Convolutional Hough Matching Networks
—Supplementary Material—

Juhong Min Minsu Cho
POSTECH CSE & GSAI
http://cvlab.postech.ac.kr/research/CHM/

In this supplementary material, we provide additional results and analyses, and implementation details.

1. Additional results and analyses

Analysis on scale-space maxpool. To further analyze the results in Fig. 7 of our main paper, we visualize maxpooled positions of predicted matches on sample pairs of SPair-71k [9], PF-PASCAL [4], and PF-WILLOW [3]. Figure S3 shows the results and describes how we visualize them. Due to large scale-variations in pairs of SPair-71k, our model collects winners of scale-space vote, *i.e.*, CHM$(\cdot; k^{\text{6D}}_{\text{psi}})$, from diverse positions in scale-space. In contrary, objects in PF-PASCAL and PF-WILLOW exhibit relatively small scale-variations, thus encouraging our model to collect winners of the vote mostly from the original scales. We observe that the maxpooled positions typically depend on scales of object's parts as seen in Fig. S3.

Learned CHM kernels. Figure S4 describes how we visualized Fig. 3 of our main paper. For straightforward visualization of high-dimensional geometry on 2D plane, we use tesseracts and their arrangement on a 2D grid to represent 4D and 6D tensors respectively. Learned kernels of $k^{\text{6D-4D}}_{\text{psi}}$ (ours), $k^{\text{6D-4D}}_{\text{full}}$, and $k^{\text{6D-4D}}_{\text{iso}}$ are respectively visualized in Figs S5, S6, and S7.

Interestingly, the weight patterns of kernels $k^{\text{6D-4D}}_{\text{psi}}$ and $k^{\text{6D-4D}}_{\text{full}}$ are remarkably similar; the weights for matches with large offsets and closer distance are learned to be higher (darker) while those with small offsets and far distance are learned to be lower (brighter). Moreover, learned weight patterns of 4D maps in second, fourth, sixth, and eighth rows of $k^{\text{6D}}_{\text{full}}$ in Fig. S6 are noticeably similar to each other. We also observe that patterns in first and last rows, and patterns in third and seventh rows of $k^{\text{6D}}_{\text{full}}$ are similar to each other as well. In contrast, $k^{\text{nD}}_{\text{iso}}$ is unable to express diverse weight patterns due to its parameter-sharing constraint that enforces full isotropy. This observation reveals that our kernel $k^{\text{nD}}_{\text{psi}}$ in CHMNet clearly benefits from its reasonable parameter-sharing strategy, in terms of both efficiency and accuracy as demonstrated in Tab. 2 of our main paper.

2. Additional implementation details

Coordinate normalization. Following the work of [6], we use height and width normalized coordinates to ensure numerical stability of loss gradients such that

$$\begin{bmatrix}-1\\-1\end{bmatrix} \leq \mathbf{P}_{ij:} \leq \begin{bmatrix}1\\1\end{bmatrix}, \tag{1}$$

where $\mathbf{P}$ is a set of coordinates on a dense regular grid used for flow formation. This normalization gives spatial bounds $[-1, 1]$ to the intermediate output coordinates $\hat{\mathbf{P}}'$, $\mathbf{k}$, and $\hat{\mathbf{k}}'$.

Hyperparameters. During training, the learning rates of CHM layers and backbone feature extractor are set to 1e-3 and 1e-5, respectively with batch size of 16. The distance threshold τ in Eqn.11 is set to 0.1. We set the standard deviation of Gaussian kernel $\mathbf{G} \in \mathbb{R}^{30\times 30}$ to 17.

Implmentation of high-dimensional convolution. As PyTorch [11] supports only upto 3D convolution, we must manually implement a (dense) high-dimensional convolutions. We first demonstrate the original implementation of 4D convolution [12], and how we efficiently re-implemented the same 4D convolution and improved it for high-dimensional convolution. Given B correlation tensors in a minibatch[1] $\mathbf{C} \in \mathbb{R}^{B\times H\times W\times H'\times W'}$ and a 4D kernel $\mathbf{K} \in \mathbb{R}^{k\times k\times k\times k}$, we denote each 4D piece of $\mathbf{C}$ by $\mathbf{C}_i \coloneqq \mathbf{C}_{:i:::} \in \mathbb{R}^{B\times W\times H'\times W'}$ and each 3D tensor in $\mathbf{K}$ by $\mathbf{K}_i \coloneqq \mathbf{K}_{i:::} \in \mathbb{R}^{k\times k\times k}$. The work of [12] implements 4D convolution f_{4D} by performing H times of following operation:

$$\begin{aligned} f_{\text{4D}}(\mathbf{C})_i =& f_{\text{3D}}(\mathbf{C}_{i-p}, \mathbf{K}_1) + f_{\text{3D}}(\mathbf{C}_{i-p+1}, \mathbf{K}_2) \\ &+ \ldots + f_{\text{3D}}(\mathbf{C}_{i+p}, \mathbf{K}_k) + b \end{aligned} \tag{2}$$

where f_{3D} is a function that performs 3D convolution on $\mathbf{C}_*$ across the batch given 3D kernel $\mathbf{K}_*$, p is a padding size[2], and b is a bias term. As a result, f_{4D} in Equation 2 performs kH times of 3D convolutions.

