Latent Space Regression in GANs for Invertible Image Generation

Miria Feng
Georgia Gabriela Sampaio
Stanford University
{miria0, gsamp}@stanford.edu

Abstract

Generative Adversarial Networks (GANs) are one of the most popular generative architectures, yet how the generator convert unstructured latent code into high fidelity output is still an open question. In this paper, we aim to better understand and manipulate the concepts GANs learns about the world by probing the latent vector space. Specifically, we aim to take a step towards achieving an invertible representation from the image manifold output of GANs to the latent code input. Our approach combines a regressor model composed of a simple encoder with a pre-trained GAN generator. The underlying assumption is that a pre-trained generator, without any additional interventions, already represents certain compositional properties in its latent code space. In order to quantify the effects of our invertible representation, we conduct experiments with three types of generators (DCGAN, GANsformer, and ProGAN) in combination with three regressor models. Given each fixed pre-trained generator, we train the respective regressor network to predict the latent code from an input image, while adding various masking modifications. We investigate the compositional properties of the pre-trained generators across the CelebA-HQ, COCO2017, and FFHQ datasets, and ultimately show that the combination of the regressor and generator forms a strong image prior. Our results demonstrate that the regressor model is capable of projecting the given masked image into a reasonable part of the latent code space, which the generator is then able to map onto the output image manifold.

1. Introduction

Generative Adversarial Networks (GANs) are deep learning models that estimate how data points are generated in a probabilistic framework. Through adversarial training between the generator and discriminator models, we are able to generate incredibly high fidelity images from latent code space input. As a result, GANs have been successfully implemented on a variety of generative tasks. For example, the high resolution of ProGAN [6] is capable of generating realistic images of 1024 x 1024 pixels, while StyleGAN [7] and CycleGAN [22] allow us to control the style of the output images to a certain degree. The notable advantages of GANs are low sampling cost and high fidelity performance on image generation, but the major disadvantage arises from the fact that we cannot model the probability distribution, and therefore cannot infer the latent input z from an image sample.

Previous works [13], [8] aim to achieve a more disentangled representation of the latent code space, in order to better understand the mapping from the noise distribution to the output image manifold. More recently, the research of [2] proved the effectiveness of working from a more invertible representation for a multitude of generative tasks. Better understanding of this non-trivial invertible mapping is a key component for future work in GANs, and will ultimately lead to more controllable image generation with a direct way of manipulating GANs even in an unsupervised environment. Hence interpreting the inverse mapping of the GANs' generator has great potential value.

In this project, we aim to take a step towards achieving an invertible representation of the generative model from the latent code space to image manifold. In order to approach this challenging task, we examine three unique frozen pre-trained generators across three different datasets (CelebA-HQ, COCO2017, and FFHQ) and probe their respective latent code space with individually trained regressor models. Our goal is to invert the latent code from each input image. In order to quantitatively analyze the effects of our proposed regressor – generator models, we conduct experiments with inpainting by masking out a certain number of pixels during input and allowing the models to complete the missing sections. Our metrics include the commonly used Inception Score (IS) [15], which evaluates the diversity and fidelity of the generated image outputs, as well as the Frechet Inception Distance (FID) [4], which measures the quality of high resolution generated outputs. We validate the underlying assumption that there already exists deep compositional properties within the latent code space

of each pre-trained generator, which gives us a clearer understanding of how GANs are able to generate outputs of such high fidelity. Our experiments further demonstrate that the combination of the regressor – generator model is able to form stronger image priors for generation. The following project is outlined as follows: in section 2 we introduce related work, section 3 outlines the datasets utilized, section 4 discusses our method in inverting latent code representation and metrics, section 5 introduces model architecture and experiments conducted. Finally, section 6 summarizes our results, conclusions, limitations, and promising directions for future work.

2. Related Work

A wide variety of methods have been applied to the challenging task of understanding the latent space in GANs. In this section, we explore the methods that are most closely related to our problem statement.

