How GAN Generators can Invert Networks in Real-Time

Rudolf Herdt RHERDT@UNI-BREMEN.DE
Maximilian Schmidt SCHMIDT4@UNI-BREMEN.DE
Daniel Otero Baguer OTERO@UNI-BREMEN.DE
Jean Le'Clerc Arrastia LECLERC@UNI-BREMEN.DE
Peter Maaß PMAASS@UNI-BREMEN.DE
Center of Industrial Mathematics, University of Bremen, Germany

Editors: Berrin Yanıkoğlu and Wray Buntine

Abstract

In this work, we propose a fast and accurate method to reconstruct activations of classification and semantic segmentation networks by stitching them with a GAN generator utilizing a 1x1 convolution. We test our approach on images of animals from the AFHQ wild dataset, ImageNet1K, and real-world digital pathology scans of stained tissue samples. Our results show comparable performance to established gradient descent methods but with a processing time that is two orders of magnitude faster, making this approach promising for practical applications.

Keywords: Computer Vision; Deep Learning; GAN; Network Inversion

1. Introduction

In this work, we efficiently reconstruct the extracted features of a ResNet50 (He et al., 2016) classification network trained on ImageNet1K (Deng et al., 2009) and a ResNet34 backbone of a semantic segmentation network trained on digital pathology scans of histochemically stained tissue samples, by inverting them using a pretrained, frozen generative adversarial network (GAN) (Goodfellow et al., 2014). For the ResNet50 trained on ImageNet1K, we invert it using a StyleGAN2 (Karras et al., 2020) and a BigGAN (Brock et al., 2019). For the ResNet34 trained on digital pathology data, we use a custom GAN trained on digital pathology data. The architecture is shown in Appendix A. In the following, we use the term "feature extractor" for both the classification and semantic segmentation network.

In order to bring the feature extractor and GAN generator together, we learn a linear transformation in the form of a 1x1 convolution to stitch them. The 1x1 convolution is trained to map from a hidden layer of the feature extractor into a hidden layer of the GAN generator. Utilizing this mapping, we can quickly reconstruct activations of the feature extractor, by transferring them through the convolutional connection into the GAN generator, i.e., we use the GAN generator as a decoder to invert the feature extractor. Here, only the 1x1 convolution is being trained. Both the feature extractor and GAN generator are kept frozen.

Through the stitching, we investigate the similarity between representations learned by the GAN generators and the feature extractors, and also which combination of layers is more compatible.

2. Related Work

We use model stitching (Lenc and Vedaldi, 2015) to stitch a feature extractor with a GAN, which allows us to reconstruct activations of the feature extractor in real-time (on average, it takes 0.034s per image using StyleGAN2). A 1x1 convolution is used as the stitching layer, which is methodically closely related to the work of Bansal et al. (2021). They applied a sequence of batch norm, 1x1 convolution, and batch norm to stitch two image classification networks together in a hidden layer. In contrast, we do the stitching in order to see how similar the learned representations of hidden layers of a feature extractor are to the learned representations of hidden layers of a GAN generator by comparing the reconstructed images with the original images. In addition, we reconstruct activations of the feature extractor by sending them through the GAN generator.

Those activations can also be reconstructed through gradient descent (Mahendran and Vedaldi, 2014). The disadvantage is that it is slow since several forward-backward passes through the network are needed. We use gradient descent as in Olah et al. (2017) and gradient descent without any regularization as baselines to compare our GAN method against. Another method is to reconstruct the activations by training a decoder network to invert a feature extractor from a hidden layer (Dosovitskiy and T.Brox, 2016). But such an approach makes it necessary to train a new decoder for every new layer in the feature extractor where activations should be reconstructed from. In contrast, with our method, it is only necessary to train a new 1x1 convolution and keep the same GAN generator. Finally, in the work of Dosovitskiy and T.Brox (2016), the decoder was trained to minimize a reconstruction error in the input space of the classification network, whereas we train the stitching layer to minimize a reconstruction error at the hidden layer where we transfer from.

3. Methodology

In this section, we show an overview of our GAN-based reconstruction method and the gradient descent methods we use as a baseline. In addition, we describe the evaluation metrics used in the experiments.

3.1. Overview

Figure 1 shows an overview of our approach. For combining a feature extractor with a GAN generator in a hidden layer, i.e., stitching LayerX of the feature extractor and LayerY of the GAN generator, we train a 1x1 convolution to transfer between the two layers. After the 1x1 convolution is trained, we can invert the feature extractor at LayerX by propagating the activations from LayerY till the output of the GAN generator, as shown in steps 1 - 3 in Figure 1.

First, the activations at LayerX are extracted. Then we transfer them through the 1x1 convolution to get them from the latent space of LayerX into the latent space of LayerY. After that, the activations are scaled to match the spatial size of LayerY. For example, layer b32.conv0 of StyleGAN2 has a spatial size of 32x32. Therefore, before injecting our activations into that layer, we scale them to a spatial size of 32x32 via nearest neighbor scaling. Finally, we inject the activations into LayerY of the GAN generator while generating an output with the generator from a random seed. This means that we still get variations

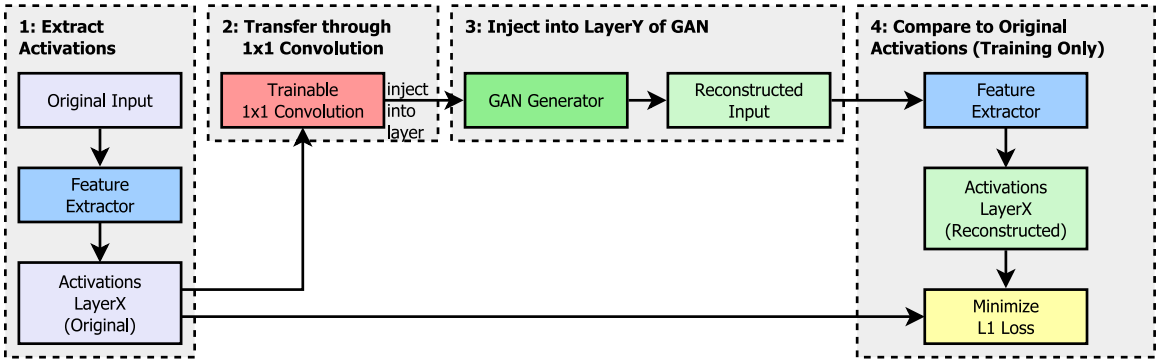

Figure 1: Usage and training process for stitching a GAN generator and a feature extractor in a hidden layer.

from the used seed or noise after the injection layer. The output of the GAN generator is referred to as reconstructed input in Figure 1.

