Appendix

A Additional Editing Examples

Figures 7, 8, 9, 10, 11, and 12 show additional results of our editing method to change a model to achieve a variety of effects across an entire distribution of generated images. Each figure illustrates a single low-rank change of a StyleGAN v2 model derived from the user gestures shown in the top row. The twelve pairs of images shown below the top row of each figure are the images that score highest in the context direction d, out of a random sample of 1000: that is, these are images that are most relevant to the user's context selection. For each image, both the output of the unmodified original model and the modified model are shown. All changes are rank-one changes to the model, except Figure 10, which is rank ten, and Figure 12, which is rank three.

B Solving for Λ Algebraically

To strengthen our intuition, here we describe the closed-form solution for Λ in the linear case. Recall from Equations 13 and 15:

$$W_1 k_* = v_* \tag{20}$$

$$W_1 = W_0 + \Lambda d^T \tag{21}$$

In the above we have written $d = C^{-1}k_*$ as in Eqn. 16 for berevity. Then we can solve for both W_1 and Λ simultaneously by rewriting the above system as the following matrix product in block form:

$$\left[\begin{array}{c|c} W_1 & \Lambda \end{array}\right] \left[\begin{array}{c|c} I & k_* \\ \hline -d^T & 0 \end{array}\right] = \left[\begin{array}{c|c} W_0 & v_* \end{array}\right] \tag{22}$$

$$\left[\begin{array}{c|c} W_1 & \Lambda \end{array}\right] = \left[\begin{array}{c|c} W_0 & v_* \end{array}\right] \left[\begin{array}{c|c} I & k_* \\ \hline -d^T & 0 \end{array}\right]^{-1} \tag{23}$$

In practice, we do not solve this linear system because a neural network layer is nonlinear. In the nonlinear case, instead of using matrix inversion, Λ is found using the optimization in Equation 17.

C Implementation details

Datasets To compare identical model edits in different settings, we prepare a small set of saved editing sessions for executing an change. Each session

