Supplementary Material: Visual Boundary Prediction: A Deep Neural Prediction Network and Quality Dissection

Jyri J. Kivinen, Christopher K. I. Williams, Nicolas Heess

A Architecture Illustrations

Figure I illustrates a diagonally-tiled convolutional mcRBM (TmcRBM) model instance (with different receptive field sizes and stride than what are used in our experiments). Figure II illustrates a diagonally-tiled convolutional mcDBN (TmcDBN) model instance (note that in the illustration the receptive fields are much smaller than in the model considered in our experiments, and the connections are drawn only from few selected hidden units to their receptive fields). Figure III illustrates a two-stream model for contour prediction.

Figure IV illustrates two three-stream networks and their respective deep-only network. These were not studied in this paper.

B Training details

B.1 Generative pretraining

The mean and covariance filters of the TmcRBMs were initialized to small random values, the mean and covariance biases were set to -2 and to 2, respectively, and their learning rates were globally scaled versions of 0.05,0.0025,0.0025,0.0005, respectively, similar to in Ranzato and Hinton [2010]. Only the regular parameter learning rates were annealed, using $\frac{1}{t}$-type annealing, starting at epoch 200, and ending at 0.25 of the initial rates by the end of training at epoch 800. The 32 negative particles (of size 142×142) were drawn and updated by a single step of HMC-sampling with 20 Leapfrog steps, with step-size set automatically to maintain 90 percent sample accept rate according to an exponentially weighted moving average with smoothing factor of 0.9. We used a L_1-weight decay, with rate 0.001.

The mcDBNs were trained in the usual greedy manner, layer-by-layer, using exactly the same hyperparameter settings as above, except the all of parameters had the same learning rate, 0.00025 divided by the number of feature plane grid locations (as in the shallow models).

B.2 Discriminative training

The shallow model parameters had equal learning rates, starting from $\alpha = \frac{0.05\cdot\beta}{M\cdot T}$, where M denotes the batch size, and T the number of sites the parameter was shared over per image. β was set to 1 and 10 for the feature extraction and read-out parameters, respectively. In the deeper models, the learning rates in the shallow stream were set to 0.5 times those of the above, and those for the deeper stream parameters were set with $\beta = 5$, and setting T as for the shallow layer.

B.3 Tuning the prediction performance

Our full prediction model (called Enhanced Two-Stream) uses five 'tunings' for effective prediction performance. Two of the modifications involve changes in setting the batch data for training: we standardize extracted the image patches[1], subtracting their mean from them and dividing by their standard deviation; we also set the training labels for each site as the average of the annotations[2]. The third modification encourages hidden unit activation levels (measured as an exponentially weighted moving average of the observed levels with smoothing factor of 0.9) to be at/near specific levels, by adding to the objective function cross-entropy penalties between the target and measured levels. This was applied to the final hidden unit layer in the deep stream with target activation level of 0.1, with non-sparsity penalty of 0.001, and also to the mean hidden units (with target activation level of 0.1 (grey-domain)/0.025 (colour-domain), with non-sparsity penalty of 0.01). The fourth modification, called rotation-averaging, applies prediction with the network of rotated versions (16 angles in total around the clock) of the images, back-rotates the results and averages them. We note that the resulting network prediction is still expected not to be fully transformation-equivariant but this approach was effective for our purposes. Finally, as done on several

[1] this makes the network more robust to data variations (shifts and scaling in global intensity within the extracted training patches) not considered to be relevant to the problem of deciding whether a site should be a boundary or not

[2] This makes optimizing the main objective faster.

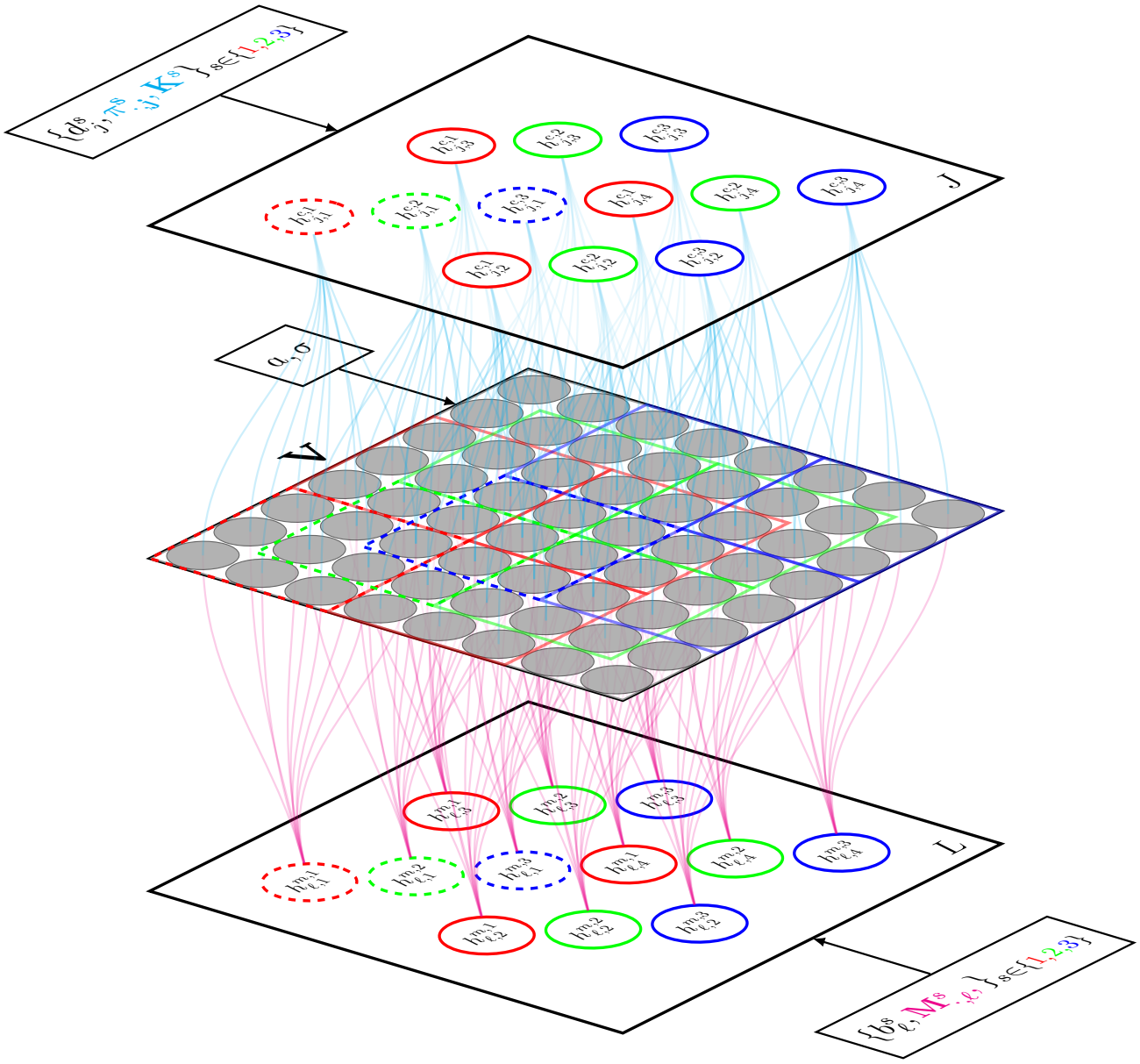

Figure I: A graphical illustration of a diagonally-tiled-convolutional mcRBM. The visible units visualized in the middle layer connect to the J covariance hidden unit layers of the model at the top, and to the L mean hidden unit layers of the model at the bottom. Within these layers the hidden units are partitioned into different sets (red,green,blue), associated with different parameters for their hidden units. Each of the hidden units connect to a region of visible units, and the filter applications within a set are non-overlapping and tile a certain-sized region of visible units, and those of different sets are offset diagonally with a stride between the neighboring sets.
.

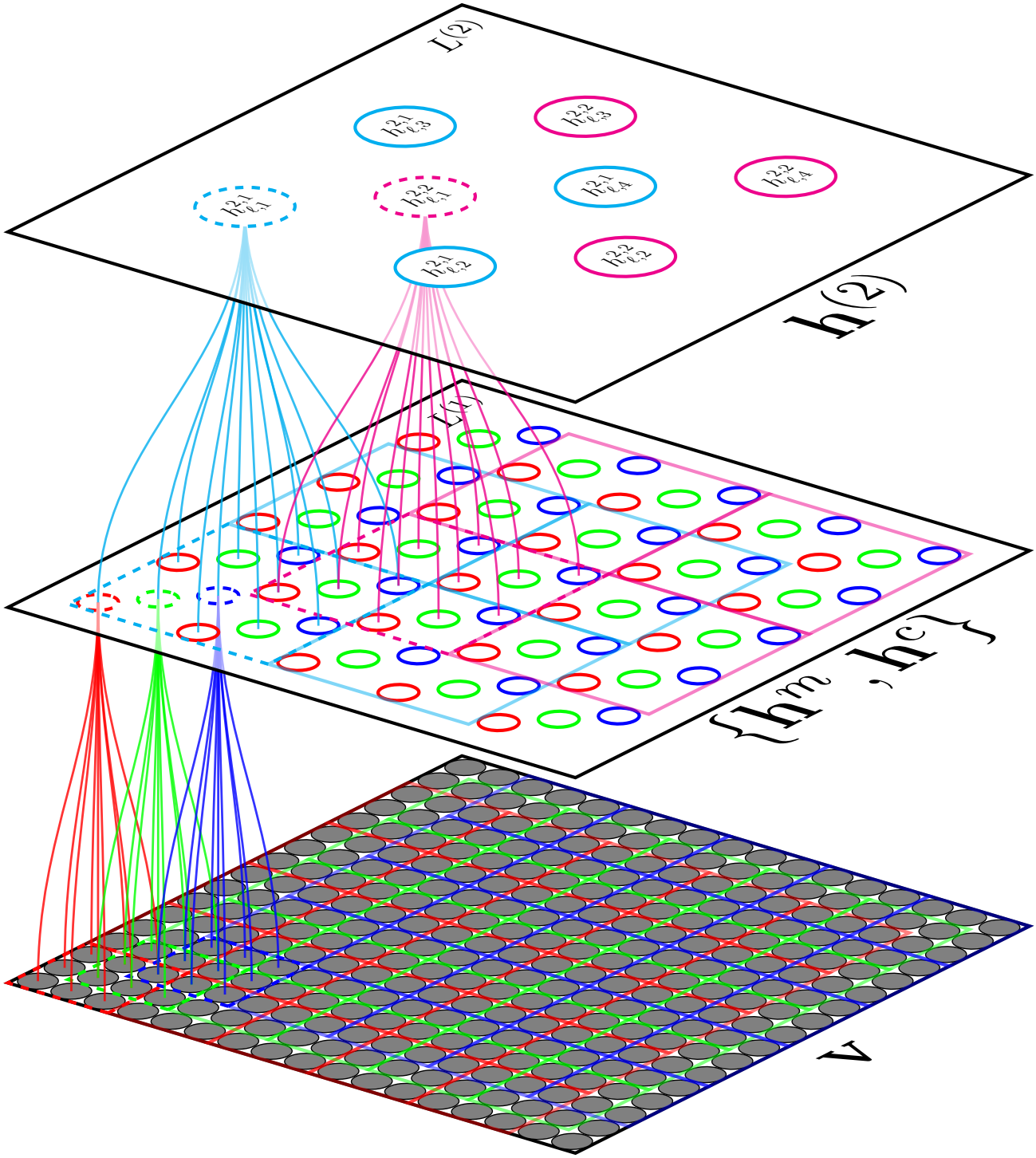

Figure II: A graphical illustration of a diagonally-tiled-convolutional mcDBN. The visible units $\mathbf{v}$ visualized in the bottom layer connect to the covariance hidden unit $\mathbf{h}^c$ and mean hidden unit $\mathbf{h}^m$ layers ($L^{(1)}$ feature planes in total) in the middle, which connect to the second layer hidden units at the top (having $L^{(2)}$ feature planes). Within the hidden unit layers, the units are partitioned into different sets (red,green, and blue in the first layer, and cyan and magenta in the second layer), associated with different parameters for their hidden units. Each of the hidden units connect to a region of units below, and the filter applications within a set are non-overlapping and tile a certain-sized region of units, and those of different sets are offset diagonally with a stride between the neighboring sets. Connections are drawn only from few selected hidden units to their receptive fields.