For training, we need to compare the reconstructed and original input and train the 1x1 convolution to minimize a loss between the two. To compare them, we again use the feature extractor, as shown in step 4 in Figure 1, and optimize the 1x1 convolution to minimize the L1 loss in LayerX. We do not minimize the loss in the input layer, because we expect the network to increasingly discard unnecessary low-level information the deeper the input propagates through the network. For example, individual positions of the dots on leopard fur, which the network would then be unable to reconstruct properly. Since we aim to reconstruct the activations of LayerX, we also use this layer to compute the loss.

3.2. Evaluation

To evaluate how good our GAN-based inversion method is, we can first look at some reconstructions, i.e., if the reconstructions do not resemble the original images, then the method has failed for this combination of layers. To get a more comparable quantification of the reconstruction quality, we compare the activations of the reconstructed and the original input using L1 loss and cosine similarity, averaged over 300 or 500 images.

With this, we can investigate for a given layer in the feature extractor, which layer in the GAN reconstructs it best, i.e., which layer in the GAN is most compatible with it.

Similar to the training, we do not compare the original input and the result of the inversion method in the input layer but in a hidden layer of a second feature extractor. The motivation for this is that we want to know how accurately the features extracted by LayerX were reconstructed, not whether they also match low-level features that LayerX has already discarded. And the reason for using a second network for the comparison is to reduce overfitting effects, especially with gradient descent without regularization (see also Section 3.5).

The whole process is shown in Figure 2. First, we extract the activations α^x from the original input from the test network at LayerX'. Second, we generate a reconstruction of the original input from the activations at LayerX from the network we wish to invert, using either gradient descent or our GAN method for the reconstruction process. Third, we pass

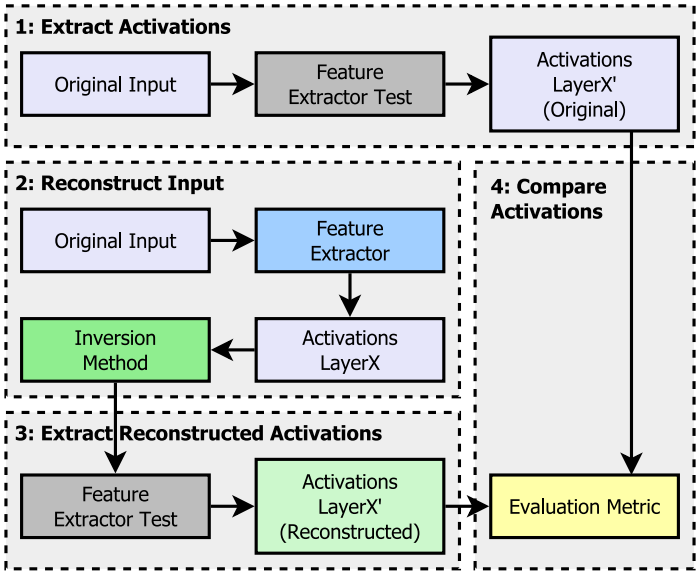

Figure 2: Comparing inversion methods at test time.

the reconstruction into the second network (test network) and extract its activation $\alpha^{x\prime}$ at LayerX'. Lastly, in step 4 we compare both activations α^{x} and $\alpha^{x\prime}$ from the test network using two different evaluation metrics (see Section 3.4).

3.3. Gradient Descent

Algorithm 1 describes how we perform gradient descent, as proposed by Olah et al. (2017). When using no regularization, both the functions g and h in (*) are the identity function, and σ is the sigmoid function to keep the values of the reconstructed image in a 0 to 1 range. When using regularization, h is an inverse Fourier transform followed by correlating colors so that z' is in color de-correlated Fourier space. In addition, g is one-pixel jittering to reduce noise in the inversion. With one-pixel jittering, we first pad the input by 2 pixels on each side and then randomly move it up to one pixel horizontally and vertically. We perform $n = 512$ forward-backward passes through the feature extractor and use the Adam optimizer from Kingma and Ba (2015) with a learning rate of 0.05.

Algorithm 1 Gradient Descent inversion from LayerX

Input: data x, network from layers 0 to X F_X, L1 loss L
$y \leftarrow F_X(x)$
$z'_0 \sim \mathcal{N}(0, 0.01)$
for $i \leftarrow 0$ to n **do**
  loss $\leftarrow \nabla_{z'_i}[L(F_X(g(\sigma(h(z'_i)))), y)]$ (*)
  backpropagate loss
  update z'_i
end for

3.4. Evaluation Metrics

Let $x \in I$ be the original input image and F_X be the feature extractor from layers 0 to LayerX, i.e., the layer from which we want to reconstruct the activations. Further, let L_X be the latent space of LayerX, $f : L_X \to I$ an inversion method (steps 2 - 3 in Figure 1 or gradient descent), and F'_X the test network from layers 0 to LayerX'.