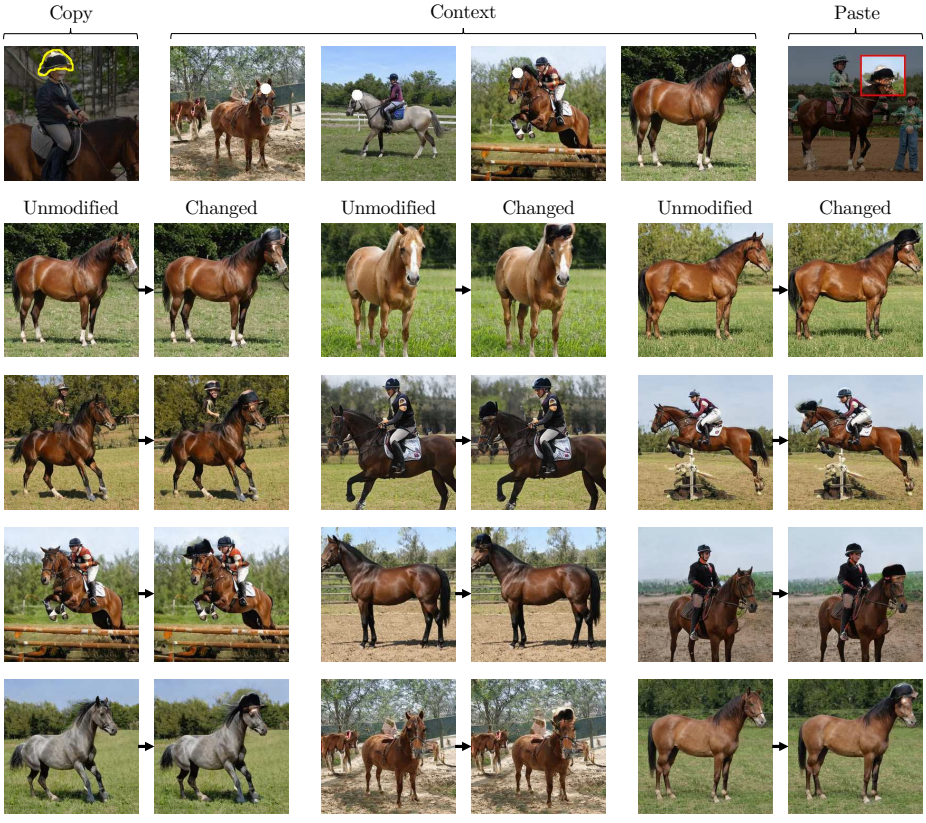

Fig. 7: **Giving horses a hat to wear.** After one hat is pasted onto an example of a horse, and after the user has pointed at four other horse heads, the model is changed so that horses in a variety of poses, settings, shapes, and sizes all get a hat on their head. This is not merely a re-balancing of the distribution of the model. This change introduces a new kind of image that was not generated before. The original training data does not include hats on horses, and the original pretrained StyleGANv2 does not synthesize hats on any horses.

corresponds to a set of masks that a user has drawn in order to specify a region to copy and paste, together with any number of context regions within generated images for a model. Benchmark editing sessions are included with the source code.

Large-scale datasets are used only for pretraining the generative models. The generative models we use are trained on the following datasets. The face model is trained on Flickr-Faces-HQ (FFHQ) [37], a dataset of 70,000 1024×1024 face images. The outdoor church, horse, and kitchen models are trained on LSUN image datasets [77]. LSUN provides 126,000 church images, 2.2 million kitchen images, and 2 million horse images at resolutions of 256×256 and higher.

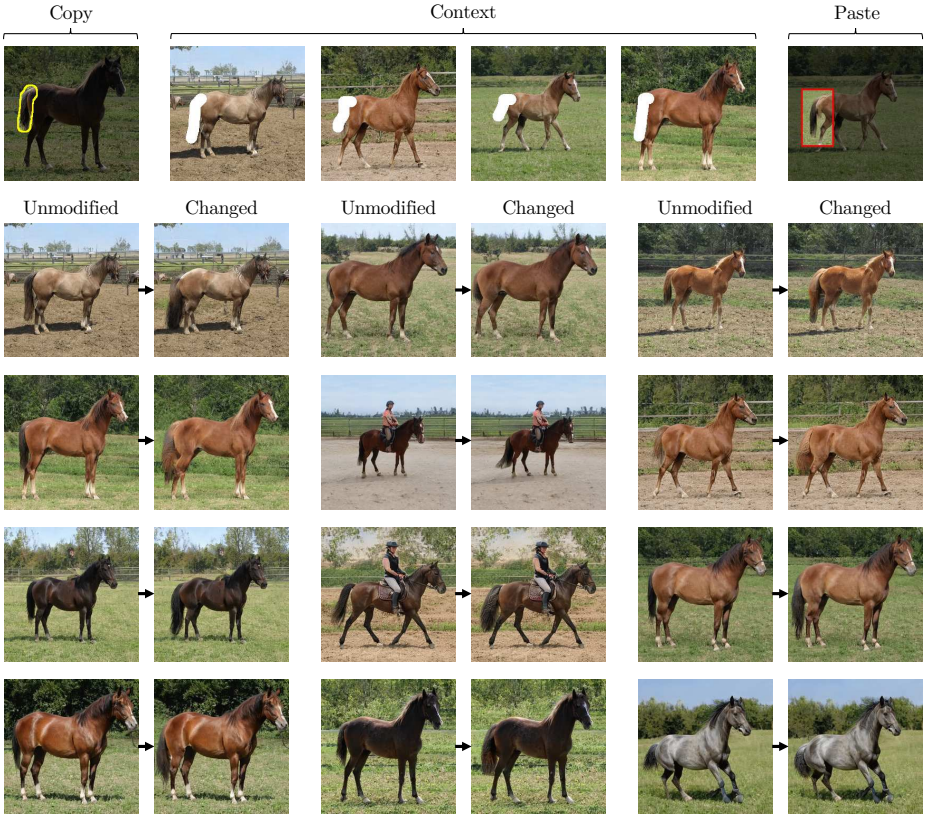

Fig. 8: **Giving horses a longer tail.** Notice that the color, shape, and occlusions of the tail vary to fit the specific horse, but in each case the tail is made longer, as demonstrated in the pasted example.

