Sparse, Smart Contours to Represent and Edit Images
Supplemental

1 Implementation Details

Our models were implemented in Tensorflow. We will release our trained models and code. To achieve the editing effects, we built a simple graphical user interface (GUI) that allows editing the contour map using simple operations such as moving, scaling or erasing sets of contours. Note that although the user edits the contour map, the underlying features change in the appropriate manner (for example, if some contours are removed, then the features associated with those contours are also removed).

Computing Input Representation: Edge probability maps were extracted using [1] followed by non-maximum suppression as a post processing. The computed maps were binarized by keeping x percentile of edges, where x is the desired percentage of nonzero pixels in the binary edge map. We group the edges into contours and filter out short contours (length less then 10 pixels). Gradients are computed by simple forward differences.

LFN and HFN: Both LFN and HFN are "U-Nets" [2]. We adopt the notations in [2] and denote a convolution layer with k filters followed by batch normalization by `Ck` for ReLU activation, and `Clk` for LeakyReLU (slop 0.2). For 256×256 images the number of filters in the encoder is given by: `Cl64-Cl128-Cl256-Cl512-Cl512-Cl512`, hence the spatial resolution of the bottleneck is 4×4. The number of filters in each layer of the decoder is given by: `C512-C512-C256-C128-C64-C64` (filters in decoder are bigger because of the skip connections). All convolutional filters are size 4×4, and we use strides of size 2.

Dilated-Patch Discriminator Our discriminator (Fig. 3(c)) is a combination of a "patch discriminator" [2] and a branch of dilated convolution filters. Let $D_r k$ denote a dilated convolution with sampling rate r, leaky ReLU and k filters. Then our patch discriminator architecture is given by `Cl64-Cl128-Cl256` followed by 4 parallel dilated convolutions $\{\mathtt{D}_2\mathtt{256}, \mathtt{D}_4\mathtt{256}, \mathtt{D}_8\mathtt{256}, \mathtt{D}_{12}\mathtt{256}\}$, which are concatenated, and a final convolution layer with a single channel output.

The network used for learning the input representation (see Sec. 4) is similar and consists of a branch of dilated convolutions, each followed by regular a convolution layer. That is, $\{\mathtt{D}_4\mathtt{32} - \mathtt{C3}, \mathtt{D}_8\mathtt{256} - \mathtt{C3}, \mathtt{D}_{12}\mathtt{256} - \mathtt{C3}, \mathtt{D}_{16}\mathtt{256} - \mathtt{C3}\}$, which are added to form the final output.

Training hyper-parameters and details For each dataset, LFN and HFN were trained from scratch (weights were initialized from a Gaussian distribution with mean 0 and standard deviation 0.02). We used Adam optimizer with $\beta = 0.5, \epsilon = 1e^{-4}$, and batch size of 16 in all training runs except when training on 512×512 where we used size of 8. We used learning rate of 0.0002 for the generator, and 0.00002 for the discriminator, with decay rate of 0.98 every 10000 steps. During training we resized the images such that the small dimension is at the desired resolution and then randomly cropped to desired size. The relative weight of the adversarial loss vs. reconstruction loss was 100 in all our experiments.

During the HFN training, the weights of the LFN remained fixed, and we alternate between stepping the discriminator and generator in ratio of 2:1 in favor of the discriminator. When working with learned features, the weights feature network remained fixed during HFN training.

We trained on the VGG dataset at two spatial resolutions: 256×256 and 512×512. For Birds and Dogs, we used the original train/test splits, and for the VGG dataset we filtered out low resolution images from the train and test sets, to avoid reconstruction of JPEG artifacts. The VGG has 30227 samples and was trained for 210 epochs for both LFN and HFN; CUB Birds has 8855 samples and was trained for 720 epochs; Dogs dataset has 12000 training samples and was trained for 500 epochs.

Texture Loss We use the texture loss in Section 6.1 to evaluate our reconstructions. This loss was defined in [3].

References

[1] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *ICCV*, 2013. 1

[2] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016. 1

[3] I. Ustyuzhaninov, W. Brendel, L. Gatys, and M. Bethge. What does it take to generate natural textures? 2016. 2