Quadtree Generating Networks: Efficient Hierarchical Scene Parsing with Sparse Convolutions

Kashyap Chitta[1*] Jose M. Alvarez[2] Martial Hebert[3]
[1] Autonomous Vision Group, MPI for Intelligent Systems and University of Tübingen
[2] NVIDIA [3] The Robotics Institute, Carnegie Mellon University
kashyap.chitta@tue.mpg.de josea@nvidia.com hebert@cs.cmu.edu

Abstract

Semantic segmentation with Convolutional Neural Networks is a memory-intensive task due to the high spatial resolution of feature maps and output predictions. In this paper, we present Quadtree Generating Networks (QGNs), a novel approach able to drastically reduce the memory footprint of modern semantic segmentation networks. The key idea is to use quadtrees to represent the predictions and target segmentation masks instead of dense pixel grids. Our quadtree representation enables hierarchical processing of an input image, with the most computationally demanding layers only being used at regions in the image containing boundaries between classes. In addition, given a trained model, our representation enables flexible inference schemes to trade-off accuracy and computational cost, allowing the network to adapt in constrained situations such as embedded devices. We demonstrate the benefits of our approach on the Cityscapes, SUN-RGBD and ADE20k datasets. On Cityscapes, we obtain an relative 3% mIoU improvement compared to a dilated network with similar memory consumption; and only receive a 3% relative mIoU drop compared to a large dilated network, while reducing memory consumption by over 4×.

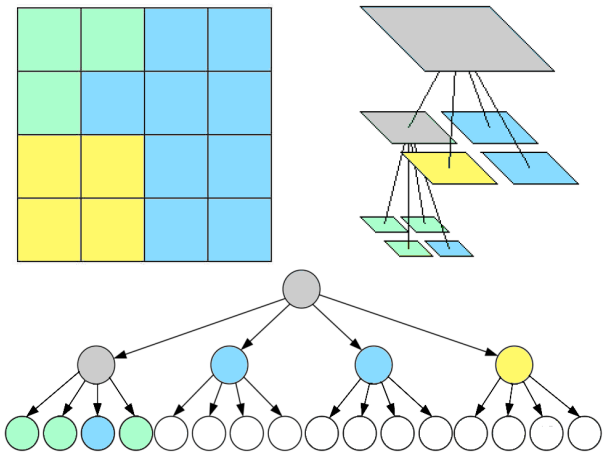

Figure 1. Efficient quadtree representation of a segmentation mask with three semantic classes (green, blue and yellow). Each node in the tree has four children, which represent quadrants of the corresponding image region in a clockwise order. The leaf nodes of the quadtree represent the pixel-level annotation. Some of the nodes in the quadtree (gray) are composite, and must be further split into individual classes; whereas others (white), whose parents in the quadtree structure are not composite, can be removed from the representation, achieving lossless compression (best viewed in color).

1. Introduction

Semantic segmentation is the problem of assigning a class to every pixel in an image. It is a challenging task as it requires a combination of coarse contextual reasoning and fine pixel-level accuracy [7, 16]. Fully Convolutional Networks (FCNs) [26] and their variants [4,28,34,46] have become a standard tool for this task, leveraging the representational power of deep Convolutional Neural Networks (CNNs) trained end-to-end with backpropagation. In contrast to traditional methods using low-level features [1], FCN-based approaches extract information at various scales using pooling and subsampling layers to perform contextual reasoning and obtain significant accuracy improvements. However, a common limitation of all these FCN-based methods is that they build upon convolutional layers, which require dense pixel grids at their input and output. Compute and memory requirements for dense pixel grids scale quadratically with the resolution of the input image, thus increasing the inference time required to process high-resolution images [33]. In addition, at train time, this large memory requirement limits the number of samples and spatial resolution used in the training batch. Due to this limitation, most recent architectures can fit just two cropped images per GPU while training [4, 46]. Existing tricks to circumvent this such as CPU/GPU activation swapping in-

*Work completed while author is at CMU and NVIDIA.

troduce significant overhead in training times.

In this paper, we propose a novel encoder-decoder based segmentation approach to reduce memory consumption in the most computationally demanding layers of a segmentation network. Our key idea is to decompose the target segmentation mask into a linear quadtree representation, as shown in Fig. 1. The use of quadtree representations enables us to replace several dense convolutional layers of the network's decoder with sparse convolutions [11,30] that are significantly more efficient than their dense counterparts, and scale linearly in terms of compute and memory requirements to higher resolutions. In our experiments, the quadtree representation of the high-resolution labels used consumes $20\times$ to $30\times$ less memory than the corresponding dense pixel-wise representation. Further, our proposed efficient and high-performance decoder enables the use of less computationally demanding backbone networks. Overall, our networks are $2\times$ to $4\times$ more efficient than strong baselines, with similar or better performance.

In short, our contributions are: (i) a novel approach to use quadtrees to represent semantic segmentation masks for deep neural networks; (ii) a novel architecture, Quadtree Generating Networks (QGNs), able to generate quadtrees from input images efficiently using sparse convolutional layers; and (iii) a loss function to train QGNs using high-resolution images. We demonstrate the benefits of our approach on three public semantic segmentation datasets. Our results consistently outperform existing networks with similar memory requirements and perform similarly to current state-of-the-art models while significantly reducing the memory and computational requirements. Moreover, at inference time, our approach is flexible, and allows a trade-off to be made between accuracy and computational requirements without any modification of a trained network.

2. Related Work

Scene Parsing. Semantic segmentation of complex scenes has seen considerable progress in recent years through FCN-based methods. Recent approaches fall under three broad categories: methods to improve contextual reasoning, methods to improve feature resolution, and methods that allow fast inference while maintaining reasonable performance. The first category of works propose modules, typically incorporated at the final layers of the encoder, to augment the feature representation and improve contextual reasoning. Though the receptive field of deep CNNs is theoretically large, studies have shown that their effective receptive field is small [27]. Atorus Spatial Pyramid Pooling (ASPP) in DeepLabv3 [4] and the Pyramid Pooling Module (PPM) in PSPNet [46] are two modules which apply kernels of various large scales to the feature map– ASPP employs dilated convolutions whereas PPM uses average pooling for capturing global context. Scale Adaptive Convolutions (SAC) [45] achieve a similar effect by modifying the convolutional operator to build larger receptive fields. More recently, PSANet [47], DANet [9] and CCNet [20] encode context using self-attention mechanisms [39,40].

