HyperSeg: Patch-wise Hypernetwork for Real-time Semantic Segmentation

Yuval Nirkin*
Facebook AI & Bar-Ilan University

Lior Wolf
Facebook AI & Tel Aviv University

Tal Hassner
Facebook AI

Abstract

We present a novel, real-time, semantic segmentation network in which the encoder both encodes and generates the parameters (weights) of the decoder. Furthermore, to allow maximal adaptivity, the weights at each decoder block vary spatially. For this purpose, we design a new type of hypernetwork, composed of a nested U-Net for drawing higher level context features, a multi-headed weight generating module which generates the weights of each block in the decoder immediately before they are consumed, for efficient memory utilization, and a primary network that is composed of novel dynamic patch-wise convolutions. Despite the usage of less-conventional blocks, our architecture obtains real-time performance. In terms of the run-time vs. accuracy trade-off, we surpass state of the art (SotA) results on popular semantic segmentation benchmarks: PASCAL VOC 2012 (val. set) and real-time semantic segmentation on Cityscapes, and CamVid. The code is available: https://nirkin.com/hyperseg.

1. Introduction

Semantic segmentation plays a crucial role in scene understanding, whether the scene is microscopic, telescopic, captured by a moving vehicle, or viewed through an AR device. New mobile applications go beyond seeking accurate semantic segmentation, and also requiring real-time processing, spurring research into real-time semantic segmentation. This domain has since become a leading testbed for new architectures and training methods, with the goals of improving both accuracy and speed. Recent work added capacity [5, 6] and attention mechanisms [20, 45, 49] to improve performance. When runtime is not a concern, the image is often processed multiple times by the model and the results are accumulated. In this paper, we attempt to improve the performance in a different way: by providing the network with additional adaptivity.

We add this adaptivity using a meta-learning technique, often referred to as *dynamic networks* or *hypernetworks* [13]. These networks are used for tasks ranging from text analysis [13, 50] to 3D modeling [26, 42], but rarely for generating image-like maps. The reason is that the hypernetworks, as suggested by previous methods, do not fully capture the signals of high resolution images.

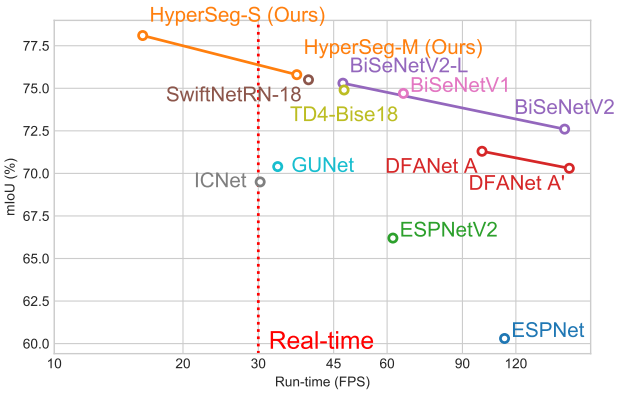

Figure 1. *Run-time / accuracy trade-off comparison on the Cityscapes [8] test set.* Our models (in orange) achieve the best accuracy and the best run-time vs. accuracy trade-off relative to all previous real-time methods.

Semantic segmentation map are an especially interesting case. They are generated by a coarse to fine pyramid, where each level of the process can benefit from adaptation, since these effects accumulate from one block to the next. Moreover, since every part of the image may contain a different object, such adaptation is best done *locally*.

We thus offer a novel encoder-decoder approach, in which the encoder's backbone is based on recent advances in the field. The encoded signal is mapped to dynamic network weights using an internal U-Net, while the decoder consists of dynamic blocks with spatially varying weights.

The proposed architecture achieves SotA accuracy vs. runtime trade-off on the most widely used benchmarks for this task: PASCAL VOC 2012 [11], CityScapes [8], and CamVid [2]. For CityScapes and CamVid, the SotA accuracy result is obtained under the real-time conditions. Despite using an unconventional architecture that employs locally connected layers with dynamic weights, our method is very efficient (See Fig. 1 for our run-time / accuracy trade-off relative to other methods.)

To summarize, our contributions are:

*Performed this work while an intern at Facebook.

- A new hypernetwork architecture that employs a U-Net within a U-Net.
- Novel dynamic patch-wise convolution with weights that vary both per input and per spatial location.
- SotA accuracy vs. runtime trade-off on the major benchmarks of the field.

2. Related work

Hypernetworks. Hypernetwork [13], are networks that generate weight values for other networks (often referred to as *primary networks*). Hypernetworks are useful as a modeling tool, e.g., as implicit functions for image-to-image translation [9, 24], 3D scene representation [26, 42], and also for avoiding compute and data heavy training cycles during neural architecture search (NAS) [55] and continual learning [48]. To our knowledge, however, hypernetworks were never proposed for semantic segmentation, as we propose to do here.

Locally connected layers. Connectivity in locally connected layers follows a spatial pattern that is similar to conventional convolutional layers but without weight sharing. Such layers played an important role in the early days of deep learning, mostly due to computational reasons [10, 36, 47].

Locally connected layers were introduced as an accuracy enhancing component in the context of face recognition, where these were motivated by the need to model each part of the face in a different way [43]. However, subsequent face recognition methods tend to use conventional convolutions, e.g., [41]. Partial sharing of weights, in which convolutions are shared within image patches, was proposed for the analysis of facial actions [58].

As far as we can ascertain, we are the first to propose locally connected layers in combination with hypernetworks or in the context of semantic segmentation or, more generally, in image-to-image mapping.

Semantic segmentation. Early semantic segmentation methods used feature engineering and often relied on data driven approaches [15, 16, 17, 46]. To our knowledge, Long et al. [28] were the first to show end-to-end training of convolutional neural networks (CNN) for semantic segmentation. Their fully convolutional network (FCN) output dense, per-pixel predictions of variable resolutions, based on a classification network backbone. They incorporated skip connections between the early and final layers, for combining coarse and fine information. Subsequent methods added a post processing step based on conditional random fields (CRF) to further refine the segmentation masks [3, 4, 59]. Nirkin et al. overcame limited, scarce segmentation labels by utilizing motion from videos [31]. U-Nets [38] used encoder-decoder pairs, concatenating the last feature maps of the encoder, in each stride, with corresponding upsampled feature maps from the decoders.