[1] We omit channel sizes of the tensor for brevity.
[2] We set $p = \lfloor k/2 \rfloor$ in our experiment.

In this work, we implement a fast version of the 4D convolution which performs significantly smaller number of 3D convolutions compared to the original one. We first reshape the correlation tensor of a minibatch as $\mathbf{C} \in \mathbb{R}^{BH\times W\times H'\times W'}$ and make k copies of it. Using the 3D kernels $\{\mathbf{K}_i\}_{i=1}^{k}$, we apply 3D convolution $f_{3\mathrm{D}}$ on each copy and denote its output by $\hat{\mathbf{C}}^i = f_{3\mathrm{D}}(\mathbf{C}, \mathbf{K}_i)$. We again reshape the tensors $\{\hat{\mathbf{C}}^i\}_{i=1}^{k}$ to have size $B \times H \times W \times H' \times W'$ and perform the following:

$$f_{4\mathrm{D}}(\mathbf{C})_i = \hat{\mathbf{C}}^1_{i-p} + \hat{\mathbf{C}}^2_{i-p+1} + ... + \hat{\mathbf{C}}^k_{i+p} + b. \quad (3)$$

Note that the number of 3D convolution operations in our implementation is H times smaller compared to that in the original implementation [12] (k (ours) vs. kH [12]). Given a 4D correlation tensor $\mathbf{C} \in \mathbb{R}^{16\times 30\times 30\times 30\times 30}$, our implementation takes about 0.7 ms while the implementation of [12] takes about 150 ms on a machine with an Intel i7-7820X CPU and an NVIDIA Titan-XP GPU. A high-dimensional convolutions ($\geq$ 5D) are implemented in a similar manner; our implementation of 6D convolution with input in $\mathbb{R}^{16\times 15\times 15\times 3\times 15\times 15\times 3}$ takes about 180 ms on the same machine.

We also manually implement parameter-sharing kernels $k^{\mathrm{nD}}_{\mathrm{psi}}$ and $k^{\mathrm{nD}}_{\mathrm{iso}}$: Before applying convolution, we instantiate high-dimensional kernel filled with zeros and assign parameters to their corresponding indices by addition.

3. Qualitative results

The proposed convolutional Hough matching allows a flexible non-rigid matching and even multiple matching surfaces or objects. To demonstrate the ability of the CHM in matching multiple objects, we visualize some qualitative results of our method (CHMNet) on some toy images with multiple instances in Fig. S1. Top 300 confident matches predicted by our model (CHMNet) are mostly on *common* instances in the input pairs of images. Replacing convolutional Hough matching (learnable local voting layer) to regularized Hough matching [1, 8] (non-learnable global voting layer) severely damages the model predictions; the confident matches become noisy and unreliable, mostly being scattered on background. Without CHM layers, the model fails to localize common instances in the images. Figure S8 also visualizes sample pairs of PF-PASCAL with top 300 confident matches predicted by each model. Our model effectively discriminates between semantic parts and background clutters as seen in the second row of Fig. S8. The absence of CHM layers severely harms the model predictions as seen in the third and last rows of Fig. S8. These results reveal that the proposed CHM layers effectively find reliable matches between common instances across different images while being robust to background clutter even in presence of multiple instances.

Figure S1: Multiple instance matching with top 300 confident matches.

Figure S2: Failure cases on SPair-71k [9] dataset in presence of extreme changes in view-point, large intra-class variation, and deformation. We show the keypoints of ground-truth correspondences in circles and the predicted keypoints in crosses with a line that depicts matching error.

