Supplementary material

Vision-as-Inverse-Graphics: Obtaining a Rich 3D Explanation of a Scene from a Single Image

A. Probabilistic Chain Rule within PHNs

In a case of a multidimensional grid-based GMM, it may be undesirable to make predictions directly in the d-dimensional Hough space, as this has $\mathcal{O}(N^d)$ mixture components, where N is the average number of components per dimension. Assuming the number of hidden units in the layer before the softmax one is on average U, the number of weights in the *softmax* layer is $\mathcal{O}(UN^d)$, which would likely lead to over-fitting.

Fortunately PHNs allows us to decompose $p(\mathbf{z}|\mathbf{x}_i)$ into a number of simpler networks via the probabilistic chain rule. We decompose our network which in a direct 3D case has $\mathcal{O}(UN^3)$ weights into two networks with a GMM of dimension 2 and 1, so $\mathcal{O}(UN^2+UN)=\mathcal{O}(UN^2)$ weights. In general we can decompose any grid-based PHN to a single PHN for each dimension, leading to a total of $\mathcal{O}(dUN)$ weights in the last layers. Thus using more but smaller networks we can obtain a GMM which if predicted directly would lead to a network that would have too many weights compared to the dataset size we have.

A.1. PHN Decomposition

We use the following decomposition, where $\mathbf{z}=\mathbf{z}_1\cup\mathbf{z}_2$:

$$p(\mathbf{z}|\mathbf{x}_i)=p(\mathbf{z}_1,\mathbf{z}_2|\mathbf{x}_i)=p(\mathbf{z}_1|\mathbf{z}_2,\mathbf{x}_i)p(\mathbf{z}_2|\mathbf{x}_i). \quad (1)$$

We note that we need a decomposition-composition procedure as we do not simply multiply the values, but create the whole density that represents the product in the whole space. We decompose $H=p(\mathbf{z}|\mathbf{x}_i)$ into two PHNs:

$$H_1(\mathbf{z}_1|\mathbf{z}_2,\mathbf{x}_i)=p(\mathbf{z}_1|\mathbf{z}_2,\mathbf{x}_i), \quad (2)$$
$$H_2(\mathbf{z}_2|\mathbf{x}_i)=p(\mathbf{z}_2|\mathbf{x}_i). \quad (3)$$

H_1 predicts the probabilistic vote conditioned on the observation and a Hough space variable (or a set of variables) given which there is a simpler solution.

A.2. PHN Composition

Suppose $\dim(H_1)=d_1$ and $\dim(H_2)=d_2$, with $\dim(H)=d=d_1+d_2$. We want to compose the GMM obtained from H_1 whose components are indexed by $(i_{\mathbf{z}_1})$ and H_2 whose components are indexed by $(i_{\mathbf{z}_2})$ into a d-dimensional GMM represented by H and indexed by $(i_{\mathbf{z}_1},i_{\mathbf{z}_2})$. We can obtain $H=H_1\otimes_{\mathbf{PHN}}H_2$ representing the density $p(\mathbf{z}|\mathbf{x}_i)$ from these two models as follows: we iterate through the values in the $\mathbf{z}_2$ dimension at the positions of the components as defined by the grid, denoted by $\mathbf{z}_2(i_{\mathbf{z}_2})$. Then we evaluate H_1 by conditioning on each of these values, obtaining $N_{\mathbf{z}_2}$ GMM slices in the $\mathbf{z}_1$-space. We also predict GMM in the $\mathbf{z}_2$-space using H_2. The GMM mixing coefficients are given by:

$$H(i_{\mathbf{z}_1},i_{\mathbf{z}_2}|\mathbf{x}_i)=H_1(i_{\mathbf{z}_1}|\mathbf{x}_i,\mathbf{z}_2(i_{\mathbf{z}_2}))H_2(i_{\mathbf{z}_2}|\mathbf{x}_i), \quad (4)$$

which directly represents a probability distribution in the desired grid, where each component mean is at the location of respective indices and the covariance matrix is the same as before the decomposition.

B. Integration for Log-Likelihood Computation in the Joint Posterior

$$p(\mathbf{z}|X)\propto\frac{\prod_{i=1}^{n}p(\mathbf{z}|\mathbf{x}_i)}{p(\mathbf{z})^{n-1}}, \quad (5)$$

To perform the inference, find the MAP value and sample from the posterior $p(\mathbf{z}|X)$ as in eq. 5 we do not need to find a normalising constant. However, we need it to compute the log-likelihood for the purposes of evaluating the framework.

We wish to compute the normalising constant $1/Z(X)$ for the RHS of eq. 5. We estimate $Z(X)$ by numerical integration of the predicted density in the $\mathbf{z}$-space.