We compare the activations $\alpha^{x\prime} = F'_X(f(F_X(x)))$ and $\alpha^x = F'_X(x)$ by computing a similarity metric $d(\alpha^{x\prime}, \alpha^x)$ with $d : L'_X \times L'_X \to \mathbb{R}$. For d, we use the following two evaluation metrics:

- Cosine similarity: Let C be a matrix of shape (H, W) with

$$C_{ij} = \frac{\alpha^{x\prime}[:, i, j] \cdot \alpha^x[:, i, j]}{\max(||\alpha^{x\prime}[:, i, j]||_2 \cdot ||\alpha^x[:, i, j]||_2, \epsilon)}.$$

  Then $d(\alpha^{x\prime}, \alpha^x) = \text{mean}(C)$. This means we first compute pixelwise cosine similarity, followed by computing the mean of the resulting matrix.

- L1 loss: $d(\alpha'_x, \alpha_x) = \text{mean}(|\alpha'_x - \alpha_x|)$

Both cosine similarity and L1 loss work pixelwise. They measure whether concepts have been correctly reconstructed and also whether they have been reconstructed at the correct position.

3.5. Convolutional Downsampling

Strided convolutions tend to introduce checkerboard noise (Odena et al., 2016), and strided convolutional downsampling tends to introduce noise in the gradient (Olah et al., 2017). In the extreme setting of a 1x1 convolution with stride 2, which both the ResNet50 and ResNet34 we utilize have, only one-quarter of the input pixels would be used based on a fixed grid pattern, and the other three-quarters would be discarded. Then only this one-quarter of pixels would receive any gradient in the backward pass, which introduces noise when reconstructing the input with gradient descent, as shown in Figure 3.

This behavior is also reflected in our evaluations: Suppose both the network we invert with gradient descent and the testing network, where we compare the reconstruction to the original image, have convolutional downsampling. In that case, the measured quality is relatively higher compared to when the testing network does not have convolutional downsampling. We additionally use VGG, which has max pooling downsampling for the AFHQ wild and ImageNet1K data, and a ResNet34 with bilinear downsampling for the Digipath data.

4. Experiments

4.1. Setup

In this section, we describe the datasets, models, and hardware we use in our experiments[1].

1. The code is available at: https://github.com/herdtr/gan-stitching

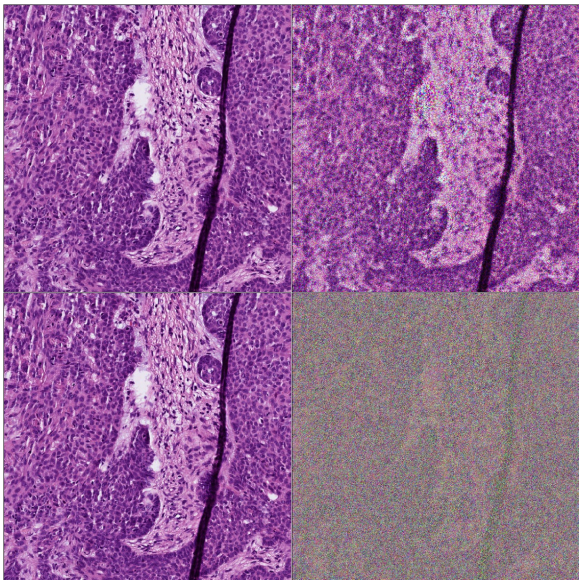

Figure 3: Comparison between inverting a ResNet34 and ResNet34 using bilinear downsampling from Layer2. The top line shows the result for a ResNet34 using bilinear downsampling and the bottom line for regular ResNet34. Left is both times the real input image, and on the right is the resulting inversion, generated using gradient descent without any regularization. Both networks are randomly initialized, and they are untrained.

4.1.1. Datasets

We conduct our experiments on three different datasets, once on tissue scans in digital pathology data (Digipath), on Animal Faces-HQ (AFHQ) containing wild animal faces (Choi et al., 2020), and on ImageNet1K (Deng et al., 2009). The images in the Digipath data have a spatial size of 600x600, whereas the AFHQ wild images have a size of 512x512. Here, we downsample them from 512x512 to 224x224. For the ImageNet1K images, we downsample and crop them to 224x224.

An in-house dataset and models that are not publicly available are used for our experiments on digital pathology data, whereas both the AFHQ and ImageNet1K datasets are available on Kaggle. Table 1 shows the number of training and validation samples for the datasets we use. For the ImageNet1K validation data, we used every hundredth image of the whole validation data of 50 000 images resulting in 500 validation images.

Table 1: Train- and validation set sizes for the three datasets Digipath, AFHQ wild, and ImageNet1K.

	Digipath	AFHQ	Imagenet1K
Training Size	156 680	4738	1 281 167
Validation Size	300	500	500
Used Image Size	600x600	224x224	224x224

4.1.2. Hardware and Software

For all our experiments, we use a Linux server with 4 Nvidia RTX A6000 GPUs. On the software side, we use PyTorch (Paszke et al., 2019) version 1.11.0 and Torchvision version 0.12.0.

4.1.3. Training

All networks in our experiments are running in eval mode. We only train a 1x1 convolution to map between two layers. The weights of the feature extractor and the GAN generator are kept frozen. We use Adam with a learning rate of 0.01 as an optimizer and train with a batch size of 8.

4.2. Results on the AFHQ and ImageNet1K Datasets

In this section, we describe our results from the experiments on the AFHQ wild and ImageNet1K data. We stitch a ResNet50 from Torchvision models trained on ImageNet, once with a StyleGAN2 trained on AFHQ wild data at a resolution of 512x512, and once with a BigGAN trained on ImageNet at a resolution of 128x128. In both cases, we scale the output of the GAN generator to 224x224 and compare the results to gradient descent with and without regularization. As test networks, we use ResNet34 and VGG19 from Torchvision models, both trained on ImageNet. The stitching is performed at 4 points of the ResNet50, namely at the output of Layer1 to Layer4. When stitching on the AFHQ wild data, we train for 30 epochs, i.e., 142 140 images in total. For the stitching on the ImageNet1K data, we train for a single epoch that involves 1 281 167 images.