The second category of work focuses on the performance at boundaries between classes and other fine details in images, which is directly tied to feature resolution. Typically, pre-trained classification architectures give feature maps that are 32 times smaller than the input resolution. Early approaches on improving this aspect of segmentation include the use of skip connections [26], deconvolutional architectures [2, 24, 28], or leveraging the features learned at all layers in the network to make predictions [14,29,34]. Since then, dilated convolutions have become the standard technique to handle the resolution issue, by maintaining a fairly high feature resolution throughout the network, while also increasing the receptive field as a subsampling or pooling layer would [43]. State-of-the-art segmentation networks based on the ResNet encoder architecture [15] typically dilate the last two residual blocks, giving output feature maps that are only 8 times smaller than the input resolution. However, given the design of ResNets, it becomes a huge computational burden to extract output feature maps at this scale. For example, with the ResNet-101 architecture, *78 of the 101 layers* would now have 4 or 8 times greater memory consumption.

The third category includes work on designing decoders and other efficient alternatives to dilation [5, 42]. However, the quadratic scaling of standard operations used in these techniques, such as deconvolution and bilinear upsampling, remains a crucial issue, which has seen little to no study in recent semantic segmentation literature [41]. In the proposed approach, we focus on this issue. Rather than architectural modifications, we aim to achieve better scaling through a novel representation of the network outputs.

Quadtree and octree representations. A quadtree is a hierarchical data structure with a long history in image processing literature [35,37]. Quadtrees were prominently used in some of the earliest work on semantic segmentation with graphical models, due to their hierarchical structure and efficiency [3, 8]. In contrast, quadtrees in the deep learning era have only recently been applied, in a very limited setting, for tasks involving sparse, binary input images [21]. Our proposed networks, on the other hand, do not make any sparsity assumptions on their inputs.

The family of octree representations, which are the 3D analog of quadtrees, has had significant impact on 3D deep learning. In this domain, sparse convolutions on octrees alleviate the burden of cubic scaling [31]. Our work is closely related to Octree Generating Networks, which perform high resolution 3D reconstruction from single images using a deconvolutional architecture [38]. The key differences are: (i) a novel, flexible loss formulation, that allows the user

to trade-off between accuracy and computational cost; (ii) deeper decoder architectures and more quadtree levels than those typically used with octrees; (iii) inclusion of elements such as encoder-to-decoder skip connections and adaptive level-wise loss weighting; and (iv) quadtree representations over semantic labels of complex scenes with a large number of semantic classes (up to 150); as opposed to work in the 3D setting, which has largely been restricted to binary representations (such as voxel occupancy grids).

Anytime inference. Anytime predictors are a class of algorithms that produce a crude initial prediction for each test sample, and continue to refine it as allowed by a compute budget [12, 18]. There are several real-world situations where inference-time computational costs of scene parsing are critical, which prompted research into anytime predictors for semantic segmentation in the era of fixed feature extractors [13, 25]. However, anytime prediction with learned CNN based feature extractors has been evaluated primarily in the image classification setting [17, 19, 23]. In contrast, our work with QGNs involves anytime inference for semantic segmentation, through the inherent flexibility of the quadtree representation.

3. Quadtree Generating Networks

In this section, we describe our deep network architecture used to generate an efficient quadtree prediction for a given input image, the Quadtree Generating Network (QGN). Its architecture is shown in Fig. 2. We begin by introducing the quadtree and T-pyramid representations, used in calculating the loss function for training QGNs. We then describe the three main components of our architecture: the sparse convolutional decoder, prediction layers and skip connections. We limit our focus in this study to the setting where the encoder subsamples inputs by a factor of 32. Correspondingly, we use a sparse convolutional decoder with 5 blocks– each block consists of several sparse convolutions followed by an upsampling operation.

3.1. Quadtrees and T-Pyramids

A quadtree is a tree data structure in which each internal node has exactly four children [35], see Fig. 1. Quadtrees are the two-dimensional analog of octrees and are most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions. A function defined on a pixel grid, such as a segmentation map, can be converted to a quadtree by this kind of recursive subdivision of cells– if every pixel within a sub-cell has the same function value, it need not be further divided, and becomes a leaf of the quadtree assigned that specific function value.

The set of cells at a certain resolution in a quadtree is referred to as a quadtree level, denoted by l. If a fixed number of levels is desired, the recursive subdivision process can be started not from the whole grid, but from some initial coarse resolution. Then, the maximal quadtree cell size, or size of the root node, is given by this initial resolution. For example, a maximal cell size of 32×32 can be partitioned into a quadtree with 6 levels.

An efficient way of storing a quadtree is using hash tables: each node in the quadtree is indexed by its key, which is a tuple of the level and spatial location (l, x, y), and stores a value v, which may be discrete or continuous. In the case of semantic segmentation, the value v comes from the set $\{1, ..., k\}$ where k is the number of classes. The quadtree Q is the set of all key-value pairs,

$$Q = \{(l, x, y, v)\}. \tag{1}$$

To train networks using quadtree representations, we would need to be able to compute the distance between two different quadtrees, the prediction $\hat{Q}$ and ground truth Q. To this end, we would need to be able to query, for any cell (l, x, y) in $\hat{Q}$, the corresponding value in Q. This is not trivial, as the two quadtrees may have a different structure. The ground truth value corresponding to a prediction with the wrong structure would be undefined. To resolve this discrepancy, we store the ground truth Q in the a more generalized form, called a tree-pyramid (T-pyramid), denoted as T. A T-pyramid is a 'complete' quadtree; every node of the T-pyramid has four children except leaf nodes; and all leaves are on the same level, which corresponds to individual pixels in the image [37]. To efficiently construct this kind of representation for a segmentation mask with classes $(1, ..., k)$, we propose the use of a merging operator $\mathcal{M}$ that is applied to patches of size 2×2,

$$\mathcal{M}\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = \begin{cases} a, & \text{if } a = b = c = d \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

where 0 is the value assigned to a newly introduced class that represents a mixture of multiple classes. We refer to this as the 'composite' class (gray cells in Fig. 1). This class helps to encode information about the quadtree structure in the T-pyramid representation, as any cell belonging to this class has children at a lower level in the original quadtree. Cells with any value other than the composite class have no children, or are not present in the original quadtree.

