Dynamic Latent Scale GAN for GAN inversion

Jeongik Cho

jeongik.jo.01@gmail.com

Abstract

Generator of generative adversarial networks (GAN) maps latent random variable into data random variable. GAN inversion is mapping data random variable to latent random variable by inverting the generator of GAN.

When training the encoder for generator inversion, using the mean squared error causes the encoder to not converge because there is information loss on the latent random variable in the generator. In other words, it is impossible to train an encoder that inverts the generator as it, because the generator may ignore some information of the latent random variable.

This paper introduces a dynamic latent scale GAN, a method for training a generator that does not lose information from the latent random variable, and an encoder that inverts the generator. When the latent random variable is an i.i.d. (independent and identically distributed) random variable, dynamic latent scale GAN dynamically scales each element of the latent random variable during GAN training to adjust the entropy of the latent random variable. As training progresses, the entropy of the latent random variable decreases until there is no information loss on the latent random variable in the generator. If there is no information loss on the latent random variable in the generator, the encoder can converge to invert the generator.

The scale of the latent random variable depends on the amount of information that the encoder can recover. It can be calculated from the element-wise variance of the predicted latent random variable from the encoder.

Since the scale of latent random variable changes dynamically in dynamic latent scale GAN, the encoder should be trained with a generator during GAN training. The encoder can be integrated with the discriminator, and the loss for the encoder is added to the generator loss for fast training. Also, dynamic latent scale GAN can be used for continuous attribute editing with InterFaceGAN.

1. Introduction and previous works

The generator of a generative adversarial network (GAN) [2] is trained to map latent random variable to data random variable. Generally, simple distribution such as normal or uniform i.i.d. random variable is used as a latent random variable.

Inverting generator is finding an inverse mapping of a generator of GAN. It can be used for feature learning or can various useful applications such as data manipulation.

There are learning-based methods, optimization-based methods, or hybrid methods for GAN inversion. Many methods and useful applications for GAN inversion are introduced in the GAN inversion survey paper [1].

Among the learning-based methods, [11], [12], [13] used conditional GAN [14] to train an encoder that inverts the generator. However, those methods are difficult to train model, and the performance of the model is not good.

[15], [16] used mean squared error (MSE) loss to train encoder to recover latent random variable. [18], [7] added reconstruction loss to MSE loss for the better performance. [19], [20], [6] used model (StyleGAN [5], [9]) specific methods.

In this paper, I introduce a dynamic latent scale GAN (DLSGAN), a learning-based method for training an encoder that inverts the generator of GAN. Previous works used MSE loss to train the encoder when the latent random variable is an i.i.d. random variable. However, simply using MSE loss causes convergence problem because information loss on latent random variable occurs in the generator. If there is information loss on the latent random variable in the generator, the encoder cannot be converged to invert the generator. Dynamic latent scale GAN dynamically adjusts the scale of the latent random variable so that there is no information loss in the generator. It means the encoder can be converged to invert the generator.

The scale of the latent random variable depends on the amount of information that the encoder can recover. It can be calculated from the element-wise variance of the predicted latent random variable from the encoder. DLSGAN traces the predicted latent random variable from the encoder during training to approximate the element-wise variance of the predicted latent random variable.

Since the scale of a latent random variable dynamically changes dynamically in dynamic latent scale GAN, the encoder should be trained with a generator during GAN training. The encoder can be integrated with the discriminator, and the loss for the encoder is added to the generator loss for fast training.

I also introduce a method to edit attributes of data continuously with DLSGAN.

2. Problem of using MSE

The generator G maps the latent random variable Z to the data random variable X (i.e., $X = G(Z)$). However, there is no guarantee that the generator G uses all the information of the latent random variable Z. For example, when the latent random variable Z has too many dimensions, the generator G can be trained to ignore some elements of the latent random variable Z. Or, the generator G can be trained so that some elements of the latent random variable Z have relatively more information than others. In other words, different latent codes z_1 and z_2 sampled from latent random variable Z can be mapped to the same or nearly similar generated data points $G(z_1)$ and $G(z_2)$. It means that information loss on the latent random variable Z occurs in the generator G, and the encoder E cannot perfectly recover the latent random variable Z from the generated data random variable $G(Z)$ and cannot converge.

