Supplementary Material for Learning Sparse High Dimensional Filters: Image Filtering, Dense CRFs and Bilateral Neural Networks

Varun Jampani[1], Martin Kiefel[1,2] and Peter V. Gehler[1,2]
[1]Max Planck Institute for Intelligent Systems, Tübingen, 72076, Germany
[2]Bernstein Center for Computational Neuroscience, Tübingen, 72076, Germany
{varun.jampani, martin.kiefel, peter.gehler}@tuebingen.mpg.de

This supplementary material contains a more detailed overview of the permutohedral lattice convolution in Section 1, more experiments in Section 2 and additional results with experiment protocols for the experiments presented before in Section 3.

1. General Permutohedral Convolutions

A core technical contribution of this work is the generalization of the Gaussian permutohedral lattice convolution proposed in [1] to the full non-separable case with the ability to perform backpropagation. Although, conceptually, there are minor difference between non-Gaussian and general parameterized filters, there are non-trivial practical differences in terms of the algorithmic implementation. The Gauss filters belong to the separable class and can thus be decomposed into multiple sequential one dimensional convolutions. We are interested in the general filter convolutions, which can not be decomposed. Thus, performing a general permutohedral convolution at a lattice point requires the computation of the inner product with the neighboring elements in all the directions in the high-dimensional space.

Here, we give more details of the implementation differences of separable and non-separable filters. In the following we will explain the scalar case first. Recall, that the forward pass of general permutohedral convolution involves 3 steps: *splatting*, *convolving* and *slicing*. We follow the same splatting and slicing strategies as in [1] since these operations do not depend on the filter kernel. The main difference between our work and the existing implementation of [1] is the way that the convolution operation is executed. This proceeds by constructing a *blur neighbor* matrix K that stores for every lattice point all values of the lattice neighbors that are needed to compute the filter output.

The blur neighbor matrix is constructed by traversing through all the populated lattice points and their neighboring elements. This is done recursively to share computations. For any lattice point, the neighbors that are n hops away are the direct neighbors of the points that are $n-1$ hops away. The size of a d dimensional spatial filter with width $s+1$ is $(s+1)^d$ (*e.g.*, a 3×3 filter, $s=2$ in $d=2$ has $3^2=9$ elements) and this size grows exponentially in the number of dimensions d. The permutohedral lattice is constructed by projecting a regular grid onto the plane spanned by the d dimensional normal vector $(1,\ldots,1)^\top$. See Fig. 1

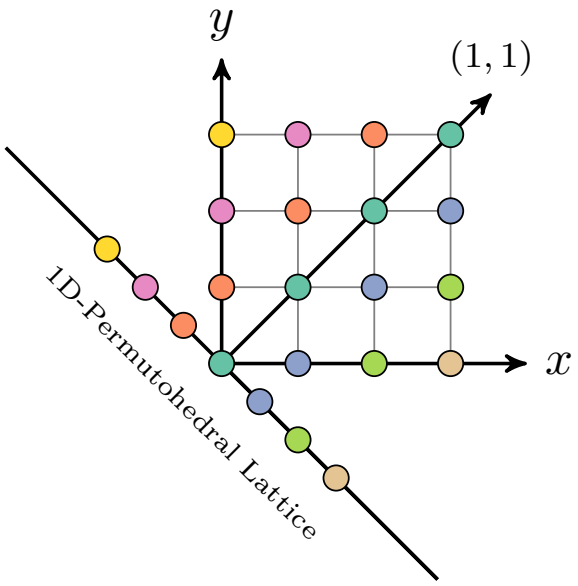


Figure 1. **Illustration of 1D permutohedral lattice construction.** A 4×4 (x,y) grid lattice is projected onto the plane defined by the normal vector $(1,1)^\top$. This grid has $s+1=4$ and $d=2$ $(s+1)^d=4^2=16$ elements. In the projection, all points of the same color are projected onto the same points in the plane. The number of elements of the projected lattice is $t=(s+1)^d-s^d=4^2-3^2=7$, that is the (4×4) grid minus the size of lattice that is 1 smaller at each size, in this case a (3×3) lattice (the upper right (3×3) elements).

for an illustration of 1D lattice construction. Many corners of a grid filter are projected onto the same point, in total $t = (s+1)^d - s^d$ elements remain in the permutohedral filter with s neighborhood in $d-1$ dimensions. If the lattice has m populated elements, the matrix K has size $t \times m$. Note that, since the input signal is typically sparse, only a few lattice corners are being populated in the *slicing* step. We use a hash-table to keep track of these points and traverse only through the populated lattice points for this neighborhood matrix construction.

Once the blur neighbor matrix K is constructed, we can perform the convolution by the matrix vector multiplication

$$l' = BK, \tag{1}$$

where B is the $1 \times t$ filter kernel (whose values we will learn) and $l' \in \mathbb{R}^{1 \times m}$ is the result of the filtering at the m lattice points. In practice, we found that the matrix K is sometimes too large to fit into GPU memory and we divided the matrix K into smaller pieces to compute Eq. 1 sequentially.

In the general multi-dimensional case, the signal l is of c dimensions. Then the kernel is of size $B \times t$ and K stores the c dimensional vectors accordingly. When the input and output points are different, we slice only the input points and splat only at the output points.

2. Additional Experiments

In this section we discuss more use-cases for the learned bilateral filters, one use-case of BNNs and two single filter applications for image and 3D mesh denoising.