Let T_0 represent the level 0, or leaf nodes, of the T-pyramid. $T_1 = \mathcal{M}(T_0)$ is obtained by applying $\mathcal{M}$ to all the 2×2 patches in T_0. Similarly, we can continue building the pyramid to as many levels as desired, by recursively applying $\mathcal{M}$,

$$T_{l+1} = \mathcal{M}(T_l). \tag{3}$$

Loss function. We now formulate the loss function between a quadtree prediction $\hat{Q}$, and the corresponding ground truth T-pyramid label T. At each level l in $\hat{Q}$, the level loss $\mathcal{L}_l$ is computed as:

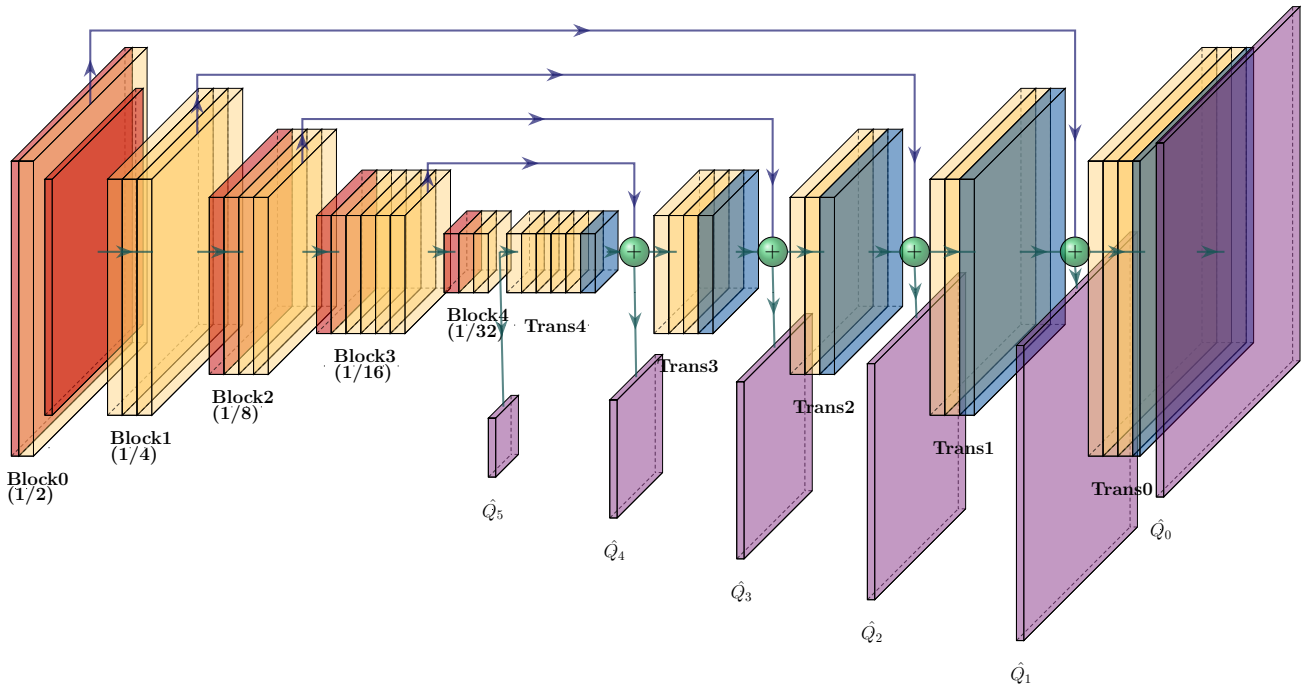

Figure 2. QGN network architecture, consisting of an encoder, decoder, skip connections and prediction layers. Subsampling layers in the encoder are shown in red, and upsampling layers in the decoder in blue. Prediction layers with the softmax activation are shown below the main architecture in purple. The first such prediction layer is placed immediately after the encoder, for which the output $\hat{Q}_5$ is of resolution $\frac{1}{32}$ compared to the input, and forms the root node of the quadtree. Prediction layers after each skip connection the decoder produce additional quadtree levels $\hat{Q}_4, ..., \hat{Q}_0$ (best viewed in color).

$$\mathcal{L}_l = \frac{1}{N} \sum_{i=1}^{N} \mathcal{H}(v^i, \mathcal{T}(l, x^i, y^i)), \tag{4}$$

where $\mathcal{H}$ is the cross-entropy loss function over the $k+1$ classes that may be taken by v^i (composite + k actual classes). $\mathcal{T}$ is a query function on the T-pyramid that returns the value of the corresponding hash table key,

$$\mathcal{T}(l, x, y) = v. \tag{5}$$

The overall loss is a weighted sum of the loss at each level, given by:

$$\mathcal{L} = \sum_{l \in Q} \beta_l \mathcal{L}_l. \tag{6}$$

Loss weighting. We experiment with two approaches for weighting the loss at different levels of the quadtree (Eq. 6): (i) **Fixed**, where we scale the weight assigned to each layer by a multiplicative hyper-parameter γ with respect to the preceding layer's weight (with $\beta_0 = 1$),

$$\beta_{l+1} = \gamma \beta_l; \tag{7}$$

and (ii) **Adaptive**, where we use a running average value of the loss at that scale as a weight, as in [17]:

$$\beta_l^{i+1} = \delta \beta_l^i + (1 - \delta) \mathcal{L}_l^i, \tag{8}$$

where i is the training iteration index, each β_l is initialized to 1, and $\delta = 0.99$ in our experiments.

Sparse activations. To compute $\mathcal{L}_l$ as defined in Eq. 4, the desired output from our network architecture is a quadtree prediction $\hat{Q}$. Our decoder generates the entries of $\hat{Q}$ in a level-by-level manner.

Let the network activation A_l have a resolution corresponding to the quadtree level $\hat{Q}_l$. For example, after passing through a typical state-of-the-art deep encoder, which subsamples the input by a factor of 32, the activations A_5 would correspond to the quadtree level $\hat{Q}_5$.

A_l is typically stored as a tensor of dimensions $W_l \times H_l \times C_l$. However, in our decoder, we store the activations as hash tables,

$$A_l = \{(x, y, v)\}, \tag{9}$$

where $x \in \{1, ..., W_l\}$, $y \in \{1, ..., H_l\}$, and $v \in \mathbb{R}^{C_l}$. Note that, the T-pyramid labels (Eq. 3) are quadtrees, whose values v are class indices. On the other hand, sparse activations (Eq. 9) are hash tables, whose values v are float numbers identical to those used in the tensor representation.

3.2. Architecture Components

Sparse convolutions. Sparse convolutions are the fundamental building blocks of our decoder. For any sparse activation A_l (Eq. 9), we define a to be a set of *active sites*,

which can be *any subset* of all spatial locations in A_l. A sparse convolutional layer with kernel W, bias b and active sites a operates on only the set of values at these active sites, generating an new activation map $A'_l = (x, y, v')$ with the same active sites as the input, through a transformation to its values:

$$v' = \begin{cases} Wv + b, & \text{if } (x, y) \in a \\ 0, & \text{otherwise.} \end{cases} \quad (10)$$

Using this procedure, multiple sparse convolutional layers can be used to propagate activations through the network in hash table form, without changing the active sites [10].