GANs inversion aims to invert a target image into the latent space of a pre-trained GAN model such that the image can be faithfully reconstructed from the inverted code by the generator. Despite the success of GANs in generating highly realistic outputs for applications, there still lacks theoretical understanding of GAN inversion [19]. Being able to invert a given image back to its latent code mapping will provide meaningful insight to the precise generative properties of GANs, with additional practical applications in areas such as image restoration and image manipulation. Notably, [16] have recently successfully interpreted the latent space of GANs for the specific task of semantic face editing with a novel framework [17], while many other authors have adapted various approaches to solving this challenging problem. For example, the authors of [20] examine the application of using learned semantics in the latent space of GANs for image editing by inverting a real image back to its latent code. They propose an in-domain inversion method which aims to reconstruct the input image but also ensure the latent code can be semantically meaningful for editing. This technique involves first learning a novel domain-guided encoder to project the target image to the latent code space, followed by domain-regularized optimization with the encoder as a regularizer to fine-tune the latent code. The purpose of this two step technique is to create an inversion method which is capable of reconstructing the target image at the pixel level, while aligning certain semantic knowledge encoded at the latent code level. However, the additional requirement of the regularizer - necessary to ultimately ensure that the inverted latent code does not deviate too far in the latent code space - is complicated and non-trivial to execute.

Other approaches to GAN inversion involve directly optimizing the latent code for individual images [10]. Due to the highly non-convex nature of this optimization problem, it is typically computationally complicated and without optimality guarantees. The research of [12] present new theoretical results on the recovery of latent code from network output, and examines the effects of GAN inversion on inpainting applications as well as to mitigate the problem of mode collapse during GAN training. Unfortunately, this approach of rigorous theoretical proof requires strict assumptions and conditions in order to find a bijective mapping between the low dimensional latent code space and the high dimensional image space. In practical applications, a less strict set of conditions is desired.

Alternatively, the authors of [3] have approached learning invertible disentangled interpretations of variational auto-encoders. Motivated by this approach, other researchers have attempted to train an extra encoder beyond the GAN [11], [21] which functions as an inverse generator model to find some invertible representation in the latent code space. The advantage of this method is that it overcomes the difficulty in training, and avoids the challenges of seeking an explicit bijective mapping function. This pioneering work explored the applications of this method on image searching, although given the high fidelity capabilities of GAN in recent works, deeper applications should be explored. We still have numerous open questions about inverted code, such as how well does the inverted code represent the target image? Can the inverted code support image manipulation by using the knowledge learned by GANs? Does the inverted code even lie in the same space as the original latent space of GAN?

Our work is motivated by the research of [1] and uses encoders as regressors to explore the disentanglement and invertibility of GANs' latent space. In our work, we approximate image inversion as latent space regression, using the emergent priors of a pre-trained GAN. Such regressor has lower reconstruction accuracy than techniques that are based on latent space optimization. However, we can investigate the learned priors in a more efficient way, which makes probing the latent space less computationally expensive.

Model	Dataset	Num Images	Image Resolution
DCGAN	COCO2017	14,500	256 x 256
GANsformer	FFHQ	56,000	256 x 256
ProGAN	CelebA-HQ	30,000	1024 x 1024

Table 1: Dataset specifics and respective Models

3. Dataset

We utilize three datasets: the CelebA-HQ [6], the COCO2017 [9] dataset, and the FFHQ [7] dataset of human faces. Table 1 shows the corresponding models and dimensions of each dataset. In each case, we split the dataset into

train/validation/test sets of 60-20-20, respectively. In order to be consistent with existing work, we will further partition the COCO dataset into a subset of approximately 8500 training samples, 3000 testing samples, and 3000 validation samples.

Each dataset is selected to be most suitable for its respective model. We take into consideration future experimentation with caption – image conditional generation and select the COCO dataset to be integrated with the DCGAN [14] model. Since the GANsformer model is largely motivated by the original StyleGAN architecture which was trained on the FFHQ dataset, we elect to integrate this dataset with our GANSformer generator for consistency. Finally, we select to implement the ProGAN model with the CelebA-HQ dataset in order to demonstrate the high resolution generative capabilities of this model, as well as in maintaining consistency with the original documentation of ProGAN during training. During our training procedure, we further process the dataset by adding a masking element to the images in order to encourage more flexible recovery of latent code. The design of this additional masking element is intended to allow the generator to produce images that are both likely under the prior, yet still consistent with the observed areas of the image. Further details regarding this methodology is outlined in section 4, and the application of $Mask\ M$ to the images of the dataset during training is illustrated in Figure 3.