Generators We rewrite two different generative model architectures: Progressive GAN and StyleGAN v2. The Progressive GAN generator has 18.3 million parameters and 15 convolutional layers; we edit a model pretrained on LSUN kitchens. We also edit StyleGAN v2 [40]. StyleGAN v2 has 30 million parameters and 14 convolutional layers (17 layers for the higher-resolution faces model). We edit StyleGAN v2 models trained on FFHQ faces, LSUN churches, and LSUN horses. All the model weights were those published by the original GAN model authors. For StyleGAN v2, we apply the truncation trick with multiplier 0.5 when running the model.

Metrics To quantify undesired perceptual differences made in edits, we use the Learned Perceptual Image Patch Similarity (LPIPS) [81] metric to compare unedited images to edited images. We use the default Alexnet-based LPIPS network weights as published by the original LPIPS authors. To focus the measurement on undesired changes, we follow the method of the GAN projection

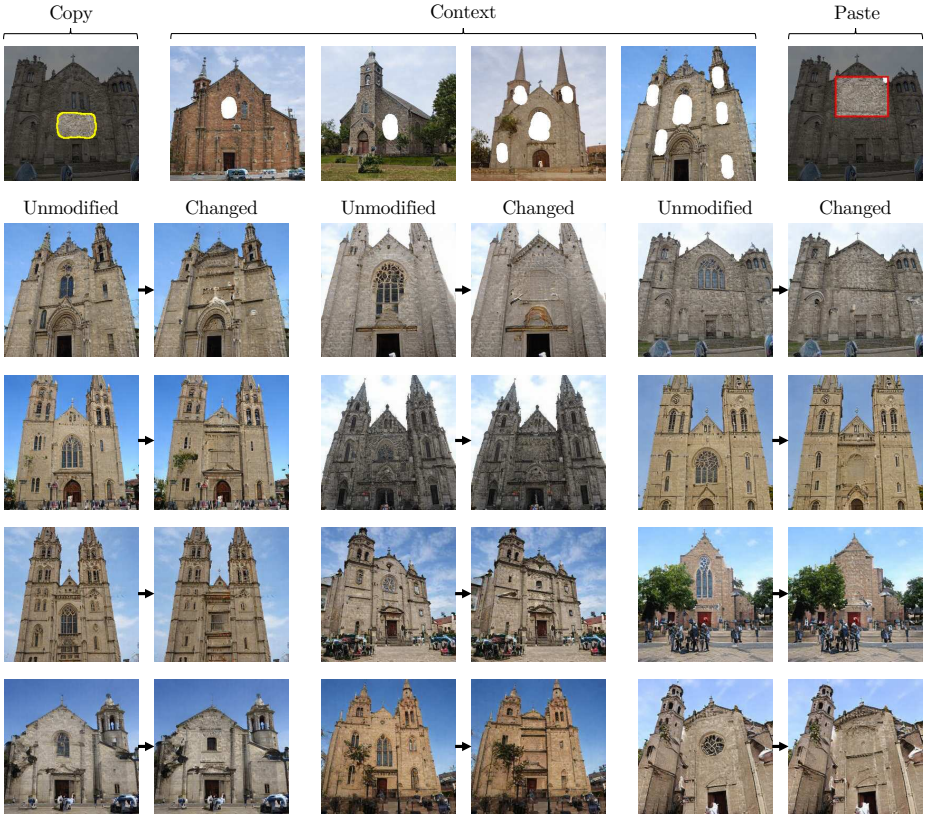

Fig. 9: **Removing main windows from churches.** The modified model will replace the central window with a blank wall, or with a wall with some different details.

work [32] and mask out portions of the image that we intend to change, as identified by a semantic segmentation network. For faces, we segment the image using a modified BiSeNet [76] as published by ZLL as faceparsing-Pytorch [84]. For churches, we segment the image using the Unified Perceptual Parsing network [73].

To quantify the efficacy of the change, we also use pretrained networks. To detect whether a face image is similing, we use a Slim-CNN [66] facial attribute classifier. To determine if domes have successfully been edited to other types of objects, we again use the Unified Perceptual Parsing network, and we count pixels that have changed from being classified as domes to buildings or trees.