Upsampling. After a set sparse convolutions, each block in our decoder is differentiated from the next block through an upsampling operation. Upsampling changes the resolution, and hence the quadtree level of the activation, from l to $l-1$. We perform this operation through 2×2 nearest neighbor interpolation, an operation that doubles the resolution of the activation (i.e., $W_{l-1} = 2W_l$ and $H_{l-1} = 2H_l$). A_{l-1} is obtained as follows:

$$A_{l-1} = \{(2x, 2y, v), (2x+1, 2y, v), (2x, 2y+1, v), (2x+1, 2y+1, v)\} \; \forall \; (x, y, v) \in A_l. \quad (11)$$

Skip connections. To allow the deeper layers in the decoder to access high-frequency information from early in the encoder, otherwise lost due to subsampling, we incorporate additional 1×1 convolutional skip connections from the encoder to its corresponding block in the decoder, as shown at the top of Fig. 2. Skip connections are essential for a decoder with our choice of upsampling operation, since the upsample described in Eq. 11 produces outputs A_{l-1} that have identical values in local 2×2 blocks at the finer resolution. By adding non-identical features from the encoder to the identical features coming from the previous block in the decoder, we obtain a unique feature value at each spatial resolution in A_{l-1}.

Prediction layers. A 1×1 convolutional prediction layer is used to generate $\hat{Q}_l$ from the activation A_l (downward arrows in Fig. 2). Prediction layers use a softmax activation over $k+1$ channels, corresponding to the k-way classification problem in the dataset, and the additional composite class. At train time, the level-wise loss from Eq. 4 is computed at each of these prediction layers. Inference at test-time is conducted by accumulating the predictions made at the leaf nodes of $\hat{Q}$. In case the composite class has the highest confidence at any leaf node, the class with second-highest confidence is assigned as the prediction at that pixel instead.

3.3. Propagation Schemes

An important component of the optimization is the choice of active sites a that are propagated at each level of the decoder, which we call the propagation scheme. The choice of propagation scheme is extremely flexible– for example, certain high-priority classes could be propagated, while efficiency is maintained by not propagating entries assigned to other classes. As another use case, an entire block of the decoder could be left unpropagated by choosing no active sites, when fine detailed segmentation is not necessary. In this work, we focus our attention on three different propagation schemes: (i) All; (ii) GT Composite; and (iii) Predicted Composite.

All. All pixels are propagated through the complete decoder till the leaf nodes. Serves as a baseline with the maximum possible memory consumption, and can be used during both training and inference, given sufficient resources.

Ground Truth Composite (GTC). The quadtree representation of the ground truth segmentation mask is used to decide which pixels to propagate. *Only pixels of the composite class in the ground truth* are selected as active sites at any given layer in the network. Due to the sparsity in ground truth, this leads to significantly reduced memory consumption. In practice, this mode of operation is useful when there is access to the ground truth (during training) but cannot be used for inference. However, in our experiments, we include results for inference with this propagation scheme, as an upper bound on the performance of QGNs in a low-memory consumption setting. Similar upper bounds have been used previously to identify the gaps in performance and improve an algorithm (eg. [32]).

Predicted Composite (PC). Only pixels predicted by the network to be of the composite class are propagated. This can lead to high sparsity, and also does not require access to ground truth, and is therefore suitable for inference. However, it relies heavily on the network's performance on the composite class, and, in practice, we found this scheme to be unreliable for training from scratch. Gains in performance could be obtained by fine-tuning a model obtained with GTC by using PC, and other variations on the propagation scheme that combine different aspects of All, GTC and PC; but we leave this investigation to future work.

4. Experiments

In this section, we demonstrate the benefits of Quadtree Generating Networks on three publicly available semantic segmentation datasets: Cityscapes [6], SUN-RGBD [36] and ADE20K [49]. We also include ablation studies where we first analyze sparsity of each dataset and then, the memory consumption of our approach compared to typical dilated ResNet based architectures.

Implementation. Our QGN encoder and decoder architectures are based on ResNets [15]. We use the ResNet-50 architecture as our encoder unless otherwise specified. For our decoder, we use a transposed ResNet, similar to the network proposed in [22]. To implement sparse convolutions

Table 1. **Cityscapes:** Pixel accuracy (%) and mean IoU (%) of the proposed approach with different propagation schemes and loss weighting mechanisms. Adaptive loss weighting, and fixed weighting with $\gamma = 1$, give the best results.

Train Mode	Loss	Eval Mode					
		All		GTC		PC	
		Acc	mIoU	Acc	mIoU	Acc	mIoU
All	$\gamma = 0.75$	96.25	77.49	95.84	72.17	95.38	71.13
	$\gamma = 1$	96.24	77.95	95.94	72.97	95.42	71.53
	$\gamma = 1.25$	96.17	77.25	95.95	73.02	95.35	71.52
	Adaptive	**96.33**	**78.20**	95.76	72.89	95.20	71.33
GTC	$\gamma = 0.75$	15.88	13.58	97.76	80.13	95.16	72.16
	$\gamma = 1$	32.59	17.25	97.69	80.67	**95.42**	72.44
	$\gamma = 1.25$	19.98	14.39	97.66	80.31	95.41	72.56
	Adaptive	31.00	18.25	**97.81**	**81.50**	95.33	**73.00**

in the decoder, we use the *SparseConvNet* framework for PyTorch, introduced with submanifold sparse convolutional networks [11]. Please refer to the supplementary material for more details regarding the experimental setup and decoder architecture.

4.1. Results on Cityscapes

Cityscapes [6] is a street scene segmentation dataset, consisting of 2048×1024 fine pixel-annotated images collected from streets in various European cities. There are 2,975 images in the training set and 500 images for validation. The segmentation task is over 19 commonly observed classes in these street scenes, such as road, car, pedestrian, traffic sign, etc. The high-resolution images in this dataset make existing segmentation networks consume large amounts of memory. QGNs are an ideal solution to address this challenge.

In our first experiment, we evaluate the three propagation schemes (All, GTC and PC as detailed in Section 3.3), when QGNs are trained using All and GTC. While GTC evaluation is not practically applicable, the gap in performance between GTC and PC evaluation helps us to better understand the strengths and weaknesses of the proposed algorithm. We also compare the two loss weighting approaches from Eq. 7 and Eq. 8 in this experiment. Our results are summarized in Table 1.

As shown, when training with the All propagation scheme (that has higher memory consumption), best results are observed for adaptive loss weighting, when the network is also evaluated with the All propagation scheme, achieving 78.20 mIoU. When evaluated with GTC or PC, the different loss weighting approaches do not significantly impact performance, and the network obtains 73.02 and 71.53 mIoU respectively. Importantly, this shows that the same networks that obtain 78.20 mIoU when evaluated with All, can be adaptively modified at inference time with a different propagation scheme such as PC, and still achieve competitive mIoU.