Some proposed replacing strided convolutions with dilated convolutions, a.k.a. atrous convolutions [4, 54]. This approach produced more detailed segmentations by enlarging the receptive field of the logits but also drastically increased computational costs. Another approach for expanding the receptive field is called spatial pyramid pooling (SPP) [18, 57], in which features from different strides are average pooled and concatenated together, after which the information is fused by subsequent convolution layers. Subsequent work combined atrous convolutions with SPP (ASPP), achieving improved accuracy, yet with even higher computational cost [4, 5, 6]. To further improve accuracy, some proposed inference strategies of applying networks multiple times on multi-scale and horizontally flipped versions of the input image, and combining the results using average pooling [5, 6].

Recently, Tao et al. [45] utilized attention to better combine the inference strategy predictions, taking scale into account. Finally, others proposed axial attention, performing attention along the height and width axes separately, to better model long range dependencies [20, 49] .

Real-time segmentation. The goal of some methods is to achieve the best trade-off between accuracy and computations, with an emphasis on maintaining real-time performance. Real-time methods typically adopt an architecture composed of an encoder based on an efficient backbone and a relatively small decoder.

One early example of inexpensive semantic segmentation is the SegNet [1], which uses an encoder-decoder architecture with skip connections and transposed convolutions for upsampling. ENet [34] proposed an architecture based on the ResNet's bottleneck block [19], achieving a high frames per second (FPS) rate but sacrificing considerable accuracy. ICNet [56] utilizes fused features extracted from image pyramids, reporting better accuracy than previous methods. GUNet [29] performed upsampling guided by the fused feature maps from the encoder, extracted from multi-scale input images. SwiftNet [32] suggested an encoder-decoder with SPP and 1×1 convolutions for reducing the number of dimensions before each skip connection.

Subsequent methods benefited from progress in efficient network architecture [25, 51], such as depth-wise separable convolutions [7, 21] and inverted residual blocks [40] which we also use in our work. BiSeNet [52] proposed an additional, coarser downsampling path that is fused with the finer resolution main network before upsampling. BiSeNetV2 [51] extends BiSeNet by offering a more elaborate fusion of the two branches and additional prediction heads from intermediate layers for boosting the training. Finally, TDNet [22] proposed a network for video semantic segmentation, by circularly distributing sub-networks over

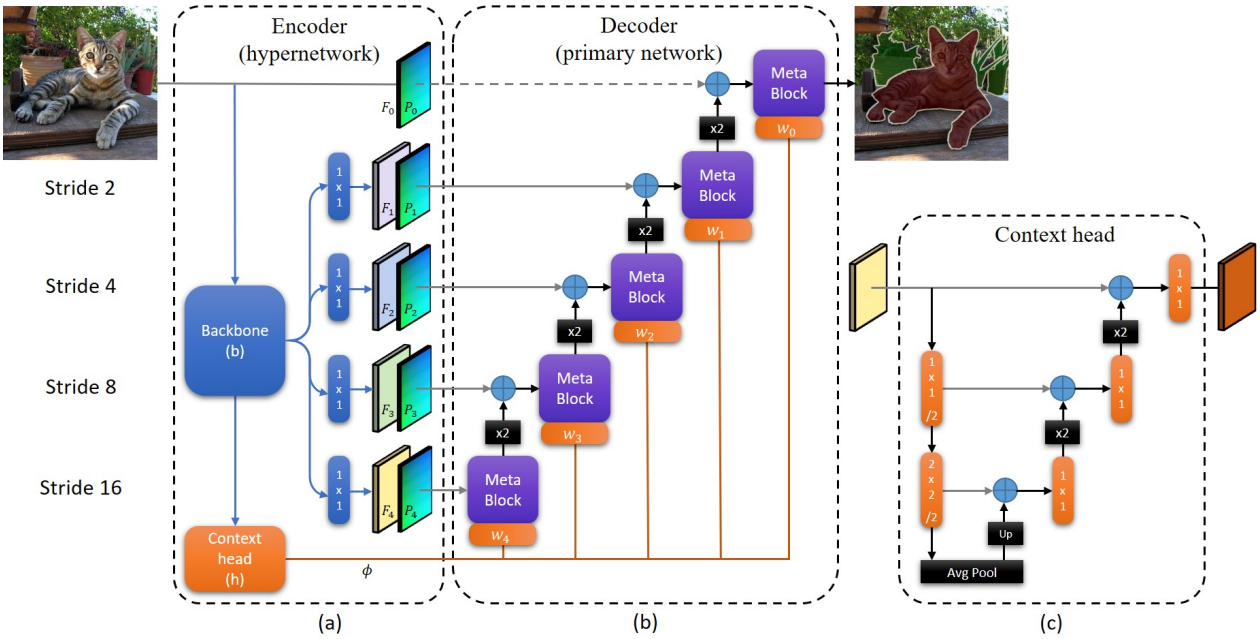

Figure 2. *Method overview.* (a) The hypernetwork encoder based on an EfficientNet [44] backbone, b, with its final layer replaced by the context head, h. (b) The primary network decoder, d, and layers w_i of the weight mapping network, w, embedded in each meta block. The input to the decoder, d, are the input image and the features, F_i, concatenated with positional embedding, P_i. Its weights are determined *dynamically* for each patch in the image. Gray arrows represent skip connections, $\times 2$ blocks are bilinear upsampling, and the blue '+' signs are concatenations. (c) The context head is designed as a nested U-Net. Please see Sec. 3 for more details.

sequential frames, leveraging temporal continuity.

3. Method

Overview. Our proposed hypernetwork encoder-decoder approach is illustrated in Fig. 2 and Fig. 3. Similarly to U-Net–based methods [38], we employ skip connections between corresponding layers of the encoder and the decoder. Our network, however, uses encoder and subsequent blocks which we refer to as the *context head* and *weight mapper*, in the spirit of hypernetwork design. Skip connections, therefore, connect different encoder levels with the levels of a hierarchical primary network, which serves as our decoder. Moreover, our decoder weights vary between patches at each stride level.