User studies Human realism measurements are done using Amazon Mechanical Turk (AMT). For each baseline editing method, 500 pairs of images are generated comparing an edited image using our approach to the same image edited using a baseline method, and two AMT workers are asked to judge which of the pair is

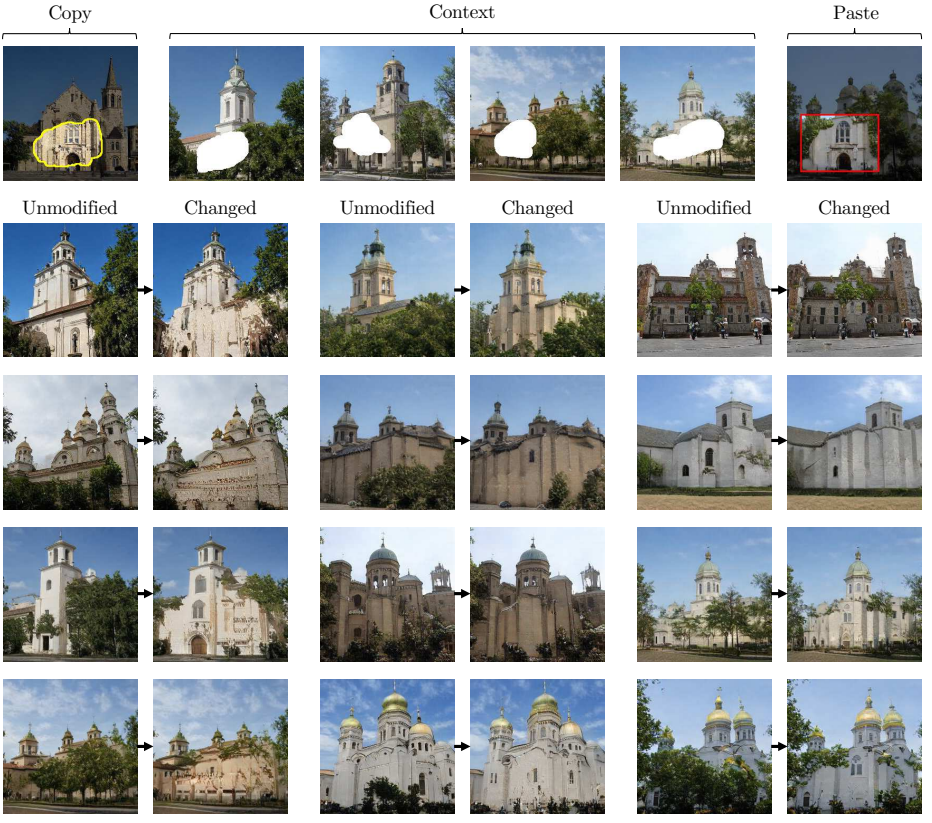

Fig. 10: **Reducing the occlusion of buildings by trees.** This edit removes the trees in front of buildings. Note that the model can still synthesize trees next to buildings.

more realistic, for a total of 1000 comparative judgements. We do not test the fantastical domes-to-trees edit, which is intended to be unrealistic.

D Rank Reduction for D_S

In this section we discuss the problem of transforming a user's *context* selection $K \in \mathbb{R}^{N \times T}$ (Section 4) into a constraint subspace $D_S \in \mathbb{R}^{N \times S}$, where the desired dimensionality $s \ll t$ is smaller than the number of given feature samples T provided in K.

We shall think of this as a lossy compression problem. Use P to denote the probability distribution of the layer $L-1$ features (unconditioned on any user selection), and think of K as a discrete distribution over the user's t context examples. We can then use cross-entropy $H(K, P)$ to quantify the information in K, measured as the message length in a code optimized for the distribution P. To

Fig. 11: **Removing earrings.** Removing one set of earrings generalizes to many different types of earrings appearing in different poses.

express this information measure in the setting used in Section 3, we will model P as a zero-centered Gaussian distribution $P(k) = (2\pi)^{-n/2} \exp -k^T C^{-1} k/2$ with covariance C.