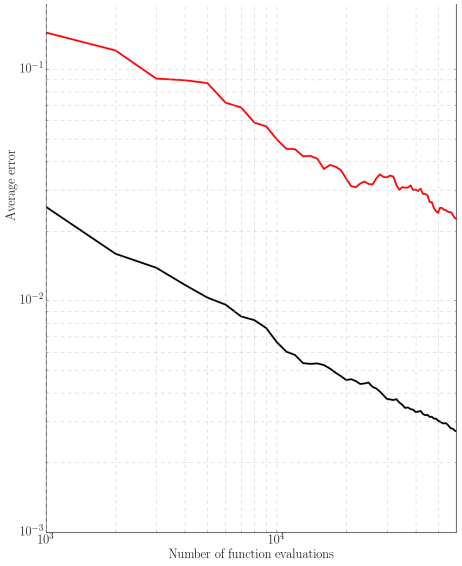

Figure 1. Average error using a uniform sampling (red) and IS (black) in the 2D case.

The function was implemented in C to make the computations faster, and we experimented with integral computation using classical quadrature. We found that in a case of 3 dimensions it was not converging in a desirable time even for

Detector		PHNs		Predictor		
Class	Scale	H^1	H^2	Shape	Azimuth	Lighting
			VGG-16			
C-50-6	C-50-6	C-50-6	C-50-6	C-50-6	C-20-6	C-20-6
C-50-6	C-50-6	C-50-6	C-50-6	C-50-6	C-20-6	C-20-6
C-50-6	C-50-6	C-50-6	C-50-6	C-50-6	C-20-6	C-20-6
F-200	F-200	F-50	F-30	F-50	F-50	F-100
Softmax-2	Sigm-1	F-30	F-30	Softmax-15	Sigm-2	Sigm-5
		Softmax-135	Softmax-10			
			Learning rate			
Class	Scale	H^1	H^2	Shape	Azimuth	Lighting
0.001	0.0002	0.001	0.001	0.001	0.0001	0.0001

Table 1. The configurations of the main models, which are 18 or 19 layers deep, and learning rates. Layer types are: C – Convolutional, F – Fully connected, Sigm – Sigmoid and Softmax. The layers are described as follows: layer type - number of units - filter size.

a single prediction, as it would be much too slow to evaluate all test examples. This integration method would be even slower for dimensions above 3.

Therefore, we evaluate the integral using Monte Carlo integration by sampling from the Hough space. One can sample uniformly in Hough space, but for faster convergence we use Importance Sampling MC integration, where the auxiliary density is obtained as follows: we first evaluate the function on a hypercube centered around each Gaussian. We then define a d-dimensional histogram (normalized to sum to 1) with the weight in each cell being mean value of the function around each Gaussian centre. This leads to sampling a few orders of magnitude more frequently in the regions of a high density.

Importance Sampling (IS) results in a significant speed-up relative to uniform sampling. We conducted experiments in 2D, where we can evaluate the ground-truth integral to high precision using a Python quadrature method `scipy.integrate.dblquad` that uses a technique from the Fortran library QUADPACK. Figure 1 shows the relative error plotted against the number of function evaluations on a log-log scale. We can note that for 1% relative error, we need to evaluate the function only 5k times when using IS, as opposed to around 500k in the uniform approach. The slopes of the lines are close to $-\frac{1}{2}$, as is expected for Monte Carlo integration where the error decreases as $N^{-\frac{1}{2}}$, where N is the number of function evaluations. As can be seen from the plot, the error after the same number of function evaluations is one order of magnitude lower for IS over uniform, and as the error decreases with a square root of the number of function evaluations, integration is two orders of magnitude faster. This allows us to accurately evaluate a single integral in a few seconds, rather than minutes.

C. Details of the networks

Table 1 shows the network configurations and learning rates used for training. We use all 13 convolutional layers of VGG-16 as the core but on 128×128 pixel input. VGG uses padding, but in our convolutional layers we do not use it. The fully connected layers of the detector networks are implemented as filter 1×1 convolutional layers, so they can be efficiently applied in a sliding window manner. We use dropout in the detector networks with $p = 0.5$ and in predictor networks with $p = 0.2$, in the convolutional layers on top of VGG ones. In PHNs a better convergence was obtained when the network was pre-trained using a lower variance scaling factor β to encourage the network to activate all Gaussian components. For azimuth prediction recall that we predict the sine and cosine of the angle.

D. Re-projection error

Figure 2 shows the examples of overlaid checkerboards using ground-truth and predicted cameras.