The qualitative comparisons to the recent semantic correspondence approaches [5, 7, 8, 10, 12] are visualized in Figs. S9, S10, and S11. We warp source images to target images using predicted correspondences: Given source keypoints, each model predicts their corresponding positions in target image by using its own keypoint transfer scheme, *e.g.*, nearest neighbor assignment [8, 10], hard-assignment by taking mostly likely match [5, 7, 12] or soft argmax (ours). Using the keypoint correspondences, we compute thin plate spline (TPS) transformation parameters [2] and apply the transformation to source image to align target image. Figure S9 shows the results on PF-PASCAL. Figures S10 and S11 show the results on SPair-71k. Our model effectively warp the source images to align the source objects to the target ones based on predicted correspondences even in presence of large view-point, illumination, and scale differences. Representative failure cases of our model are shown in Fig. S2.

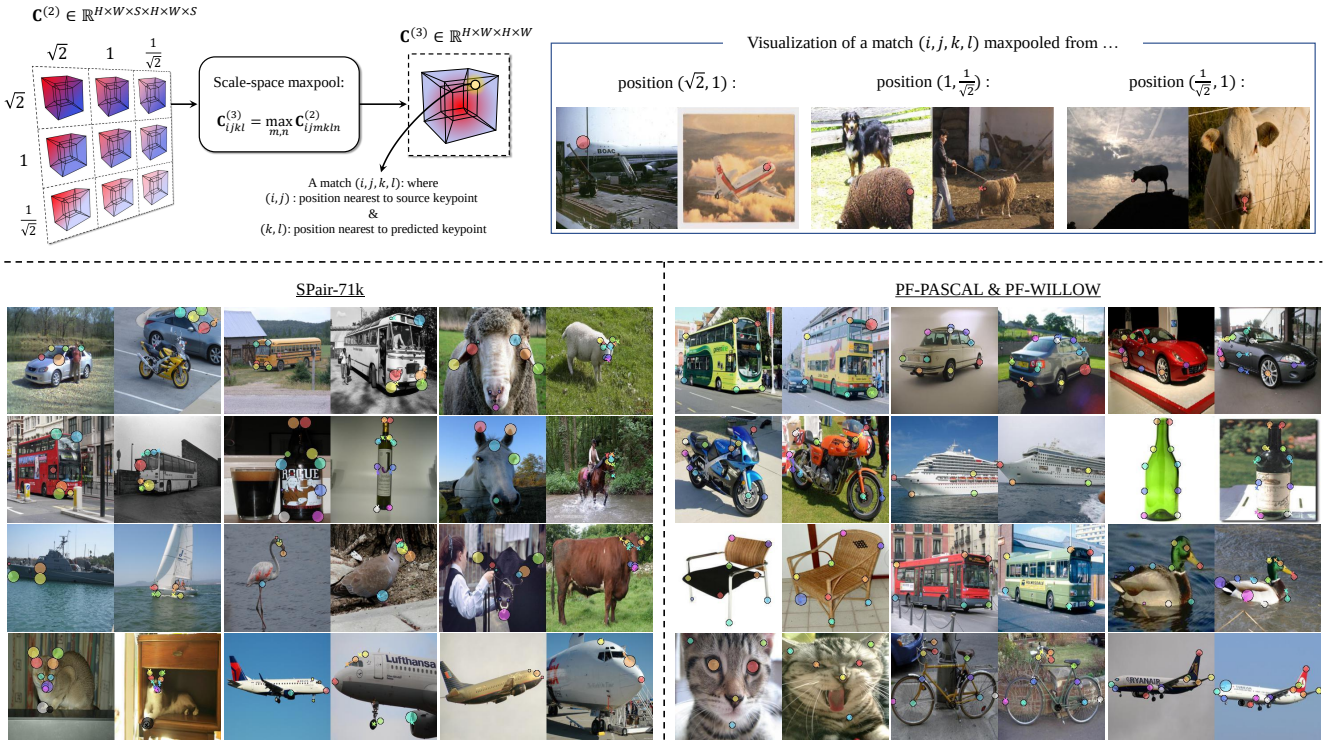


Figure S3: Visualization of maxpooled position in scale-space. In each image pair, we show source keypoints (given) and their corresponding target keypoints (predicted) in circles in left and right images respectively. The size (large, medium, and small) of each circle indicates maxpooled position in scale-space. If both circles of a match are large, its match score is pooled from position $(\sqrt{2}, \sqrt{2})$ in scale space. If the size of one circle is medium and that of the other is small, its match score is from position $(1, 1/\sqrt{2})$ and so on. We show ground-truth target keypoints in crosses with a line that depicts matching error. Best viewed in electronic form.