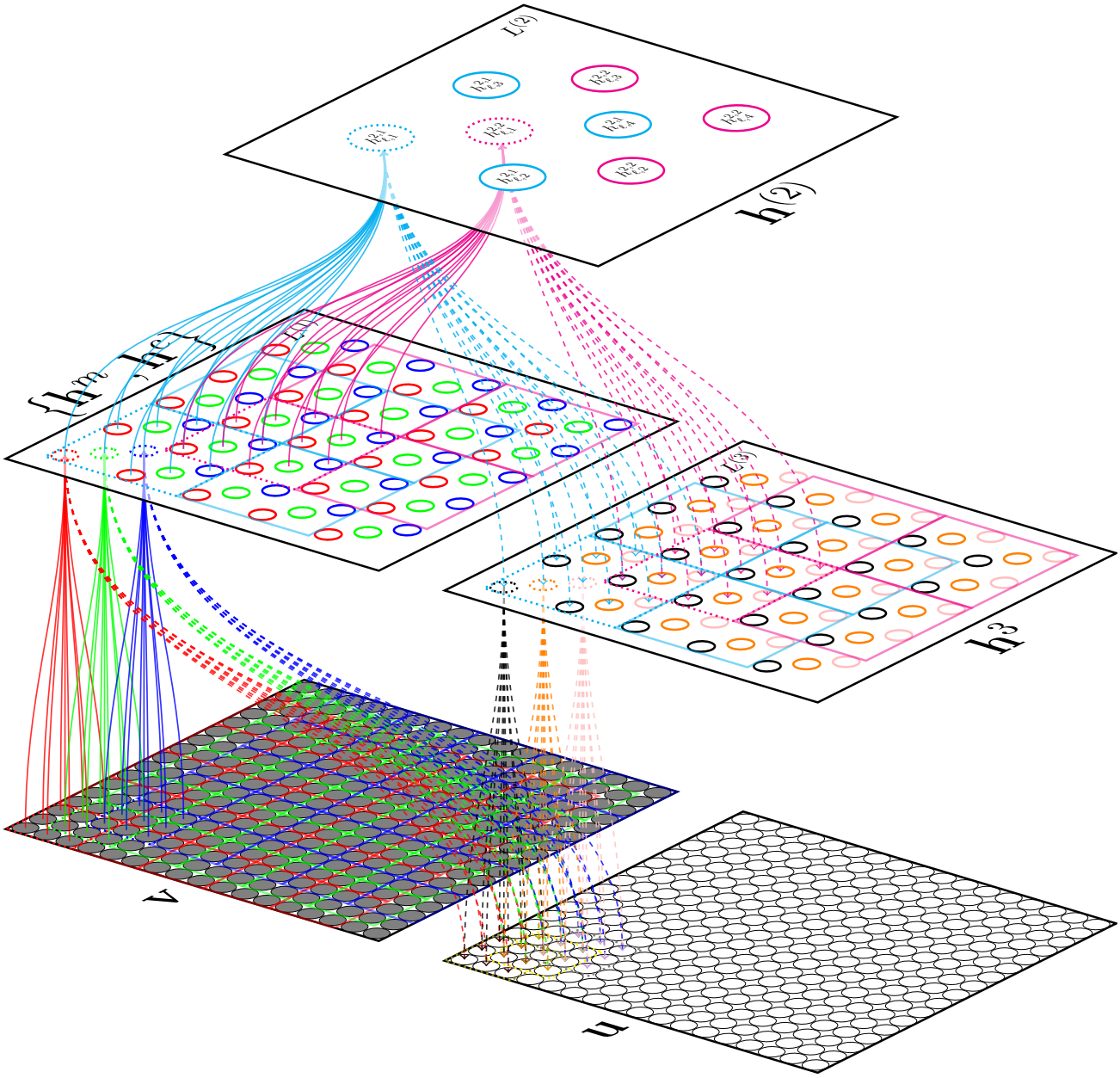

Figure III: A graphical illustration of a two-stream model for contour prediction. The visible units $\mathbf{v}$ in bottom left send information to the covariance and mean hidden units directly above, which send information to the second layer hidden units at the top and also to the contour units $\mathbf{u}$ at the bottom right. The second layer hidden units send information to the hidden units $\mathbf{h}^3$ below right, which send information to the contour units below. As before, within the hidden unit layers, the units are partitioned into different sets, associated with different parameters for their hidden units. Receptive and send connections are shown only for few of the hidden units.

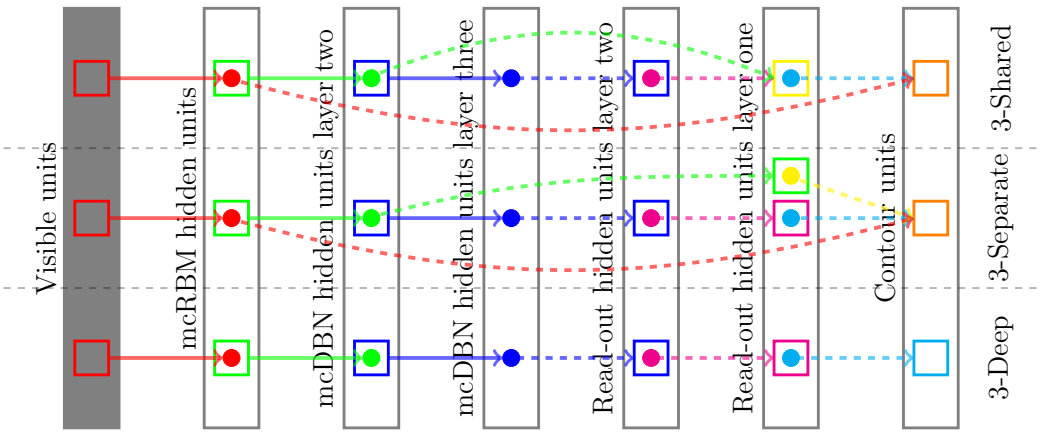

Figure IV: 3-Stream Deep Networks. Layer dots illustrate hidden units, blocks receptive fields (no size-consistency). The solid and dashed arrows denote feature extraction and read-out parameters, respectively.

other works Dollár and Zitnick [2013], Mairal et al. [2008], we apply non-maximum suppression by Canny.

C Boundary Prediction Statistics

C.1 Quality Assessment via MSSIM-scores

Table I reports statistics based on BSDS500 test image-specific MSSIM-scores, with quality assessment between the prediction result by a method and the average of the human annotations.

C.2 Speed

Our unoptimized GPU implementation of the two-stream model inference takes 0.1 to 0.2 s per test image (grey-scale; 3 times for colour-domain where twice the number of features). Our current implementation of the enhanced two-stream inference is serial over the orientations (each of the 16 orientation-specific results computed on a GPU one after another from CPU) but are working on an enhanced parallel and optimized implementation. With the rotation-averaging our maximal runtimes (colour) are on the order of thirty seconds. These compare very favorably with the following figures quoted in Lim et al. [2013], which are 60s (gPb local), 100s (SCG local), 240s (gPb global), 280s and (SCG global). We have confirmed the gPb global figure on our local machines. We note that Catanzaro et al. [2009] have reduced the runtime of gPb global to 1.8s using GPUs, but our basic two-stream is significantly faster and scales better to image size. In general the evidence is that one may typically obtain 10x speedups by going to GPUs, when comparing tuned implementations for *both* CPU and GPU [Lee et al., 2010]. We note the speeds of Sketch tokens [Lim et al., 2013](1 s) and that of in Dollár and Zitnick [2013]

	Percentiles of A-B				
B	0	25	50	75	100
	A = Us:				
Us (grey)	-0.038	-0.000	0.007	0.013	0.060
DollárZitnick	-0.005	0.070	0.110	0.161	0.393
SketchTokens	0.209	0.386	0.469	0.529	0.755
SCG (global)	-0.016	0.053	0.083	0.120	0.304
gPb	0.020	0.145	0.212	0.281	0.512
	A = Us (grey):				
DollárZitnick	-0.002	0.062	0.104	0.148	0.358
SketchTokens	0.217	0.380	0.460	0.529	0.723
SCG (global)	-0.023	0.042	0.072	0.116	0.261
gPb	0.013	0.138	0.204	0.280	0.514
gPb (grey)	0.065	0.240	0.322	0.418	0.662
	Percentiles of MSSIM				
	0	25	50	75	100
Us	**0.523**	**0.655**	**0.704**	**0.757**	0.865
Us (grey)	0.517	0.645	0.694	0.752	**0.890**
DollárZitnick	0.317	0.512	0.583	0.649	0.799
SketchTokens	0.056	0.156	0.227	0.316	0.496
SCG (global)	0.394	0.555	0.613	0.668	0.809
gPb	0.257	0.400	0.482	0.555	0.726
gPb (grey)	0.148	0.274	0.359	0.455	0.650

Table I: BSDS500 **MSSIM**-Score Statistics. gPb and DollárZitnick refer to [Arbelaez et al., 2011, gPb-owt-ucm] and Dollár and Zitnick [2013], respectively. All models are for colour-data unless mentioned (grey).

(best in performance, 1/6 s) are comparable to us in speed without the rotation-averaging. Our results in colour are the following: ODS: 0.725, OIS: 0.74, AP: 0.712. We note Sketch Tokens is there clearly better in terms of AP and Dollár and Zitnick [2013] is better also w.r.t. F-scores (see Table 1). If we do not consider non-maximum suppression by the Canny-method either, our F-scores are higher (ODS: 0.736, OIS: 0.75, AP: 0.695) but the AP is further weakened.

We have a similar trend within our method under grey-scale. In particular our scores without rotation-averaging are: ODS: 0.712, OIS: 0.724, AP: 0.707, and further without non-maximum suppression by the Canny-method: ODS: 0.719 (0.742,0.697) OIS: 0.733 (0.758,0.710) AP: 0.697. While we are still clearly better than the gPb in terms of any score, our AP is weaker in comparison to the SCG.

D Boundary Prediction Examples

Figures V, VII, VII, VIII, IX, X, XI, XII, XIII show BSDS500 test image data contour prediction examples with our approach and several competing methods, each individually rescaled. Not rescaled versions of first five of them are shown in Figures XIV, XVI, XVI, XVII, and XVIII, respectively. Non-rescaled results for images in Figure 2 are shown in Figure XIX.

E Dissecting the Deep Visual Boundary Prediction Network

We considered a dissection of our deep architecture onto several subparts which result in feed-forward neural networks. We discuss the results for shallow and deep architectures in turn. In addition to the different stream architectures, we also assess (i) the relative importance of the mean and covariance units, and (ii) the effect of generative pre-training [Hinton et al., 2006]. In the latter case the three options were (a) initialization of the parameters to those of the mcRBM/mcDBN followed by supervised fine-tuning, (b) fixing (i.e. freezing) the learned unsupervised mcRBM/mcDBN parameters, and (c) starting them from random settings (initialized as those in the generative training). We did not explore the full combinatorial space of settings, but focused on the most important cases, and considered only the standard BSDS protocol.

We did not consider Canny non-maximum suppression and did not apply rotation-averaging but focused only on the basic neural component. Also their assumed simplifications: there was no sparsity encouragement, each training image patch was not standardized individually, and corresponding annotation label patch was binary and randomly chosen from the set of available ones instead of their average when optimizing the cross-entropy defined by the objective function. Furthermore, the methods here (which are neural networks) were trained on/had access to a subset of the data available for training (200 images and their annotations out the 300 available for training and validation purposes [Martin et al., 2014]).