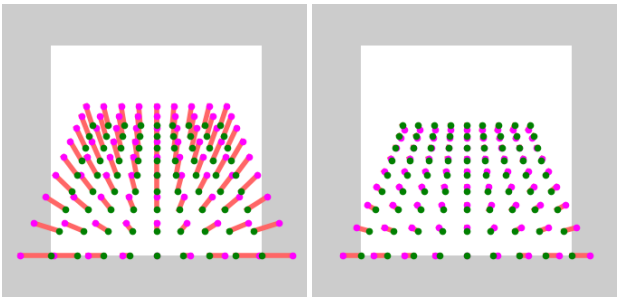

Figure 2. Re-projection error, the green checkerboard is the ground-truth, the magenta one is the prediction, and the pink lines show the errors. The white background is the image frame . Examples of re-projection error (from left): 6.5% and 2.0%.

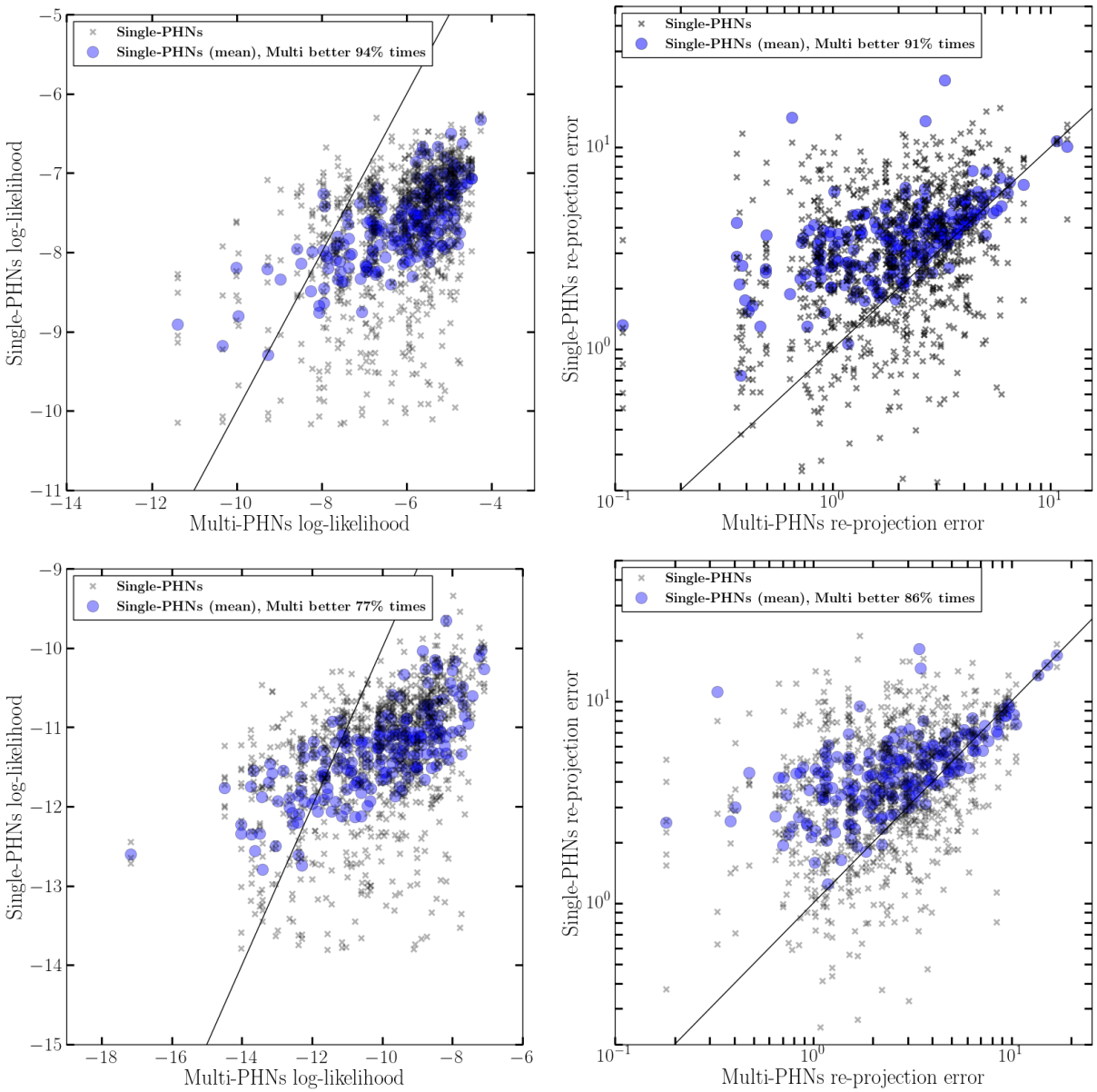

Figure 3. PHNs predictions for 2D (top) and 3D case (bottom), left: log-likelihood (higher better), right: re-projection error (lower better, log-log axes).