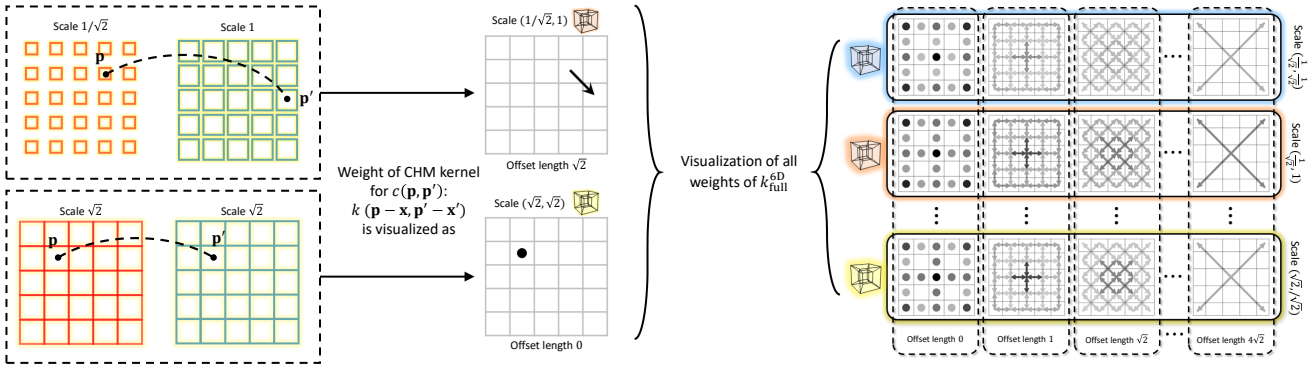


Figure S4: Description of visualizing learned weights of high-dimensional kernels: (Left) The arrows represent the offset vectors relative to the kernel position $(\mathbf{x}, \mathbf{x}')$, and the circles mean zero offset. (Right) For straightforward visualization, we decompose a high-dimensional kernel into multiple 4D kernels (tesseracts) and visualize learned weights of each 4D kernel as a set of maps consisting of offset vectors. Darker offsets mean larger weights while brighter ones mean smaller weights.

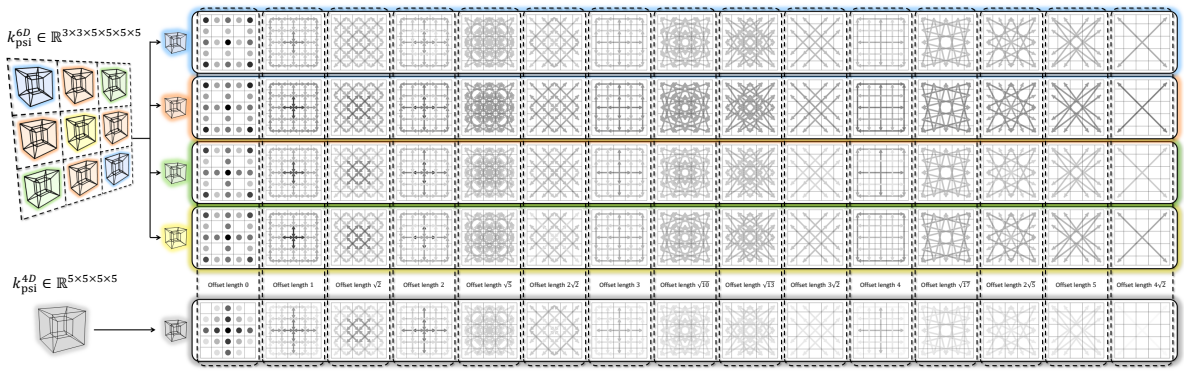


Figure S5: Learned $k_{\text{psi}}^{\text{6D-4D}}$ used in CHMNet. The 6D kernel ($k_{\text{psi}}^{\text{6D}}$) consists of *four* 4D kernels each of which has 55 parameters.