E.1 Dissecting Boundary Prediction with Shallow Networks

We consider first boundary prediction with the shallow networks. Table II (bottom) summarizes the results, and Figure XXI shows precision-recall curves, for the BSDS500 test set. We observe that when the parameters are not fine-tuned [mcRBM fixed entries], the covariance units tend to carry more contour information, as the performance without the mean units nearly matches that of the full model, while performance without the covariance units is clearly weaker. Figure XX visualizes filters of the full shallow model, without and with fine-tuning. Fine-tuning results in clear changes in the filter appearances and properties. We can observe for example that the mean feature extraction filters change in appearance to be more localized, and the associated read-out weights become larger in amplitude thus influencing the prediction more. The checker-patterned fine-detail covariance feature extraction filters have mostly disappeared.

When fine-tuning is in place, the performances increase significantly, especially so for the mean-only model, but the relative order of the models is still maintained. Interestingly, initialization of the feature extraction parameters from the generatively trained mcRBM yields better results than random initialization.

Figure 2 shows example inference with a shallow network for a large test image. For comparison, Canny edge detection (obtained with default settings in the matlab implementation) which produces a binary edgemap output is also shown. The feature extraction parameters of the shallow network is trained with the mcRBM parameter initialization, and fine-tuning. We see that it correctly places probability mass on locations where humans have placed annotations, but also in regions with repeated local structure. See Appendix D for more examples.

E.2 Dissecting Boundary Prediction with Deeper Networks

Table II summarizes the test-set results for the deep stream and two-stream networks. Similar to the earlier

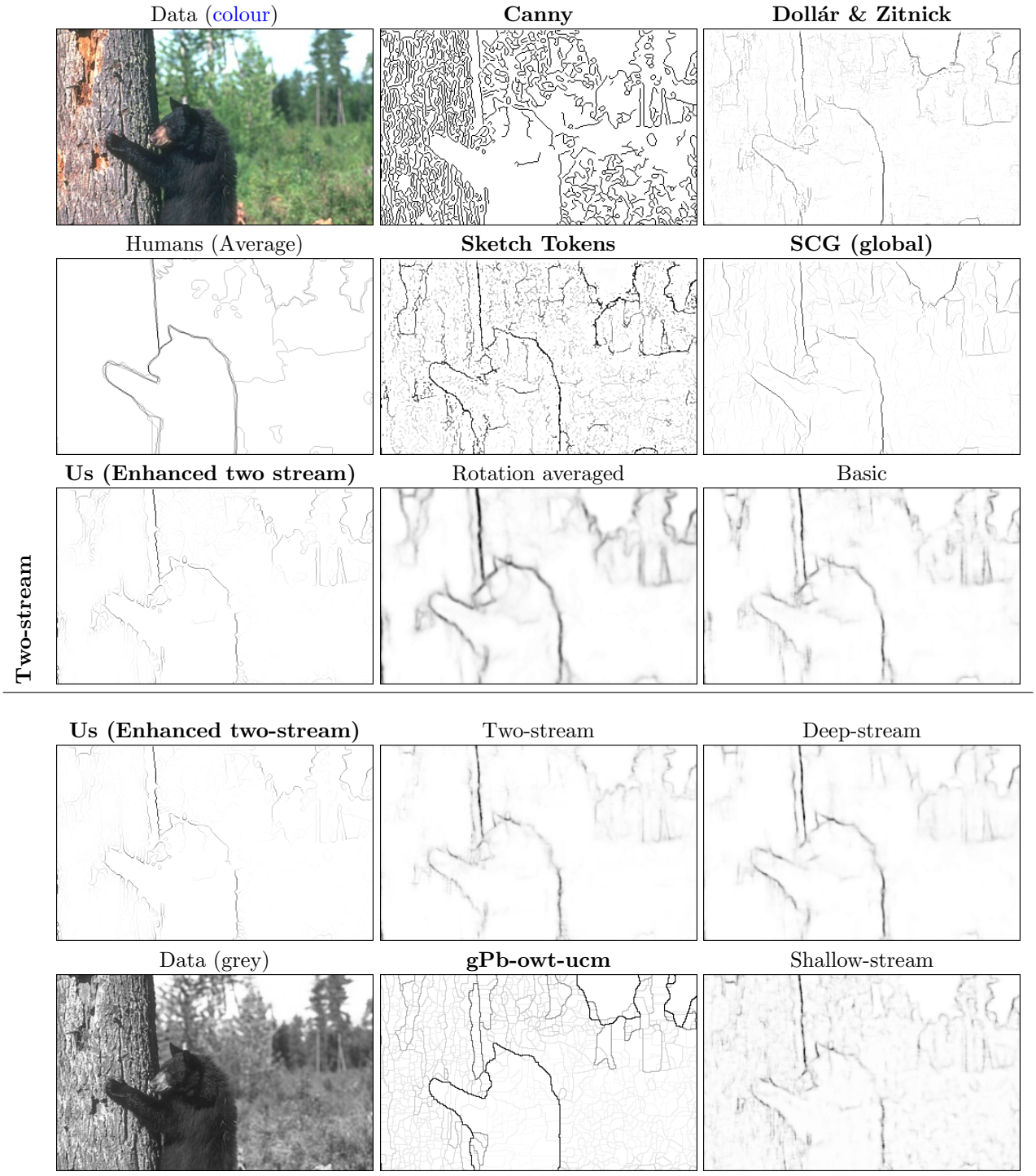

Figure V: Contour Prediction Result Examples on the BSDS500, Individually Rescaled to Fill Full Intensity Range. Best viewed on screen.

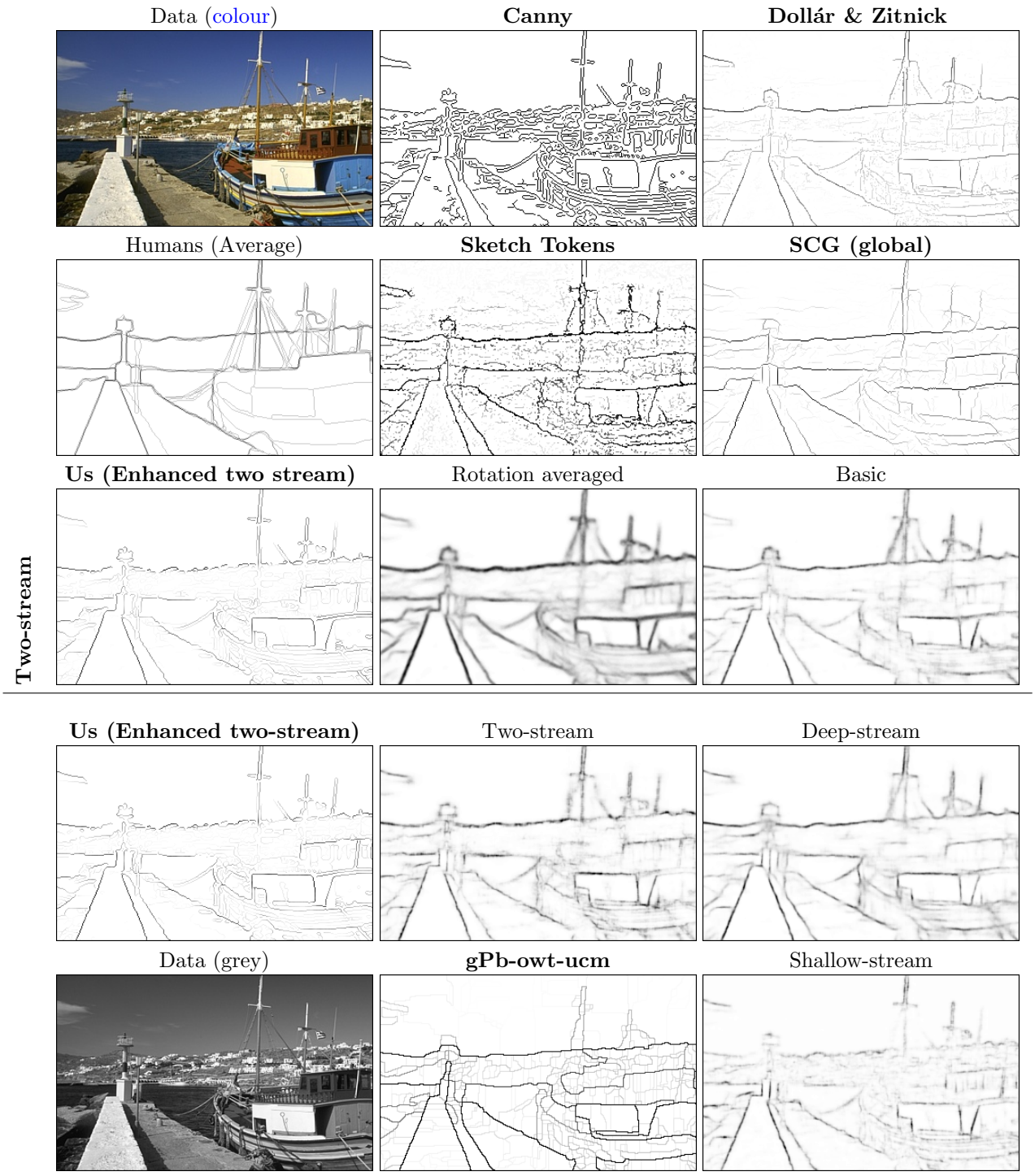

Figure VI: Contour Prediction Result Examples on the BSDS500, Individually Rescaled to Fill Full Intensity Range. Best viewed on screen.

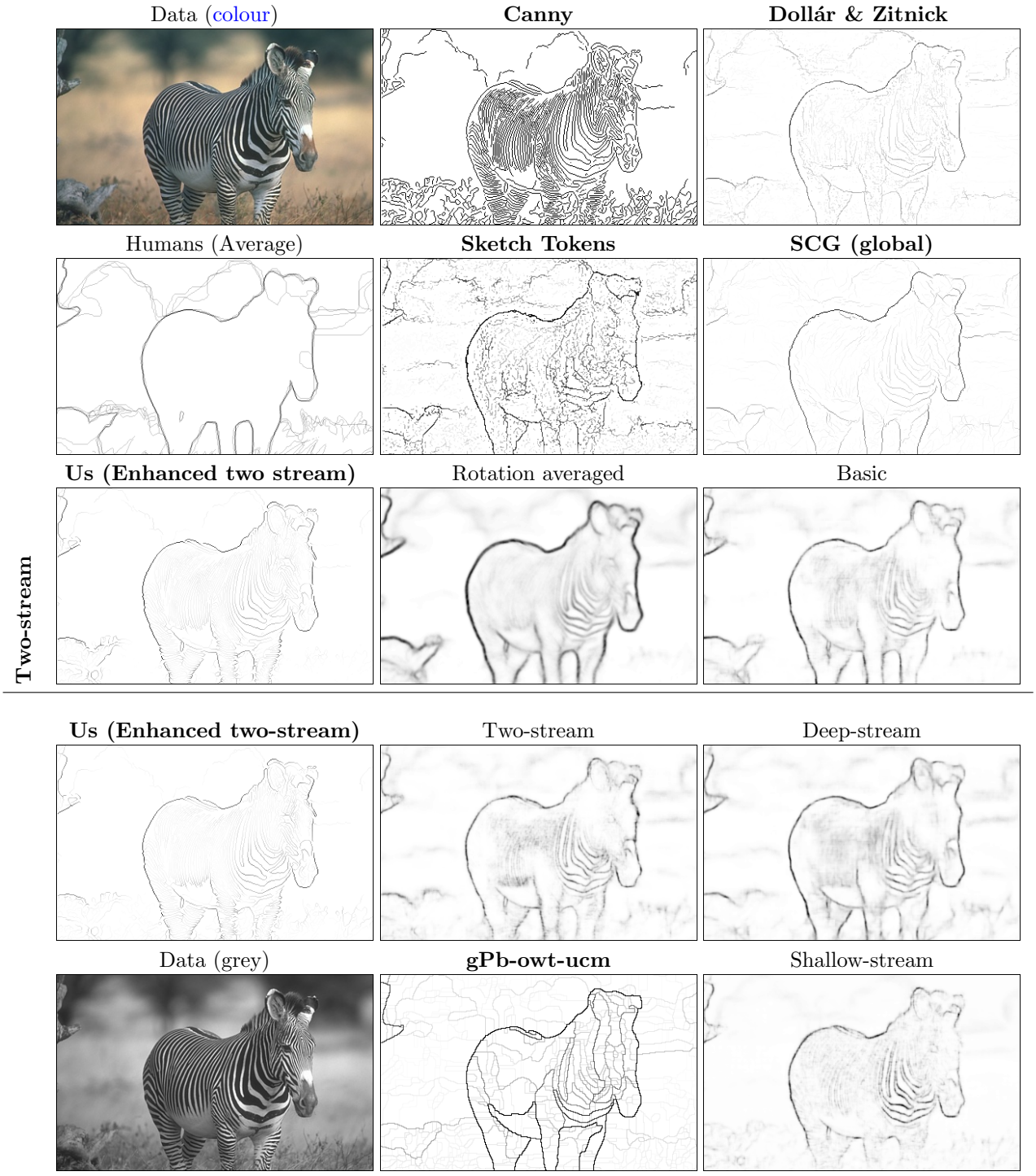

Figure VII: Contour Prediction Result Examples on the BSDS500, Individually Rescaled to Fill Full Intensity Range. Best viewed on screen.