E. Further PHN results

Figure 3 gives a detailed comparison of Single-PHNs vs Multi-PHNs for both the 2D (top row) and 3D (bottom row) cases. The left column shows log likelihoods, the right re-projection error. Blue dots indicate the mean Single-PHN prediction for a given image, this is to be compared with the Multi-PHN score (horizontal axis). For the log likelihood case higher is better, so when the Multi-PHN wins the blue dots lie *below* the diagonal line. For the re-projection error lower is better, so when the Multi-PHN wins the blue dots lie *above* the diagonal line. For each text image we show all Single-PHN predictions as well as the mean (blue dot) so that the variability can be seen.

F. More examples of prediction

Figure 4 presents more examples of prediction for real images.

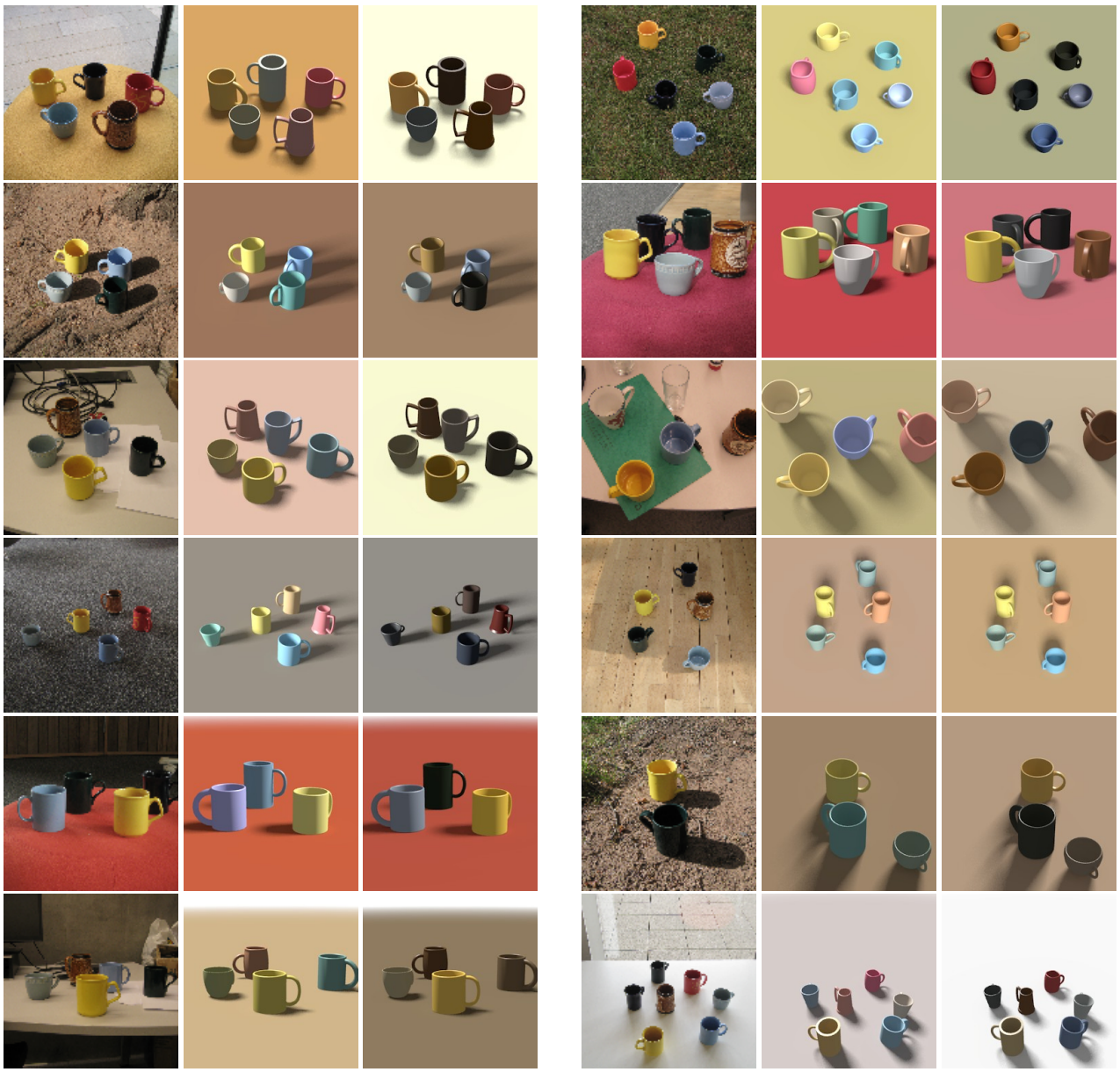

Figure 4. Results on real scenes. The bottom 4 examples have errors in detection. In each the order is the input image, predicted 3D scene, and the result after iterative refinement.