If we the normalize the basis using the ZCA whitening transform Z, we can express P as a spherical unit normal distribution in the variable $k' = Zk$. This yields a concise matrix trace expression for cross entropy:

$$\text{Let } C = U\Sigma U^T \text{ be the eigenvector decomposition} \tag{24}$$

$$Z \triangleq C^{-1/2} = U\Sigma^{-1/2}U^T \tag{25}$$

$$k' \triangleq Zk \tag{26}$$

$$K' \triangleq ZK \tag{27}$$

$$P(k') = (2\pi)^{-n/2} \exp(-k'^T k'/2) \tag{28}$$

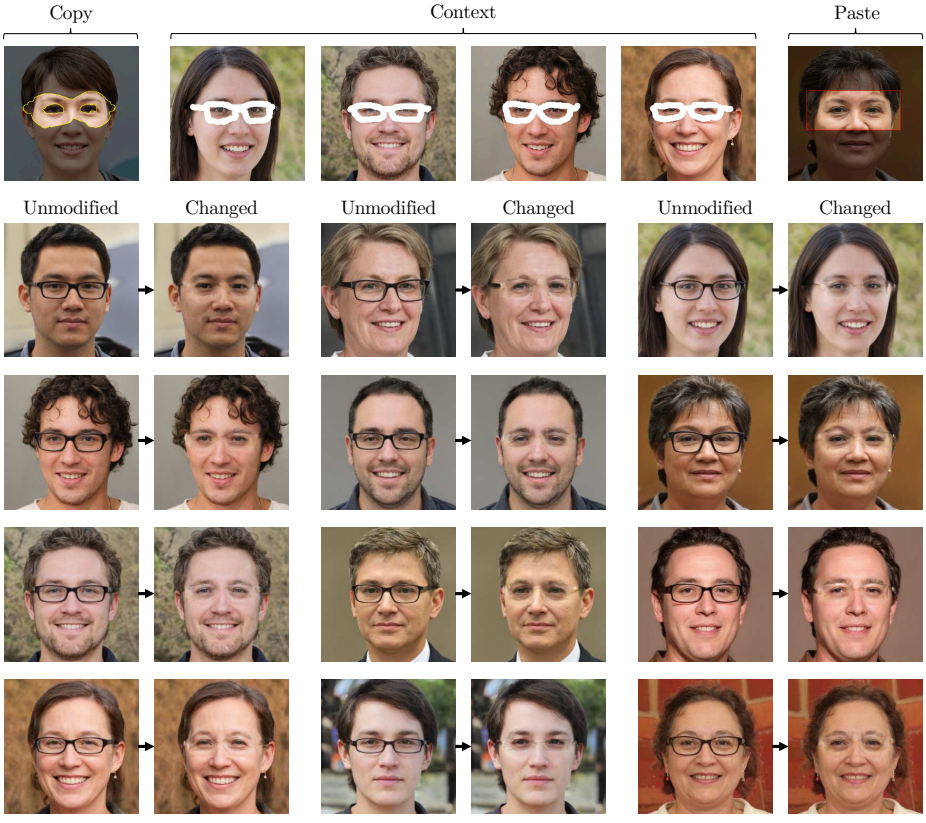

Fig. 12: **Removing glasses.** Note that glasses of different shapes are removed, and most facial structure is recovered. This is a rank-three change. Although most of the glasses have been removed, this edit did not remove the temples (side parts) of some glasses, and did not remove refraction effects.

$$H(K', P) = \sum_{k' \in K'} \frac{1}{t} \log P(k') \tag{29}$$

$$= \frac{1}{2t} \sum_{k' \in K'} k'^T k' + \frac{n}{2t} \log 2\pi \tag{30}$$

$$= \frac{1}{2t} \operatorname{Tr}\left(K'^T K'\right) + \frac{n}{2t} \log 2\pi \tag{31}$$

In other words, by assuming a Gaussian model, the information in the user's context selection can be quantified the trace of a symmetric matrix given by inner products over the whitened context selection.