In comparison, networks trained with GTC perform poorly in the All evaluation mode. This is likely due to the fact that the training distribution observed by the deeper layers in the GTC decoder is very different from the training distribution for All, and the final prediction layer is not suitable for assigning labels to all the pixels in an image. When evaluating with the GTC propagation scheme for this model, we observe the best results for adaptive loss weighting. The obtained 81.50 mIoU is an upper bound on performance with QGNs in the low-memory consumption regime. Interestingly, this performance exceeds that of state-of-the-art architectures such as DeepLabv3 and CCNet, which rely on deeper networks and additional context modules. Finally, while evaluating with PC, both $\gamma = 1$ and adaptive loss weighting perform well. The gap in performance between GTC and PC evaluation helps us isolate the impact of the quadtree 'structure prediction' and final 'label assignment' tasks performed by the QGN on performance. Based on the observed gap (8.5 points), we see that 'structure prediction' is the main challenge for improving our approach.

Table 2. **Cityscapes:** Activation memory consumption (GB), FLOPs ($\times 10^{12}$), and mean IoU (%) of the proposed approach in comparison to state-of-the-art dilation-based methods on the validation set, for input resolution 2048×1024. QGN results are with adaptive loss weighting. Our approach significantly outperforms dilated networks with similar memory consumption, and provides competitive results to state-of-the-art models that are computationally more intensive.

Method	Model	Memory	TFLOPs	mIoU
Dilation	DRN-C-42 [43]	3.77	1.07	70.9
	DRN-D-105 [43]	15.15	1.91	75.6
	DeepLabv3 [4]	14.27*	1.97*	79.3
	CCNet [20]	14.33*	1.55*	79.8
QGN (Ours)	Train-All-Eval-All	5.85	0.48	78.2
	Train-GTC-Eval-PC	3.66	0.25	73.0

* indicates an estimate based on architecture details

Comparison to dilated convolutions. We focus now on comparing our best results with PC evaluation to existing semantic segmentation networks that use dilated convolutions in the encoder. Table 2 summarizes our results in terms of memory consumption, FLOPs and mIoU. As shown, QGNs with Train-GTC-Eval-PC provide a 3% relative mIoU improvement over the DRN-C-42 network, which consumes a similar amount of memory. Further, they only have a 3% less relative mIoU compared to the DRN-D-105 network, which consumes $4\times$ more memory and $6\times$ more compute. DeepLabv3 and CCNet provide slightly better results than our proposal. However, they require deeper backbone networks, specialized context modules, and at least $2.5\times$ more memory and $3\times$ more compute than QGNs.

From these results, we can conclude that adaptive loss weighting, and fixed weighting with $\gamma = 1$, give the best performance. Further, training with GTC leads to maximum performance when evaluating with PC, but training

Table 3. **SUN-RGBD:** Pixel accuracy (%) and mean IoU (%) of the proposed approach on the 13-way and 37-way validation sets. All methods use $\gamma = 1$. Relative performance of the 3 propagation schemes is similar across both datasets.

Train Mode	Eval Mode					
	All		GTC		PC	
	Acc	mIoU	Acc	mIoU	Acc	mIoU
SUN-RGBD-13						
All	**85.32**	**62.70**	81.92	55.76	82.59	56.65
GTC	66.91	36.03	**87.12**	**64.57**	**84.08**	**59.19**
SUN-RGBD-37						
All	**80.69**	**44.11**	77.26	37.88	78.05	39.13
GTC	65.24	28.10	**82.36**	**45.36**	**79.26**	**41.10**

with All gives networks that can be adaptively modified at inference time with competitive results when evaluated with all 3 propagation schemes. Of special interest is the low memory requirement of our approach (see Table 2): contrary to methods such as DeepLabv3 or CCNet which need to process full resolution images in parts on a standard GPU with 12GB of memory, our proposal can process 2 or 3 frames in parallel on the same GPU at full resolution. As a result, our architecture could be used to process consecutive frames in batches allowing for improved throughput, or perform multi-scale fusion during inference to improve performance.

4.2. Results on SUN-RGBD

The focus of our next experiment is to analyze the impact of increasing the number of classes on different propagation schemes. To this end, we use the SUN-RGBD dataset [36], an indoor scene segmentation benchmark that consists of two segmentation tasks (13 and 37 categories) with 5,285 RGB-D training images and 5,050 test images. We use the same experimental setting as the one used for the first Cityscapes experiment.

Table 3 summarizes our results. As shown, with respect to the propagation scheme, the trend remains similar to Cityscapes: the GTC upper bound provides the best performance, followed by All, and then PC. On SUN-RGBD-37, our QGN based on a ResNet-50 encoder evaluated using All provides competitive results (44.11 mIoU) to the 45.90 of RefineNet [24] which builds upon a ResNet-152 backbone, and is the state-of-the-art for this task. Regarding the influence of the number of classes, our results show no significant difference in terms of relative mIoU of the best GTC and PC results with respect to All (1.03 and 094 for SUN-RGBD-13; and 1.03 and 0.93 on SUN-RGBD-37). These results suggest that the number of classes has no significant impact on the performance of our approach.

4.3. Results on ADE20k

ADE20k [48, 49] is a recent scene parsing benchmark with a large number of classes (150 total classes, with 35 stuff classes, i.e., road, sky, wall... which occupy 61% of the pixels in the dataset, and 115 thing classes, i.e., table, person, car... which occupy 32% of the pixels). The dataset consists of natural images of a variety of indoor and outdoor scenes, with 20,210 training images and 2,000 validation images. The large proportion of thing classes makes ADE20k a challenging task that requires a lot of contextual reasoning. In order to counter these challenges, existing results involve the use of dilated convolutions, in combination with a specialized module after the encoder that improves contextual reasoning [20, 44–47].

For our experiments on this task, we use a modified version of QGNs, inserting a PSPNet [46] module in between our encoder and decoder. This allows us to check the compatibility of QGNs with specialized context modules. We compare our approach to state-of-the-art results in terms of pixel accuracy and mIoU with both ResNet-50 and ResNet-101 backbones.

Our results are shown in Table 4. With PSPNet, QGNs outperform dilation on ResNet-101, showing the potential of QGNs as a generic approach that can be paired with exist-

Table 4. **ADE20k:** State-of-the-art performance comparisons in terms of pixel accuracy (%) and mean IoU (%) on the validation set. All QGN results are with the Train-All-Eval-All models, and $\gamma = 1$. All Dilation based numbers are obtained from the respective papers. With PSPNet, QGNs outperform Dilation, and provide competitive results to other recent context modules in literature.

Context	Encoder	Method	Acc	mIoU
PSPNet [46]	ResNet-50	Dilation	80.76	42.78
		QGN (Ours)	80.43	41.42
	ResNet-101	Dilation	81.39	43.29
		QGN (Ours)	81.67	43.91
SAC [45]	ResNet-101	Dilation	81.86	44.30
EncNet [44]			81.69	44.65
PSANet [47]			81.51	43.77
CCNet [20]			-	45.22

Table 5. Percentage of pixels in the three datasets (at original resolutions) assigned to each level in a 6-level quadtree representation. The final column indicates the average ratio of memory occupied by the efficient quadtree representation in comparison to dense pixel grid segmentation masks for that dataset (in %). We observe that a large fraction of the pixels belong to larger cells, with less than 3% at the leaf (Q_0) nodes, leading to compression ratios of 95% or more. See supplemental for a further visual comparison of quadtree representations and dense pixel grids.