Our proposed model involves three sub-networks: the *backbone*, b (shown in blue, in Fig. 2(a), the *context head*, h (orange box in Fig. 2(a), also detailed in Fig. 2(c)), and the primary network, acting as the decoder, d (Fig. 2(b)). In addition, the decoder consists of multiple *meta blocks*, visualized in detail in Fig. 3(a). Each meta block, $i = 0 \dots n$, includes an additional weight mapping network component, w_i, represented as orange boxes in Fig. 2(b).

Information flow. The weights of the three networks: θ^b, θ^h, and θ^w, are fixed during inference and learned during the training process, while θ^{m_i}, the weights of the decoder meta block, m_i, are predicted dynamically at inference time. The encoder's backbone, b, maps an input image, $I \in \mathbb{R}^{3\times H\times W}$, to a set of feature maps, $F_i \in \mathbb{R}^{C_i\times \frac{H}{2^i}\times \frac{W}{2^i}}, i \in [1,5]$, of different resolutions, where H and W are the number of pixels along the height and width of the image correspondingly.

The context head, $h : \mathbb{R}^{C_n\times \frac{H}{2^n}\times \frac{W}{2^n}} \rightarrow \mathbb{R}^{C_n\times \frac{H}{2^n}\times \frac{W}{2^n}}$, maps the last feature map from b to a signal, ϕ. This signal is then fed to $w : \mathbb{R}^{C_n\times \frac{H}{2^n}\times \frac{W}{2^n}} \rightarrow \mathbb{R}^{(\sum_i |\theta^{m_i}|)\times \frac{H}{2^n}\times \frac{W}{2^n}}$ which generates the weights for meta blocks of the primary network, d. Note that these weights vary from one spatial location to the next. We define a fixed positional encoding, $P^{H,W} \in \mathbb{R}^{2\times H\times W}$, such that in each position (i,j), $P^{H,W}_{i,j} = (\frac{2i-H+1}{H-1}, \frac{2j-W+1}{W-1}), i \in [0,H), j \in [0,W)$.

Finally, given the input image and the feature maps, $F_1, \dots, F_n$, their corresponding positional encodings of the same resolutions, $P_0, \dots, P_n$, and the weights, θ^d, the decoder, d, outputs the segmentation prediction, $S \in \mathbb{R}^{C\times H\times W}$, where C is the number of classes in the semantic segmentation task.

Our entire network is, therefore, defined by the following set of equations:

$$F_1, \dots, F_n = b(I|\theta^b), \tag{1}$$

$$\phi = h(F_n|\theta^h), \tag{2}$$

$$\theta^{m_i} = w_i(\phi|\theta^w), i = 0, \dots, n, \tag{3}$$

$$S = d(\{F_i\}, \{P_i\}|\{\theta^{m_i}\}), \tag{4}$$

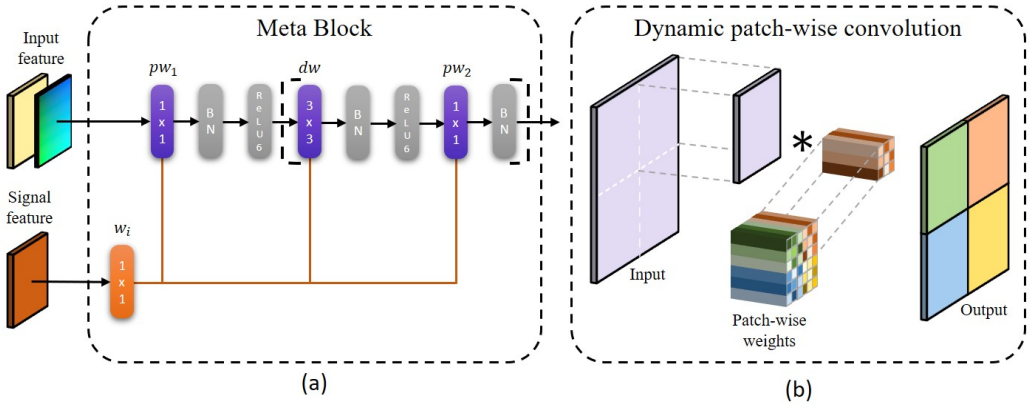

Figure 3. (a) *The meta block based on the inverted residual block* [40]. Each purple layer represents a dynamic patch-wise convolution with weights generated by the orange layer, w_i. (b) *Visualization of the dynamic patch-wise convolution operation.* Each color represents weights corresponding to a specific patch and '*' is the convolution operation. Please see Sec. 3.3 for more details.

where the weights of each network are specified explicitly after the separator sign.

3.1. The encoder and the hypernetwork

The first component of the hypernetwork is the backbone network, b (blue box in Fig. 2(a)). It is based on the EfficientNet [44] family of models. Sec. 4 provides details of this network. In our work, the head of the backbone architecture is replaced with our context head, h. The backbone outputs the feature map, F_i, in each stride. In order to decrease the size of the decoder, we augment it with additional 1×1 convolutions that reduce the number of channels of each F_i by a factor r_i. The exact values for r_i are detailed in Sec. B in the supplemental material.

The last feature map of b, the backbone network, is of size $\frac{H}{2^n} \times \frac{W}{2^n}$. Each pixel in this feature map encodes a patch in the input image. These patches have little overlap, and the limited receptive field can lead to poor results in case of large objects that span multiple patches. The context head, h, therefore, combines information from multiple patches.

We detail the structure of h in Fig. 2(c). Network h uses the nested U-Net structure introduced by Xuebin et al. [35]. In our implementation, we employ 2×2 convolutions with a stride of two, that output half the number of channels of the input. Such convolutions are computationally cheaper than 3×3 convolutions, which require padding for the low resolution feature maps processed by h, and this padding can significantly increase the spatial resolution. The bottom-most feature map is average pooled to extract the highest level context, and then upsampled to the previous resolution using nearest neighbor interpolation. Finally, in the upsampling path of h, at each level, we concatenate the feature map with its corresponding upsampled feature map, followed with a fully connected layer.