To reduce the rank of the user's context selection, we wish to project the elements of K' by discarding information along the $R = N - S$ most uninformative directions. Therefore, we seek a matrix $Q_R^* \in \mathbb{R}^{N \times R}$ that has R orthonormal

columns, chosen so that the projection of the samples $Q_R Q_R^T K'$ minimize cross-entropy with P:

$$Q_R^* = \arg\min_{Q_R} H(Q_R Q_R^T K', P) \tag{32}$$

$$= \arg\min_{Q_R} \mathrm{Tr}\left(K'^T Q_R Q_R^T Q_R Q_R^T K'\right) \tag{33}$$

$$= \arg\min_{Q_R} \mathrm{Tr}\left(Q_R^T K' K'^T Q_R\right) \tag{34}$$

The trace minimization Eqn. 34 is an instance of the well-studied trace optimization problem [46] that arises in many dimension-reduction settings. It can be solved by setting the columns of Q_R^* to a basis spanning the space of the eigenvectors for the smallest R eigenvalues of $K'_{\mathsf{ctx}} K'^T$.

Denote by $Q_S^* \in \mathbb{R}^{N \times S}$ the matrix of orthonormal eigenvectors for the S *largest* eigenvalues of $K'_{\mathsf{ctx}} K'^T$. Then we have $(I - Q_R^* Q_R^{*T})k' = Q_S^* Q_S^{*T} k'$, i.e., erasing the uninteresting directions of Q_R^* is the same as preserving the directions Q_S^*. This is the S-dimensional subspace that we seek: it is the maximally informative low-dimensional subspace that captures the user's context selection.

Once we have Q_S^* within the whitened basis, the same subspace can be expressed in unwhitened row space coordinates as:

$$D_S = Z^T Q_S^* = Z Q_S^* \tag{35}$$

E Axis-aligned rank reduction for D_S

The identification of axis-aligned units most relevant to a user's context selection can also be analyzed using the same rank-reduction objective as Section D, but with a different family for P. Instead of modeling P as a Gaussian with generic covariance C, we now model it as an axis-aligned Gaussian with diagonal covariance $\Sigma = \mathrm{diag}(\sigma_i)$. Then the optimal basis Q_S^* becomes the unit vectors for the unit directions e_i that maximize the expected ratio

$$\sum_{k \in K_{\mathsf{ctx}}} \frac{\left(e_i^T k\right)^2}{\sigma_i^2} \tag{36}$$

In Section 5.2 this scoring is used to identify the units most relevant to watermarks in order to apply GAN dissection unit ablation.

F Experiment Details and Results

Table 2 shows the quantitative results of comparing our method with various baselines on editing a StyleGANv2 [40] LSUN church [77] model. For both edits, our method modifies the 7th convolution layer of the generator, with Adam optimizer [41], 0.05 learning rate, 2001 gradient iterations, and projecting to a low-rank change every 10 iterations (and also after the optimization loop).

For `domes` → `trees`, a rank 1 edit is performed. (These settings are also the defaults provided in the user interface, and were used for video demos.) For `domes` → `spires`, a rank 10 edit is performed.

For the StyleGANv2 FFHQ [39] edit shown in main paper 1, our method modifies the 9th convolution layer of the generator, also with Adam optimizer [41], 0.05 learning rate, 2001 gradient iterations, and projecting to a low-rank change every 10 iterations (and also after the optimization loop).

For all experiments, the baseline that finetunes all weights uses the Adam optimizer [41] with 2001 iterations and a learning rate of 10^{-4}.

G Reflection Experiment Details

In Section 5.3, we found the rank-one rule reversal change for the abstract window lighting rule as follows.

1. **Generation**: we use the GAN to generate 15 images in two ways, one adding windows, and one removing windows, by activating and deactivating window-correlated units. The window correlated units are identified using dissection [7].
2. **Annotation**: a user masks illuminated regions of the 15 images far from the windows that show reflected light that differs between the pairs.
3. **Optimization**: we optimize a change in the weights of the layer to reverse the behavior of the reflected light in the masked areas, to match dark output when there is a window and bright output when there is no window. This optimization is constrained to one direction by using an SVD reduction to rank one every 10 iterations.