Table 2: Test results for the AFHQ wild data. The best results are highlighted in bold. The evaluation time is the total time needed for all 500 validation images.

		AFHQ				
Layer	Method	Cosine Similarity	L1 Loss	Cosine Similarity VGG	L1 Loss VGG	Evaluation Time
Layer1	BigGAN	0.930 ± 0.011	0.195 ± 0.024	0.733 ± 0.040	0.071 ± 0.009	**4s**
	StyleGAN2	0.954 ± 0.008	0.162 ± 0.019	**0.801** ± 0.029	**0.061** ± 0.007	6s
	StyleGAN2$_I$	0.951 ± 0.008	0.167 ± 0.019	0.799 ± 0.008	0.062 ± 0.008	6s
	StyleGAN2$_{DC}$	0.734 ± 0.019	0.422 ± 0.031	0.413 ± 0.036	0.119 ± 0.009	6s
	GD with Reg	0.933 ± 0.010	0.196 ± 0.023	0.705 ± 0.032	0.078 ± 0.009	251s
	GD no Reg	**0.974** ± 0.007	**0.117** ± 0.020	0.722 ± 0.033	0.074 ± 0.006	238s
Layer2	BigGAN	0.835 ± 0.017	0.158 ± 0.013	0.614 ± 0.032	0.054 ± 0.004	**3s**
	StyleGAN2	0.878 ± 0.015	0.135 ± 0.011	**0.716** ± 0.024	0.046 ± 0.003	6s
	StyleGAN2$_I$	0.871 ± 0.015	0.139 ± 0.011	0.707 ± 0.024	0.047 ± 0.003	6s
	StyleGAN2$_{DC}$	0.540 ± 0.018	0.288 ± 0.014	0.264 ± 0.015	0.083 ± 0.004	6s
	GD with Reg	0.905 ± 0.009	0.119 ± 0.008	0.711 ± 0.023	**0.045** ± 0.003	395s
	GD no Reg	**0.931** ± 0.013	**0.098** ± 0.011	0.670 ± 0.024	0.048 ± 0.003	380s
Layer3	BigGAN	0.316 ± 0.018	0.137 ± 0.005	0.180 ± 0.018	0.031 ± 0.001	**3s**
	BigGAN+1	0.724 ± 0.038	0.083 ± 0.009	0.486 ± 0.042	0.026 ± 0.002	**3s**
	StyleGAN2	0.778 ± 0.032	0.070 ± 0.007	0.579 ± 0.039	0.023 ± 0.002	6s
	StyleGAN2$_I$	0.741 ± 0.031	0.077 ± 0.007	0.523 ± 0.033	0.025 ± 0.002	6s
	StyleGAN2$_{DC}$	0.309 ± 0.020	0.143 ± 0.006	0.162 ± 0.016	0.032 ± 0.002	6s
	GD with Reg	**0.840** ± 0.023	**0.060** ± 0.006	**0.651** ± 0.036	**0.021** ± 0.002	530s
	GD no Reg	0.762 ± 0.036	0.074 ± 0.008	0.492 ± 0.036	0.025 ± 0.002	515s
Layer4	StyleGAN2	0.527 ± 0.117	0.868 ± 0.108	0.287 ± 0.095	0.128 ± 0.013	**6s**
	GD with Reg	**0.697** ± 0.054	**0.763** ± 0.089	**0.428** ± 0.061	**0.121** ± 0.012	589s
	GD no Reg	0.341 ± 0.029	0.945 ± 0.060	0.153 ± 0.027	0.122 ± 0.013	574s

Table 3: Test results for the ImageNet1K data. The best results are highlighted in bold. The evaluation time is the total time needed for all 500 validation images.

		ImageNet1K				
Layer	Method	Cosine Similarity	L1 Loss	Cosine Similarity VGG	L1 Loss VGG	Evaluation Time
Layer1	BigGAN	0.907 ± 0.028	0.230 ± 0.049	0.696 ± 0.087	0.080 ± 0.018	**3s**
	StyleGAN2	0.926 ± 0.023	0.209 ± 0.043	0.768 ± 0.059	0.070 ± 0.014	6s
	StyleGAN2$_I$	0.931 ± 0.022	0.202 ± 0.042	**0.775** ± 0.059	**0.068** ± 0.015	6s
	GD with reg	0.911 ± 0.021	0.235 ± 0.043	0.710 ± 0.046	0.081 ± 0.013	251s
	GD no reg	**0.955** ± 0.014	**0.159** ± 0.034	0.707 ± 0.056	0.079 ± 0.010	238s
Layer2	BigGAN	0.818 ± 0.030	0.163 ± 0.022	0.591 ± 0.055	0.054 ± 0.007	**3s**
	StyleGAN2	0.833 ± 0.026	0.157 ± 0.018	0.645 ± 0.043	0.051 ± 0.005	6s
	StyleGAN2$_I$	0.848 ± 0.024	0.150 ± 0.018	0.666 ± 0.047	0.050 ± 0.006	6s
	GD with reg	0.883 ± 0.015	0.131 ± 0.012	**0.675** ± 0.032	**0.046** ± 0.004	395s
	GD no reg	**0.909** ± 0.020	**0.112** ± 0.015	0.646 ± 0.034	0.048 ± 0.004	380s
Layer3	BigGAN	0.376 ± 0.053	0.137 ± 0.015	0.213 ± 0.050	0.030 ± 0.003	**3s**
	BigGAN+1	0.676 ± 0.044	0.099 ± 0.017	0.440 ± 0.053	0.026 ± 0.003	**3s**
	StyleGAN2	0.557 ± 0.070	0.116 ± 0.021	0.361 ± 0.061	0.029 ± 0.003	6s
	StyleGAN2+1	0.583 ± 0.062	0.112 ± 0.020	0.374 ± 0.056	0.028 ± 0.003	6s
	StyleGAN2$_I$	0.634 ± 0.048	0.103 ± 0.018	0.421 ± 0.050	0.027 ± 0.003	6s
	StyleGAN2$_I$ + 1	0.669 ± 0.044	0.097 ± 0.017	0.449 ± 0.047	0.026 ± 0.003	6s
	GD with reg	**0.782** ± 0.041	**0.079** ± 0.013	**0.574** ± 0.053	**0.022** ± 0.003	530s
	GD no reg	0.725 ± 0.050	0.089 ± 0.015	0.468 ± 0.050	0.025 ± 0.003	515s