Dataset	Split	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0	Ratio
Cityscapes	train	66.34	14.21	9.18	5.52	3.02	1.70	3.07 %
	val	65.12	14.44	9.53	5.85	3.22	1.81	3.25 %
SUN-RGBD	train-13	56.26	18.22	11.78	7.09	4.20	2.43	4.24 %
	val-13	55.57	18.50	11.98	7.22	4.27	2.46	4.29 %
	train-37	56.11	18.27	11.82	7.12	4.22	2.44	4.25 %
	val-37	55.40	18.54	12.03	7.25	4.29	2.47	4.31 %
ADE20k	train	47.48	21.40	14.44	8.68	5.05	2.93	5.09 %
	val	47.25	21.44	14.49	8.74	5.10	2.96	5.14 %

Table 6. Average forward pass activation memory consumption with different model architectures (in GB) for segmentation of an image from the validation datasets used in our experiments. R50 denotes ResNet-50 and R101 denotes ResNet101. All and GTC correspond to the propagation schemes detailed in Section 3. Dil denotes dilation in the encoder, and the corresponding 'Decoder Only' memory refers to a single 1×1 convolution as a decoder. The architectures with the **lowest** and *highest* memory consumption for each backbone are marked in **bold** and *italic* respectively. QGNs consume significantly less memory than dilation based networks.

Architecture	**Decoder Only**			**Encoder-Decoder**			**Encoder-Decoder**		
Dataset	**Dil**	**All**	**GTC**	**R50-Dil**	**R50-All**	**R50-GTC**	**R101-Dil**	**R101-All**	**R101-GTC**
Cityscapes	0.15	2.28	0.09	*7.52*	5.85	**3.66**	*13.89*	7.44	**5.26**
SUN-RGBD-13	0.01	0.27	0.02	*0.98*	0.71	**0.46**	*1.72*	0.91	**0.66**
SUN-RGBD-37	0.04	0.31	0.03	*1.01*	0.75	**0.47**	*1.75*	0.95	**0.67**
ADE20k	0.17	0.53	0.08	*1.31*	1.05	**0.60**	*2.18*	1.29	**0.84**

ing context modules in literature. The achieved 43.91 mIoU is also competitive to the state-of-the-art obtained with more recent context modules. Our results indicate that there may be further gains by using QGNs as a drop-in replacement for dilation in these newer segmentation approaches.

4.4. Memory footprint analysis

Dataset sparsity. To evaluate the efficiency of quadtree representations, we convert the provided ground truth segmentation masks in each dataset into 6-level quadtrees and analyze the sparsity of the representation in Table 5. We observe that there is significant redundancy across all datasets, with less than 3% of the pixels in the dataset being assigned to the leaf nodes of the quadtree. The proposed representation eventually occupies orders of magnitude less memory than the typical dense pixel grid. The compression ratio is maximum (around 97%, or $33\times$) for higher-resolution images with few classes (Cityscapes) and minimum (around 95%, or $20\times$) for lower-resolutions with more classes (ADE20k). Further analysis of the dataset sparsity, in terms of visualizations of the quadtree representation for validation images from the Cityscapes and ADE20k datasets, is provided in the supplemental material.

Activation memory footprint. The actual reduction in memory footprint achieved through QGNs depends on a number of factors besides the dataset sparsity, including the input resolution, network architecture and propagation scheme. It is important to note that the reduction in memory footprint with our approach comes only with respect to the feature activations (which form a majority of the occupied memory during training). The memory consumption of the model parameters remains the same for different propagation schemes.

To evaluate the actual memory reduction, we compare the forward pass memory consumption of QGNs with the All and GTC propagation schemes, using ResNet-50 and ResNet101 backbones, to networks based on dilated convolutions. When using dilated convolutions, dilate the last 2 residual blocks of the encoder (Block3 and Block4) so that the feature maps produced are 8 times smaller than the input resolution along each dimension. We remove the QGN decoder, and use a single 1×1 convolutional layer as a decoder with these dilated encoders to obtain the logits. We use bilinear interpolation on the logits to recover a prediction at the same resolution as the input. The QGN decoder has a parameter overhead of around 0.12GB as opposed to a single 1×1 convolutional layer, but this overhead is negligible compared to the memory gains for QGNs through the activations, as summarized in Table 6.

For the same backbone network, a QGN encoder-decoder is more memory efficient than a dilated network, even when operating with the All propagation scheme (R50-All and R101-All). Furthermore, when using the GTC propagation scheme, the decoder activations consume significantly less memory ($<$0.1GB), sometimes less than the memory consumed by a single 1×1 convolution used as a decoder with dilation. Overall, our GTC approach consumes up to $2.6\times$ less memory than a dilation based network using the same backbone architecture, while maintaining similar performance (see Tables 2 and 4).

While QGNs are designed to reduce decoder memory, a key advantage is that the QGN decoder *enables us to use a stride 32 backbone without losing performance*. As a result, our decoder is implicitly responsible for the memory reductions observed in the encoder. Existing methods perform poorly when the encoder is replaced with a higher stride network. For example, when compared to a DeepLabv3 decoder + ResNet-50 stride 32 encoder (from [4]), which only obtains 70.0 mIoU on Cityscapes, our method achieves a 4.3% better relative mIoU of 73.0 (see Table 2).

5. Conclusion

In this paper, we presented Quadtree Generating Networks (QGNs) for semantic segmentation. The key idea is to make predictions as a hierarchical quadtree instead of a dense rectangular grid, using a sparse convolutional decoder. Our results on three public benchmarks demonstrate that QGNs can process high-resolution images and outperform those approaches that have similar memory consumption. In addition, our results are competitive when compared to state-of-the-art methods while consuming $2\times$ to $4\times$ less memory. Moreover, our approach is flexible and can be adapted at inference time to trade-off between performance and memory consumption.