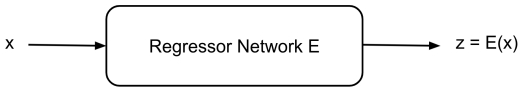

Figure 1: The input to the regressor network E, is the original target image x. The output is the generated z vector of latent code.

Figure 2: The input to the GAN G is a randomly sampled z vector, and the output is the target image $x = (G(z))$.

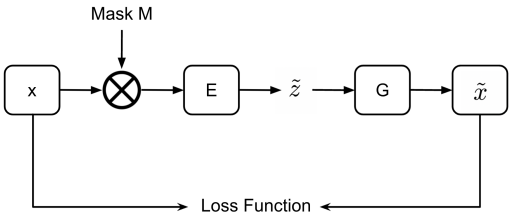

Figure 3: Complete architecture of the application.

4. Methods and Evaluation

4.1. Approach to Inverting Latent Code

GANs are capable of generating extremely high fidelity images by mapping from a predetermined input distribution in the latent code space to the output image manifold. However, this mapping is not easily invertible. Since invertibility implies we aim to find the latent z vector which best describes the generated target image x, our objective can therefore be summarized as follows:

$$z^* = \arg\min_z \left(distance(G(z), x)\right) \tag{1}$$

Where z^* is the resulting latent code z vector which minimizes the distance between the generated output image $G(z)$ and the target image x. Typically, this distance is measured using some metric such as pixel distance, and then solved iteratively using an optimizer. However, notable disadvantages of this approach are the slow convergence to solutions of poor quality, the necessary computation involved in assessing each image x independently, and the risk of converging to a local minima.

Our approach aims to overcome these disadvantages and is defined as follows: given a fixed pre-trained GAN, we train a regressor model to predict the latent code from the input image, while adding various masking modifications to encourage learning from missing pixels. By demonstrating the ability of the model to perform image reconstruction in various cases of incomplete images, we will show that the combination of the regressor and generator forms a strong image prior. The underlying assumption is that a pre-trained generator already represents certain compositional properties in its latent code, and that these properties can be manipulated using the regressor which predicts the latent code of a given image.

Two model components are involved in our architecture: Figure 1 shows the explicit goal of the regressor model and Figure 2 outlines the role of the pre-trained GAN generator. The complete architecture used during our experimentation to qualitatively evaluate the regressor-GAN model is shown in Figure 3. In the diagram, x represents the target image, $Mask\ M$ obstructs certain pixels, E is the encoding regressor model, $\tilde{z}$ is the generated latent code vector, G is the generator, and $\tilde{x}$ is the generated output.

Throughout this project, we experiment with three frozen pre-trained generators (DCGAN, GANSformer, ProGAN). For each generator, we implement a suitable regressor model to probe the latent space in order to solve equation (1) and achieve latent code recovery in GANs. We experiment with the architecture and design of the encoder E for each frozen generator, and intuitively add a masking element to the images in order to encourage more flexible re-

covery of latent code. The design of this additional masking element ultimately allows the generator to produce images that are both likely under the prior and yet still consistent with the observed areas of the image.

During training, we consider the input of this masked image as a technique similar in intent to dropout. $Mask\ M$ is the result of some random uniform noise upsampled to full resolution (with bilinear interpolation). The input image x is then subjected to be masked out at all pixels where this upsampled noise is smaller than a sampled threshold limit, thus creating arbitrary mask boundaries. At test time, the precise shape of the mask is unimportant, since the mask simply indicates to the generator where to inpaint the missing pixels. Our experiments with inpainting prove that despite modifying or masking out significant sections of the input, the generator is still able to produce output images with good overall consistency. These results suggest that the regressor – generator model pair better enforces global coherence, and that meaningful compositional properties are already inherent in the latent code space of the pre-trained generator.

4.2. Metrics

Overall, our mathematical objective is then to optimize: x vs $G(E(x))$, which represents the target image versus the recovered image, and z vs $E(x)$ which represents the original z vector versus the recovered z vector. Our loss function is thus defined as:

$$L = \mathbb{E}(\|x - G(E(x))\|^2 + L_p(x, G(E(x))) + L_z(z, E(x))) \tag{2}$$

The design of the loss function is motivated by two main considerations: the perceptual loss term L_p guides more high level reconstruction and gives a sense of semantic similarity, while the mean squared loss guides more low level pixel differences during reconstruction. The additional term of L_z latent recovery loss serves to measure the relationship between the original latent code z and the recovered latent code.