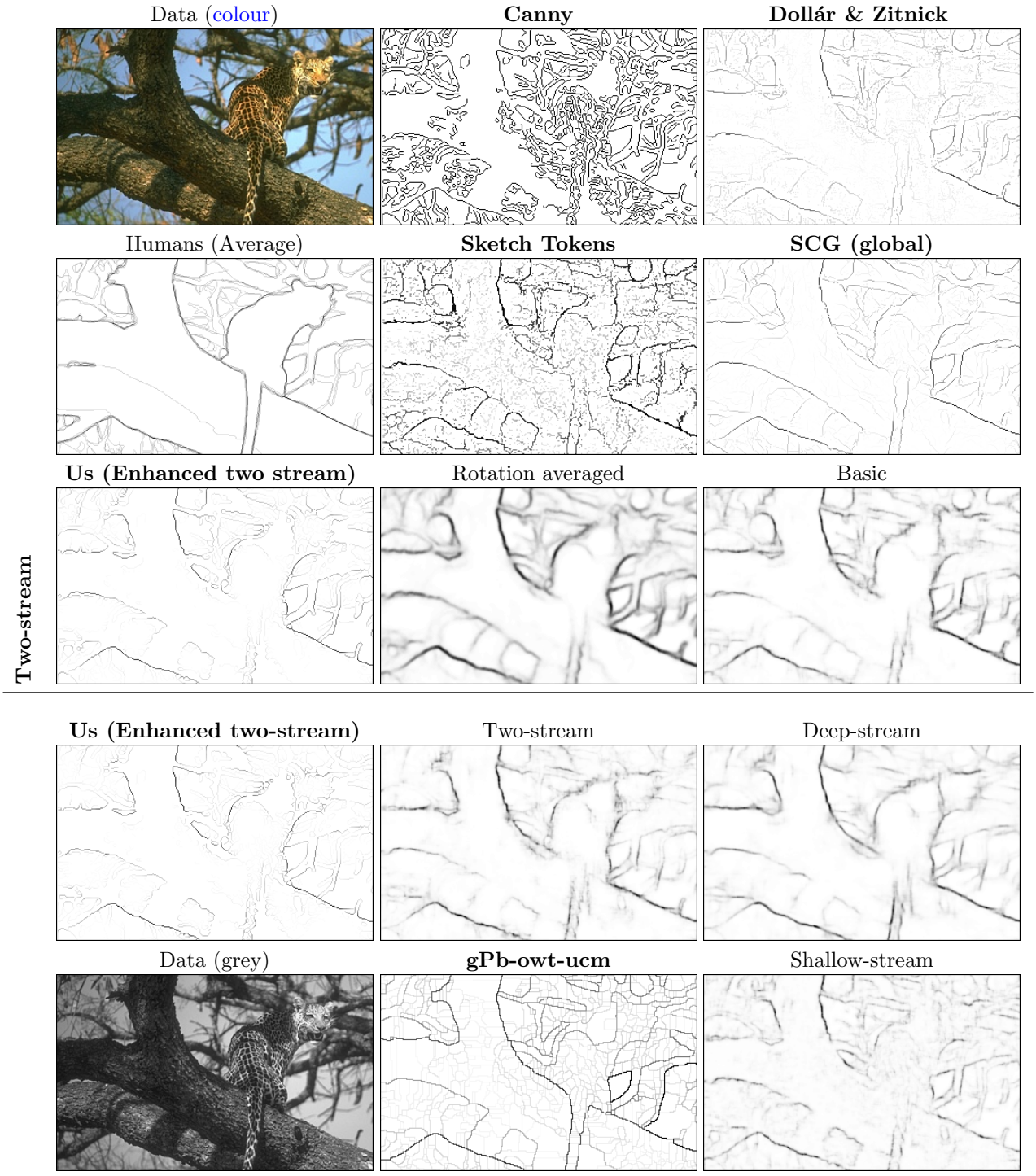

Figure VIII: Contour Prediction Result Examples on the BSDS500, Individually Rescaled to Fill Full Intensity Range. Best viewed on screen.

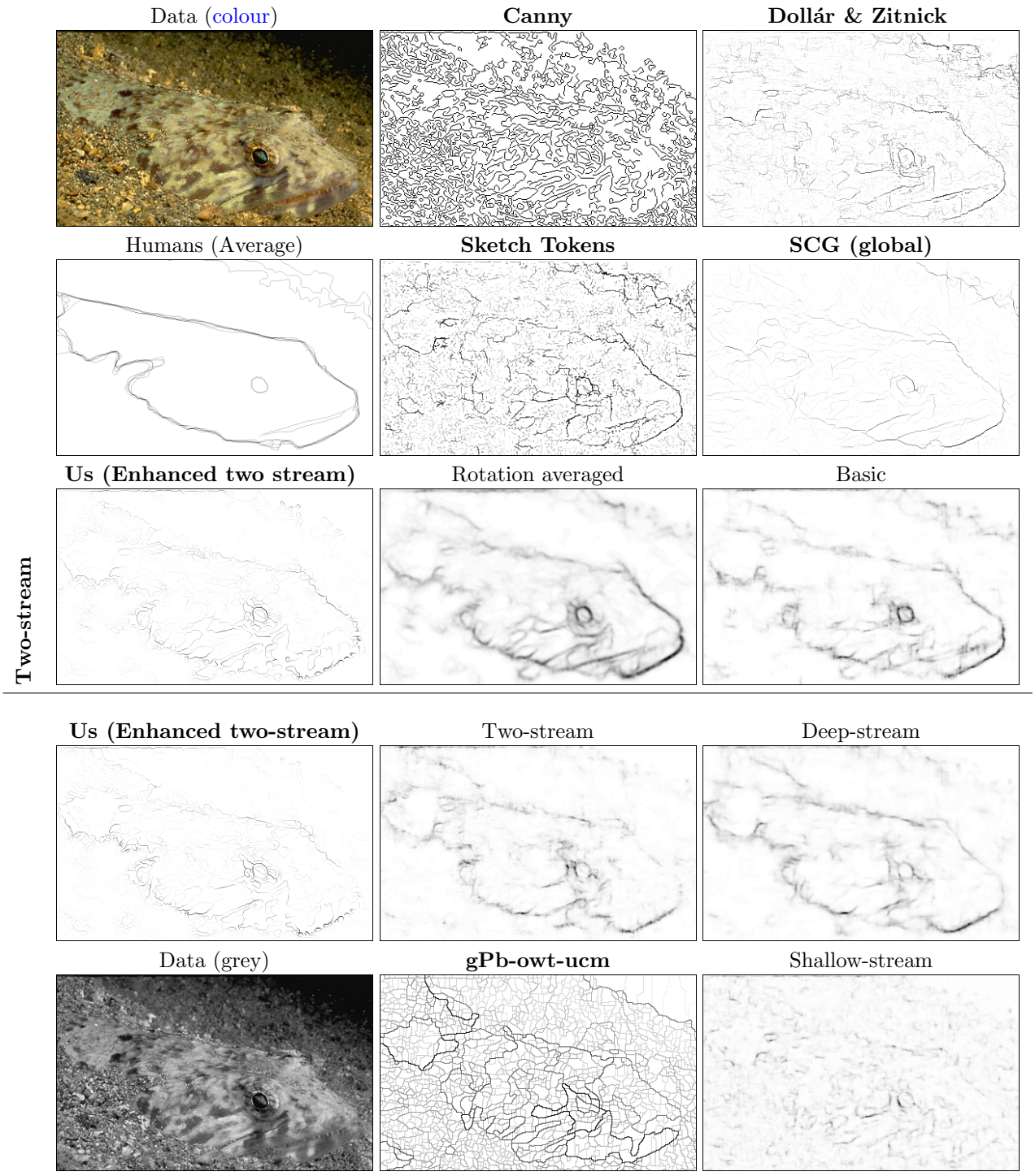

Figure IX: Contour Prediction Result Examples on the BSDS500, Individually Rescaled to Fill Full Intensity Range. Best viewed on screen.

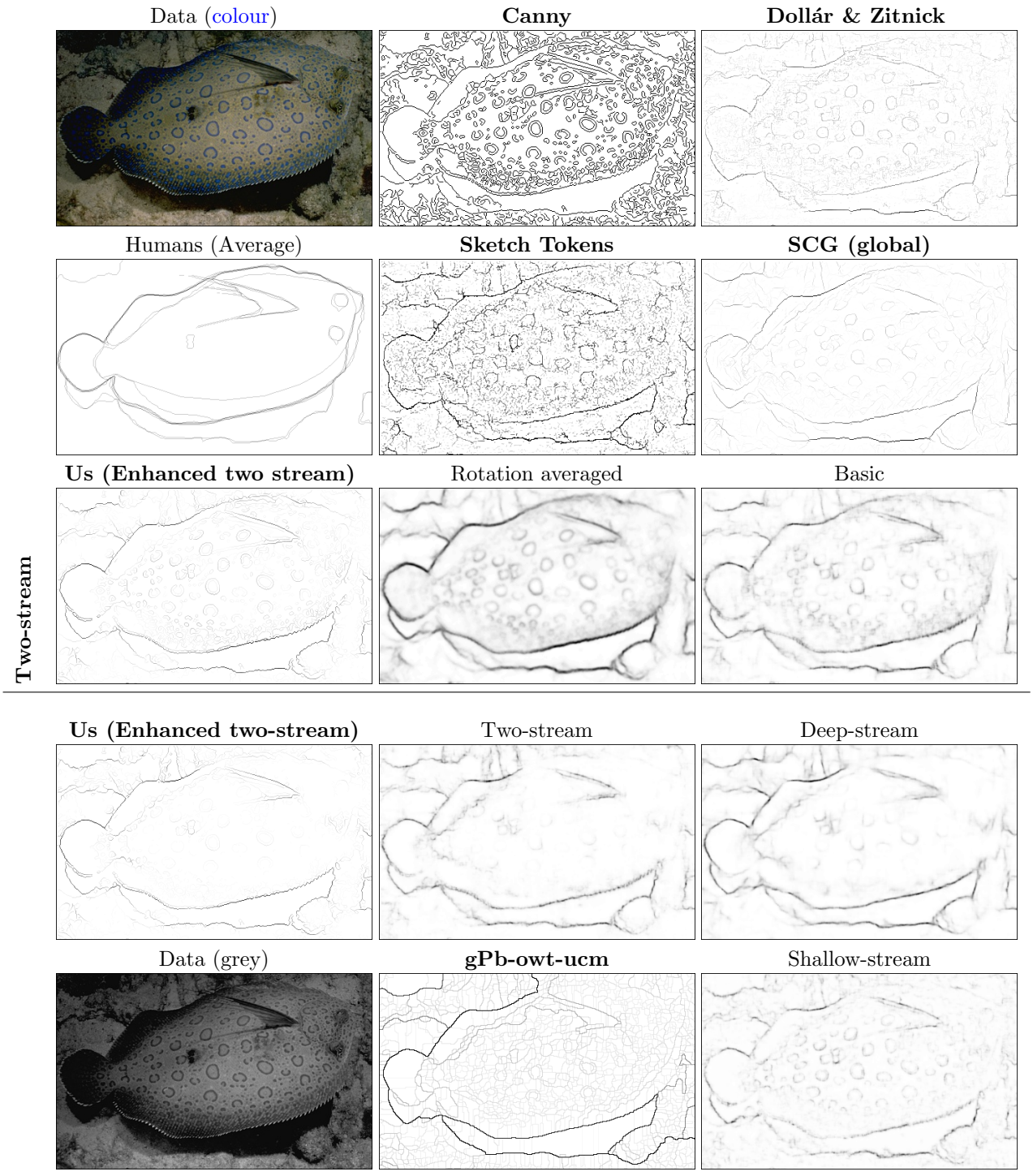

Figure X: Contour Prediction Result Examples on the BSDS500, Individually Rescaled to Fill Full Intensity Range. Best viewed on screen.