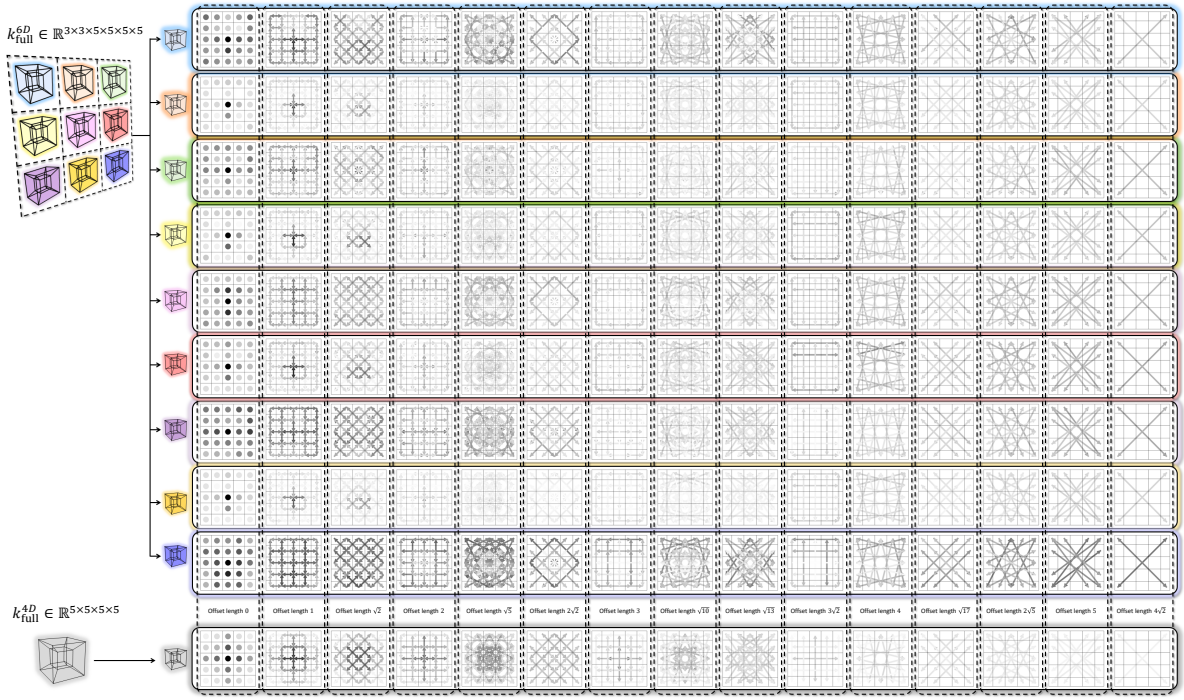


Figure S6: Learned $k_{\text{full}}^{\text{6D-4D}}$. The 6D kernel ($k_{\text{full}}^{\text{6D}}$) consists of *nine* 4D kernels each of which has 625 parameters.

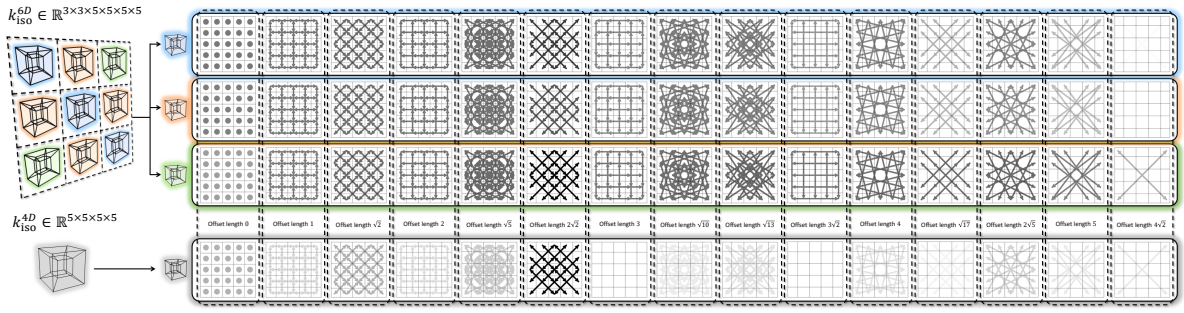
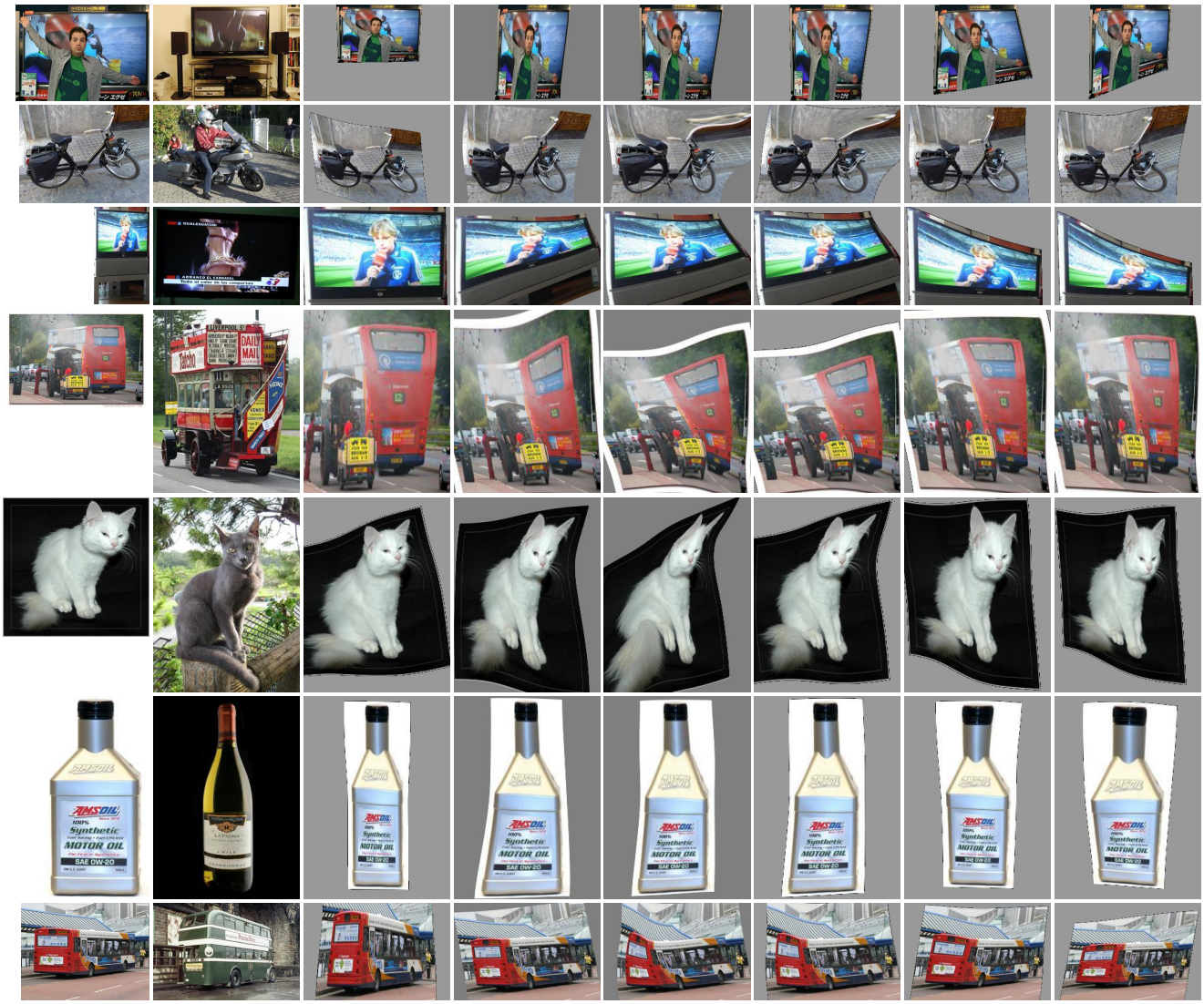