In order to quantitatively evaluate the quality of our generated image outputs, we compare the respective Frechet Inception Distance (FID) and Inception Scores (IS) scores across each of our three models. Specifically, the FID metric calculates the distance between feature vectors of target and generated images. This metric summarizes how similar these two groups of images are by using the Inception v3 model [18], and provides a metric to measure the performance of each generated outcome. Hence lower FID scores indicate stronger similarity between two groups of images, while a perfect FID score of 0.0 implies these images are absolutely identical.

In contrast, the Inception Score (IS) is an early and widely adopted metric of objective evaluation for generated images in an effort to remove subjective human evaluation. The IS involves using the pre-trained deep neural network Inception v3 model for image classification to classify the generated images. The probability of the image belonging to each class is predicted, which is then summarized into a single IS in an attempt to capture image diversity and image quality/fidelity. Unlike the FID, a high IS is better, and the lowest possible score signifies poor performance. Thus if the generated images involve both diversity and fidelity, the IS will be high, else if either criteria is not satisfied, the IS will be low.

5. Models and Experiments

In this section, we discuss the architectures of our three models and the set of experiments we conducted to evaluate the validity of our method. Additional results and latent space probing outputs as well as project code can be viewed at `https://github.com/gsamp/disentanglement`. In the following subsections, we will distinguish between various network structures, training configurations, and quantitative effectiveness of our regressor – generator approach.

5.1. Baseline DCGAN Model

In order to verify the validity of the regressor – generator method, we first implement a basic DCGAN. Our configuration is motivated by the research of [14]. We select the DCGAN as our baseline model since the implementation of deep convolution networks in the structure (as opposed to traditional fully connected layers) intuitively suggests better adherence to recognizing spatial correlations in images. This means DCGAN naturally fits image data very well, being a successful combination of convolutional neural networks and GANs. We design the Generator and Discriminator as follows:

(1) Generator: 3 hidden layers followed by 1 output layer; each hidden layer consists of conv – batch norm – relu activation; the final output layer has conv – tanh activation; each conv layer uses a kernel size of 3, stride of 1, then stride of 2, then stride of 2, and hidden dimension of 64.

(2) Discriminator: 3 layer neural network; hidden dimension of 16; except for the last layer, we utilize the conv – batch norm – Leaky ReLU ($\alpha = 0.2$) structure; at the last layer we only apply conv.

We train the DCGAN with Binary Cross Entropy loss and randomly sampled z dim = 64, batch size = 128, and learning rate of 0.0002. We use the Adam optimizer with $\beta_1 = 0.5$ and $\beta_2 = 0.999$.

We train on the images from the COCO2017 dataset. The replacement of the max-pooling layers with convolutions layer means that the DCGAN is easier to train than the traditional GAN. Finally, we note that the batch normalization layers in both the generator and discriminator are able to prevent covariant shift, and therefore introduces more stable training. We select the negative slope of the Leaky ReLU activation to be $\alpha = 0.2$ (after experimentation) in the discriminator to prevent the "dying relu" problem and to further speed up training. These techniques cumulate in more successful learning and a more stable model.

After training the DCGAN, we freeze the generator and experiment with the regressor model to probe the latent code space.

(3) Encoder: We design the regressor as a 5 layer convolutional neural network with hidden dimensions of $(32, 64, 128, 256, 512)$. Leaky ReLU activation is used throughout the regressor, with a negative slope $\alpha = 0.1$ for more sensitive learning. Each of the layers follows the conv - batchnorm – leaky relu pattern, and we set all kernels to 3, strides to 2, and padding to 1. In training the regressor model, we select a learning rate of 0.0001.

5.2. GANsformer: Visual Generative Transformer

The Gansformer unifies GANs and Transformers for the task of image generation. The main innovation introduced by the GANsformer is the change in the architectural design of the discriminator and generator that comprise the architecture of the GAN. Specifically, rather than using multiple layers of convolution, the generator and discriminator in the GANsformer are built using a novel architecture called Bipartite Transformer [5].