Acknowledgements

We would like to thank Yash Patel for several insightful comments and discussions.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *PAMI*, 2012. 1

[2] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *PAMI*, 2017. 2

[3] C. A. Bouman and M. Shapiro. A multiscale random field model for bayesian image segmentation. *IEEE Trans. Img. Proc.*, 1994. 2

[4] L. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv e-prints*, 2017. 1, 2, 6, 8

[5] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. *ECCV*, 2018. 2, 11

[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. 5, 6

[7] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *PAMI*, 2013. 1

[8] X. Feng and C. K. I. Williams. Training bayesian networks for image segmentation. *Proc. SPIE*, 1998. 2

[9] J. Fu, J. Liu, H. Tian, Z. Fang, and H. Lu. Dual attention network for scene segmentation. *arXiv e-prints*, 2018. 2

[10] B. Graham. Sparse 3D convolutional neural networks. *BMVC*, 2015. 5

[11] B. Graham, M. Engelcke, and L. van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. *CVPR*, 2018. 2, 6, 11

[12] J. Grass and S. Zilberstein. Anytime algorithm development tools. *SIGART Bull.*, 1996. 3

[13] A. Grubb, D. Munoz, J. A. Bagnell, and M. Hebert. SpeedMachines: Anytime Structured Prediction. *arXiv e-prints*, 2013. 3

[14] B. Hariharan, P. A. Arbeláez, R. B. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. *CVPR*, 2015. 2

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2, 5, 11

[16] X. He, R. S. Zemel, and M. A. Carreira-Perpinan. Multiscale conditional random fields for image labeling. In *CVPR*, 2004. 1

[17] H. Hu, D. Dey, M. Hebert, and J. A. Bagnell. Learning Anytime Predictions in Neural Networks via Adaptive Loss Balancing. *AAAI*, 2018. 3, 4

[18] H. Hu, A. Grubb, J. A. Bagnell, and M. Hebert. Efficient Feature Group Sequencing for Anytime Linear Prediction. *UAI*, 2016. 3

[19] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger. Multi-Scale Dense Networks for Resource Efficient Image Classification. *ICLR*, 2018. 3

[20] Z. Huang, X. Wang, L. Huang, C. Huang, Y. Wei, and W. Liu. CCNet: Criss-Cross Attention for Semantic Segmentation. *arXiv e-prints*, 2018. 2, 6, 7

[21] P. K. Jayaraman, J. Mei, J. Cai, and J. Zheng. Quadtree convolutional neural networks. In *ECCV*, 2018. 2

[22] J. Jiang, L. Zheng, F. Luo, and Z. Zhang. Rednet: Residual encoder-decoder network for indoor rgb-d semantic segmentation. *arXiv e-prints*, 2018. 5, 11

[23] H. Lee and J. Shin. Anytime Neural Prediction via Slicing Networks Vertically. *arXiv e-prints*, 2018. 3

[24] G. Lin, A. Milan, C. Shen, and I. Reid. RefineNet: Multi-path refinement networks for high-resolution semantic segmentation. In *CVPR*, 2017. 2, 7

[25] B. Liu and X. He. Learning Dynamic Hierarchical Models for Anytime Scene Labeling. *ECCV*, 2016. 3

[26] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 1, 2

[27] W. Luo, Y. Li, R. Urtasun, and R. Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *NIPS*. 2016. 2

[28] H. Noh, S. Hong, and B. Han. Learning Deconvolution Network for Semantic Segmentation. *ICCV*, 2015. 1, 2

[29] P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollár. Learning to refine object segments. In *ECCV*, 2016. 2

[30] M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun. Sbnet: Sparse blocks network for fast inference. *CVPR*, 2018. 2, 11

[31] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. In *CVPR*, 2017. 2

[32] G. Roig, X. Boix, R. D. Nijs, S. Ramos, K. Kuhnlenz, and L. V. Gool. Active map inference in crfs for efficient semantic segmentation. In *ICCV*, 2013. 5

[33] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Trans. Int. Transp. Sys.*, 19(1):263–272, Jan 2018. 1

[34] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *MICCAI*, 2015. 1, 2

[35] H. Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16(2):187–260, June 1984. 2, 3

[36] S. Song, S. P. Lichtenberg, and J. Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *CVPR*, 2015. 5, 7

[37] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. Thomson-Engineering, 2007. 2, 3

[38] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. *ICCV*, 2017. 2

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. *NIPS*, 2017. 2

[40] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local Neural Networks. *CVPR*, 2017. 2

[41] Z. Wojna, V. Ferrari, S. Guadarrama, N. Silberman, L.-C. Chen, A. Fathi, and J. Uijlings. The Devil is in the Decoder. *BMVC*, 2018. 2

[42] T. Xiao, Y. Liu, B. Zhou, Y. Jiang, and J. Sun. Unified Perceptual Parsing for Scene Understanding. *ECCV*, 2018. 2, 11

[43] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016. 2, 6

[44] H. Zhang, K. Dana, J. Shi, Z. Zhang, X. Wang, A. Tyagi, and A. Agrawal. Context encoding for semantic segmentation. In *CVPR*, 2018. 7

[45] R. Zhang, S. Tang, Y. Zhang, J. Li, and S. Yan. Scale-adaptive convolutions for scene parsing. In *ICCV*, 2017. 2, 7

[46] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *CVPR*, 2017. 1, 2, 7, 11

[47] H. Zhao, Y. Zhang, S. Liu, J. Shi, C. C. Loy, D. Lin, and J. Jia. PSANet: Point-wise spatial attention network for scene parsing. In *ECCV*, 2018. 2, 7

[48] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. In *CVPR*, 2017. 7

[49] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso, and A. Torralba. Semantic understanding of scenes through the ade20k dataset. *IJCV*, 2018. 5, 7

Supplementary Material

A. Implementation Details

In this section, we provide more information about the network architectures and hyper-parameters used in the experiments from Section 4.

A.1. Network Architecture

Our QGN encoder and decoder architectures are based on ResNets [15]. Similar to existing semantic segmentation papers, we replace the first 7×7 convolution of the original ResNet with three 3×3 convolutions [5, 42, 46]. We use the ResNet-50 architecture as our encoder unless otherwise specified. For our decoder, we use a transposed ResNet, similar to the network proposed in [22]. Specifically, this decoder is a residual network with 38 layers. The exact channel and layer counts for the encoder and decoder are summarized in Table 7.

Table 7. Encoder and Decoder configurations for QGNs, based on the architecture proposed in [22]. Each 'Block' subsamples or upsamples the image by a factor of 2.

Block	Encoder			Block	Decoder		
	In	Out	# Unit		In	Out	# Unit
Block0	3	64	-	Trans4	512	256	6
Block1	64	256	3	Trans3	256	128	4
Block2	256	512	4	Trans2	128	64	3
Block3	512	1024	6	Trans1	64	64	3
Block4	1024	2048	3	Trans0	64	64	3

Sparse convolutions. To implement sparse convolutions in the decoder, we use the *SparseConvNet* framework, introduced with submanifold sparse convolutional networks [11]. We also implement a sanity check that uses dense convolutions with multiplicative masks of zeros to emulate sparse convolutions. We found that the wall-clock time for training with sparse vs. dense convolutions is similar (up to +10% training time depending on dataset sparsity). More optimized implementations of sparse convolutions (eg. [30]) could translate the large gains in memory efficiency to reductions in wall-clock training time.