2.1. Recognition of subsampled MNIST

One of the strengths of the proposed filter convolution is that it does not require the input to lie on a regular grid. The only requirement is to define a distance between features of the input signal. We highlight this feature with the following experiment using the classical MNIST ten class classification problem [12]. We sample a sparse set of N points $(x, y) \in [0, 1] \times [0, 1]$ uniformly at random in the input image, use their interpolated values as signal and the *continuous* (x, y) positions as features. This mimics subsampling of a high-dimensional signal. To compare against a spatial convolution, we interpolate the sparse set of values at the grid positions.

We take a reference implementation of LeNet [11] that is part of the Caffe project [10] and compare it against the same architecture but replacing the first convolutional layer with a bilateral convolution layer (BCL). The filter size and numbers are adjusted to get a comparable number of parameters (5×5 for LeNet, 2-neighborhood for BCL).

The results are shown in Table 1. We see that training on the original MNIST data (column Original, LeNet vs. BNN) leads to a slight decrease in performance of the BNN (99.03%) compared to LeNet (99.19%). The BNN can be trained and evaluated on sparse signals, and we resample the image as described above for $N = 100\%$, 60% and 20% of the total number of pixels. The methods are also evaluated on test images that are subsampled in the same way. Note that we can train and test with different subsampling rates. We introduce an additional bilinear interpolation layer for the LeNet architecture to train on the same data. In essence, both models perform a spatial interpolation and thus we expect them to yield a similar classification accuracy. Once the data is of higher dimensions the permutohedral convolution will be faster due to hashing the sparse input points, as well as less memory demanding in comparison to naive application of a spatial convolution with interpolated values.

		Test Subsampling		
Method	Original	100%	60%	20%
LeNet	**0.9919**	0.9660	0.9348	**0.6434**
BNN	0.9903	**0.9844**	**0.9534**	0.5767
LeNet 100%	0.9856	0.9809	0.9678	**0.7386**
BNN 100%	**0.9900**	**0.9863**	**0.9699**	0.6910
LeNet 60%	0.9848	0.9821	0.9740	0.8151
BNN 60%	**0.9885**	**0.9864**	**0.9771**	**0.8214**
LeNet 20%	**0.9763**	**0.9754**	0.9695	0.8928
BNN 20%	0.9728	0.9735	**0.9701**	**0.9042**

Table 1. Classification accuracy on MNIST. We compare the LeNet [11] implementation that is part of Caffe [10] to the network with the first layer replaced by a bilateral convolution layer (BCL). Both are trained on the original image resolution (first two rows). Three more BNN and CNN models are trained with randomly subsampled images (100%, 60% and 20% of the pixels). An additional bilinear interpolation layer samples the input signal on a spatial grid for the CNN model.

2.2. Image Denoising

The main application that inspired the development of the bilateral filtering operation is image denoising [3], there using a single Gaussian kernel. Our development allows to learn this kernel function from data and we explore how to improve using a *single* but more general bilateral filter.

We use the Berkeley segmentation dataset (BSDS500) [2] as a test bed. The color images in the dataset are converted to gray-scale, and corrupted with Gaussian noise with a standard deviation of $\frac{25}{255}$.

We compare the performance of four different filter models on a denoising task. The first baseline model ("Spatial" in Table 2, 25 weights) uses a single spatial filter with a kernel size of 5 and predicts the scalar gray-scale value at the center pixel. The next model ("Gauss Bilateral") applies a bilateral *Gaussian* filter to the noisy input, using position and intensity features $\mathbf{f} = (x, y, v)^\top$. The third setup ("Learned Bilateral", 65 weights) takes a Gauss kernel as initialization and fits all filter weights on the "train" image

Figure 2. **Sample data for 3D mesh denoising.** (top) Some 3D body meshes sampled from [13] and (bottom) the corresponding noisy meshes used in denoising experiments.

set to minimize the mean squared error with respect to the clean images. We run a combination of spatial and permutohedral convolutions on spatial and bilateral features ("Spatial + Bilateral (Learned)") to check for a complementary performance of the two convolutions.

Method	PSNR
Noisy Input	20.17
Spatial	26.27
Gauss Bilateral	26.51
Learned Bilateral	26.58
Spatial + Bilateral (Learned)	26.65

Table 2. PSNR results of a denoising task using the BSDS500 dataset [2]

The PSNR scores evaluated on full images of the "test" image set are shown in Table 2. We find that an untrained bilateral filter already performs better than a trained spatial convolution (26.27 to 26.51). A learned convolution further improve the performance slightly. We chose this simple one-kernel setup to validate an advantage of the generalized bilateral filter. A competitive denoising system would employ RGB color information and also needs to be properly adjusted in network size. Multi-layer perceptrons have obtained state-of-the-art denoising results [4] and the permutohedral lattice layer can readily be used in such an architecture, which is intended future work.

Figure 3. **4D isomap features for 3D human bodies.** Visualization of 4D isomap features for a sample 3D mesh. Isomap feature values are overlaid onto mesh vertices.

2.3. 3D Mesh Denoising

Permutohedral convolutions can naturally be extended to higher (> 2) dimensional data. To highlight this, we use the proposed convolution for the task of denoising 3D meshes.

We sample 3D human body meshes using a generative 3D body model from [13]. To the clean meshes, we add Gaussian random noise displacements along the surface normal at each vertex location. Figure 2 shows some sample 3D meshes sampled from [13] and corresponding noisy meshes. The task is to take the noisy meshes as inputs and recover the original 3D body meshes. We create 1000 training, 200 validation and another 500 testing examples for the experiments.