3. Dynamic latent scale GAN

To prevent information loss on the latent random variable Z that occurs in the generator G, I introduce a DLSGAN that dynamically adjusts the scale of each element of the latent random variable Z so that no information loss occurs in generator G.

Assume the latent random variable Z is d_z-dimensional i.i.d. random variable and encoder E is trained to predict latent random variable Z from generated data random variable $G(Z)$ with MSE loss. Each element of the predicted latent random variable $Z' = E(G(Z))$ follows a distribution with mean μ, but variances less or equal to σ^2 because some elements of latent random variable Z cannot be fully recovered.

At this time, the variance of each element of predicted latent random variable Z' represents information on the latent random variable Z that can be recovered from the generated data random variable $G(Z)$. If the variance of n-th predicted latent random variable Z'_n is zero, it means that encoder E cannot recover any information of n-th latent random variable Z_n from generated data random variable $G(Z)$. On the other hand, if the variance of n-th predicted latent random variable Z'_n is σ^2, it means encoder E can recover all information of n-th latent random variable Z_n from generated data random variable $G(Z)$. Therefore, if the element-wise variance of the predicted latent random variable Z' and the element-wise variance of the latent random variable Z are the same, it means that there is no information loss in the generator G, and the encoder can converge to predict latent random variable Z from generated data random variable $G(Z)$.

DLSGAN dynamically adjusts the scale of each element of latent random variable Z according to the variance of each element of predicted latent random variable Z' so that the variance of the latent random variable Z and predicted latent random variable Z' are equal. Since the dynamic latent scale GAN requires both the encoder E and the generator G to be trained together, it is efficient to integrate the encoder E into the discriminator D. For the same reason, generator G and encoder E can be trained cooperatively. That is, encoder loss L_{enc} (objective function) can be added to generator loss L_g (objective function).

The following algorithm shows the process of obtaining the loss (objective function) for training DLSGAN.

```
$function\ GetLoss(D, G, Z, x, v)$:
1    $z \leftarrow sample(Z)$
2    $s \leftarrow \frac{\sqrt{d_z v}}{\|\sqrt{v}\|_2}$

3    $a_g, z' \leftarrow D(G(z \circ s))$
4    $L_{enc} \leftarrow \frac{1}{d_z}\|(z - z') \circ s\|_2^2$
5    $a_r, \_ \leftarrow D(x)$

6    $L_d \leftarrow f_d(a_r, a_g) + \lambda_{enc} L_{enc}$
7    $L_g \leftarrow f_g(a_g) + \lambda_{enc} L_{enc}$

8    $v \leftarrow update(v, z'^2)$

9    $return\ L_d, L_g, v$
```

Algorithm 1. Algorithm to obtain DLSGAN loss

In the above algorithm, D, G, and x represent discriminator, generator, and real data point, respectively. Real data point x is a data point sampled from real data random variable X. Since encoder E is integrated with discriminator D, discriminator D outputs two values: 1-dimensional adversarial value and d_z-dimensional predicted latent code. v represents the element-wise variance of the predicted latent random variable Z'. It is ideal to approximate v for every training step, but for efficiency, v is approximated through predicted latent codes from the past k train steps.

In line 1, for convenience, it is assumed that latent random variable Z has a mean of 0 and a standard deviation of 1. $sample$ is a function that samples a single value from a random variable. z is latent code, which is sampled from latent random variable Z.

In line 2, s is the latent scale vector. $\sqrt{vec}$ represents the element-wise square root of the example vector vec. $\|vec\|_2$ represents L2 norm of example vector vec.