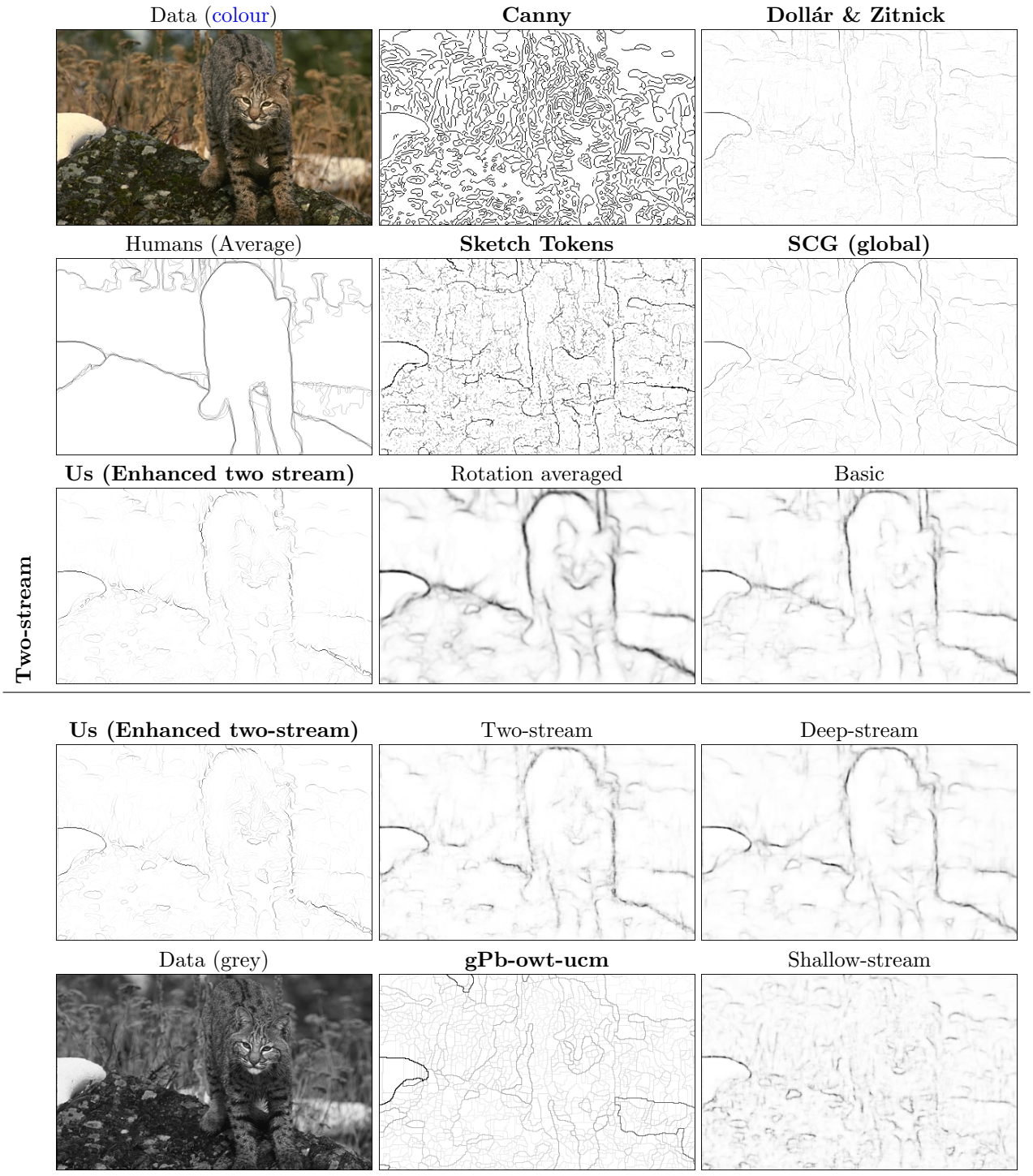

Figure XI: Contour Prediction Result Examples on the BSDS500, Individually Rescaled to Fill Full Intensity Range. Best viewed on screen.

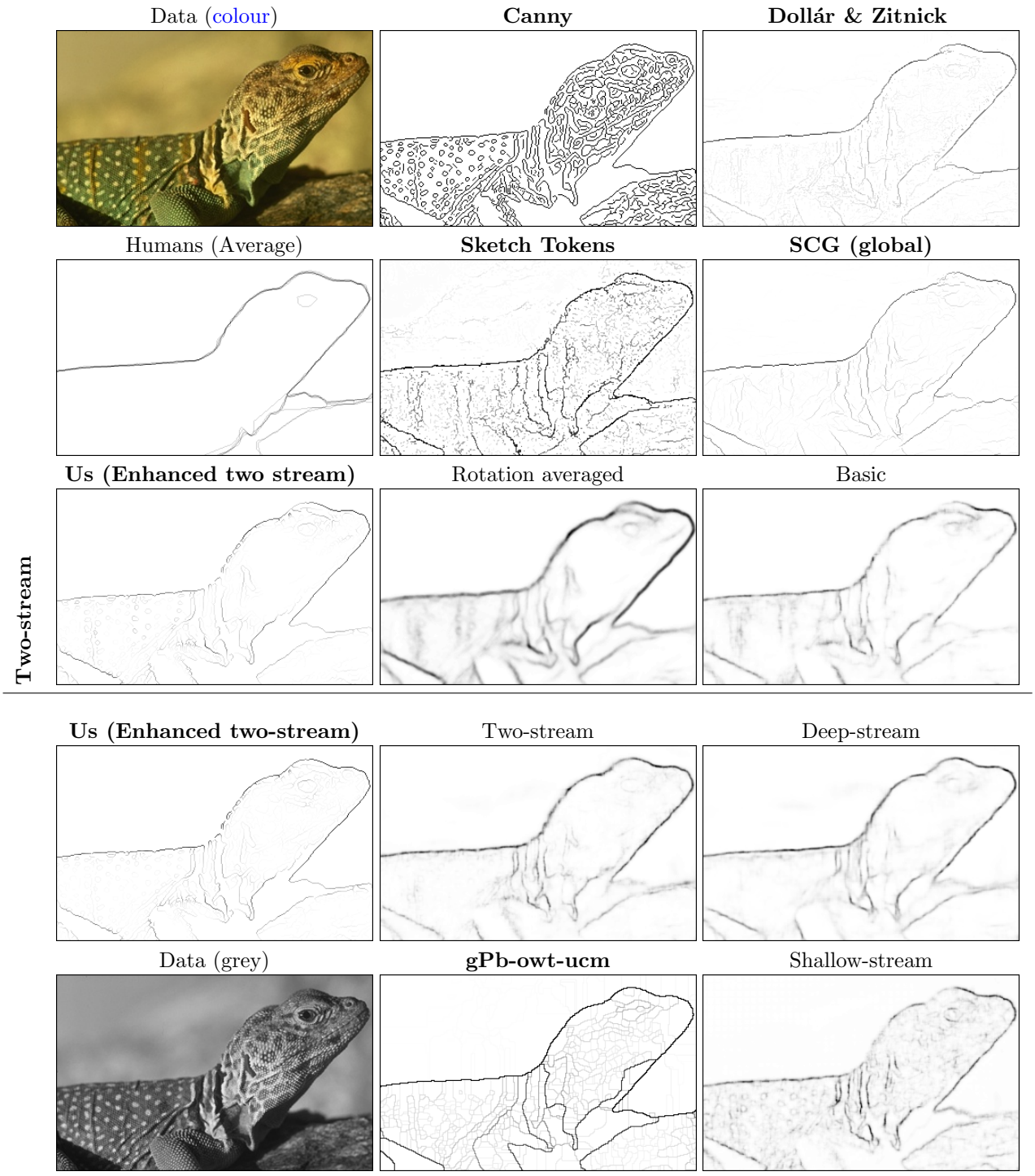

Figure XII: Contour Prediction Result Examples on the BSDS500, Individually Rescaled to Fill Full Intensity Range. Best viewed on screen.

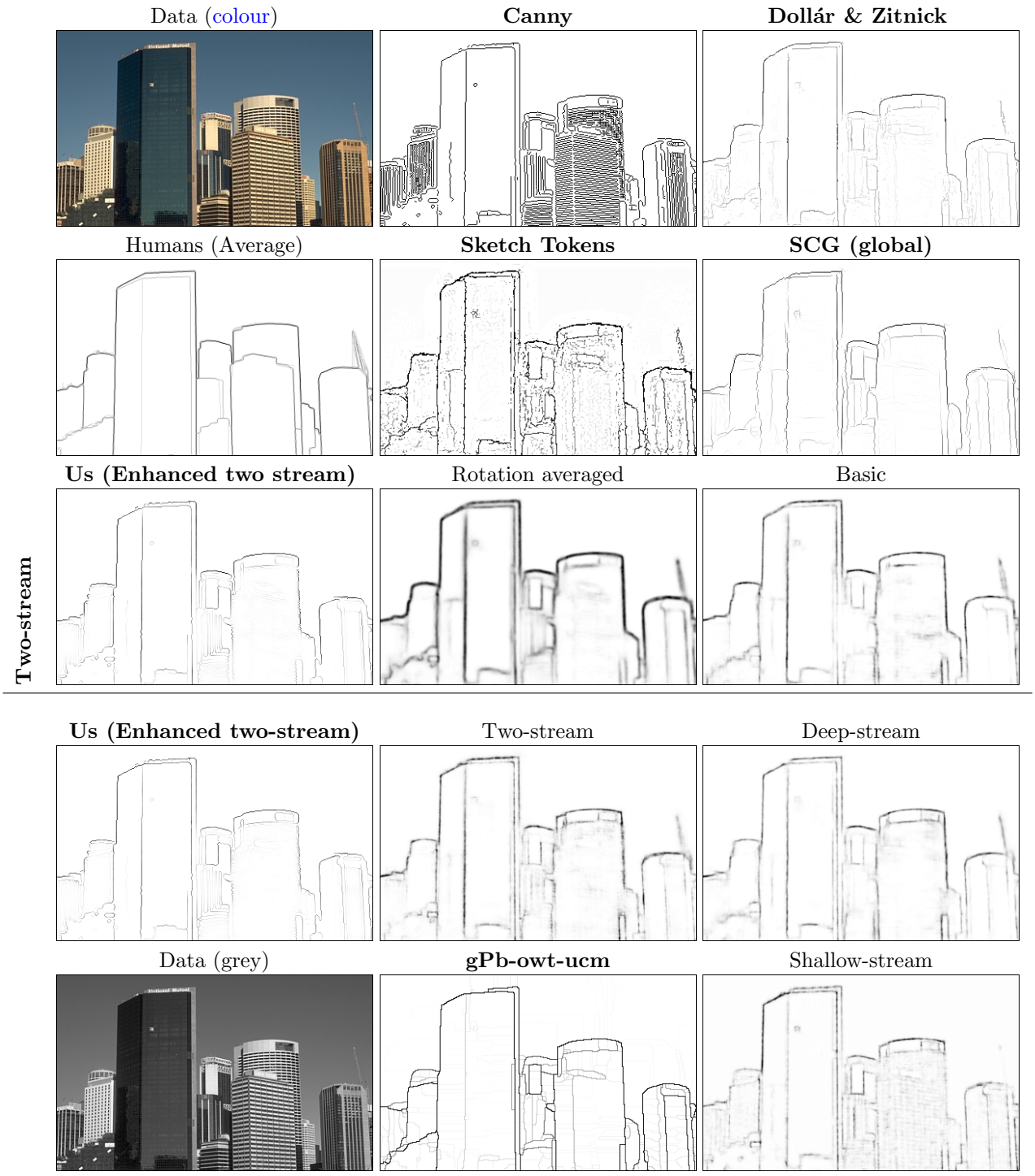

Figure XIII: Contour Prediction Result Examples on the BSDS500, Individually Rescaled to Fill Full Intensity Range. Best viewed on screen.