While the weight mapping network, $w = [w_0, \ldots, w_n]$, is a key part of our hypernetwork, it is more efficient in our hierarchical network to divide w into parts and attach these parts to primary network blocks (Fig. 2(b)). Hence, instead of directly following h, the layers, $w_0, \ldots, w_n$, of the weight mapping network are embedded in each of the meta blocks of d. The rationale is that the mapping from context to weights incurs a large expansion in memory, which can become a performance bottleneck. Instead, the weights are generated right before they are consumed, minimizing the maximum memory consumption and better utilizing the memory cache. Each w_i is a 1×1 convolution with channel groups, g_{w_i}, and is detailed next.

3.2. The decoder (the primary network)

The decoder, d, shown in Fig. 2(b), consists of $n + 1$ meta blocks, $m_0, \ldots, m_n$, illustrated in Fig. 3(a). Block, m_0, corresponds to the input image, and each of the blocks, m_i, $i = 1..n$, corresponds to the feature map, F_i, of the encoder. Each block is followed by bilinear upsampling and concatenation with the next finer resolution feature map.

Unlike conventional schemes, by employing a hypernetwork, the weights of the decoder, d, depend on the input image. Moreover, the weights of d are not only conditioned on the input image but also *vary between different regions of the image*. With this approach, we can efficiently combine low level information from the stem of the network with high level information from the bottom layers. This allows our method to achieve *higher accuracy using a much smaller decoder*, thus enabling real-time performance. The hypernetwork can be seen as a type of attention, and similar to some attention-based methods [20, 33], d benefits from knowing the position information of the pixels. For this reason, we augment the input image and the encoder's feature maps with additional positional encoding.

The design of $m_0, \ldots, m_n$ is based on the inverted residual block of MobileNetV2 [40]: a point-wise convolution, pw_1, followed by depth-wise convolution, dw, and another point-wise convolution, pw_2, without an activation function. Instead of regular convolutions, our network employs

Dataset	Batch	lr_0	p	t
PASCAL VOC 2012 [11]	32	10^{-4}	3.0	3.2M
Cityscapes [8]	16	10^{-3}	0.9	1.4M
CamVid [2]	16	10^{-3}	2.0	240k

Table 1. *Training hyperparameters for each benchmark.*

Method	Backbone	mIoU	GFLOPs	Params
Auto-DeepLab-L [27]	-	73.6	79.3*	44.42
DeepLabV3 [5]	ResNet-101	78.5	249.2*	58.6*
DFN [53]	ResNet-101	79.7	-	-
SDN [12]	DenseNet-161	79.9	-	238.5
DeepLabV3+ [6]	Xception-71	80.0	177	43.48
HyperSeg-L	EfficientNet-B3	**80.6**	**8.21**	**39.6**

Table 2. *Results on the PASCAL VOC 2012, val. set [11].* '$\star$', represents metrics that were computed by us using open source (listed in Sec. D in the supplemental material).

dynamic, patch-wise convolutions, described in the next section. For very small patches – smaller than 4×4 in our large model; smaller than 8×8 in our smaller models – the meta block includes only pw_1. The total meta parameters, $\theta^{m_i} \in \mathbb{R}^{(|\theta^{pw_1}|+|\theta^{dw}|+|\theta^{pw_1}|)\times\frac{H}{2^n}\times\frac{W}{2^n}}$, required by each m_i is the combined meta parameters of all dynamic convolutions in m_i: $\theta^{m_i} = \theta^{pw_1} \cup \theta^{dw} \cup \theta^{pw_2}$. The weights, θ^{m_i}, are generated by the w_i layer, embedded in m_i, given the signal, $\phi_i \in \mathbb{R}^{C_{\phi_i}\times\frac{H}{2^n}\times\frac{W}{2^n}}$. At inference, the batch-normalization layers of m_i are fused with w_i; more details are provided in Sec. E in the supplemental material.

Employing the full signal in each m_i is inefficient both computationally and in the number of trainable parameters, because ϕ is directly mapped into a large number of weights. We thus instead divide the channels of ϕ into parts, $C_{\phi_0}, \ldots, C_{\phi_n}$, which are relative in size to the required number of weights of each meta block. The division of the channels is defined using the following procedure:

$$C_{\phi_0}, \ldots, C_{\phi_n} = \text{divide\_channels}(\\ C_n, max(g_{w_0}, \ldots, g_{w_n}), |\theta^{m_0}|, \ldots, |\theta^{m_n}|), \quad (5)$$

where divide_channels$(\cdot)$ is detailed in Sec. A in the supplemental material. This routine ensures that each part is proportional to its allocated signal channels, is divisible by $max(g_{w_0}, \ldots, g_{w_n})$ for the grouped convolutions in w, and is allocated a minimal number of channels.

The number of groups, g_{w_i}, is an important hyperparameter, since it controls the amount of computations and trainable parameters invested in producing the weights for m_i. Increasing g_{w_i} reduces the computations and trainable parameters in direct proportions, as can be seen from the following equations:

$$|\theta^{w_i}| = \frac{|\theta^{m_i}| \cdot C_{\phi_i}}{g_{w_i}}, \quad (6)$$

$$\text{FLOPs}_{w_i} = \frac{|\theta^{m_i}| \cdot C_{\phi_i} \cdot \frac{H}{2^n} \cdot \frac{W}{2^n}}{g_{w_i}}. \quad (7)$$

In the supplemental material, we study the effect of different values of g_{w_i} (Sec. C), and report the exact values of g_{w_i} used in our tests (Sec. B).