In line 3, $\circ$ represents element-wise multiplication. $G(z \circ s)$ is the generated data point with scaled latent code $z \circ s$. The discriminator D outputs the adversarial value a_g and the predicted latent code z'. When all elements of v are the same, i.e., when the variance of all elements of predicted latent random variable Z' are the same, the scaled latent random variable $Z \circ s$ has the largest differential entropy. On the other hand, when the variance of only one element of the predicted latent random variable is not 0, and the other elements are 0, the scaled latent random variable $Z \circ s$ has the least differential entropy. $\sqrt{d_z}$ is a constant multiplied to make the scaled latent random variable $Z \circ s$ equal to latent random variable Z when the differential entropy of the scaled latent random variable $Z \circ s$ is the largest. The differential entropy of the scaled latent random variable $Z \circ s$ dynamically changes according to the variance of predicted latent random variable Z' during GAN training. As GAN training progresses, the scaled latent random variable $Z \circ s$ converges to have an optimal entropy representing the real data random variable X through generator G.

In line 4, vec^2 means the element-wise square of the example vector vec. avg is a function that calculates the average of a vector. L_{enc} is encoder loss. The encoder loss L_{enc} is equal to the MSE loss between the scaled latent code $z \circ s$ and the scaled predicted latent code $z' \circ s$.

In line 5, a_r is the adversarial value of real data point x. "_" represents not using value. Since the latent code of the real data point x is unknown, the predicted latent code for the real data point x is not used in DLSGAN training.

In lines 6 and 7, f_d and f_g are adversarial loss functions for discriminator D and generator G, respectively. One can find many adversarial losses in [4]. λ_{enc} is encoder loss weight. One can see encoder loss L_{enc} is added to both generator loss L_g and discriminator loss L_d. This means that the generator G and discriminator D are trained cooperatively to reduce the encoder loss L_{enc}.

In line 8, $update$ function update the predicted latent random variable variance v with the new predicted latent code z'. Since the mean of the predicted latent random variable Z' become automatically zero, z'^2 (element-wise square of predicted latent code z') can be considered as the sample variance of predicted latent random variable Z'. A moving average or an exponential moving average can be used for the $update$ function.

Note that the latent code input to the generator should always be scaled by the scale vector s. Therefore, the generated data point is $G(z \circ s)$, and the recovered data point of x is $G(z_x \circ s)$, where z_x is the predicted latent code of the real data point x.

4. Attribute edit dynamic latent scale GAN

InterFaceGAN [17] showed that the latent random variable of GAN learns disentangled representation after linear transformation. InterFaceGAN edits data attributes continuously through the decision boundary of SVM that predicts attributes of latent code. However, InterFaceGAN can edit attributes only when the latent code of the data point is known, and training SVM requires a highly complex method.

In this paper, I introduce AEDLSGAN for integrating InterFaceGAN into the DLSGAN training stage without training a separate classifier or SVM. AEDLSGAN assumes that randomly initialized weights of the SVM are already ideal. Therefore, the only needs to do is to train DLSGAN to fit the SVM. The following algorithm shows the process of computing the loss for training AEDLSGAN.

```
$function\ get\_loss(D, G, Z, X, v, w)$:
1  $z \leftarrow sample(Z)$
2  $x, c_x \leftarrow sample(X)$

3  $s \leftarrow \frac{\sqrt{d_z} v^{\circ 1/2}}{\|v^{\circ 1/2}\|_2}$
4  $a_g, z' \leftarrow D(G(z \circ s))$
5  $L_{enc} \leftarrow \frac{1}{d_z} \|(z - z') \circ s\|_2^2$
6  $a_r, \_ \leftarrow D(x)$

7  $c_y \leftarrow to\_discrete((z \circ s) w)$
8  $L_d \leftarrow f_d(a_r \circ c_x, a_g \circ c_y) + \lambda_{enc} L_{enc}$
9  $L_g \leftarrow f_g(a_g \circ c_y) + \lambda_{enc} L_{enc}$

10 $v \leftarrow update(v, z'^{\circ 2})$

11 $return\ L_d, L_g, v$
```

Algorithm 2. Algorithm to obtain AEDLSGAN loss

In line 2 of Algorithm 2, c_x represents a discrete attribute vector (conditional vector) of real data point x.

In lines 4 and 6, a_g and a_r represent an adversarial vector of generated data and real data, respectively. In Algorithm 2, CAGAN [21] loss is used as a class conditional GAN. Therefore, a_g and a_r are vectors, not one-dimensional values.