A standard transformer model is a set of multi-head self-attention and feed-forward layers, such that a Transformer is considered a stack of several such layers. The Bipartite Transformer generalizes the self-attention operator, featuring instead a bipartite graph between the latent space and image features. Two forms of attention are computed over the bipartite graph: Simplex attention and Duplex attention. The simplex attention distributes information in a single direction over the Bipartite Transformer, while the Duplex attention supports bidirectional interaction between the latent code and the image features. [5] As a consequence, the model supports bidirectional interaction between the latent space and the image features, which allows for the refinement and interpretation of each in light of the other. These design choices yield a model architecture with improved latent space disentanglement, a suiting attribute for our evaluation of latent space disentanglement. [5]

(1) Encoder: The encoder consists of a 4 layer convolutional neural network with hidden dimensions $(32, 32, 32, 64)$. The last layer is a fully-connected layer

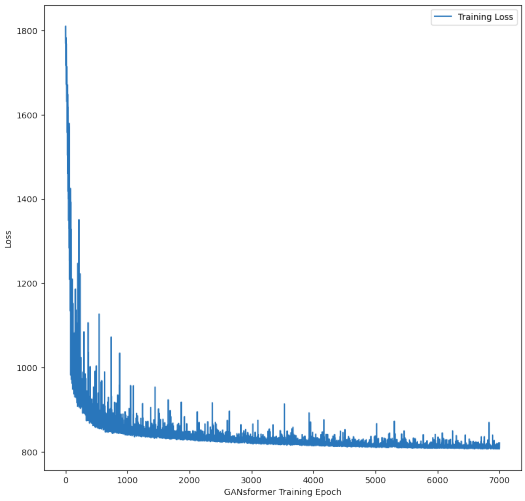

Figure 4: GANsformer training loss over epoch.

used to flatten the representation of the output latent space z. Each convolutional layer is followed by a ReLU nonlinearity. All kernels are set to 2; the first two strides are set 2, and the last two strides are set to 4. We use a batch size of 8 and learning rate of 0.001 in accordance with [5]. The model was trained using L2 loss with the Adam optimizer in its standard setting: $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e^{-8}$.

(2) Generator: The generator architecture is motivated by the StyleGAN model [7], which features a feed-forward mapping network that outputs an intermediate vector w that interacts directly with each convolution through the synthesis network, which globally controls the feature maps statistics of every layer [5]. In the GANsformer, however, instead of controlling the statistics of all features globally, the attention layers are used to perform adaptive and local modulation. The latent vector z is split into k components, $z = [z_1, ..., z_k]$, which is passed through a shared mapping network, resulting in a set of intermediate latent variables. During synthesis, after each convolutional later in the generator, the feature map and latent code mediate their interaction through the attention layer (either simplex or duplex). [5].

The first layer of the generator is an embedding layer for the latent vector. The generator expects a latent vector of size $(17, 32)$, where the $17th$ row consists of a one dimensional $(1, 32)$ random sample from a Gaussian, and the remaining $(16, 32)$ vector comprises the latent vector already reparameterized. The output shape of this layer is thus a $(16, 32)$ layer which is fed into the mapping portion of the generator.

The mapping portion of the generator consists of 4 attention layers: the first three attention layers are followed by two dense layers, and the fourth attention layer is followed by three dense layers. The last layer of the mapping portion

of the generator is a global layer. The synthesis portion of the generator consists of 12 attention layers. We use an iteration of this generator pre-trained on the FFHQ dataset of human faces [7] for all our experiments.

5.3. ProGAN: High Resolution Generation

Following the successful implementation of the regressor – generator model on our baseline DCGAN and on the GANsformer, we then experiment with higher resolution generation by probing the latent code space of the ProGAN generator on the challenging CelebA-HQ dataset. We select the CelebA-HQ dataset for the ProGAN due to its higher resolution images (1024 x 1024), which take advantage of the capabilities of ProGAN. What sets ProGAN apart from traditional GAN is the gradual growth of the generator during training. Instead of training all layers of the generator and discriminator simultaneously, the layers of ProGAN are trained one at a time, thus allowing the generator to handle extremely high fidelity and high resolution versions of images (unlike our previous models). We implement the regressor – generator model with a pre-trained ProGAN generator whose architecture is outlined in Figure 6. The respective regressor model is motivated by the ResNet-18 architecture (refer to Figure 5 for detailed build) and defined as follows:

- modify output dimensions to match number of our input latent dimensions
- Adam optimizer with learning rate of 0.0001
- specifically requires 1080TI GPU acceleration to train for 256 x 256 - 1024 x 1024 resolution
- we found a batch size of 16 to be optimal trade-off for the 256 x 256 resolution and train for $500k$ batches
- for 1024 x 1024 generation we alter to batch size 4 and $400k$ batches

Some examples of generated results are shown in Figure 7.

