

# 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 [14]. Similar to existing semantic segmentation papers, we replace the first  $7 \times 7$  convolution of the original ResNet with three  $3 \times 3$  convolutions [5, 41, 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 [21]. 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 [21]. 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 [10]. 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. [29]) 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 ( $561 \times 427$  to  $730 \times 530$ , 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  $\alpha_0$  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  $\rho$  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.  $Q_5, \dots, Q_0$  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).


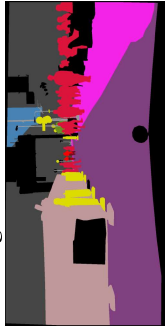

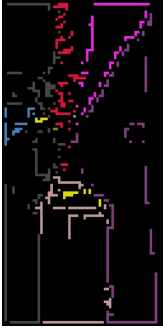

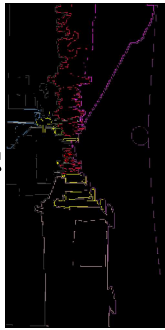

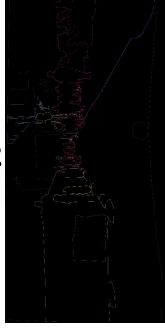


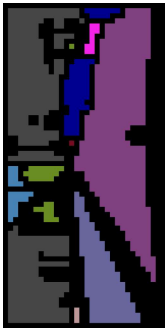
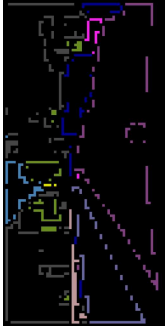
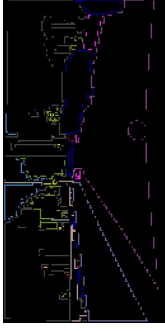
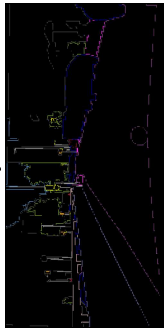
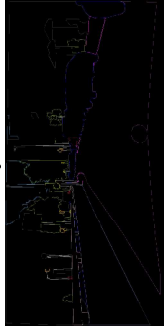


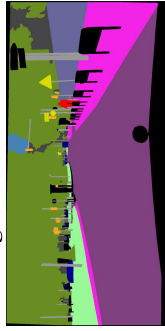
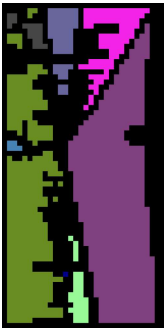


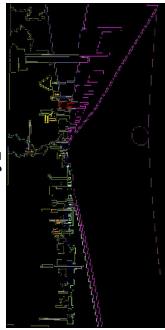
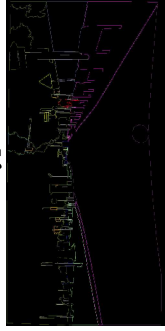

<p>Image</p>  <p>Segmentation Mask</p> 	<p><math>Q_5</math></p>  <p><math>Q_4</math></p>  <p><math>Q_3</math></p>  <p><math>Q_2</math></p>  <p><math>Q_1</math></p>  <p><math>Q_0</math></p> 
<p>Image</p>  <p>Segmentation Mask</p> 	<p><math>Q_5</math></p>  <p><math>Q_4</math></p>  <p><math>Q_3</math></p>  <p><math>Q_2</math></p>  <p><math>Q_1</math></p>  <p><math>Q_0</math></p> 
<p>Image</p>  <p>Segmentation Mask</p> 	<p><math>Q_5</math></p>  <p><math>Q_4</math></p>  <p><math>Q_3</math></p>  <p><math>Q_2</math></p>  <p><math>Q_1</math></p>  <p><math>Q_0</math></p> 

Table 9. Visualization of dataset sparsity using examples from the ADE20k validation set.  $Q_5, \dots, Q_0$  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).

Image	$Q_5$	$Q_4$	$Q_3$
			
Segmentation Mask	$Q_2$	$Q_1$	$Q_0$
			
Image	$Q_5$	$Q_4$	$Q_3$
			
Segmentation Mask	$Q_2$	$Q_1$	$Q_0$
			
Image	$Q_5$	$Q_4$	$Q_3$
			
Segmentation Mask	$Q_2$	$Q_1$	$Q_0$
			