3.3. Dynamic patch-wise convolution

We illustrate the operation of the dynamic patch-wise convolution (DPWConv), the layers, pw_1, dw, and pw_2 of m_i, in Fig. 3(b). Given an input feature map, $X \in \mathbb{R}^{C_{in}\times H\times W}$, and a grid of weights, $\theta \in \mathbb{R}^{C_{out}\times\frac{C_{in}}{G}\times K_h\times K_w\times N_h\times N_w}$, where C_{in} and C_{out} are the channel numbers for the input and output, G is the number of channel groups, H and W are the input's height and width, K_h and K_w are the height and width of the kernel, and N_h and N_w are the number of patches along the height and width axes, we define output patches as follows:

$$O_{i,j} = X_{i,j} * \theta_{i,j}, \quad (8)$$

where $*$ is the convolution operation, $i \in [0, N_h)$ and $j \in [0, N_w)$ are the patch indices, $X_{i,j}$ is a patch of X in the grid location (i, j), and $\theta_{i,j}$ are the corresponding weights from the weights grid. We first apply padding to the entire input feature map, X, and then at each patch, $X_{i,j}$, we wrap the adjacent pixels from neighboring patches.

4. Experimental results

We experiment on three popular benchmarks: PASCAL VOC 2012 [11], Cityscapes [8], and CamVid [2]. We report results using the following standard measures: class mean intersection over union (mIoU), frames per second (FPS), billion floating point operations (GFLOP), and number of trainable parameters.

FPS is measured using established protocols [32]: We record FPS for the elapsed time between data upload to GPU through to prediction download. Our model is implemented in PyTorch without specific optimizations. Finally, we use a batch size of 1 to simulate real-time inference. Similar to most previous methods, we measure FPS on a NVIDIA GeForce GTX 1080TI GPU (i7-5820k CPU and 32GB DDR4 RAM). GFLOPs and trainable parameters are calculated using the pytorch-OpCounter library [60], also used by others [32].

We experiment with large, medium, and small variants of our model, *HyperSeg-L*, *HyperSeg-M*, and *HyperSeg-S*, respectively. The models share the same template and are named according to their size as reflected by their parameter numbers. Both HyperSeg-M and HyperSeg-S omit the

Method	Backbone	Resolution	mIoU (%) val	mIoU (%) test	FPS	GFLOPs	Params (M)
ERFNet [37]	-	1024 × 512	-	69.7	41.7	21.7⋆	2.0⋆
ESPNet [30]	ESPNet	1024 × 512	-	60.3	112.9	-	-
ESPNetV2 [30]	ESPNetV2	1024 × 512	66.4	66.2	61.9⋆	2.7	**1.3**⋆
ICNet [56]	PSPNet50	2048 × 1024	-	69.5	30.3	-	-
GUNet [29]	DRN-D-22	1024 × 512	69.6	70.4	33.3	-	-
DFANet A' [25]	Xception A	1024 × 512	-	70.3	**160.0**	**1.7**	7.8
DFANet A [25]	Xception A	1024 × 1024	-	71.3	100.0	3.4	7.8
SwiftNetRN-18 [32]	ResNet18	2048 × 1024	75.4	75.5	39.9	104.0	11.8
BiSeNetV1 [52]	ResNet18	1536 × 768	74.8	74.7	65.5	75.2⋆	49.0
BiSeNetV2 [51]	-	1024 × 512	73.4	72.6	156.0	21.2	-
BiSeNetV2-L [51]	-	1024 × 512	75.8[1]	75.3	47.3	118.5	-
TD4-Bise18 [22]	BiseNet18	2048 × 1024	75.0	74.9	47.6	-	-
HyperSeg-M	EfficientNet-B1	1024 × 512	76.2	75.8	36.9	7.5	10.1
HyperSeg-S	EfficientNet-B1	1536 × 768	**78.2**	**78.1**	16.1	17.0	10.2

Table 3. *Real-time semantic segmentation results on Cityscapes [8].* '-' Implies that the metric was not reported. '⋆', denotes that the specific metric was computed by us using available open source (listed in Sec. D in the supplemental material). [1]Reported using horizontal mirroring and multi-scale (confirmed by open source).

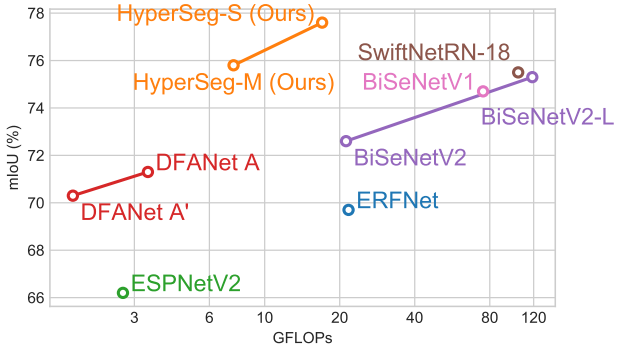

Figure 4. *GFLOPs vs. accuracy trade-off on Cityscapes [8]. Our models (in orange) attain a significantly better trade-off than previous methods.*

finest resolution level of d; we bilinearly upsample predictions to the input resolution from their previous level. In HyperSeg-S the channels of the layers in m_i are halved, relative to those of the largest model. We provide model backbone and resolution details, separately, for each experiment. For other hyperparameter values, see Sec. B in the supplemental material.

4.1. Training details

We initialize our network using weights pretrained on ImageNet [39] for θ^b, and using random values sampled from the normal distribution for θ^h and θ^w. The Adam optimizer [23] was used for training, with $\beta_1 = 0.5$ and $\beta_2 = 0.999$. Following others [5, 6], we use a polynomial learning rate scheduling that decays the initial learning rate, lr_0, after i iterations by a factor of $(1 - \frac{i}{t})^p$, where t is the total number of iterations and p is a scalar constant. The exact values for each dataset are listed in Tab. 1.

For the Cityscapes and Camvid benchmarks, we apply the following image augmentations: random resize with scale range $[0.5, 2.0]$, crop, and horizontal flipping with probability 0.5. For PASCAL VOC we use a similar horizontal flip, and random resize with scale range of $[0.25, 1.0]$. We further randomly rotate the images, in the range of $-30°$ to $30°$, jitter colors to manipulate brightness, contrast, saturation, and hue, and finally pad the images to a resolution of 512×512. We train all our models on two Volta V100 32GB GPUs.