Mesh Representation: The 3D human body meshes from [13] are represented with 3D vertex locations and the edge connections between the vertices. We found that this signal representation using global 3D coordinates is not suitable for denoising with bilateral filtering. Therefore, we first smooth the noisy mesh using mean smoothing applied to the face normals [17] and represent the noisy mesh vertices as 3D vector displacements with respect to the corresponding smoothed mesh. Thus, the task becomes denoising the 3D vector displacements with respect to the smoothed mesh.

Isomap Features: To apply permutohedral convolution, we need to define features at each input vertex point. We use 4 dimensional isomap embedding [16] of the given 3D mesh graph as features. The given 3D mesh is converted into weighted edge graph with edge weights set to Euclidean distance between the connected vertices and to infinity between the non-connected vertices. Then 4 dimensional isomap embedding is computed for this weighted edge graph using a publicly available implementation [15]. Fig. 3 shows the visualization of isomap features on a sample 3D mesh.

	Noisy Mesh	Normal Smoothing	Gauss Bilateral	Learned Bilateral
Vertex Distance (RMSE)	5.774	3.183	2.872	**2.825**
Normal Angle Error	19.680	19.707	19.357	**19.207**

Table 3. **Body Denoising.** Vertex distance RMSE values and normal angle error (in degrees) corresponding to different denoising strategies averaged over 500 test meshes.

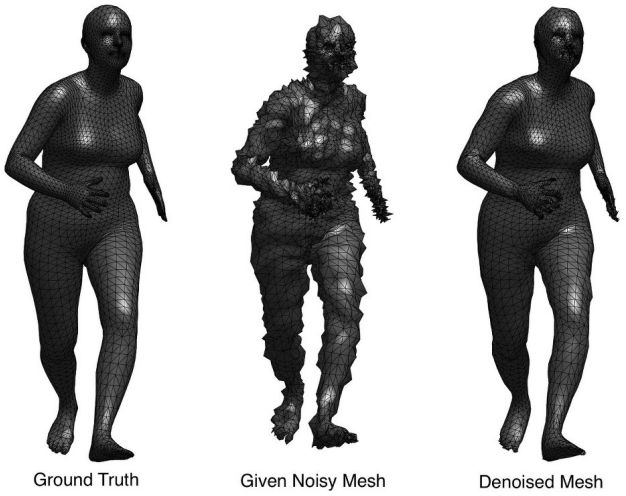


Figure 4. **Sample Denoising Result.** Ground truth mesh (left), corresponding given noisy mesh (middle) and the denoised result (right) using learned bilateral filter.

Experimental Results: Mesh denoising with a bilateral filter proceeds by splatting the input 3D mesh vectors (displacements with respect to smoothed mesh) into the 4D isomap feature space, filtering the signal in this 4D space and then slicing back into original 3D input space. The Table 3 shows quantitative results as RMSE for different denoising strategies. The normal smoothing [17] already reduces the RMSE. The Gauss bilateral filter results in significant improvement over normal smoothing with and learning the filter weights again improves the result. A visual result is shown in Figure 4.

3. Additional results

This section contains more qualitative results for the experiments of the main paper.

3.1. Lattice Visualization

Figure 5 shows sample lattice visualizations for different feature spaces.

3.2. Color Upsampling

In addition to the experiments discussed in the main paper, we performed the cross-factor analysis of training and testing at different upsampling factors. Table 4 shows the PSNR results for this analysis. Although, in terms of PSNR, it is optimal to train and test at the same upsampling factor, the differences are small when training and testing upsampling factors are different. Some images of the upsampling for the Pascal VOC12 dataset are shown in Fig. 6. It is especially the low level image details that are better preserved with a learned bilateral filter compared to the Gaussian case.

		Test Factor			
		2×	**4×**	**8×**	**16×**
Train Factor	**2×**	**38.45**	36.12	34.06	32.43
	4×	38.40	**36.16**	**34.08**	32.47
	8×	38.40	36.15	**34.08**	32.47
	16×	38.26	36.13	34.06	**32.49**

Table 4. **Color Upsampling with different train and test upsampling factors.** PSNR values corresponding to different upsampling factors used at train and test times on the 2 megapixel image dataset, using our learned bilateral filters.

3.3. Depth Upsampling

Figure 7 presents some more qualitative results comparing bicubic interpolation, Gauss bilateral and learned bilateral upsampling on NYU depth dataset image [14].

3.4. Character Recognition

Figure 8 shows the schematic of different layers of the network architecture for LeNet-7 [12] and DeepCNet(5, 50) [6, 8]. For the BNN variants, the first layer filters are replaced with learned bilateral filters and are learned end-to-end.

3.5. Semantic Segmentation

Some more visual results for semantic segmentation are shown in Figure 9. These include the underlying DeepLab CNN[5] result (DeepLab), the 2 step mean-field result with Gaussian edge potentials (+2stepMF-GaussCRF) and also corresponding results with learned edge potentials (+2stepMF-LearnedCRF). In general, we observe that mean-field in learned CRF leads to slightly dilated classification regions in comparison to using Gaussian CRF thereby filling-in the false negative pixels and also correcting some mis-classified regions.