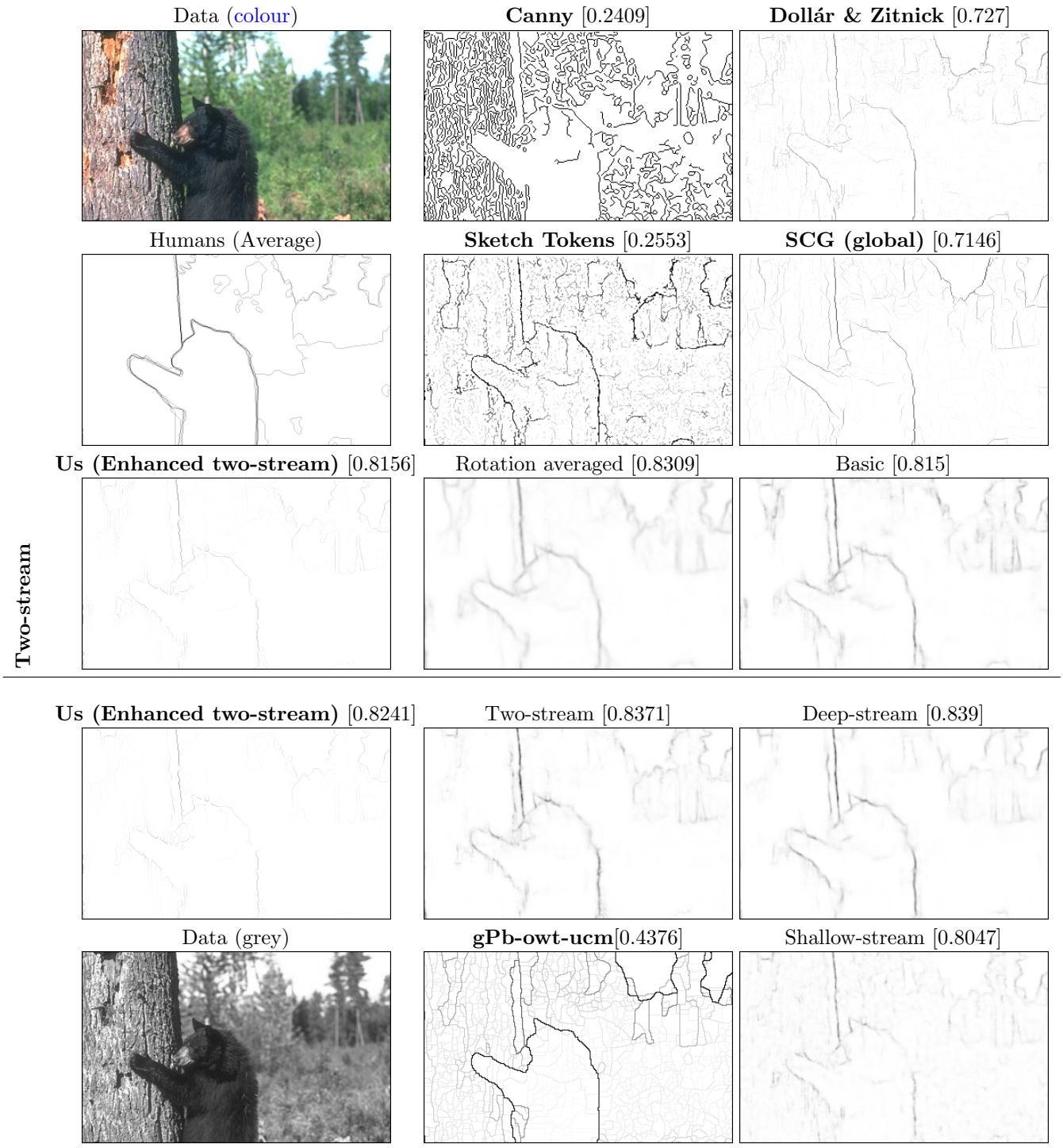

Figure XIV: Contour Prediction Result Examples on the BSDS500. Numbers in square brackets are MSSIM scores. Best viewed on screen.

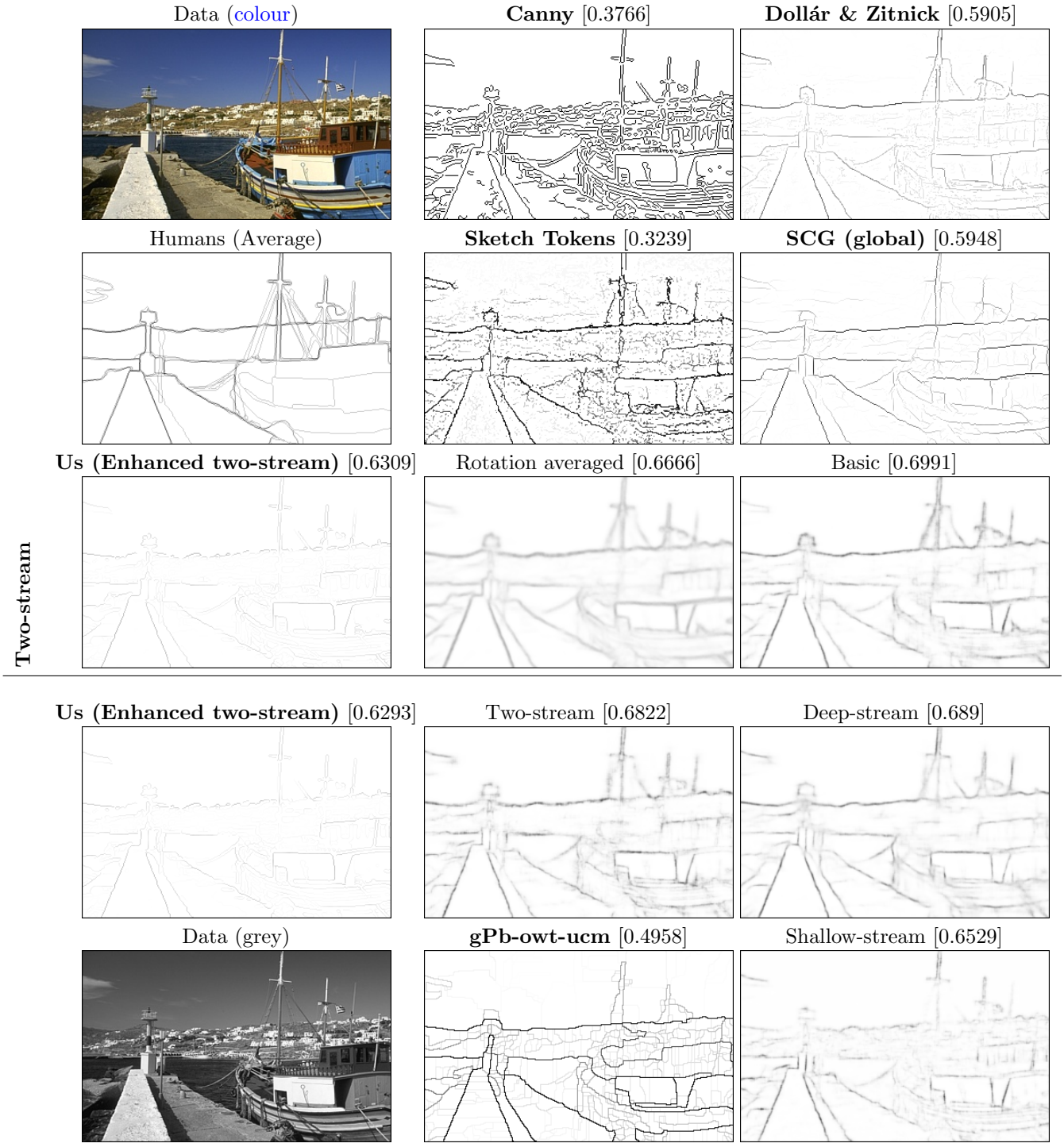

Figure XV: Contour Prediction Result Examples on the BSDS500. Numbers in square brackets are MSSIM scores. Best viewed on screen.

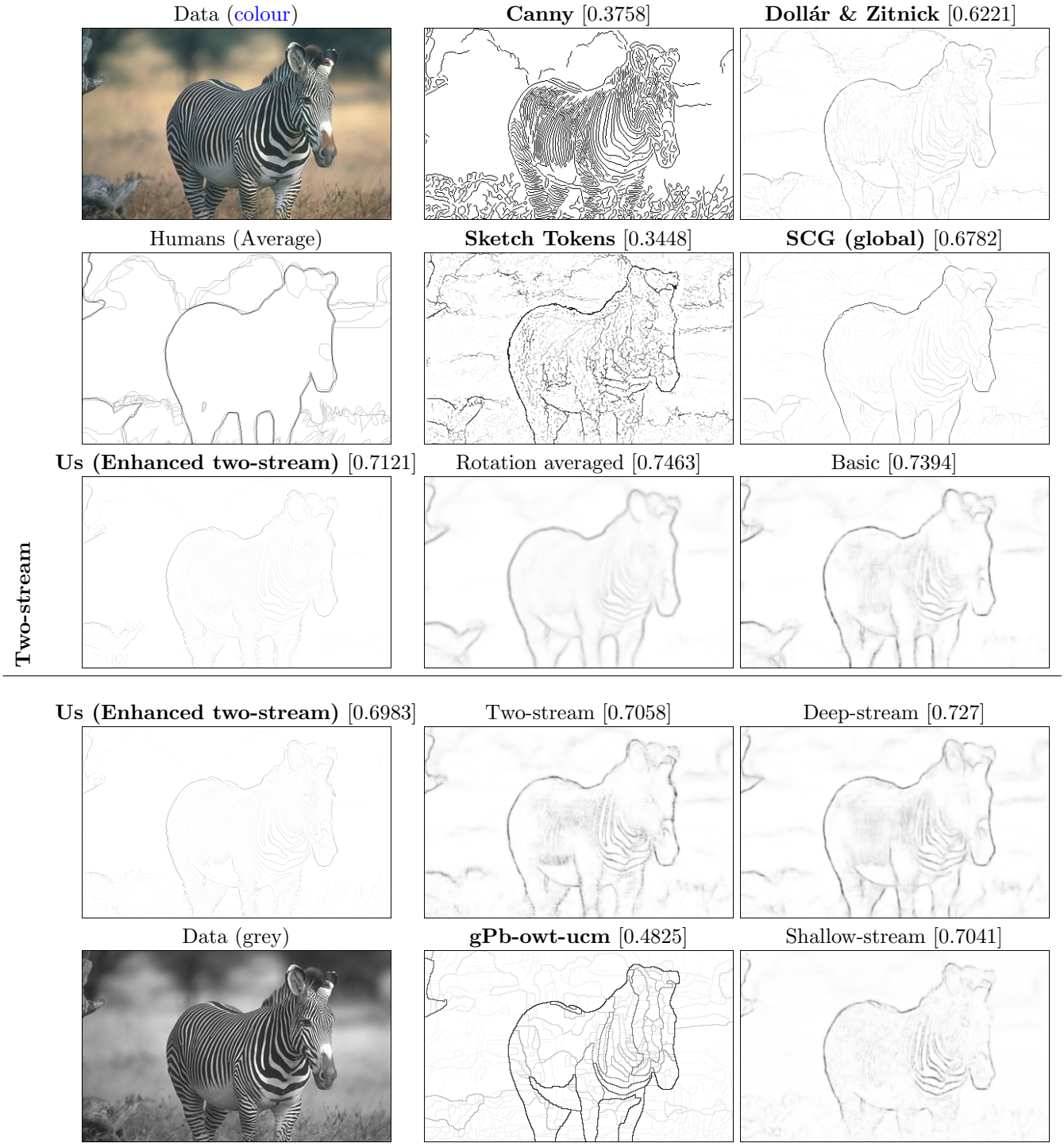

Figure XVI: Contour Prediction Result Examples on the BSDS500. Numbers in square brackets are MSSIM scores. Best viewed on screen.