4.2. PASCAL VOC 2012 benchmark tests

PASCAL VOC 2012 [11] contains images of varying resolutions, up to 500×500, representing 21 classes (including a class for the background). This set originally contained 1,464 training, 1,449 validation, and 1,456 test images. Its training set was later extended by others to a total of 10,582 images [14]. This set is not typically used to evaluate real-time segmentation methods but the low resolution images allow for quick experimentation. For this reason, we chose this benchmark for our initial tests.

Tab. 2 reports accuracy, FLOPs, and number of trainable parameters for our model and those of existing work. We chose methods that reported results on the PASCAL VOC validation set, without inference strategies (e.g., without horizontal mirroring and multi-scale testing). Besides increasing inference times by several factors, these techniques can blur the contributions of the underlying methods. As evident from the results, compared with previous work, our methods achieve the best mIoU, with lower GFLOPs and a

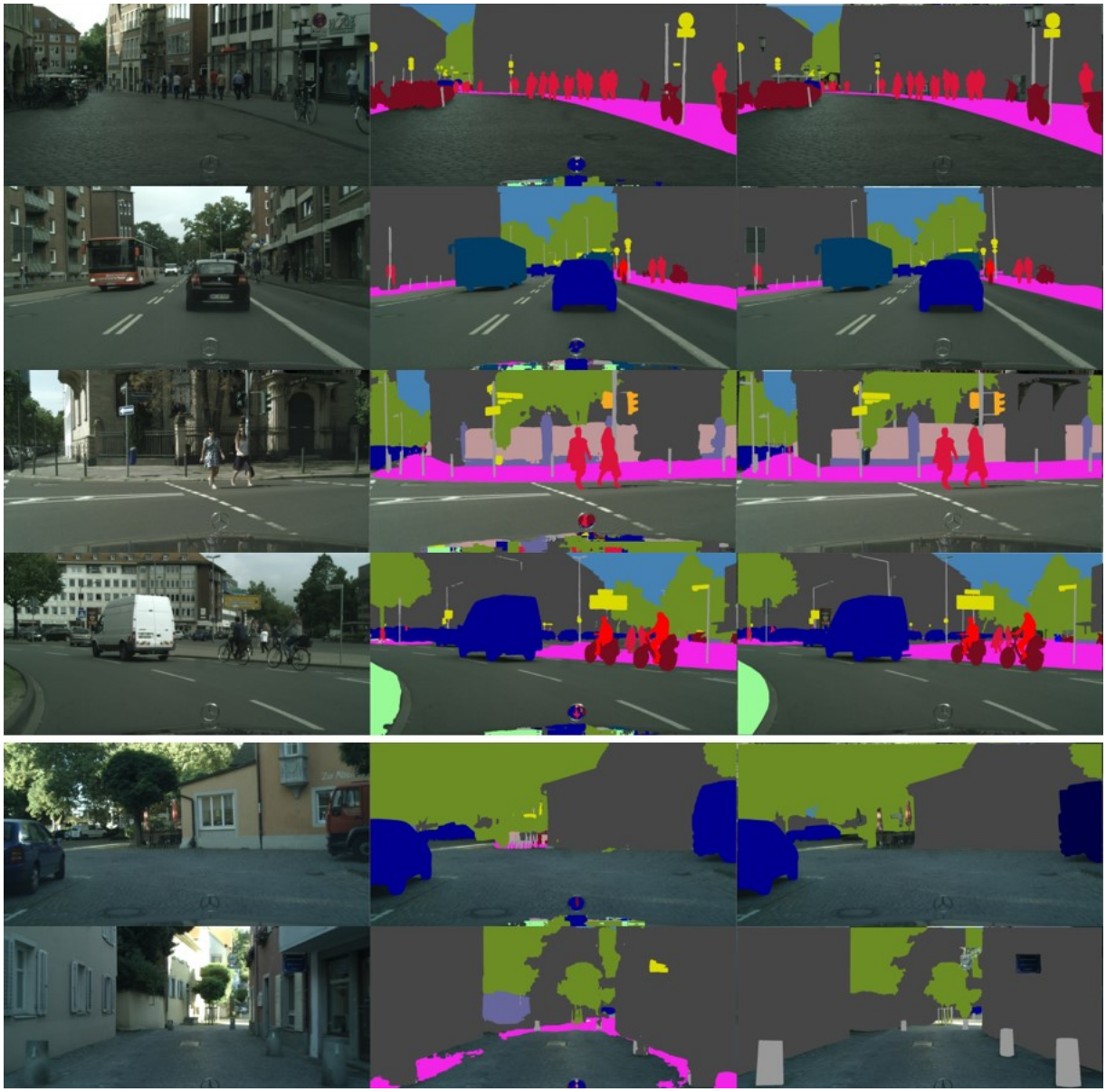

Figure 5. *Qualitative results on Cityscapes [8] validation set images.* Left to right: input, our result, and ground truth. The first four rows showcase our model's performance in diverse scenes. The last two rows provide sample failures. Please note that the reflective car hood region is ignored in evaluation.

smaller number of trainable parameters.

4.3. Cityscapes benchmark tests

Cityscapes [8] provides 5k images of urban street scenes, labeled with 19 classes. Image resolutions are 2048×1024 pixels, and are typically downsampled or cropped during training time. The images are partitioned into 2,975 training, 500 validation, and 1,525 test images.

Tab. 3 compares variants of our approach using the EfficientNet-B1 backbone [44] operating on different resolutions, with previous methods. We only show previous work considered to be *fast*: Methods that run at 10 FPS or faster and report test set mIoU. Our models achieve the best accuracy on both validation and test sets, as well as the best trade-off between accuracy and run-time performance. The trade-off comparison can be best seen in Fig. 1.

Importantly, our model incurs a large run-time penalty, due to the unoptimized operations of DPWConv. Fig. 4 shows the trade-off between GFLOPs and accuracy of our method relative to previous work. Evidently, our method achieves a significantly better trade-off than other methods. While GFLOPs does not directly correlate with FPS, it does suggest the potential run-time performance, once all our functions are optimized.