Table 2 shows the results of different metrics when inverting a ResNet50 classifier trained on ImageNet data. It compares different inversion methods and measures their run time for the 500 validation images of the AFHQ data. The "Evaluation Time" column shows the accumulated time over the 500 images. Table 3 evaluates the different inversion methods on the 500 validation images of the ImageNet1K data. In Table 2, we report the metrics on the validation data for the epoch which had the highest cosine similarity on the validation data and in Table 3 we report the metrics after one epoch of training.

We do the reconstruction at the output of Layer1 to Layer4 of ResNet50, shown in the "layer" column, each time with six different methods:

1. BigGAN where the 1x1 stitching convolution is trained on the ImageNet1K data
2. StyleGAN2 where the 1x1 stitching convolution is trained on the AFHQ wild data
3. StyleGAN2 where the 1x1 stitching convolution is trained on the ImageNet1K data, denoted as StyleGAN2$_I$
4. StyleGAN2 without the 1x1 stitching convolution (i.e. directly copying the intermediate output of the feature extractor into the GAN), denoted as StyleGAN2$_{DC}$
5. By gradient descent with regularization (GD with reg)
6. By gradient descent without any regularization (GD no reg).

For the GAN methods, we need not only a starting layer in ResNet50 but also a layer in the GAN that we want to stitch into. Unless noted otherwise, we choose the first convolution of a layer that is the same number of samplings away from the output as the starting layer in ResNet50 is away from the input. Exemplary, for StyleGAN2 that means we stitch

Figure 4: Input reconstruction from activations at the output of Layer3 of ResNet50 for 8 test samples where the GAN method had the highest cosine similarity. The top row shows the original dataset images, the middle line shows their reconstructions from StyleGAN2, and the bottom line shows their reconstructions using gradient descent with regularization.

- ResNet50 Layer1 → StyleGAN2 b128.conv0 (Layer1 is two downsampling steps away from the input of ResNet50, and b128.conv0 is two upsampling steps away from the output of StyleGAN2)
- ResNet50 Layer2 → StyleGAN2 b64.conv0
- ResNet50 Layer3 → StyleGAN2 b32.conv0
- ResNet50 Layer4 → StyleGAN2 b16.conv0

The notation of the form "+1", as used in Table 3 for Layer3, refers to stitching into a layer that is one level shallower. In Section 4.4, we investigate different combinations of stitching layers. Cosine similarity and L1 loss are evaluated for each of the 500 validation images. The tables show the mean ± standard deviation.

The GAN methods take considerably less time than the gradient descent methods, as shown in the "Evaluation Time" column. For measuring the time of the GAN methods, we let it run over the dataset 10 times and take the mean run time. The reported evaluation times do not include the time for computing the evaluation metrics, only for computing the reconstructions. For example, for Layer3, the GAN methods take between 3 and 6 seconds, whereas gradient descent takes more than 500 seconds. In our experiments, we observed an average runtime improvement with StyleGAN2 of 73x with a minimum improvement of 39x and a maximum improvement of 101x compared to gradient descent without regularization. Similar improvements can also be observed compared to gradient descent with regularization. In this case, we observed an average runtime improvement of 76x with a minimum of 41x and a maximum of 103x. Our method runs faster because it only needs one forward pass through the feature extractor and the GAN generator instead of 512 forward-backward passes through the feature extractor with gradient descent.

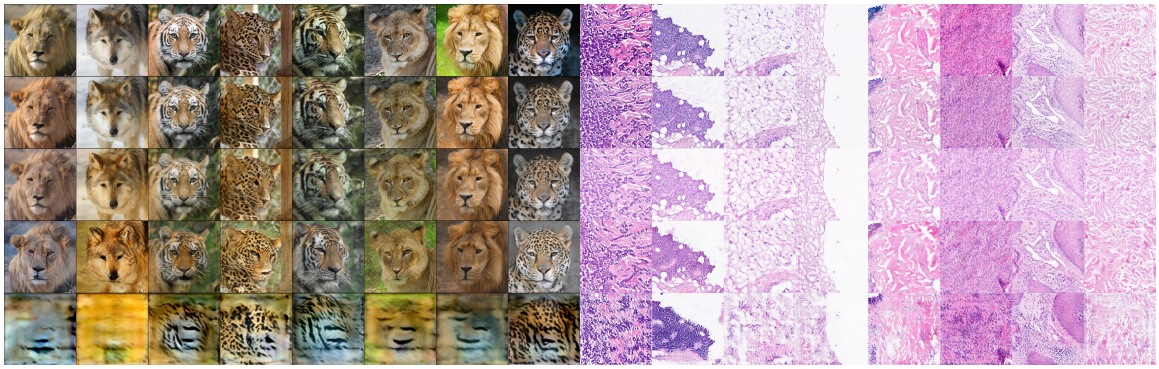

Figure 5: Comparison of the original images in the first row with our GAN-reconstructed activations down from Layer1 to Layer4 in the following rows. Left using StyleGAN2 on AFHQ data, right using a custom GAN on Digipath data.