It is observed that although the images are able to generalize well to masking of small sections, the model is unlikely to recover aspects of the target image, such as "glasses" or other unique attributes, when entire features are masked. This is understandable, since the objective based on pixel similarity introduced during training seems to encourage reconstruction methods based primarily on similar neighboring pixel colors and patterns. We hypothesize that the model becomes unable to generate high level semantic meaningful features as a consequence of this, despite the additional term of perceptual loss during training.

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	7×7, 64, stride 2
conv2_x	$56 \times 56 \times 64$	3×3 max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	7×7 average pool
fully connected	1000	512×1000 fully connections
softmax	1000	

Figure 5: Encoder model for Progan, based on ResNet-18

Generator	Act.	Output shape	Params
Latent vector	–	512 × 1 × 1	–
Conv 4 × 4	LReLU	512 × 4 × 4	4.2M
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Upsample	–	512 × 8 × 8	–
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Upsample	–	512 × 16 × 16	–
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Upsample	–	512 × 32 × 32	–
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Upsample	–	512 × 64 × 64	–
Conv 3 × 3	LReLU	256 × 64 × 64	1.2M
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Upsample	–	256 × 128 × 128	–
Conv 3 × 3	LReLU	128 × 128 × 128	295k
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Upsample	–	128 × 256 × 256	–
Conv 3 × 3	LReLU	64 × 256 × 256	74k
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Upsample	–	64 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	18k
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Upsample	–	32 × 1024 × 1024	–
Conv 3 × 3	LReLU	16 × 1024 × 1024	4.6k
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 1 × 1	linear	3 × 1024 × 1024	51
Total trainable parameters			**23.1M**

Figure 6: Generator model for Progan with frozen weights

Model	Dataset	Encoder	FID	IS
DCGAN	COCO2017	5 Layer CNN	81.72	1.08 ± 0.00
GANsformer 5k	FFHQ	4 Layer CNN	13.99	3.08 ± 0.54
GANsformer 50k	FFHQ	4 Layer CNN	7.37	1.00 ± 0.00
ProGAN	CelebA-HQ	ResNet-18	10.54	10.52 ± 0.00

Table 2: Fréchet Inception Distance (FID) and Inception Scores (IS) for each model.

5.4. Numerical Results

With the exception of the ProGAN model, all other encoder models were fine tuned with various experimentation. Each time we hold one variable constant while adjusting the others to an optimal point on each encoder model, then alternate between holding different parameters fixed. Some of our hyper-parameter search heuristics include limiting the size of kernels in convolutional layers to no larger than 3 in order to reduce computational costs, and limiting the size of the stride with padding in order to achieve the most meaningful representations. Table 2 summarizes the numerical experimental results of our three pre-trained frozen generators across our three respective regressor models, and we discuss these outcomes in greater detail below:

(1) DCGAN: Performed the weakest in terms of both FID and IS metrics. These results suggest that the encoder – generator model was unable to achieve realistic generated images and also incapable of generating great diversity outputs. In understanding the failure of the baseline model, we first examine the training data, and acknowledge that the low number of training images (of low resolution) is likely a critical factor. Additionally, since we did not fine tune the hyperparameters of the DCGAN as meticulously during training the initial generator, it is understandable that the latent code space of this generator will inherently represent significantly less compositional properties. Although the CNN based architecture of the DCGAN makes it an intuitive choice when learning from image data, the shallow number of layers in the generator and discriminator will also limit the number of features learned from the data during training.

(2) GANsformer: Ultimately generated the most realistic images with an FID score of 7.37 when trained on $50,000$ images of low resolution. We note that when trained on a smaller subset of data (5000 images), the GANsformer reports a weaker FID score but a superior IS metric; thus, when trained on a smaller dataset, it seems that this model is able to generate higher diversity but with less realism. This is reasonable, since we expect more data during training to result in the generator learning to output more realistic features, although the random shuffle of the dataset does not guarantee necessary diversity. However, as seen in Figure 4, the GANsformer demonstrated the most stable results during training.