Fig. 5 provides qualitative results of our HyperSeg-S model on Cityscapes validation images. Our model produces high quality segmentations without any apparent artifacts, due to the partitioning of images into patches. The last two rows of Fig. 5 offer sample failure cases. In the second row from the bottom, our model confuses a truck with a car. In the last row, our model fails to segment the poles and mistakenly labels pixels as wall or sidewalk.

4.4. CamVid benchmark tests

CamVid offers 701 images of driving scenes, similar to those of Cityscapes, labeled for 11 classes [2]. All images share the same resolution, 960×720. The images are partitioned into 367 training, 101 validation, and 233 test images. Following the training protocol used by all of our baselines, we train on both the training and validation sets.

Tab. 4 compares our approach with previous real-time methods pretrained on ImageNet [39]. For a fair comparison, we exclude methods that use additional outside data, other than ImageNet and CamVid. We test two variations of our model, both using the EfficientNet-B1 backbone [44]. HyperSeg-S operates on resolutions of 768×576 and HyperSeg-L on 1024×768. Both models achieve SotA mIoU by a margin relative to that reported by the previous SotA, with HyperSeg-S running at 38 FPS.

Even without outside data, our method outperforms SotA results reported by methods that use Cityscapes as additional training data: The best results using Cityscapes was reported by BiSeNetV2-L [51], which improves its performance from 73.2% mIoU, when trained without additional data, to 78.5% with this data. This is still lower, by a margin, than our method: 79.1% with 16.6FPS. In fact, their result is almost identical to our 38FPS network. Both variants of our method do not use any additional training data.

Ablation Study. We performed ablation studies on the CamVid dataset [2], to show the contribution of our meta-learning approach and the effect of using different backbones. The results are reported in Tab. 4.

In the first six middle experiments, for each model configuration we replace the EfficientNet-B1 backbone with different backbones: ResNet18, PSPNet18, and PSPNet50. We have explicitly chosen backbones that were used by previous methods. In our implementation, a fully connected layer transforms the last feature map before it is fed to the context head, to 1280 channels for the ResNet18 and PSPNet18 backbones, and 2048 channels for the PSPNet50 backbone. In the PSPNet backbones we do not use dilations in any of the convolutions. In the experiments marked with "w/o DPWConv", we replace all the dynamic patch-wise convolutions with regular convolutions, effectively eliminating all the meta-learning elements of our method.

The results clearly show that the EfficientNet-B1 backbone is superior to the ResNet18, PSPNet18, and PSPNet50 backbones, yet *our method still outperforms previous methods with those backbones*. Finally, removing meta-learning from our method causes a 1.1% drop in accuracy for the HyperSeg-S configuration, and a reduction of 0.7% in accuracy for the HyperSeg-L configuration, with only a slight improvement in FPS, *showing that meta-learning is an integral part of our method*.

Method	Backbone	mIoU (%)	FPS	Params (M)
ICNet [56]	PSPNet50	67.1	27.8	-
DFANet A [25]	Xception A	64.7	120.0	**7.8**
SwiftNetRN-18 [32]	ResNet18	72.6	85.8*	11.8
BiSeNetV1 [52]	ResNet18	68.7	116.2	49.0
BiSeNetV2 [51]	-	72.4	**124.5**	-
BiSeNetV2-L [51]	-	73.2	32.7	-
TD4-PSP18 [22]	PSPNet18	72.6	25.0	-
TD2-PSP50 [22]	PSPNet50	76.0	11.1	-
HyperSeg-S	ResNet18	77.0	32.5	16.2
HyperSeg-L	ResNet18	77.1	11.5	16.7
HyperSeg-S	PSPNet18	76.6	31.3	17.2
HyperSeg-L	PSPNet18	77.5	11.4	17.6
HyperSeg-S	PSPNet50	77.1	9.3	57.6
HyperSeg-L	PSPNet50	77.9	2.2	67.9
HyperSeg-S w/o DPWConv	EfficientNet-B1	77.3	45.5	9.9
HyperSeg-L w/o DPWConv	EfficientNet-B1	78.4	21.6	10.3
HyperSeg-S	EfficientNet-B1	78.4	38.0	9.9
HyperSeg-L	EfficientNet-B1	**79.1**	16.6	10.2

Table 4. *Real-time semantic segmentation results on CamVid [2]* (test set; no outside data). Top: previous methods. Middle: ablation study. The first six rows are variants of our models with different backbones. In comparison to the baselines, we improve accuracy and increase runtime. In the "w/o DPWConv" variants, we replace the dynamic depth-wise convolutions with regular convolutions. Those variants achieve lower accuracy compared to our full method. Bottom: Our full method. *Computed by us using open source.

5. Conclusions

We propose to marry autoencoders with hypernetworks for the task of semantic segmentation. In our scheme, the hypernetowrk is a composition of three networks: the backbone of the semantic segmentation encoder, b, a context head, h, in the form of an internal U-Net, and multiple weight mapping heads, w_i. The decoder is a multi-block decoder, where each block, d_i, implements locally connected layers. The outcome is a new type of U-Net that is able to dynamically and locally adapt to the input, thus holding the potential to better tailor the segmentation process to the input image. As our experiments show, our method outperforms the SotA methods, in this very competitive field, across multiple benchmarks.

