

## 519 A Training details

520 We trained 24 networks of each of the three types. The versions differed in the size of the neighborhood  
521 (4, 8, 12, or 20 neighbors), the amount of noise added ( $\alpha \in 0, 0.1, 0.2$ ), and the used loss (position or  
522 factor loss).

523 The parameters we trained were:

- 524 • all weights of the underlying network
- 525 • the logit transform of  $p$  for each relative position of two neighbors
- 526 • the logarithms of the diagonal entries of  $C$  for each relative position of neighbors

527 We trained models using the standard stochastic gradient descent implemented in pytorch [56] with  
528 a learning rate of 0.001, a momentum of 0.9 and a slight weight decay of 0.0001. To speed up  
529 convergence we increased the learning rate by a factor of 10 for the parameters of the prediction, i.e.  
530  $C$  and  $p$ . For the gradient accumulation for the position based loss, we accumulate 5 repetitions for  
531 the pixel model and 10 for the linear model and for predseg1. Each repetition contained 10 random  
532 negative locations. Batch size was set to fit onto the smaller GPU type used in our local cluster. The  
533 resulting sizes are listed in Table 2.

### 534 A.1 Architecture details

535 The pixel model was implemented as a single Identity layer.

536 The linear model was implemented as a single  $50 \times 11 \times 11$  convolutional layer.

537 The Predseg1 model was implemented as a sequential model with 4 processing steps separated by  
538 subsampling layers ( $1 \times 1$  convolutional layers with a stride  $> 1$ ). The first processing step was a  
539  $3 \times 3$  convolutional layer with 3 channels followed by subsampling by a factor of 3. The second step  
540 was a  $11 \times 11$  convolutional layer with 64 features followed by subsampling by a factor of 2. The  
541 third and fourth steps were residual processing blocks, i.e. two convolutional layers with a rectified  
542 linear unit non-linearity between them whose results were added to the inputs. They had 128 and 256  
543 features respectively and were separated by another subsampling by a factor of 2.

### 544 A.2 Added noise

545 To prevent individual features dimensions from becoming perfectly predictive, we added a small  
546 amount of Gaussian noise to the feature maps before applying the loss. To yield variables with mean  
547 0 and variance 1 after adding the noise we implemented this step as:

$$f_{noise} = \sqrt{1 - \alpha^2} + \alpha\epsilon \quad (13)$$

548 where  $\alpha \in [0, 1]$  controls the noise variance and  $\epsilon$  is a standard normal random variable.

549 Adding this noise did not change any of our results substantially and the three versions with different  
550 amounts of noise ( $\alpha = 0, 0.1$  or  $0.2$ ) performed within  $1 - 2\%$  in all performance metrics.

### 551 A.3 Training duration

552 Networks were trained in training jobs that were limited to either 48 hours of computation time or 10  
553 epochs of training. As listed in table 2 we used a single such job for the pixel models, 7 for the linear  
554 models and 9 for the predseg1 models. Most larger networks were limited by the 48 hour limit, not  
555 by the epoch limit.

### 556 A.4 Used computational resources

557 The vast majority of the computation time was used for training the network parameters. Computing  
558 segmentations for the BSDS500 images and evaluating them took only a few hours of pure CPU  
559 processing.

Table 2: Training parameters and training time for the different networks. Networks were trained on single V100 (32GB) or RTX8000 (48GB) GPUs depending on availability. Training times were approximately read out from the computation logs. # of training jobs indicates how many 48 hour jobs we started for each model.

Model	batch size	training time (hh:mm per epoch)				# training jobs
		4	8	12	20	
pixel (position loss)	32	0:30	1:00	1:30	2:30	1
pixel (factor loss)	32	0:45	1:20	2:00	3:15	1
linear (position loss)	6	10:20	20:20	30:15	50:00	7
linear (factor loss)	6	4:00	7:50	11:30	19:00	7
predseg1 (position loss)	16	4:05	7:45	11:25	18:40	9
predseg1 (factor loss)	24	2:20	4:40	6:45	11:10	9

560 Networks were trained on an internal cluster using one GPU at a time and 6 CPUs for data loading.  
561 We list the training time per epoch in table 2. If every job had run for the full 48 hours we would have  
562 used  $(1 + 7 + 9) \times 24 \times 2 = 816$  days of GPU processing time, which is a relatively close upper  
563 bound on the time we actually used.