The optimization is computed at each individual layer, and we use the layer that achieves the lowest loss with a rank-one change: for this experiment, this is layer 6 of the model.

H Selecting a Layer for Editing

There are two ways to view a convolutional layer: either as a computation in which information from neighboring locations is combined to detect or produce edges, textures, or shapes; or as a memory in which many independent feature mappings are memorized.

In our paper we have adopted the simple view that a layer acts as an associative memory that maps from one layer's local feature vectors to local patches of feature vectors in the next layer. This view is appropriate when layer representations have features in which neighboring locations are disentangled from one another. In practice, we find that both ProgressiveGAN and StyleGAN representations have this property. For example, if a feature patch is rendered in isolation from neighboring features, the network will usually render the same object as it does in the context of the full featuremap.

In Figures 13 and 16, we measure the similarity between patches rendered in isolation compared to same-sized patches cropped out of the full model, using Fréchet Inception Distance (FID) [29]. Lower FIDs indicate less dependence between neighboring patches, and higher FIDs indicate higher dependence between neighbors. These graphs show that layers 6-11 in StyleGANv2 and layers 4 and higher in Progressive GAN are most appropriate for editing as an associative memory. (Note that in StyleGANv2, the nth featuremap layer is the output of the $n-1$th convolutional layer, because the first featuremap layer is fixed. In Progressive GAN, the nth featuremap layer is the output of the nth convolutional layer.)

Figures 14 and 15 visualize individual patches rendered in isolation at various layers of StyleGANv2, and compare those to the entire image rendered together. Figures 17 and 18 visualize the same for Progressive GAN.

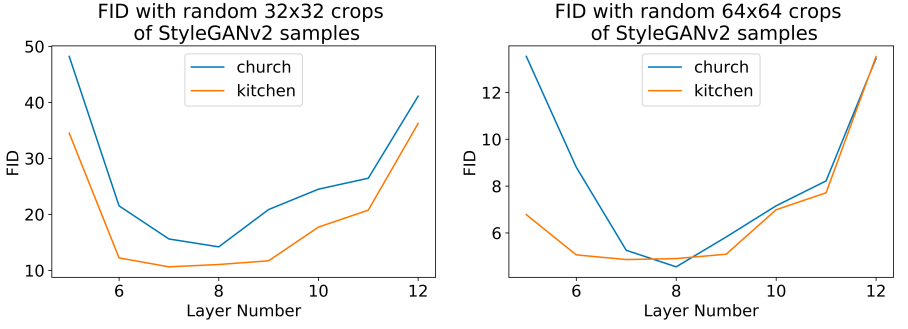

Fig. 13: FID of rendered cropped activations with respect to random crops of StyleGANv2 generated images. In StyleGANv2, the nth convolutional layer outputs the $n+1$th featuremap layer. The layer numbers above correspond to featuremap layers.

Fig. 14: Comparison of rendered cropped activations at various layers of StyleGANv2 generated LSUN church images.

Fig. 15: Comparison of rendered cropped activations at various layers of StyleGANv2 generated LSUN kitchen images.

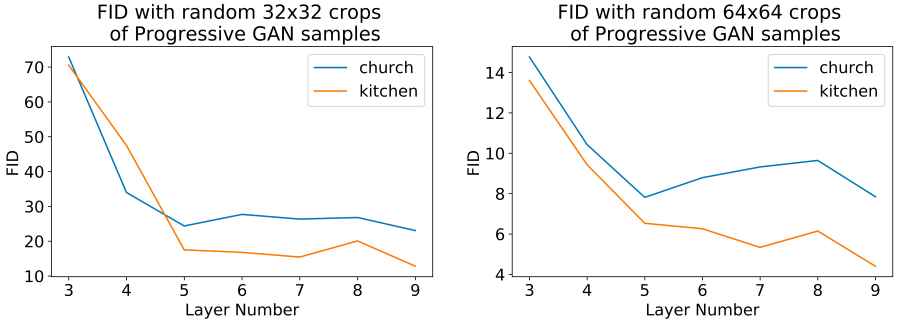

Fig. 16: FID of rendered cropped activations with respect to random crops of Progressive GAN generated images.