In Layer3 and Layer4, gradient descent with regularization performs best in cosine similarity and L1 loss. In earlier layers, it is worse than gradient descent without any regularization, likely due to the effect of one-pixel jittering. We use the one-pixel jittering to reduce noise in the reconstruction, but it forces that even if we move the reconstruction one pixel, it should still have similar activations in the hidden layer. Combining that with the fact that 1x1 (and to some extent also 3x3) convolutions with stride 2 utilize pixels in a grid pattern likely reduces the evaluation score for the ResNet34. For VGG, which uses max pooling for downsampling, gradient descent with regularization outperforms gradient descent without regularization from Layer2 downwards, and in Layer1 the GAN methods are best.

Directly copying the activations into the GAN (i.e. reconstructing without using the 1x1 stitching convolution) does not work, as shown in Figure 8 and Table 2. This is expected, since the feature extractor and GAN where trained completely unaware of each other.

Figure 4 shows 8 samples for reconstructing activation from Layer3 (with using the 1x1 stitching convolution) using StyleGAN2 and gradient descent with regularization. We can see that the reconstruction with StyleGAN2 does not introduce noise in the image, whereas gradient descent does. But the GAN method loses lighting and some color information.

Figure 5 shows reconstructions using StyleGAN2 for Layer1 to Layer4 for the AFHQ data, and using our custom GAN for the Digipath data. The top line contains the original images, and the following four lines show reconstructions from Layer1 to Layer4. Each column for the AFHQ wild data uses the same noise seed (same z vector for StyleGAN2). The reconstruction works, except for Layer4 for the AFHQ wild data, where it is reduced to a fur pattern (tiger fur pattern for a tiger image and leopard fur pattern for a leopard image). For Layer1 and Layer2, it mainly retains lower-level information, like the fur pattern of a leopard. The image also becomes slightly blurred by reconstructing it from Layer2. In contrast, in Layer3, it hallucinates a new fur pattern, and the image becomes sharper again.

Table 4: Test results for Digipath data. The best results are highlighted in bold. The evaluation time is the total time needed for all 300 validation images.

		Digipath		
layer	Method	Cosine Similarity	L1 Loss	Evaluation Time
Layer1	GAN	**0.896** ± 0.018	**0.601** ± 0.100	**14s**
	GD with reg	0.825 ± 0.043	0.786 ± 0.189	1112s
	GD no reg	0.721 ± 0.098	0.948 ± 0.067	1044s
Layer2	GAN	**0.857** ± 0.028	**0.731** ± 0.089	**10s**
	GD with reg	0.832 ± 0.037	0.771 ± 0.057	1306s
	GD no reg	0.691 ± 0.045	1.069 ± 0.053	1237s
Layer3	GAN	**0.808** ± 0.035	**0.904** ± 0.084	**11s**
	GD with reg	0.795 ± 0.023	0.931 ± 0.069	1496s
	GD no reg	0.489 ± 0.047	1.482 ± 0.156	1426s
Layer4	GAN	**0.743** ± 0.067	**0.788** ± 0.086	**10s**
	GD with reg	0.711 ± 0.055	0.842 ± 0.074	1584s
	GD no reg	0.317 ± 0.060	1.403 ± 0.141	1513s

4.3. Digipath Dataset Results

In this section, we describe our results for the digital pathology data.

4.3.1. Models

We stitch a semantic segmentation network using a ResNet34 backbone trained to segment different diseases in tissue slides in digital pathology, with a GAN generator trained unsupervised to generate tissue slides. For the test network, we use a slightly modified ResNet34 as the backbone, where we replaced the 3x3 convolutional downsamplings with bilinear downsampling followed by a 3x3 convolution with stride 1. Further, we removed the 1x1 convolutional downsampling skip connections and the max pooling layer. Those adjustments were made to reduce noise in the gradient, as visible in Figure 3.

4.3.2. Results

We train the 1x1 convolution for 240 000 samples. Table 4 shows our results evaluated on 300 images of the Digipath data. The table can be read similarly to Table 2. Again, the GAN method takes considerably less time than the gradient descent methods. We observed an average speedup of 120x with a minimum improvement of 76x and a maximum of 147x compared to gradient descent without regularization. Moreover, we observed a similar speedup compared to gradient descent with regularization, with an average improvement of 126x, a minimum of 81x, and a maximum of 154x.

Two results are noticeably different compared to the results for the AFHQ wild data. First, the GAN method performs best in the cosine similarity and L1 loss metrics over all layers, whereas for the AFHQ wild and ImageNet1K data, it only did for Layer1 testing in VGG.

Second, gradient descent without regularization performs considerably worse here compared to gradient descent with regularization, even in the earlier layers. A likely reason for those two differences is that the test network we use for the Digipath data does not use convolutional downsampling but bilinear downsampling. Therefore, it is not using input pixels in the same noise pattern as the reconstruction network. Consequently, since gradient

descent without any regularization is building in more noise, it is even stronger penalized in the metrics and performs considerably worse compared to the AFHQ wild and ImageNet1K data.

