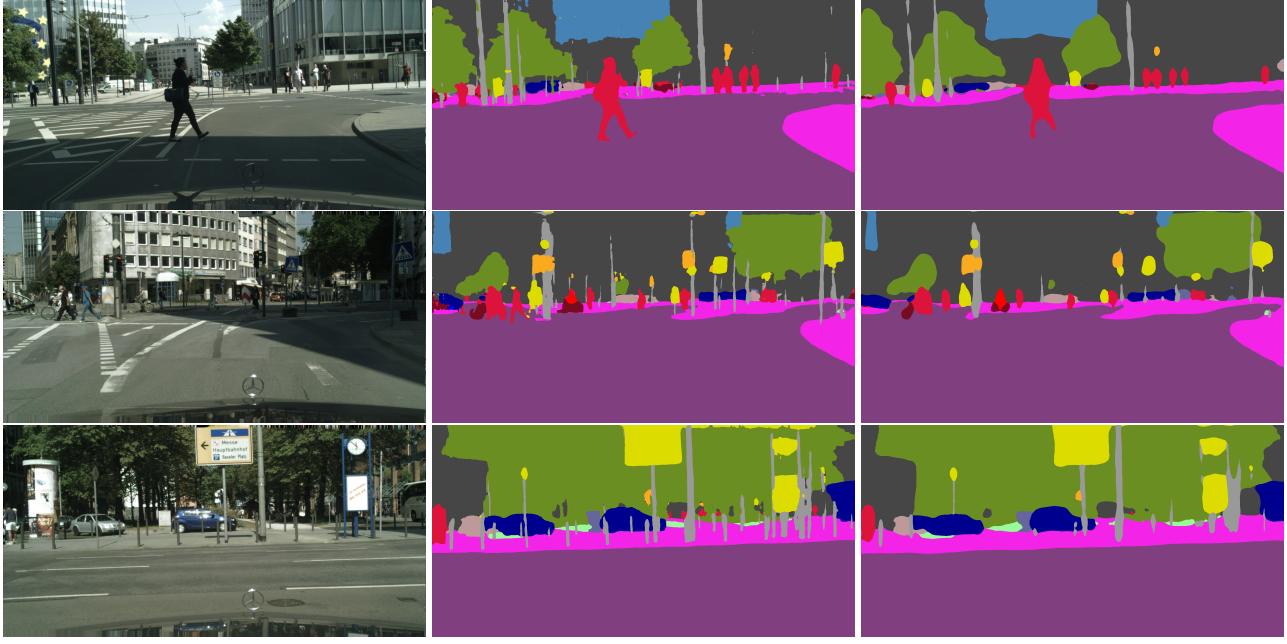


Figure 3. Visualization of Fast-SCNN’s segmentation results. First column: input RGB images; second column: outputs of Fast-SCNN; and last column: outputs of Fast-SCNN after zeroing-out the contribution of the skip connection. In all results, Fast-SCNN benefits from skip connections especially at boundaries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

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) perform 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.

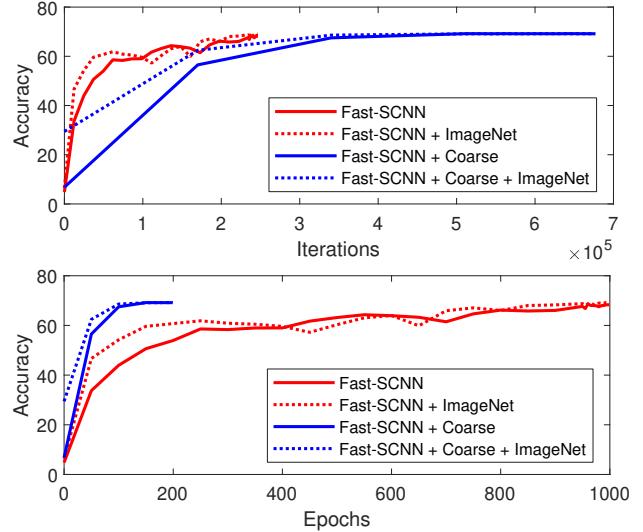


Figure 4. Training curves on Cityscapes. Accuracy over iterations (top), and accuracy over epochs are shown (bottom). Dash lines represent ImageNet pre-training of the Fast-SCNN.