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture

for image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(12):2481–2495, 2017. 2

[2] Gabriel J Brostow, Julien Fauqueur, and Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97, 2009. 1, 5, 8

[3] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014. 2

[4] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(4):834–848, 2017. 2

[5] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 1, 2, 5, 6

[6] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Eur. Conf. Comput. Vis.*, pages 801–818, 2018. 1, 2, 5, 6

[7] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1251–1258, 2017. 2

[8] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3213–3223, 2016. 1, 5, 6, 7

[9] Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic filter networks. *arXiv preprint arXiv:1605.09673*, 2016. 2

[10] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. In *Adv. Neural Inform. Process. Syst.*, pages 1223–1231, 2012. 2

[11] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vis.*, 88(2):303–338, 2010. 1, 5, 6

[12] Jun Fu, Jing Liu, Yuhang Wang, Jin Zhou, Changyong Wang, and Hanqing Lu. Stacked deconvolutional network for semantic segmentation. *IEEE Trans. Image Process.*, 2019. 5

[13] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. 1, 2

[14] Bharath Hariharan, Pablo Arbeláez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *Int. Conf. Comput. Vis.*, pages 991–998. IEEE, 2011. 6

[15] Tal Hassner, Shay Filosof, Viki Mayzels, and Lihi Zelnik-Manor. Sifting through scales. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(7):1431–1443, 2016. 2

[16] Tal Hassner and Ce Liu. *Dense Image Correspondences for Computer Vision*. Springer, 2016. 2

[17] Tal Hassner, Viki Mayzels, and Lihi Zelnik-Manor. On sifts and their scales. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1522–1528. IEEE, 2012. 2

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(9):1904–1916, 2015. 2

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 770–778, 2016. 2

[20] Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019. 1, 2, 4

[21] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 2

[22] Ping Hu, Fabian Caba, Oliver Wang, Zhe Lin, Stan Sclaroff, and Federico Perazzi. Temporally distributed networks for fast video semantic segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 8818–8827, 2020. 2, 6, 8

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[24] Sylwester Klocek, Łukasz Maziarka, Maciej Wołczyk, Jacek Tabor, Jakub Nowak, and Marek Śmieja. Hypernetwork functional image representation. In *Int. Conf. on Artificial Neural Networks*, pages 496–510. Springer, 2019. 2

[25] Hanchao Li, Pengfei Xiong, Haoqiang Fan, and Jian Sun. Dfanet: Deep feature aggregation for real-time semantic segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 9522–9531, 2019. 2, 6, 8

[26] Gidi Littwin and Lior Wolf. Deep meta functionals for shape representation. In *Int. Conf. Comput. Vis.*, pages 1824–1833, 2019. 1, 2

[27] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 82–92, 2019. 5

[28] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3431–3440, 2015. 2

[29] Davide Mazzini. Guided upsampling network for real-time semantic segmentation. *arXiv preprint arXiv:1807.07466*, 2018. 2, 6

[30] Sachin Mehta, Mohammad Rastegari, Anat Caspi, Linda Shapiro, and Hannaneh Hajishirzi. Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. In *Proceedings of the european conference on computer vision (ECCV)*, pages 552–568, 2018. 6

[31] Yuval Nirkin, Iacopo Masi, Anh Tran Tuan, Tal Hassner, and Gerard Medioni. On face segmentation, face swapping, and face perception. pages 98–105. IEEE, 2018. 2

[32] Marin Orsic, Ivan Kreso, Petra Bevandic, and Sinisa Segvic. In defense of pre-trained imagenet architectures for real-time semantic segmentation of road-driving images. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 12607–12616, 2019. 2, 5, 6, 8

[33] Niki Parmar, Prajit Ramachandran, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. In *Adv. Neural Inform. Process. Syst.*, pages 68–80, 2019. 4

[34] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016. 2

[35] Xuebin Qin, Zichen Zhang, Chenyang Huang, Masood Dehghan, Osmar R Zaiane, and Martin Jagersand. U2-net: Going deeper with nested u-structure for salient object detection. *Pattern Recognition*, 106:107404, 2020. 4

[36] Rajat Raina, Anand Madhavan, and Andrew Y Ng. Large-scale deep unsupervised learning using graphics processors. In *Int. Conf. Machine. Learning.*, pages 873–880, 2009. 2

[37] Eduardo Romera, José M Alvarez, Luis M Bergasa, and Roberto Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):263–272, 2017. 6

[38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Int. Conf. on Medical image comput. and comput. assisted intervention*, pages 234–241. Springer, 2015. 2, 3

[39] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. 6, 8

[40] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4510–4520, 2018. 2, 4

[41] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 815–823, 2015. 2

[42] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Adv. Neural Inform. Process. Syst.*, 2020. 1, 2

[43] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1701–1708, 2014. 2

[44] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. 3, 4, 7, 8

[45] Andrew Tao, Karan Sapra, and Bryan Catanzaro. Hierarchical multi-scale attention for semantic segmentation. *arXiv preprint arXiv:2005.10821*, 2020. 1, 2

[46] Moria Tau and Tal Hassner. Dense correspondences across scenes and scales. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(5):875–888, 2015. 2

[47] Rafael Uetz and Sven Behnke. Large-scale object recognition with cuda-accelerated hierarchical neural networks. In *Int. conf. on intelligent comput. and intelligent sys.*, volume 1, pages 536–541. IEEE, 2009. 2

[48] Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F Grewe. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019. 2

[49] Huiyu Wang, Yukun Zhu, Bradley Green, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. Axial-deeplab: Stand-alone axial-attention for panoptic segmentation. *arXiv preprint arXiv:2003.07853*, 2020. 1, 2

[50] Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*, 2019. 1

[51] Changqian Yu, Changxin Gao, Jingbo Wang, Gang Yu, Chunhua Shen, and Nong Sang. Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation. *arXiv preprint arXiv:2004.02147*, 2020. 2, 6, 8

[52] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *Eur. Conf. Comput. Vis.*, pages 325–341, 2018. 2, 6, 8

[53] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Learning a discriminative feature network for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1857–1866, 2018. 5

[54] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 2

[55] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*, 2018. 2

[56] Hengshuang Zhao, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia. Icnet for real-time semantic segmentation on high-resolution images. In *Eur. Conf. Comput. Vis.*, pages 405–420, 2018. 2, 6, 8

[57] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 2881–2890, 2017. 2

[58] Kaili Zhao, Wen-Sheng Chu, and Honggang Zhang. Deep region and multi-label learning for facial action unit detection. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3391–3399, 2016. 2

[59] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *Int. Conf. Comput. Vis.*, pages 1529–1537, 2015. 2

[60] Ligeng Zhu. pytorch-OpCounter. Available online: https://github.com/Lyken17/pytorch-OpCounter. 5