4.4. Different End Layer

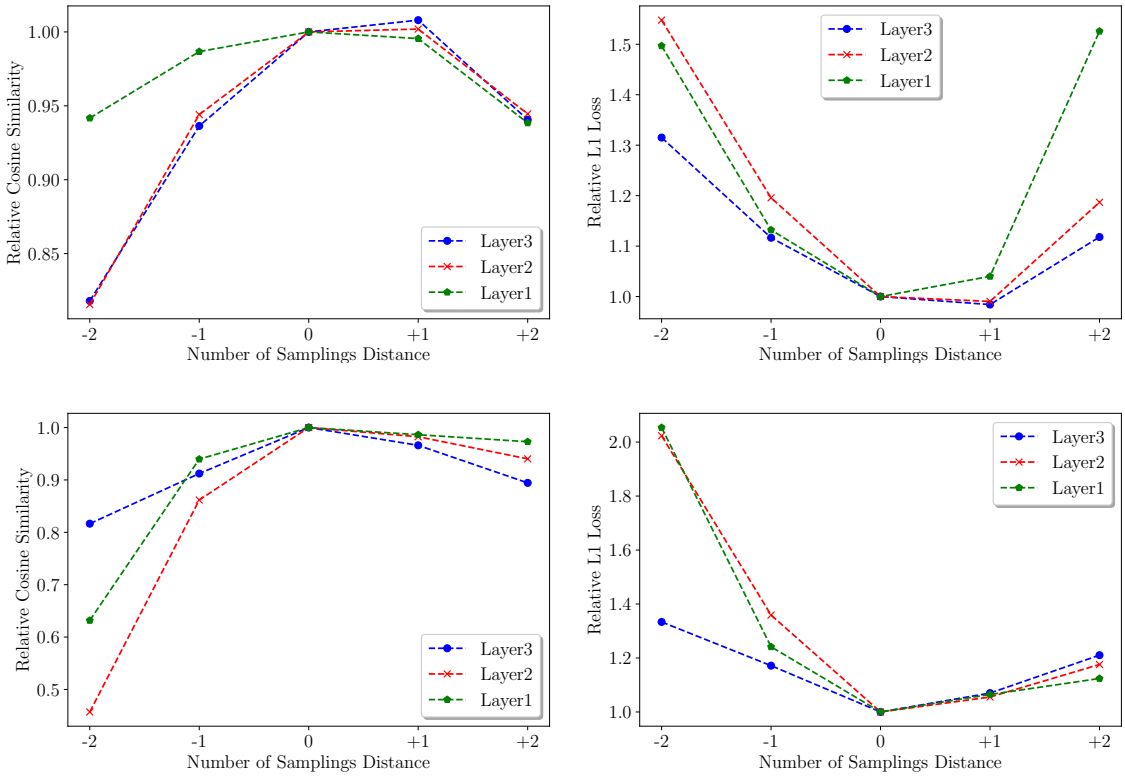

Figure 6: Result for stitching into different target layers. The top line shows results for StyleGAN2 on AFHQ wild data, and the bottom line for our custom GAN on Digipath data.

So far, for a given layer in the feature extractor, we only stitched into one layer of the GAN. Here, we evaluate the results of stitching from one layer of the feature extractor into different layers of the GAN to investigate which combination of layers is more compatible. Figure 6 shows the relative cosine similarity and L1 loss of reconstructions when stitching into different layers of StyleGAN2 (top line) and the GAN generator we use for the Digipath data (bottom line). In Table 2 and Table 4, we stitch into a layer of the GAN generator which has the same number of samplings distance from the output, as the layer in the feature extractor we start from has from the input. That combination would correspond to a "Number of Samplings Distance" of 0 in Figure 6. Here, -1 means we stitch into a layer in the GAN generator, which is one upsampling step further away from the output of the generator, and +1 means we stitch into a layer which is one upsampling step closer

to the output of the generator. As an example, Table 5 shows the different target layers in StyleGAN2 when starting from Layer3 of ResNet50.

Table 5: Target layers in StyleGAN2 when starting from Layer3.

Samplings Distance	+2	+1	0	-1	-2
Target layer	b128.conv0	b64.conv0	b32.conv0	b16.conv0	8.conv0

We plot the values relative to the value at 0 distance, and before plotting, we divide the value by the value at 0 distance. For the AFHQ wild data for Layer2 and Layer3, stitching into a layer with a +1 distance is slightly better compared to stitching into 0 distance, whereas for Layer1, it is a bit worse. For the Digipath data, it is best to stitch into a layer with 0 distance.

4.5. Additional Discussion

For BigGAN stitching into deeper layers fails (same number of samplings depth as Layer3 or deeper). This is visible in Figure 7 in the lower left where the reconstructions from Layer3 result in blue images and do not resemble the original input, and also in the metrics in Table 2 and Table 3 for Layer3 for BigGAN. BigGAN is class conditional and that conditioning is used as an additional input into the hidden layers. But for the stitching we ignore it, which likely makes the reconstruction fail for deeper layers (where the network relies on the class conditional input).

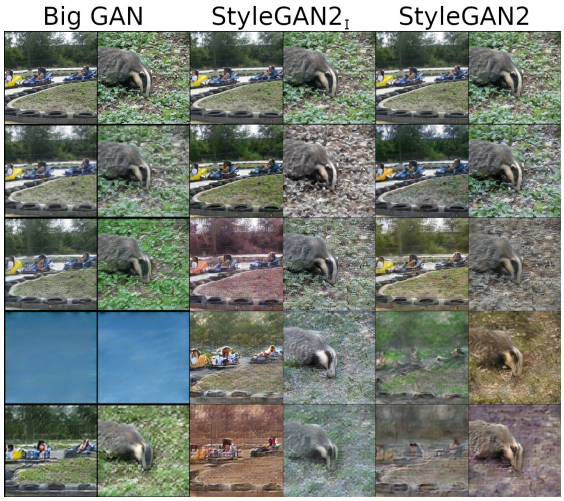

Figure 7: Samples for ImageNet1K. The rows from top to bottom are: The original dataset image, reconstruction from Layer1, Layer2, Layer3, and Layer3 with stitching into +1.

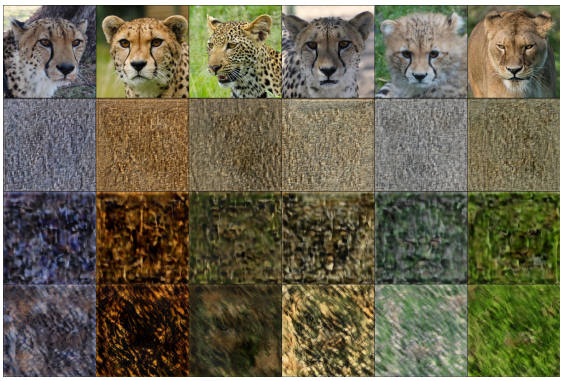

Figure 8: Reconstruction using StyleGAN2, without using the 1x1 stitching convolution (i.e. directly copying the intermediate output of the feature extractor into the GAN). The rows from top to bottom are: The original dataset image, reconstruction from Layer1, Layer2, Layer3.