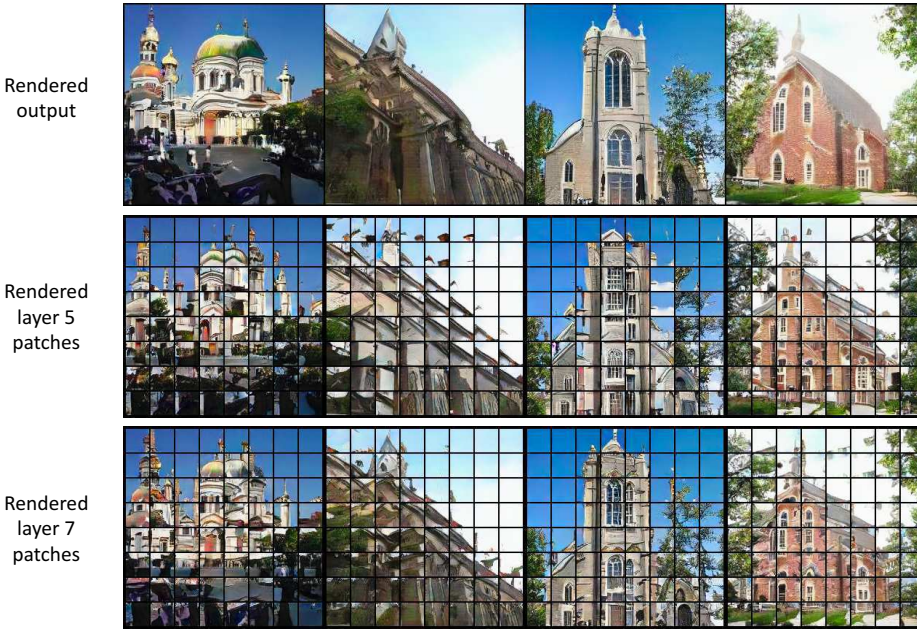

Fig. 17: Comparison of rendered cropped activations at various layers of Progressive GAN generated LSUN church images.

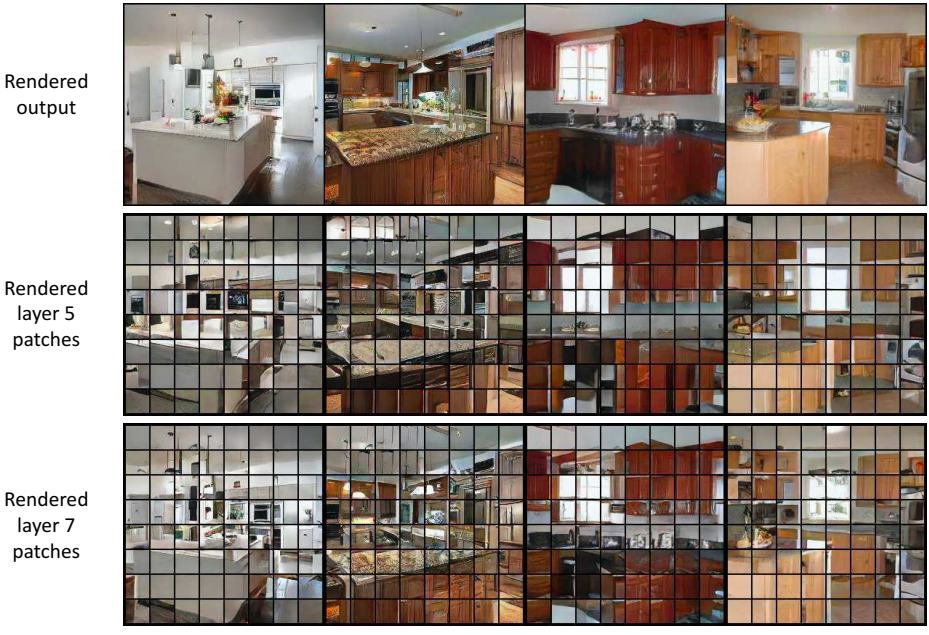

Fig. 18: Comparison of rendered cropped activations at various layers of Progressive GAN generated LSUN kitchen images.