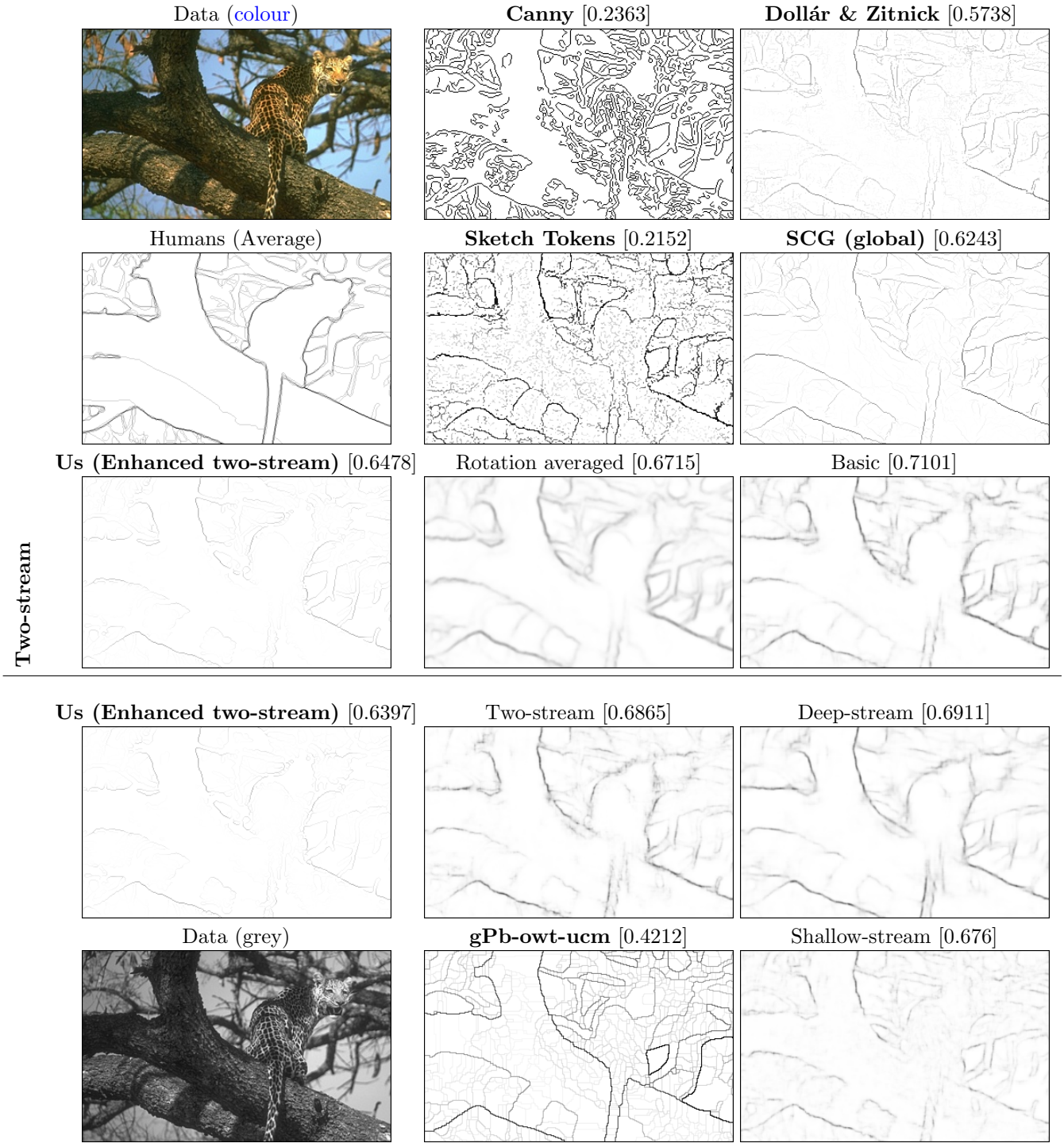

Figure XVII: Contour Prediction Result Examples on the BSDS500. Numbers in square brackets are MSSIM scores. Best viewed on screen.

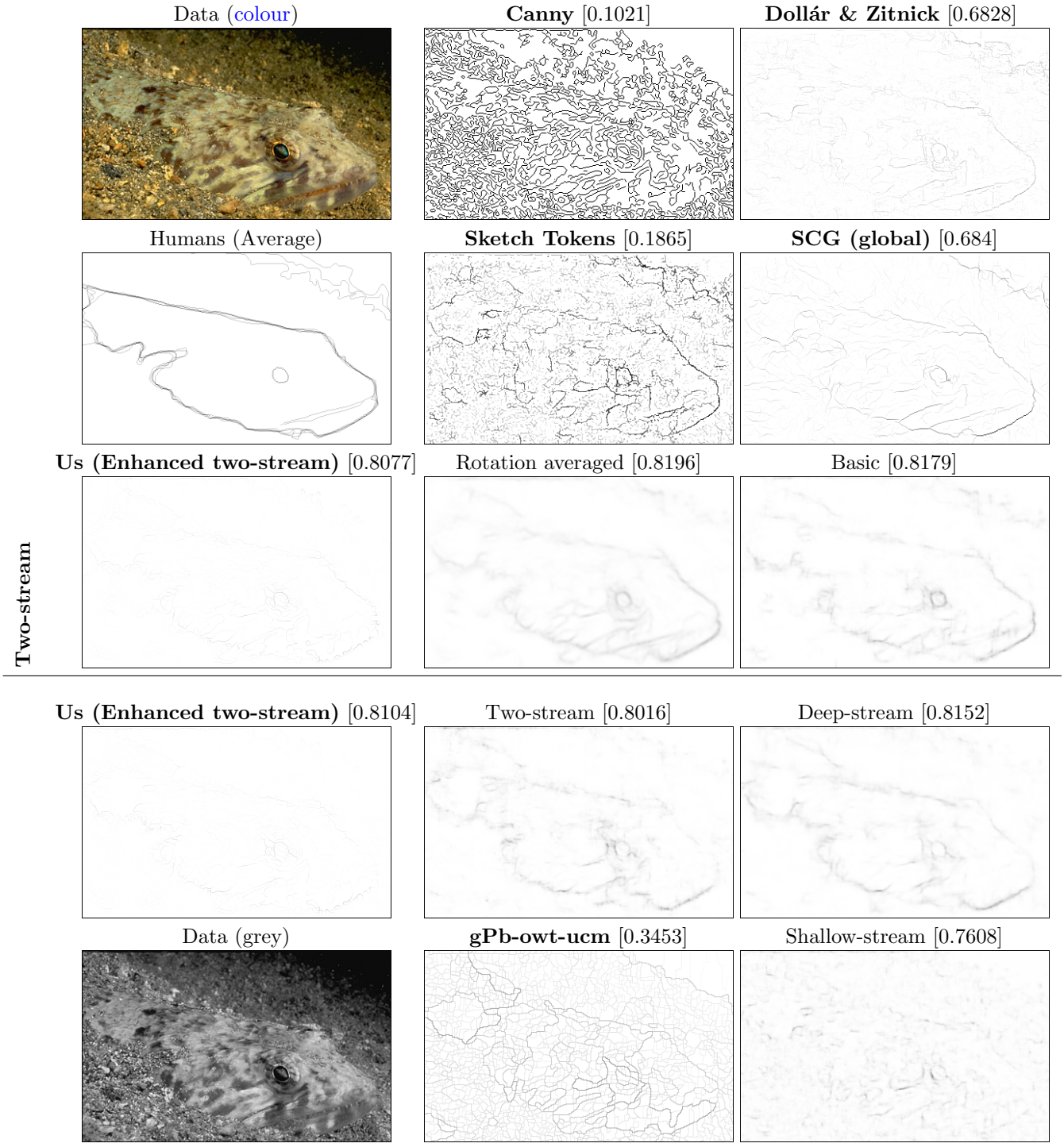

Figure XVIII: Contour Prediction Result Examples on the BSDS500. Numbers in square brackets are MSSIM scores. Best viewed on screen.

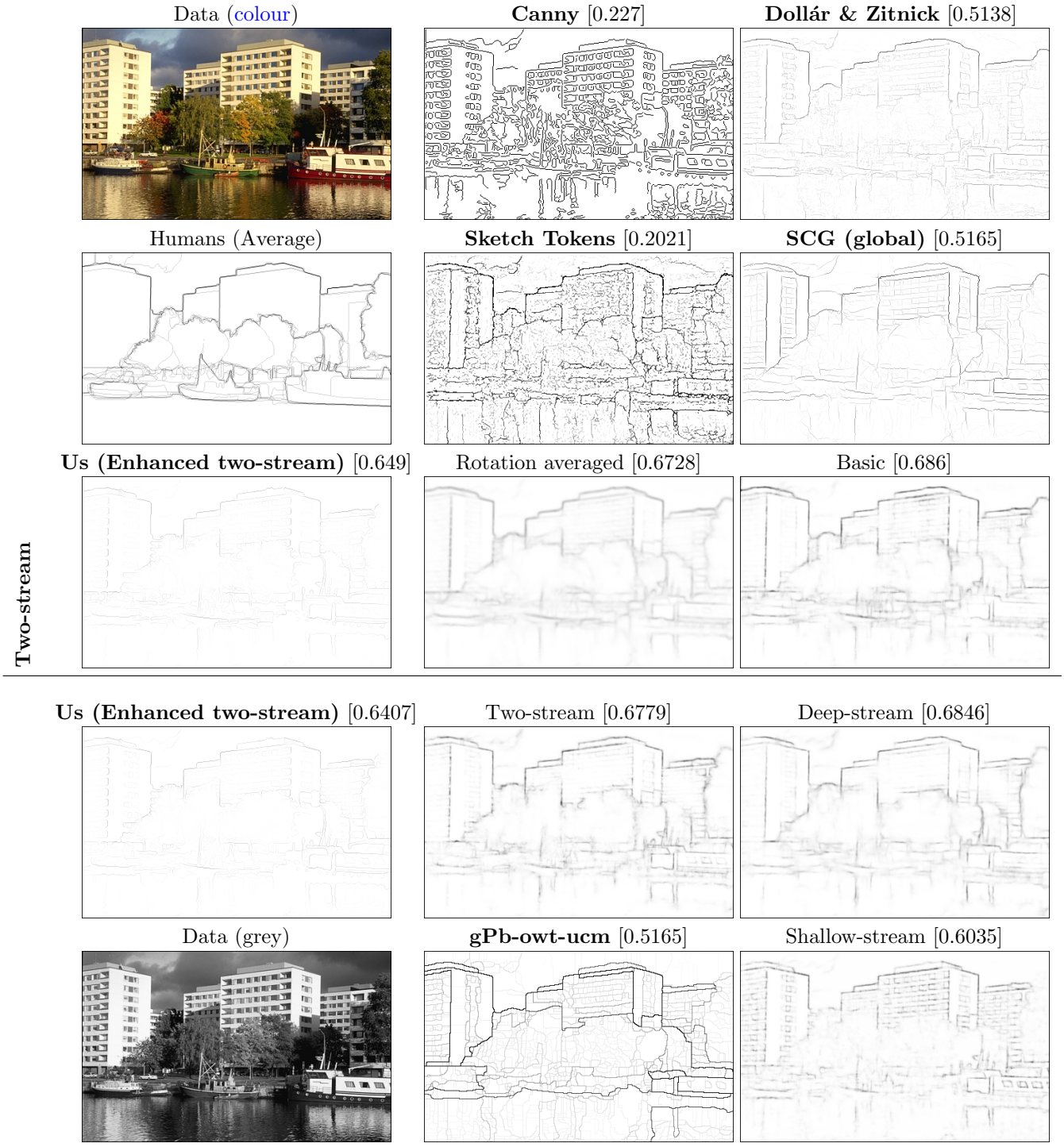

Figure XIX: Contour Prediction Result Examples on the BSDS500. Numbers in square brackets are MSSIM scores. Best viewed on screen.