In line 7, c_y represents a discrete attribute vector of generated data $G(z \circ s)$. w represents the weights of SVM. Conditional vector of generated data c_y is obtained from the matrix product of the scaled latent vector $z \circ s$ and the SVM weights w. SVM weights w are $d_z \times d_c$ size matrix, where d_c is the dimension of the attribute vector. AEDLSGAN assumes SVM weights w are already ideal, so SVM weights w never change after initialization. $to\_discrete$ is a function that converts the continuous output value of SVM into a discrete conditional vector. For example, when the mean of the latent random variable Z is 0, and the SVM weights are initialized from $N(0, \sigma^2)$, the binary step function ($f(x) = \begin{cases} 1, x > 0 \\ 0, x \leq 0 \end{cases}$) can be used to train each attribute in a balanced way.

Unlike simply adding class conditional GAN to DLSGAN, AEDLSGAN can edit attributes continuously like InterFaceGAN.

5. Experimental results and discussion

5.1 Experiment settings

I trained GAN to generate the celeb A dataset [8] resized to 128x128 resolution. As the model architecture, a partially reduced version of StyleGAN2 [9] was used. Batch operations (minibatch stddev layer) in StyleGAN2 discriminator are removed so that the encoder encodes one data point as one latent vector. As an adversarial loss, NSGAN with R1 loss [3] was used as StyleGAN2. I used encoder loss weight $\lambda_{enc} = 1.0$.

I compared the performance of DLS and MSE. I also compared the effect of the encoder loss L_{enc} on generator loss L_g. For $update$ function for DLSGAN, I used a moving average using the past 512×16 samples.

5.2 Experiment results

The following figures show the performance of DLS and MSE, with and without encoder loss L_{enc} on generator loss L_g.

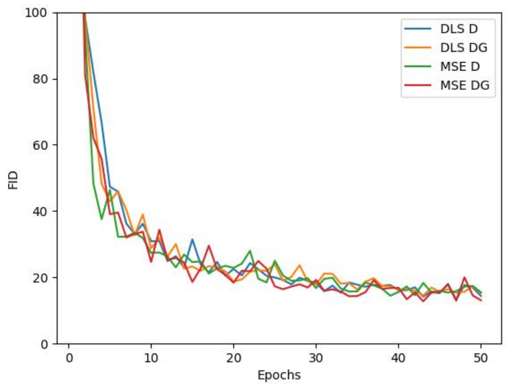


Figure 1. FID according to the encoder loss L_{enc}

Figure 1 shows the FID [10] according to the encoder loss L_{enc} for each epoch. In figure 1, DLS and MSE represent DLS (dynamic latent scale) and MSE (mean squared error), respectively. D means weighted encoder loss $\lambda_{enc}L_{enc}$ was added to only discriminator loss L_d. DG means weighted encoder loss $\lambda_{enc}L_{enc}$ was added to both discriminator loss L_d and generator loss L_g.

One can see that there is little difference in the generative performance of GAN according to the type of encoder loss L_{enc}.

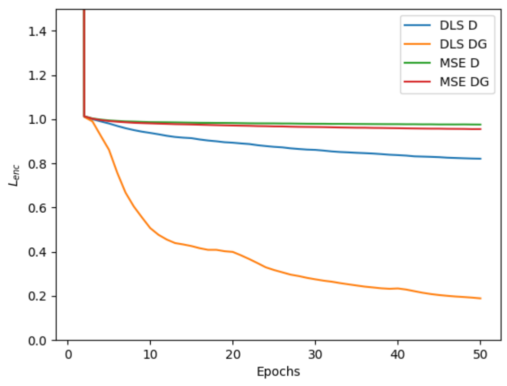


Figure 2. Average L_{enc} according to the encoder loss L_{enc}

Figure 2 shows the average encoder loss L_{enc} according to the encoder loss L_{enc} for each epoch. One can see that with MSE loss, encoder loss L_{enc} hardly changes from 1. This shows that the encoder E trained with MSE loss fails to converge due to the information loss on the latent random variable in the generator G. On the other hand, one can see that the encoder loss L_{enc} continuously decreases as training progresses with DLS. This shows that the model converges with DLS. Also, one can see that the encoder loss L_{enc} is much lower when encoder loss L_{enc} is added to the generator loss L_g.