A.2. Optimization

Input resolution. We take advantage of our efficient architecture by training QGNs with full resolution input images (without cropping) on all datasets. Within the ADE20k and SUN-RGBD datasets, resolution and aspect ratios vary widely (561×427 to 730×530, with many others). We found that zero padding to the nearest multiple of 32 pixels (maximum downsampling ratio of the encoder) during training effectively deals with this problem. During inference, we use bilinear interpolation to rescale the image dimensions to the nearest multiple of 32, make the predictions at each pixel, and rescale the resulting segmentation map to the original input dimensions with nearest neighbor interpolation.

Learning rate schedule. We optimize our networks with Stochastic Gradient Descent (SGD), with an initial learning rate α_0 of 0.02, which is decreased based on a polynomial decay function $\alpha = \alpha_0(1 - \frac{i}{i_{max}})^{\rho}$. We use a decay factor ρ of 0.9, training for a total of $i_{max} = 100,000$ iterations. Random scaling and horizontal flips are applied for data augmentation. For all our experiments, we maintain this set of hyper-parameters, without specific tuning to each task.

Class weighting. To counter the effect of class imbalance, we double the weight of the loss for the set of classes whose IoU is below the median IoU on a test subset. Specifically, we evaluate the performance of the model every $5,000$ training iterations on a subset of 100 images set aside for validation, and use these results to set the class weights.

B. Dataset Sparsity Visualization

In this section, we visualize the quadtree representations of the segmentation maps obtained using the procedure detailed in Section 3.1. Specifically, we are interesting in visualizing each level of the ground truth quadtree representation. To this end, we apply the merge operator from Eq. 2 to validation images from the Cityscapes and ADE20k datasets. The visualizations we obtain are presented in Table 8 and Table 9 respectively.

We observe that a majority of the image pixels have their ground truth encoded in Q_5, which is highly efficient due to its lower resolution. The intermediate levels coarsely encode semantic boundaries, with the leaf nodes (Q_0) encoding the finer details of these boundaries between classes. These observations mirror the statistics observed in Table 5, showcasing the huge potential for lossless compression in semantic segmentation using quadtree representations.

Table 8. Visualization of dataset sparsity using examples from the Cityscapes validation set. $Q_5, \ldots, Q_0$ are the different quadtree levels. As noted in Table 5, a large fraction of the pixels belong to larger cells (such as Q_5), while the leaf (Q_0) nodes are extremely sparse. The composite class is represented in black (best viewed in color).

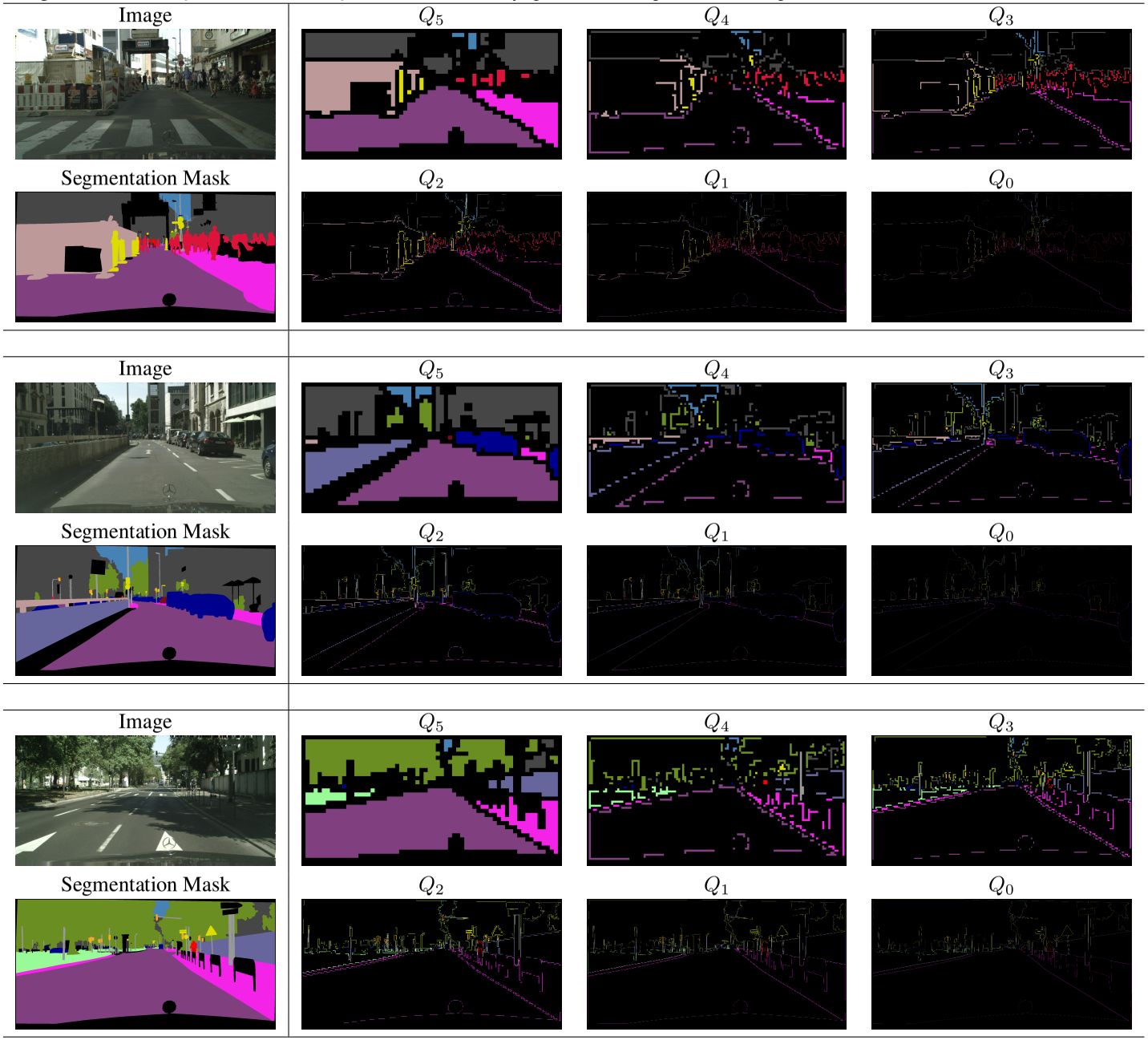

Table 9. Visualization of dataset sparsity using examples from the ADE20k validation set. $Q_5, ..., Q_0$ are the different quadtree levels. As noted in Table 5, a large fraction of the pixels belong to larger cells (such as Q_5), while the leaf (Q_0) nodes are extremely sparse. The composite class is represented in black (best viewed in color).