3.6. Material Segmentation

In Fig. 10, we present visual results comparing 2 step mean-field inference with Gaussian and learned pairwise CRF potentials. In general, we observe that the pixels belonging to dominant classes in the training data are being more accurately classified with learned CRF. This leads to a significant improvements in overall pixel accuracy. This also results in a slight decrease of the accuracy from less frequent class pixels thereby slightly reducing the average class accuracy with learning. We attribute this to the type of

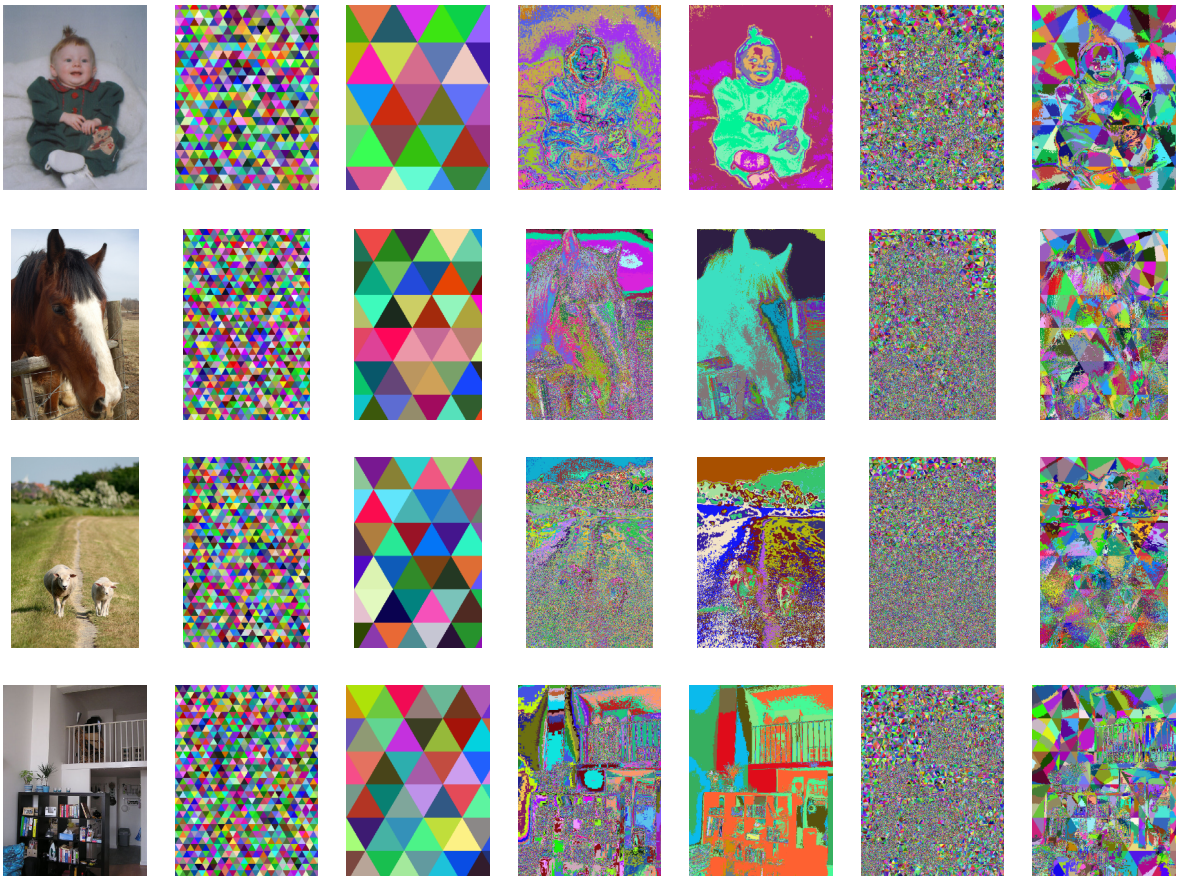
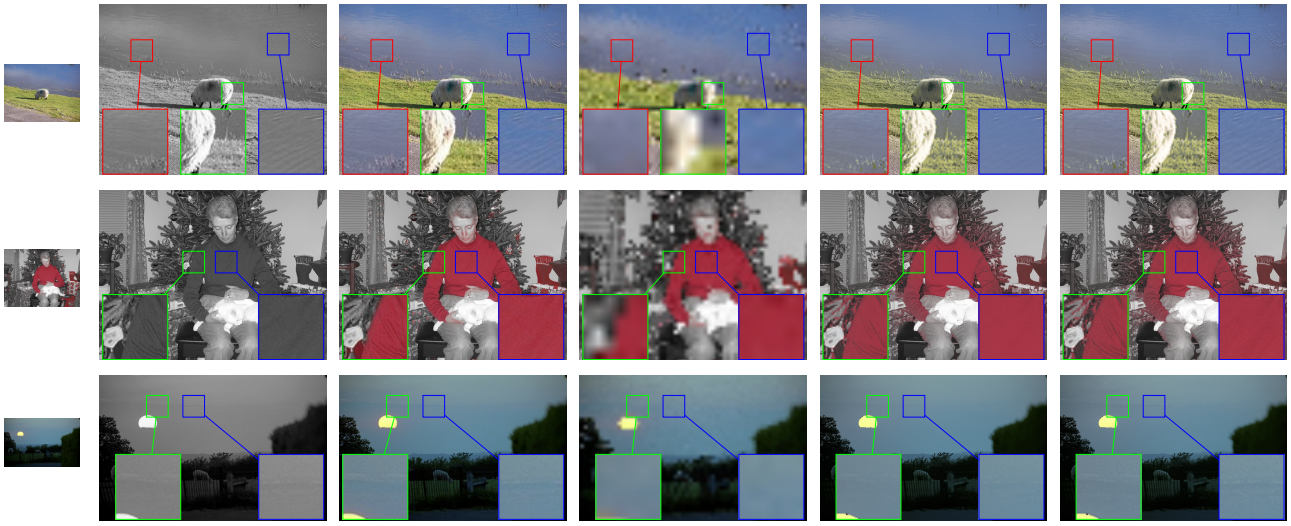
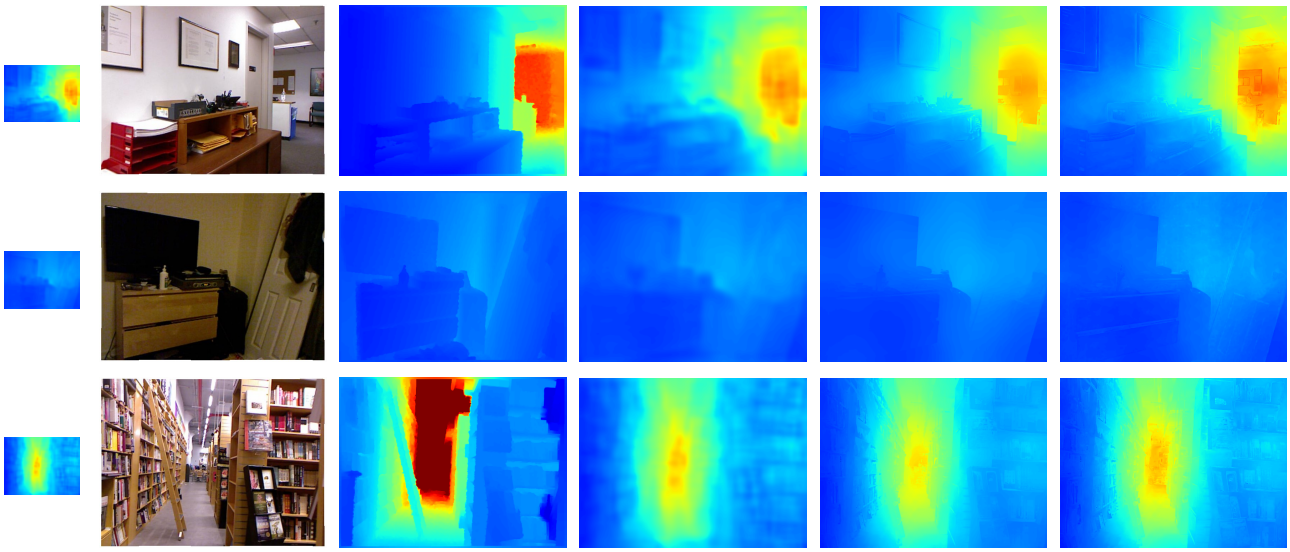