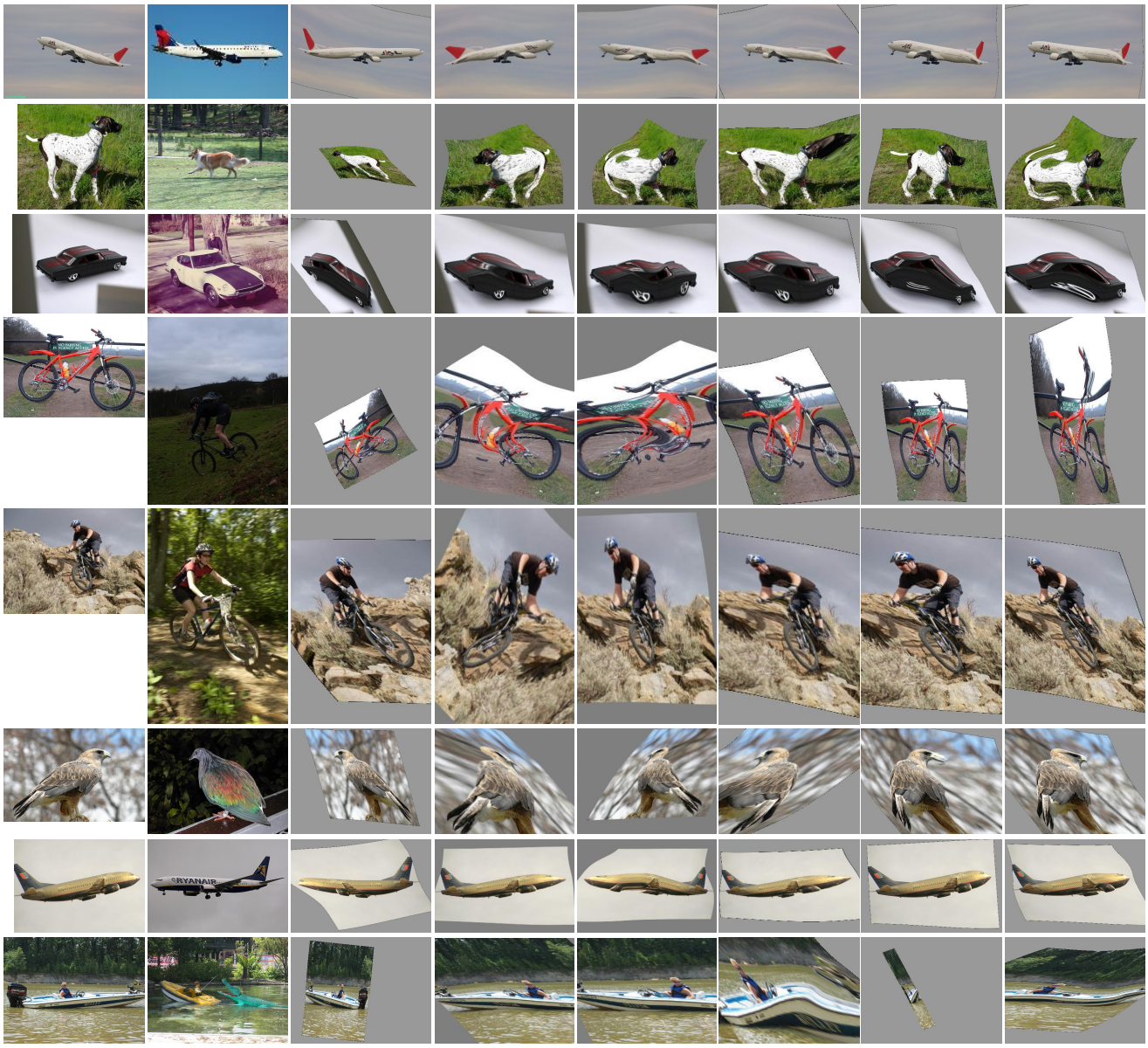
Figure S7: Learned $k_{\text{iso}}^{\text{6D-4D}}$. The 6D kernel ($k_{\text{iso}}^{\text{6D}}$) consists of *three* 4D kernels each of which has 15 parameters.