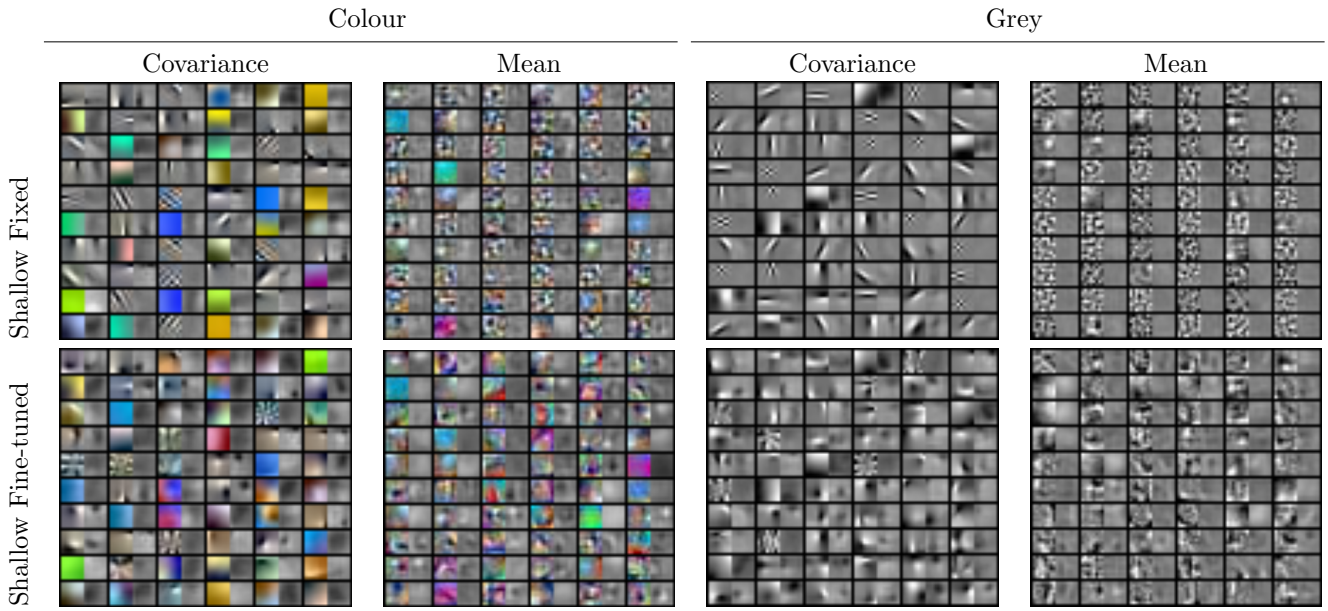

Figure XX: Random Collection of Shallow Stream Filters with and without Discriminative Fine-Tuning. The small black-boxes each contain the image feature extraction (left part) and contour prediction (right part) filter pairs. The filters connecting to the image data are normalized invidually, whereas the filters connecting to the contour data are globally normalized, so as to be on the same scale and to fill the full intensity range.

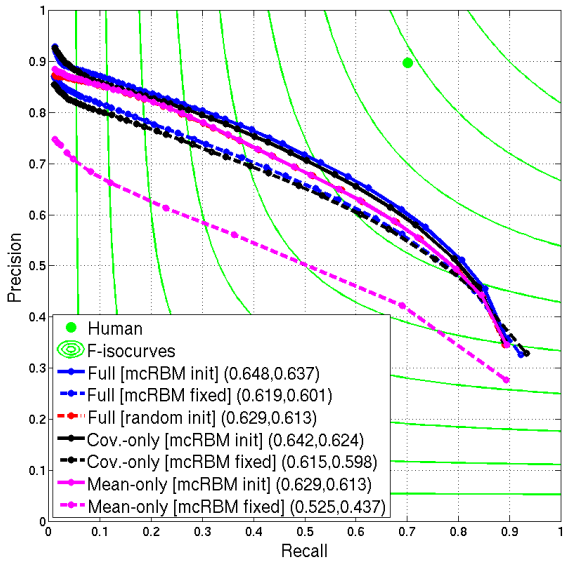

Figure XXI: Shallow Network Prediction Performance Dissection on the **BSDS500 (Grey-Scale)** as Measured by Precision-Recall Curves. The legend numbers denote the maximums of the curves w.r.t. the ODS F-measure (left), and the average precision (right).

experiments, we can see under any particular initialization setting, that the covariance-only models perform better than the mean-only models. We can also see that the generative pre-training helps the full two-stream model in an even more pronounced way than the shallow-stream model (compare random init. vs. mcRBM and mcDBN init). Allowing fine-tuning of the feature extraction parameters (compare fixed vs. init for a particular model setting) again results in improved performance. See also Figure XXII for P-R-curves of many of these cases for further verification.

When comparing the performance summaries related to the different streams, it is noticeable that the deep model is clearly better than the shallow model. This can be also seen in the Figure 2 inference example, with improved ability to filter out predictions in textured regions containing many edges. We can see for example many building windows appearing in a locally repeated fashion being removed. See Appendix D for more inference examples.

Although the numerical performance of the deep model is comparable to the two-stream model with respect to the ODS and OIS metrics (across the different cases in Table II), the average precision is clearly worse. We can see from the Figure XXII that the two-stream model is able to obtain higher recall rates, with the P-R curve of the deep-only model dropping off earlier.

Model (Parameters: Shallow,Deep)	F-score ODS	OIS	AP
Two-stream:			
MC(mcDBN fixed)	0.664	0.682	0.659
MC(random init)	0.625	0.649	0.605
MC(mcRBM init,random init)	0.667	0.684	0.663
MC(mcDBN init)	0.695	0.709	0.673
C(mcRBM fixed,mcDBN init)	0.686	0.703	0.667
C(mcDBN init)	0.692	0.708	0.683
M(mcRBM fixed,mcDBN init)	0.646	0.663	0.603
M(mcDBN init)	0.679	0.696	0.657
Deep-Stream:			
MC(mcDBN fixed)	0.625	0.641	0.537
MC(mcDBN init)	0.702	0.715	0.654
C(mcDBN init)	0.696	0.710	0.652
M(mcDBN init)	0.679	0.694	0.627
Shallow-Stream:			
MC(mcRBM fixed)	0.619	0.643	0.601
MC(random init)	0.629	0.651	0.613
MC(mcRBM init)	0.648	0.668	0.637
C(mcRBM fixed)	0.615	0.641	0.598
C(mcRBM init)	0.642	0.660	0.624
M(mcRBM fixed)	0.525	0.539	0.437
M(mcRBM init)	0.629	0.652	0.613

Table II: Deeper Network Prediction Performance Dissection on **BSDS500 (grey-scale)**. M, C and MC denote parts including only and branching from mean, covariance, and both mcRBM units, respectively.

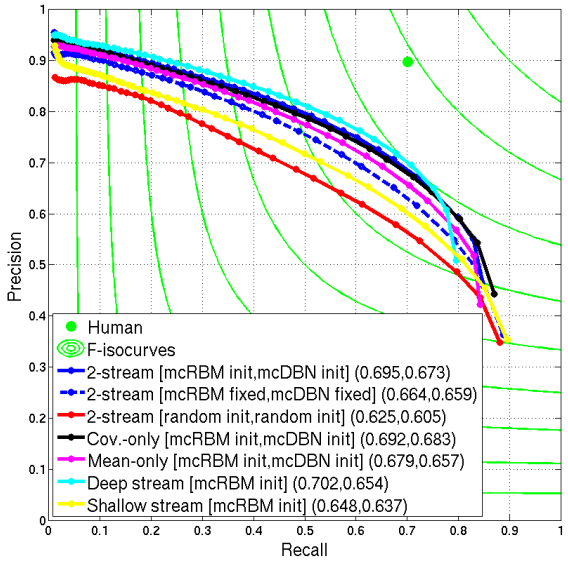

Figure XXII: Deeper Network Prediction Performance Dissection on the **BSDS500 (Grey-Scale)** as Measured by Precision-Recall Curves. The legend numbers denote the maximums of the curves w.r.t. the ODS F-measure (left), and the average precision (right).

F Boundary Prediction Quality Assessment with the Structural Similarity Index (SSIM)

The structural similarity metric [Wang et al., 2004] assesses the similarity of two signals $\mathbf{x}$ and $\mathbf{y}$ of same dimension with J elements, using a function on three comparison functions which (aim to) measure differences in signal luminance contrast, and structure:

$$\mathrm{SSIM}(\mathbf{x},\mathbf{y}) = [\ell(\mathbf{x},\mathbf{y})]^{\alpha} \cdot [c(\mathbf{x},\mathbf{y})]^{\beta} \cdot [s(\mathbf{x},\mathbf{y})]^{\gamma}, \quad (3)$$

where the signal luminance comparison function $\ell(\mathbf{x},\mathbf{y})$ operates on signal sample means ($\mu_z = \frac{1}{J}\sum_{j=1}^{J} z_j$), the contrast comparison function $c(\mathbf{x},\mathbf{y})$ operates on signal standard deviations (variance $\sigma_z^2 = \frac{1}{J-1}\sum_{j=1}^{J}(z_j-\mu_z)^2$), the structure comparison function $s(\mathbf{x},\mathbf{y})$ is a correlation-based measure between the signals, and $\alpha > 0$, $\beta > 0$ and $\gamma > 0$ denote scalars to adjust the relative importance of the three components.

The paper uses the form implemented in Wang et al. [2004] in which $\alpha = \beta = \gamma = 1$, and the specific form of the SSIM index is the following:

$$\mathrm{SSIM}(\mathbf{x},\mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \quad (4)$$

where $\sigma_{xy} = \frac{1}{J-1}\sum_{j=1}^{J}(x_j-\mu_x)(y_j-\mu_y)$, and C_1 and C_2 are scalars dependent on the dynamic range of the signals.

For image (/large lattice) quality assessment Wang et al. [2004] recommends to apply the index locally (within local windows) and average the local quality assessments for an overall quality measure, called the mean SSIM (MSSIM) index:

$$\mathrm{MSSIM} = \frac{1}{I}\sum_{i=1}^{I} \hat{\mathrm{SSIM}}(\mathbf{x}_{\mathcal{N}_j}, \mathbf{y}_{\mathcal{N}_j}), \quad (5)$$

where $\mathbf{z}_{\mathcal{N}_j}$ denotes a local (2D) window within an image, I denotes the number of local windows in the image, and $\hat{\mathrm{SSIM}}(\mathbf{x}_{\mathcal{N}_j}, \mathbf{y}_{\mathcal{N}_j})$ is an evaluation of a modification of (4) to avoid "blocking" artifacts. In particular, the local SSIM evaluations are smoothed with a circularly symmetric Gaussian weighting function $\mathbf{w}$. The modified evaluations can be then seen to apply (4) but assume $\mu_z = \sum_{j=1}^{J} w_j z_j$, $\sigma_z = \left(\frac{1}{J-1}\sum_{j=1}^{J} w_j (z_j-\mu_z)^2\right)^{\frac{1}{2}}$, and $\sigma_{xy} = \sum_{j=1}^{J} w_j (x_j-\mu_x)(y_j-\mu_y)$.

The paper uses the same settings (considered default/standard) in the evaluation as in Wang et al. [2004], with $C_1 = (0.01\cdot 255)^2$, $C_2 = (0.03\cdot 255)^2$ (the

dynamic range of images assumed to be 255), and the Gaussian smoothing filter is of 11×11 size, assumes standard deviation of 1.5 samples, and is normalized to unit sum.

We use the Matlab-implementation `https://ece.uwaterloo.ca/~z70wang/research/ssim/ssim.m` with the suggested use as described in[3] which rescales the images before the evaluation.

[3] `http://www.ece.uwaterloo.ca/~z70wang/research/ssim/`