Figure 5. **Visualization of the Permutohedral Lattice.** Sample lattice visualizations for different feature spaces. All pixels falling in the same simplex cell are shown with the same color. (x, y) features correspond to image pixel positions, and $(r, g, b) \in [0, 255]$ correspond to the red, green and blue color values.



(a) Input (b) Gray Guidance (c) Ground Truth (d) Bicubic Interpolation (e) Gauss Bilateral (f) Learned Bilateral

Figure 6. **Color Upsampling.** Color $8\times$ upsampling results using different methods (best viewed on screen).



(a) Input (b) Guidance (c) Ground Truth (d) Bicubic Interpolation (e) Gauss Bilateral (f) Learned Bilateral

Figure 7. **Depth Upsampling.** Depth $8\times$ upsampling results using different upsampling strategies.

annotation that is available for this dataset, which is not for the entire image but for some segments in the image. We have very few images of the infrequent classes to combat this behaviour during training.

3.7. Experiment Protocols

Table 5 shows experiment protocols of different experiments.

References

[1] A. Adams, J. Baek, and M. A. Davis. Fast high-dimensional filtering using the permutohedral lattice. In *Computer Graphics Forum*, volume 29, pages 753–762. Wiley Online Library, 2010.

[2] P. Arbeláez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.

[3] V. Aurich and J. Weule. Non-linear Gaussian filters performing edge preserving diffusion. In *DAGM*, pages 538–545. Springer, 1995.

[4] H. C. Burger, C. J. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with BM3D? In *Computer Vision and Pattern Recognition (CVPR)*, 2012.

[5] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected CRFs. In *International Conference on Learning Representations (ICLR)*, 2015.

[6] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR)*, pages 3642–3649, 2012.

[7] M. Everingham, L. V. Gool, C. Williams, J. Winn, and A. Zisserman. The PASCAL VOC2012 challenge results. 2012.

[8] B. Graham. Spatially-sparse convolutional neural networks. *arXiv preprint arXiv:1409.6070*, 2014.

[9] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. In *International Conference on Computer Vision (ICCV)*, 2011.

[10] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[12] Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits, 1998.

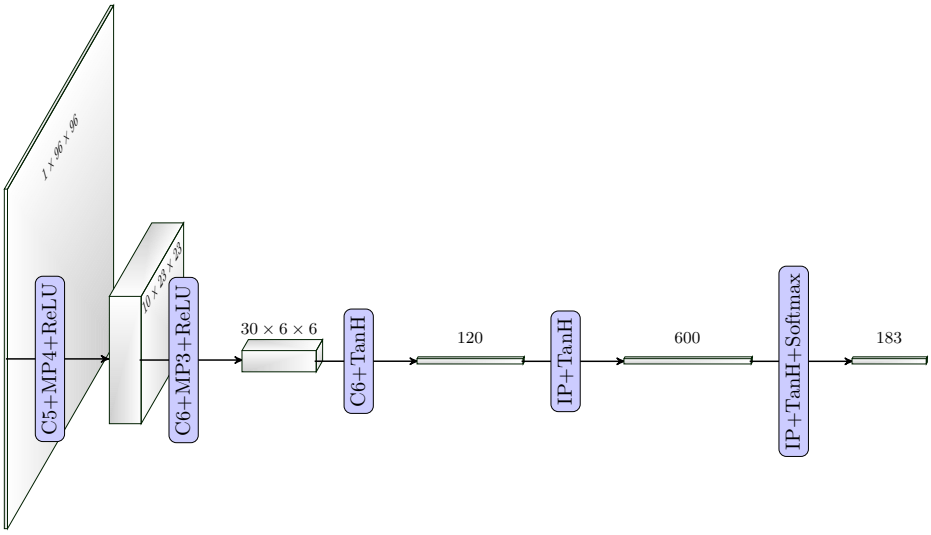
[13] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black. SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16, Oct. 2015.