Figure S8: Sample pairs with top 300 confident matches. TP and FP matches are colored in blue and red respectively.



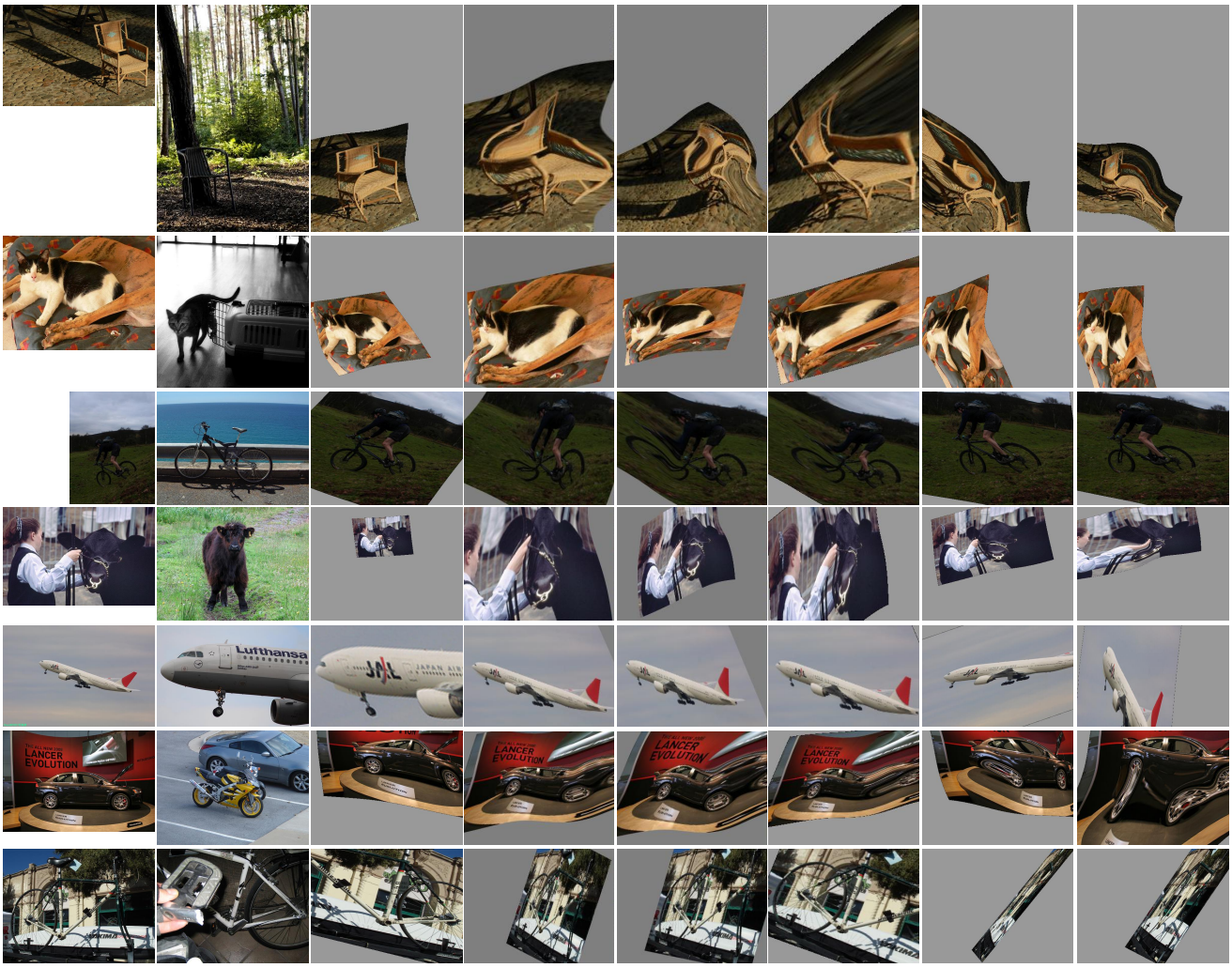
(a) Source image (b) Target image (c) CHMNet (ours) (d) DHPF (e) ANC-Net (f) HPF (g) DCCNet (h) NCNet