(3) ProGAN: Achieved the most balanced metrics, with the second best FID score and highest IS metric, while generating the highest resolution images among our models. Figure 7 demonstrates the successful inpainting results of the ProGAN generator with the ResNet-18 encoder model. Unlike previous models (which all use a 5 layer CNN encoder), we assume that the complexity of the ProGAN generator requires a more complex regressor model. The combined depth of this encoder – generator model ultimately allows it to learn a higher number of more realistic features. Finally, the ProGAN generator, which is capable of generating high resolution images, likely already represents a large number of image compositional properties in its latent code space even without any additional interventions. Therefore, the regressor model which aims to predict the latent code of each image is expected to return much more meaningful results.

6. Conclusion and Future Work

We investigate the invertible properties of latent code space for image generation via three separate regressor models and three pre-trained frozen generators. In order to take advantage of the strengths of each of the specific generators, we select a matching most suitable dataset and train a custom regressor model for each experiment. Specifically, we consider the COCO2017 dataset with the DCGAN generator, the FFHQ dataset with the GANSformer, and the CelebA-HQ dataset with the ProGAN. This regressor method allows us to probe the latent space, and better understand what is inherently encoded already in the z space of GANs without any additional interventions. Through various experimentation with inpainting, we find that our regressor - generator combination is able to create a stronger image prior thus producing more convincing output images. This is quantitively demonstrated in Table 2 and Figure 7. We find that the GANSformer trained on $50k$ images is capable of generating the most realistic images of resolution 256 x 256 with an FID score of 7.37, while the ProGAN is able to generate reasonably realistic images (FID score of 10.54) of much higher resolution ($1024x1024$). However all models achieved relatively low IS metrics, suggesting that generative diversity leaves room for improvement. The effectiveness of our project leads to promising directions for future work in latent space regression, with solid potential to improve our preliminary results. Future research will focus on:

- generalizing to multiple modalities, including captions and conditional generation with conditional and controlled GANs
- multimodal editing and more complicated image editing applications
- achieving deeper disentangled representations
- modifying the loss function to incorporate more semantic loss features

7. Contributions

Miria Feng implemented the DCGAN and ProGAN model architectures. Georgia Sampaio implemented the GANsformer architecture.

References

[1] Lucy Chai, Jonas Wulff, and Phillip Isola. Using latent space regression to analyze and leverage compositionality in gans, 2021. 2

[2] Chanho Eom and Bumsub Ham. Learning disentangled representation for robust person re-identification, 2019. 1

[3] Patrick Esser, Robin Rombach, and Björn Ommer. A disentangling invertible interpretation network for explaining latent representations, 2020. 2

[4] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018. 1

[5] Drew A Hudson and C. Lawrence Zitnick. Generative adversarial transformers. *arXiv preprint:2103.01209*, 2021. 5

[6] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2018. 1, 2

[7] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019. 1, 2, 5, 6

[8] Hadi Kazemi, Seyed Mehdi Iranmanesh, and Nasser M. Nasrabadi. Style and content disentanglement in generative adversarial networks, 2018. 1

[9] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. 2

[10] Zachary C. Lipton and Subarna Tripathi. Precise recovery of latent vectors from generative adversarial networks, 2017. 2

[11] Junyu Luo, Yong Xu, Chenwei Tang, and Jiancheng Lv. Learning inverse mapping by autoencoder based generative adversarial nets, 2017. 2

[12] Fangchang Ma, Ulas Ayaz, and Sertac Karaman. Invertibility of convolutional generative networks from partial measurements. In *Advances in Neural Information Processing Systems*, pages 9651–9660, 2018. 2

[13] David Pfau, Irina Higgins, Aleksandar Botev, and Sébastien Racanière. Disentangling by subspace diffusion, 2020. 1

[14] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016. 3, 4

[15] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016. 1

[16] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing, 2020. 2

[17] Yujun Shen, Ceyuan Yang, Xiaoou Tang, and Bolei Zhou. Interfacegan: Interpreting the disentangled face representation learned by gans, 2020. 2

[18] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015. 4

[19] Weihao Xia, Yulun Zhang, Yujiu Yang, Jing-Hao Xue, Bolei Zhou, and Ming-Hsuan Yang. Gan inversion: A survey, 2021. 2