[14] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision (ECCV)*, pages 746–760. Springer, 2012.

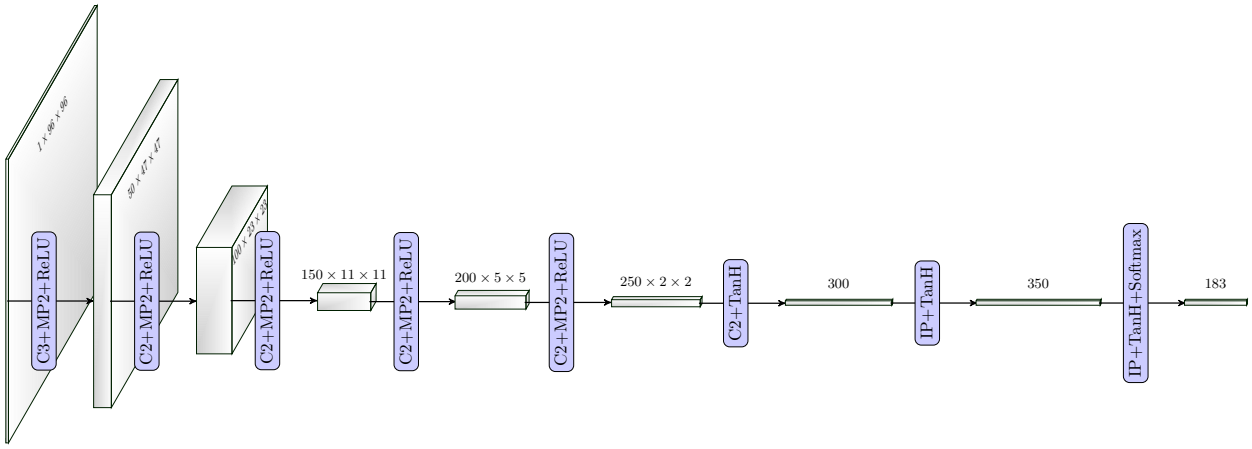
[15] J. B. Tenenbaum. A Global Geometric Framework for Nonlinear Dimensionality Reduction. http://isomap.stanford.edu/, 2000. [Online; accessed 12-October-2015].

[16] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[17] H. Yagou, Y. Ohtake, and A. Belyaev. Mesh smoothing via mean and median filtering applied to face normals. In *Geometric Modeling and Processing, 2002. Proceedings*, pages 124–131. IEEE, 2002.