#### 4.4. 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% accu-

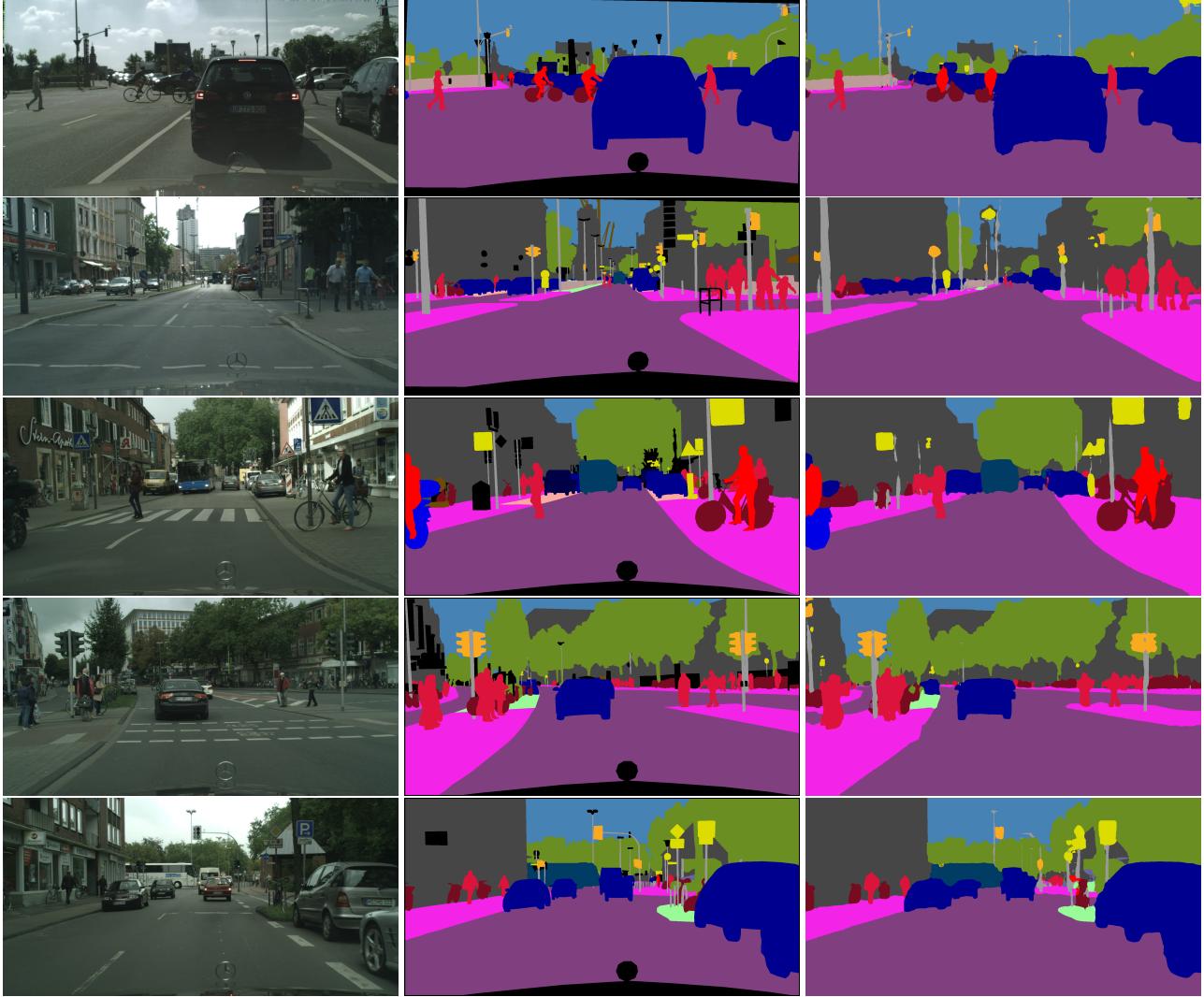


Figure 5. Qualitative results of Fast-SCNN on Cityscapes [6] validation set. First column: input RGB images; second column: ground truth labels; and last column: Fast-SCNN outputs. Fast-SCNN obtains 68.0% class level mIoU and 84.7% category level mIoU.