[20] Jiapeng Zhu, Yujun Shen, Deli Zhao, and Bolei Zhou. In-domain gan inversion for real image editing, 2020. 2

[21] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. Generative visual manipulation on the natural image manifold, 2018. 2

[22] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017. 1

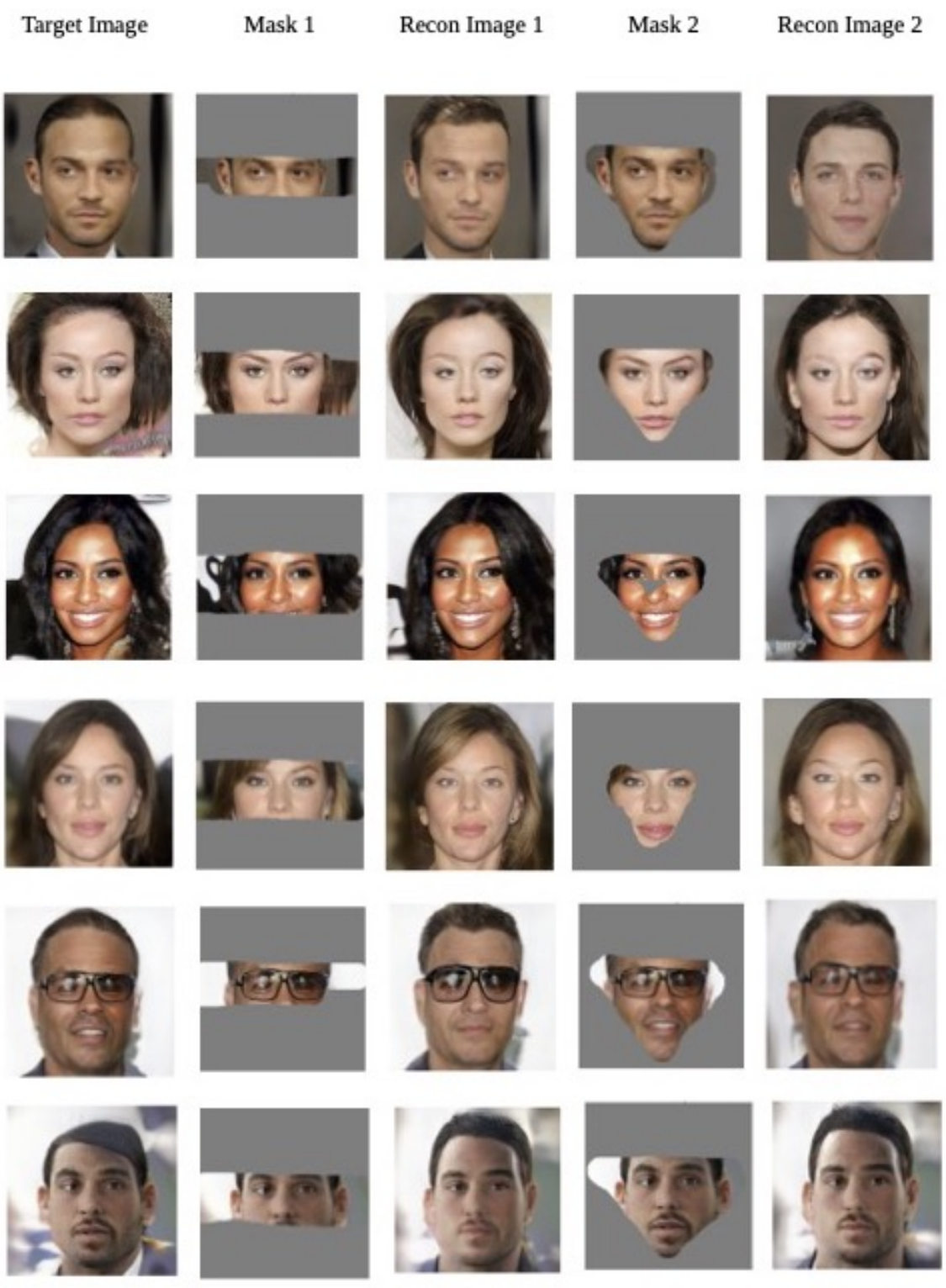

Figure 7: Generated results of ProGAN model on the CelebA-HQ dataset with two varying masks.