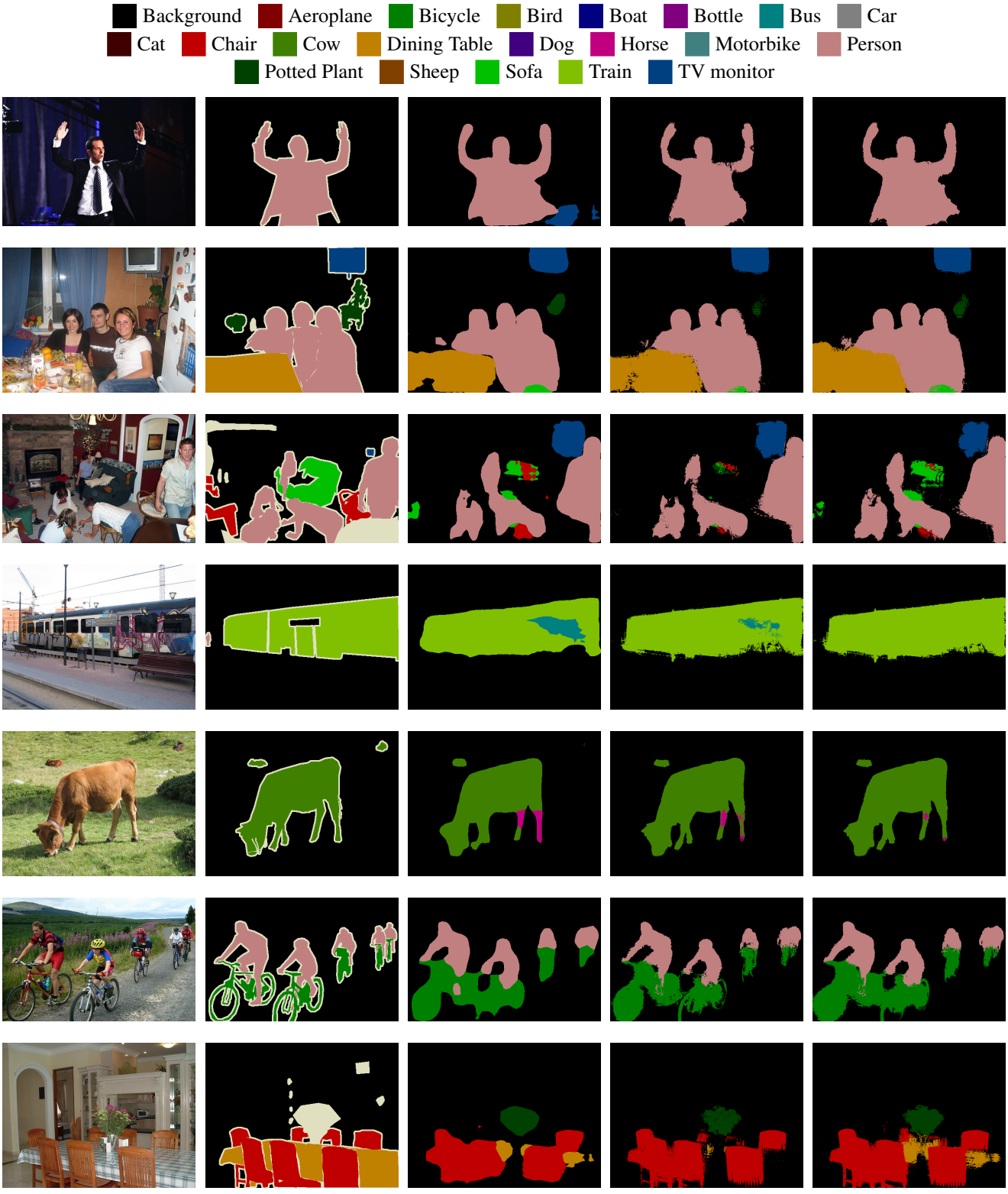


(a) LeNet-7



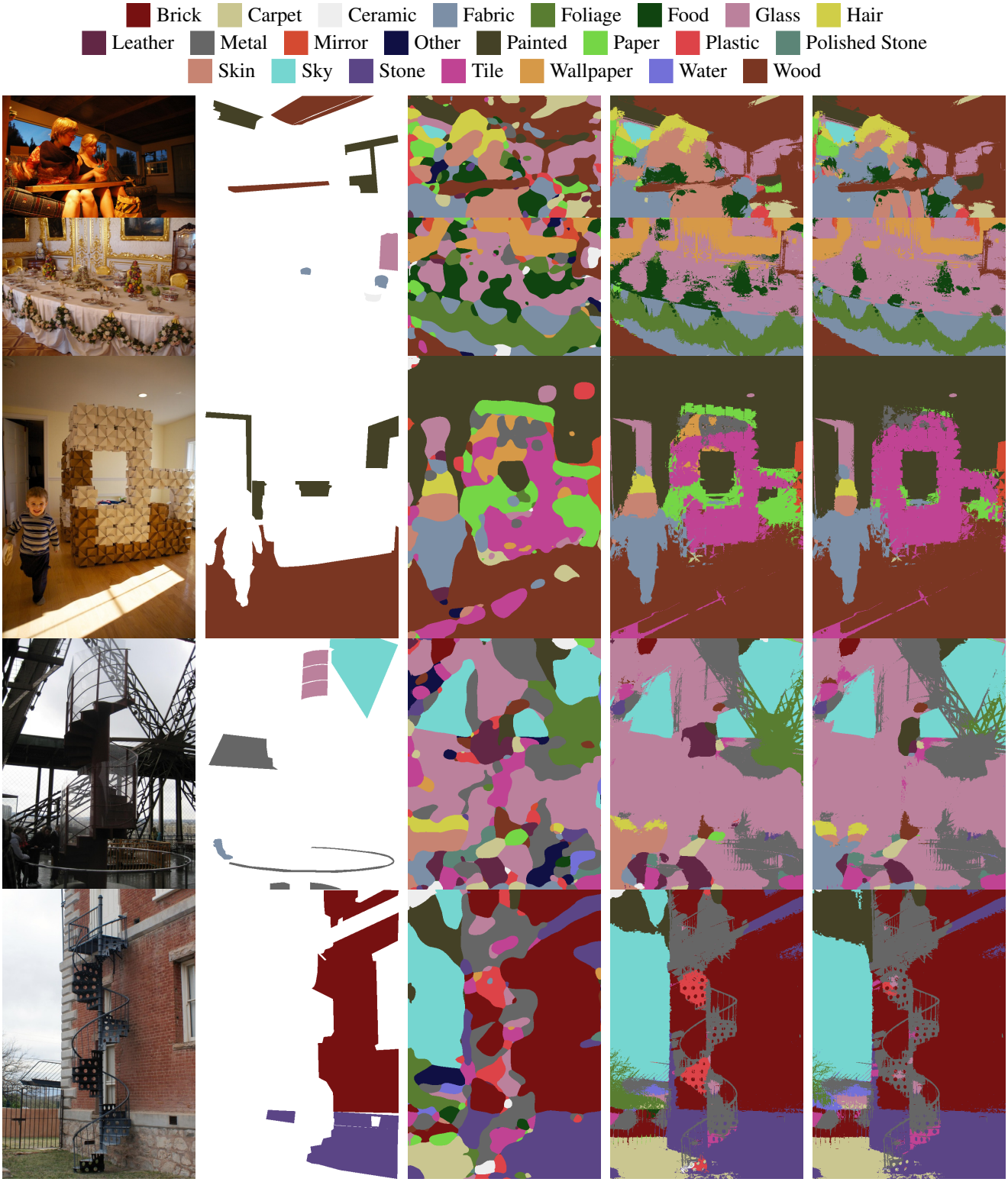
(b) DeepCNet

Figure 8. **CNNs for Character Recognition.** Schematic of (top) LeNet-7 [12] and (bottom) DeepCNet(5,50) [6, 8] architectures used in Assamese character recognition experiments.