Input Size	Class	FPS
1024 × 2048	68.0	123.5
512 × 1024	62.8	285.8
256 × 512	51.9	485.4

Table 7. Runtime and accuracy of Fast-SCNN at different input resolutions on Cityscapes’ test set [6].

uracy at 485.4 fps, which significantly improves on (anonymous) MiniNet with 40.7% mIoU at 250 fps [6]. 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 highly suitable for embedded devices.

## 5. Conclusions

We propose a fast segmentation network for above real-time scene understanding. 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. 6
- [2] V. Badrinarayanan, A. Kendall, and R. Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Im-

- age Segmentation. *TPAMI*, 2017. 1, 2, 6
- [3] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs, 2014. 2
- [4] 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. 2, 3, 5, 6
- [5] F. Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. *arXiv:1610.02357 [cs]*, 2016. 3, 4
- [6] 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. 2, 5, 6, 8
- [7] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013. 3, 6
- [8] S. Han, H. Mao, and W. J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR*, 2016. 3
- [9] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, 2015. 2
- [10] 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. 2, 3, 4, 5, 6
- [11] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized Neural Networks. In *NIPS*. 2016. 3
- [12] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, 2015. 4, 5
- [13] 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. 2
- [14] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning Filters for Efficient ConvNets. In *ICLR*, 2017. 3
- [15] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. 2015. 2
- [16] A. Lucchi, Y. Li, X. B. Bosch, K. Smith, and P. Fua. Are spatial and global constraints really necessary for segmentation? In *ICCV*, 2011. 2
- [17] D. Mazzini. Guided Upsampling Network for Real-Time Semantic Segmentation. In *BMVC*, 2018. 1, 2, 3, 4, 5, 6
- [18] S. Mehta, M. Rastegari, A. Caspi, L. Shapiro, and H. Hajishirzi. ESPNet: Efficient Spatial Pyramid of Dilated Convolutions for Semantic Segmentation. *arXiv:1803.06815 [cs]*, 2018. 2
- [19] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017. 1, 3, 4
- [20] 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. 1, 2, 3, 6
- [21] R. Poudel, U. Bonde, S. Liwicki, and S. Zach. Contextnet: Exploring context and detail for semantic segmentation in real-time. In *BMVC*, 2018. 1, 2, 3, 4, 5, 6
- [22] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *ECCV*, 2016. 3
- [23] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016. 2
- [24] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger, 2016. 2
- [25] 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. 1, 2, 6
- [26] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *MICCAI*, 2015. 2, 3, 5
- [27] 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. 2, 3, 6
- [28] 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. 2, 3, 4, 6
- [29] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *PAMI*, 2016. 1, 2, 3, 5
- [30] L. Sifre. *Rigid-motion scattering for image classification*. PhD thesis, 2014. 2
- [31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 2
- [32] F. Visin, M. Ciccone, A. Romero, K. Kastner, K. Cho, Y. Bengio, M. Matteucci, and A. Courville. Reseg: A recurrent neural network-based model for semantic segmentation, 2015. 2
- [33] S. Wu, G. Li, F. Chen, and L. Shi. Training and Inference with Integers in Deep Neural Networks. In *ICLR*, 2018. 3
- [34] 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. 1, 2, 3, 4, 5, 6
- [35] M. D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. In *ECCV*, 2014. 1, 3, 4, 5
- [36] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia. ICNet for Real-Time Semantic Segmentation on High-Resolution Images. In *ECCV*, 2018. 1, 2, 3, 4, 6
- [37] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid Scene Parsing Network. In *CVPR*, 2017. 2, 3, 4, 5, 6
- [38] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr. Conditional random fields as recurrent neural networks. In *ICCV*, December 2015. 2