Our results further show that StyleGAN2 is more general in earlier layers (same number of samplings depth as Layer1 and Layer2), but becomes more specialized in deeper layers. That is visible in Table 3 where StyleGAN2 is better than BigGAN for reconstructing activations from Layer1 and Layer2 (although this StyleGAN2 was only trained on AFHQ

wild and not on the ImageNet data). But for Layer3 it becomes worse than BigGAN, which was trained on ImageNet, but at a lower resolution of 128x128. This decline in performance suggests that StyleGAN2 specializes more in the AFHQ wild data here.

While we only use GANs for the reconstruction, it should be possible to use other generative models, e.g., variational autoencoder, in its place.

4.6. Limitations

Our method is likely to have problems when the GAN generator cannot understand the concepts of the feature extractor. We can see that in Table 3 for Layer3 the reconstructions from StyleGAN2 become worse than from BigGAN. Also class conditioning in the GAN can make problems when stitching into deeper layers of the GAN, since we ignore the class conditioning for reconstruction.

5. Conclusion and Future Work

We proposed a novel method to reconstruct activations of a feature extractor by stitching it with a GAN generator. We extensively evaluated our method and provided evidence that its accuracy is comparable to gradient descent methods while running about two orders of magnitude faster.

We also investigated how the accuracy of the reconstructions is affected when stitching into different layers of the GAN generator. Our results show that it is a good start to stitch into a layer that has the same number of samplings distance from the output of the GAN generator, as the layer we start from in the feature extractor has from the input. We observe that for most of the layers, the features learned by the feature extractor are compatible with the features learned by a GAN generator, even though the feature extractor and the GAN were trained completely unaware of each other. Only the deeper layers of a class conditional BigGAN are not compatible with a ResNet50. Further, we observe that the earlier layers of StyleGAN2 are more general, and the deeper layers become more specialized.

A weakness of the method is that the GAN generator needs to understand the concepts of the feature extractor. Otherwise, the reconstructed image is further away from the original input. This weakness might, however, be utilized for out-of-distribution (OOD) detection or uncertainty estimation.

6. Acknowledgements

R.H. is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - project number 459360854. J.L.A., D.O.B., and M.S. acknowledge the financial support by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) and the European Social Fund (ESF) within the EXIST transfer of research project “aisencia”.

References

Yamini Bansal, Preetum Nakkiran, and Boaz Barak. Revisiting model stitching to compare neural representations. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang,

and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 225–236. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/01ded4259d101feb739b06c399e9cd9c-Paper.pdf.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis, 2019.

Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. Stargan v2: Diverse image synthesis for multiple domains. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

A. Dosovitskiy and T.Brox. Inverting visual representations with convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. URL http://lmb.informatik.uni-freiburg.de/Publications/2016/DB16. arXiv:1506.02753.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 8107–8116. Computer Vision Foundation / IEEE, 2020. doi: 10.1109/CVPR42600.2020.00813. URL https://openaccess.thecvf.com/content_CVPR_2020/html/Karras_Analyzing_and_Improving_the_Image_Quality_of_StyleGAN_CVPR_2020_paper.html.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 991–999, 2015. doi: 10.1109/CVPR.2015.7298701.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5188–5196, 2014.

Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. doi: 10.23915/distill.00003. URL http://distill.pub/2016/deconv-checkerboard.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. https://distill.pub/2017/feature-visualization.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. 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 Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

Appendix A. Digipath Generator Architecture

In this section, we describe the architecture of the GAN generator we used for the experiments on the Digipath data. We use a UNet-like architecture for the generator as shown in Figure 9. The input is random Gaussian noise of a spatial size of 600x600. The content for the "Downsample", "Upsample", and "Head" layers is shown in Figure 10. Each upsample layer has two inputs. First the input from the skip connection, which is referred to as "Skip Input" in Figure 10. The second input comes from the upsample layer below it, or from the last downsample layer in the case of the first upsample layer. This second input is referred to as "Input" in Figure 10. It is upsampled via nearest neighbor scaling to the spatial size of "Skip Input", and then in the channel dimension concatenated with it.

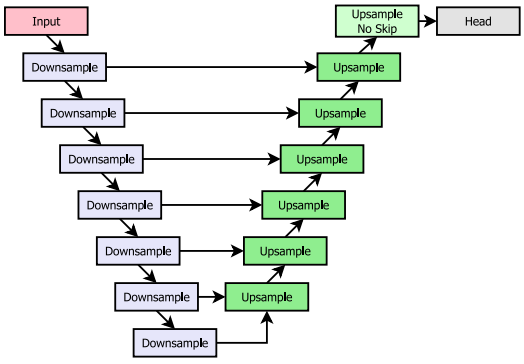

Figure 9: Architecture of the GAN generator we used for the Digipath data. The generated images have a size of 600x600x3.

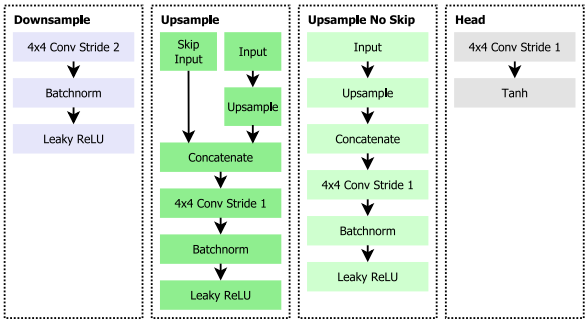

Figure 10: Layers for the GAN generator we used for the Digipath data.