(a) Input (b) Ground Truth (c) DeepLab (d) +2stepMF-GaussCRF (e) +2stepMF-LearnedCRF

Figure 9. **Semantic Segmentation.** Example results of semantic segmentation. (c) depicts the unary results before application of MF, (d) after two steps of MF with Gaussian edge CRF potentials, (e) after two steps of MF with learned edge CRF potentials.



(a) Input (b) Ground Truth (c) DeepLab (d) +2stepMF-GaussCRF (e) +2stepMF-LearnedCRF

Figure 10. **Material Segmentation.** Example results of material segmentation. (c) depicts the unary results before application of MF, (d) after two steps of MF with Gaussian edge CRF potentials, (e) after two steps of MF with learned edge CRF potentials.

					Data Statistics			Training Protocol			
Experiment	**Feature Types**	**Feature Scales**	**Filter Size**	**Filter Nbr.**	**Train**	**Val.**	**Test**	**Loss Type**	**LR**	**Batch**	**Epochs**
Single Bilateral Filter Applications											
2× Color Upsampling	Position_1, Intensity (3D)	0.13, 0.17	65	2	10581	1449	1456	MSE	1e-06	200	94.5
4× Color Upsampling	Position_1, Intensity (3D)	0.06, 0.17	65	2	10581	1449	1456	MSE	1e-06	200	94.5
8× Color Upsampling	Position_1, Intensity (3D)	0.03, 0.17	65	2	10581	1449	1456	MSE	1e-06	200	94.5
16× Color Upsampling	Position_1, Intensity (3D)	0.02, 0.17	65	2	10581	1449	1456	MSE	1e-06	200	94.5
Depth Upsampling	Position_1, Color (5D)	0.05, 0.02	665	2	795	100	654	MSE	1e-07	50	251.6
Mesh Denoising	Isomap (4D)	46.00	63	2	1000	200	500	MSE	100	10	100.0
DenseCRF Applications											
Semantic Segmentation											
- 1step MF	Position_1, Color (5D); Position_1 (2D)	0.01, 0.34; 0.34	665; 19	2; 2	10581	1449	1456	Logistic	0.1	5	1.4
- 2step MF	Position_1, Color (5D); Position_1 (2D)	0.01, 0.34; 0.34	665; 19	2; 2	10581	1449	1456	Logistic	0.1	5	1.4
- *loose* 2step MF	Position_1, Color (5D); Position_1 (2D)	0.01, 0.34; 0.34	665; 19	2; 2	10581	1449	1456	Logistic	0.1	5	+1.9
Material Segmentation											
- 1step MF	Position_2, Lab-Color (5D)	5.00, 0.05, 0.30	665	2	928	150	1798	Weighted Logistic	1e-04	24	2.6
- 2step MF	Position_2, Lab-Color (5D)	5.00, 0.05, 0.30	665	2	928	150	1798	Weighted Logistic	1e-04	12	+0.7
- *loose* 2step MF	Position_2, Lab-Color (5D)	5.00, 0.05, 0.30	665	2	928	150	1798	Weighted Logistic	1e-04	12	+0.2
Neural Network Applications											
Tiles: CNN-9×9	-	-	81	4	10000	1000	1000	Logistic	0.01	100	500.0
Tiles: CNN-13×13	-	-	169	6	10000	1000	1000	Logistic	0.01	100	500.0
Tiles: CNN-17×17	-	-	289	8	10000	1000	1000	Logistic	0.01	100	500.0
Tiles: CNN-21×21	-	-	441	10	10000	1000	1000	Logistic	0.01	100	500.0
Tiles: BNN	Position_1, Color (5D)	0.05, 0.04	63	1	10000	1000	1000	Logistic	0.01	100	30.0
LeNet	-	-	25	2	5490	1098	1647	Logistic	0.1	100	182.2
Crop-LeNet	-	-	25	2	5490	1098	1647	Logistic	0.1	100	182.2
BNN-LeNet	Position_2 (2D)	20.00	7	1	5490	1098	1647	Logistic	0.1	100	182.2
DeepCNet	-	-	9	1	5490	1098	1647	Logistic	0.1	100	182.2
Crop-DeepCNet	-	-	9	1	5490	1098	1647	Logistic	0.1	100	182.2
BNN-DeepCNet	Position_2 (2D)	40.00	7	1	5490	1098	1647	Logistic	0.1	100	182.2

Table 5. **Experiment Protocols.** Experiment protocols for the different experiments presented in this work. **Feature Types**: Feature spaces used for the bilateral convolutions. Position_1 corresponds to un-normalized pixel positions whereas Position_2 corresponds to pixel positions normalized to $[0, 1]$ with respect to the given image. **Feature Scales**: Cross-validated scales for the features used. **Filter Size**: Number of elements in the filter that is being learned. **Filter Nbr.**: Half-width of the filter. **Train**, **Val.** and **Test** corresponds to the number of train, validation and test images used in the experiment. **Loss Type**: Type of loss used for back-propagation. "MSE" corresponds to Euclidean mean squared error loss and "Logistic" corresponds to multinomial logistic loss. "Weighted Logistic" is the class-weighted multinomial logistic loss. We weighted the loss with inverse class probability for material segmentation task due to the small availability of training data with class imbalance. **LR**: Fixed learning rate used in stochastic gradient descent. **Batch**: Number of images used in one parameter update step. **Epochs**: Number of training epochs. In all the experiments, we used fixed momentum of 0.9 and weight decay of 0.0005 for stochastic gradient descent. "'Color Upsampling" experiments in this Table corresponds to those performed on Pascal VOC12 dataset images. For all experiments using Pascal VOC12 images, we use extended training segmentation dataset available from [9], and used standard validation and test splits from the main dataset [7].