Figure S9: Example results on PF-PASCAL [4]: (a) source image, (b) target image (c) CHMNet (ours), (d) DHPF [10], (e) ANC-Net [7], (f) HPF [8], (g) DCCNet [5], and (h) NCNet [12].



(a) Source image (b) Target image (c) CHMNet (ours) (d) DHPF (e) ANC-Net (f) HPF (g) DCCNet (h) NCNet

Figure S10: Example results with large view-point differences from SPair-71k [9]: (a) source image, (b) target image (c) CHMNet (ours), (d) DHPF [10], (e) ANC-Net [7], (f) HPF [8], (g) DCCNet [5], and (h) NCNet [12].



(a) Source image (b) Target image (c) CHMNet (ours) (d) DHPF (e) ANC-Net (f) HPF (g) DCCNet (h) NCNet

Figure S11: Example results with large illumination and scale differences, and truncation from SPair-71k [9]: (a) source image, (b) target image (c) CHMNet (ours), (d) DHPF [10], (e) ANC-Net [7], (f) HPF [8], (g) DCCNet [5], and (h) NCNet [12].

References

[1] Minsu Cho, Suha Kwak, Cordelia Schmid, and Jean Ponce. Unsupervised object discovery and localization in the wild: Part-based matching with bottom-up region proposals. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[2] Gianluca Donato and Serge Belongie. Approximate thin plate spline mappings. In *Proc. European Conference on Computer Vision (ECCV)*, 2002.

[3] Bumsub Ham, Minsu Cho, Cordelia Schmid, and Jean Ponce. Proposal flow. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] Bumsub Ham, Minsu Cho, Cordelia Schmid, and Jean Ponce. Proposal flow: Semantic correspondences from object proposals. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2018.

[5] Shuaiyi Huang, Qiuyue Wang, Songyang Zhang, Shipeng Yan, and Xuming He. Dynamic context correspondence network for semantic alignment. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2019.

[6] Junghyup Lee, Dohyung Kim, Jean Ponce, and Bumsub Ham. Sfnet: Learning object-aware semantic correspondence. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[7] Shuda Li, Kai Han, Theo W. Costain, Henry Howard-Jenkins, and Victor Prisacariu. Correspondence networks with adaptive neighbourhood consensus. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[8] Juhong Min, Jongmin Lee, Jean Ponce, and Minsu Cho. Hyperpixel flow: Semantic correspondence with multi-layer neural features. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2019.

[9] Juhong Min, Jongmin Lee, Jean Ponce, and Minsu Cho. SPair-71k: A large-scale benchmark for semantic correspondence. *arXiv prepreint arXiv:1908.10543*, 2019.

[10] Juhong Min, Jongmin Lee, Jean Ponce, and Minsu Cho. Learning to compose hypercolumns for visual correspondence. In *Proc. European Conference on Computer Vision (ECCV)*, 2020.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[12] Ignacio Rocco, Mircea Cimpoi, Relja Arandjelović, Akihiko Torii, Tomas Pajdla, and Josef Sivic. Neighbourhood consensus networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.