DeepView: View synthesis with learned gradient descent
Training details

1. Reducing Memory Usage

A naive implementation of our model requires a prohibitive amount of RAM for both inference and training. Internally, convolutional layers within the CNNs compute activations with as many as 128 channels. The required RAM for the activations for even a single one of these layers would be $K \times D \times N_r \times N_c \times 128$ — for even moderately sized MPIs this would require hundreds of gigabytes of GPU RAM, and this is for a single network layer within a single iteration of the network.

Reducing RAM during inference is relatively simple as we can tile the inference of each per-iteration CNN across the $D \times N_r \times N_c$ MPI volume, recomputing the gradient components for each view as needed. The tiled inference of the per-iteration CNNs can then be run on GPU. Using this strategy inference takes 50s for a 12 input $64 \times 980 \times 580$ MPI on a P100 GPU.

Reducing RAM at training time is more complex. The inference method described above is not directly applicable since we need to back-propagate the gradients. More importantly, a deep network typically requires tens or hundreds of thousands of iterations to converge and a per step time of $50s$ would make the training time impractical. Instead, during training the network is trained to produce a small 32×32 crop within a target image. By carefully considering both the CNN padding and the MPI and view geometry at each iteration we can compute both the needed crops from each input view and the minimal volume of the MPI that needs to be computed at each iteration in order to produce a given target image patch without border effects. Note that this calculation is complex since in order to produce the gradient components for a specific input view crop at iteration n we need to have available the MPI volume visible to that crop at the previous iteration $n-1$, but in order to compute that MPI volume in $n-1$ we need to have more of the input view's area etc. The required MPI volume that needs to computed for a given target crop thus increases as the number of LGD iterations increases.

To further reduce RAM we discard activations within each of the per-iteration networks, as described in [3] and tile computation along the depth plane dimension. Even with these memory optimizations, at higher resolutions we are limited to training a single example patch at a time and rely on synchronized replicas to get large effective batch sizes.

2. Generating training samples from the Spaces dataset

To generate a sample we first randomly select a scene and a random rig position. We then randomly select the input views from the set of possible view sets within the rig (see Figure 4). These views form the input to the network. The target view is then randomly selected from the remaining views across all rigs that are within 6cm of of the convex hull of the input views, and a maximum of 7cm from the plane of the input views. A 32×32 crop is then chosen from the target view.

3. Training hyperparameters

We used distributed training with synchronized replicas to increase the effective batch size. We typically used 16 replicas, but for some experiments we used less replicas but computed gradients from two examples in each replica before accumulating the batch.

As in Adler *et al*. [1] we used global gradient clipping, with a threshold of 8.0.

We use feature similarity [2, 5] as our training loss $\mathcal{L}_f$, specifically the *conv1_2*, *conv2_2* and *conv3_3* layers of a pre-trained VGG-16 network [4]. The *conv3_3* layer has a receptive field of 40×40, so we first reflect padded the 32×32 target and training crop to be 40×40. Following Chen *et al*. [2] we then computed the loss by summing the L_1 difference of the layers, weighted by the empirically determined weights of $[11.17, 35.04, 29.09]$. Finally, we divided the loss by the area of the crop 32×32, producing a loss in a more reasonable range which improved the numerical stability of global gradient clipping.

References

[1] J. Adler and O. Öktem. Learned primal-dual reconstruction. *IEEE Transactions on Medical Imaging*, 37:1322–1332, 2018. 1

[2] Q. Chen and V. Koltun. Photographic image synthesis with cascaded refinement networks. *CoRR*, abs/1707.09405, 2017. 1

[3] T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *CoRR*, abs/1604.06174, 2016. 1

[4] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1

[5] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. *CoRR*, abs/1801.03924, 2018. 1