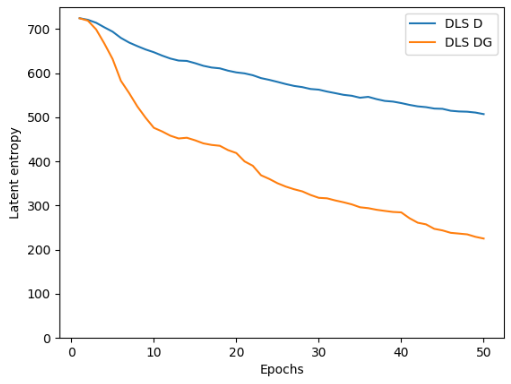


Figure 3. Differential latent entropy of DLSGAN

Figure 3 shows differential entropy of latent random variable of DLSGAN for each epoch. As encoder loss L_{enc}, one can see that the differential entropy of the latent random variable Z decreases faster when encoder loss L_{enc} is added to the generator loss L_g.

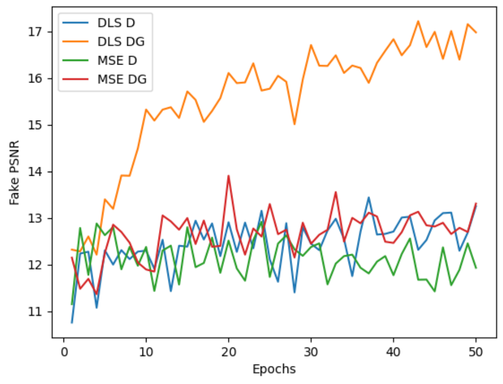


Figure 4. Average PSNR for generated images according to the encoder loss L_{enc}

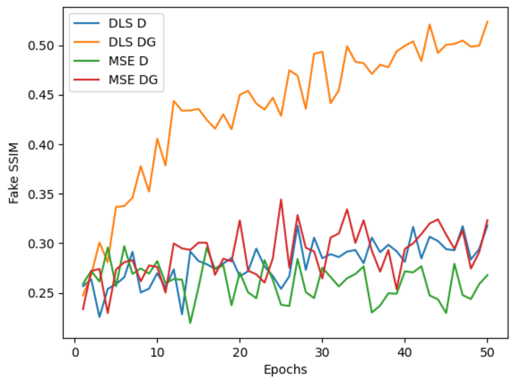


Figure 5. Average SSIM for generated images according to the encoder loss L_{enc}

Figures 4 and 5 show the average PSNR and SSIM for each epoch when reconstruction is performed on the generated images. One can see that the performance of reconstruction on generated images is much better with DLS DG. Also, both DLS and MSE perform better when the encoder loss L_{enc} is added to the generator loss L_g.

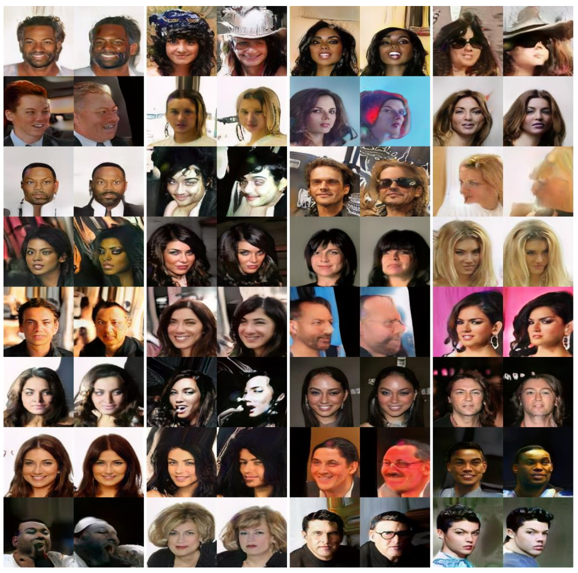


Figure 6. MSE DG generated images reconstruction example

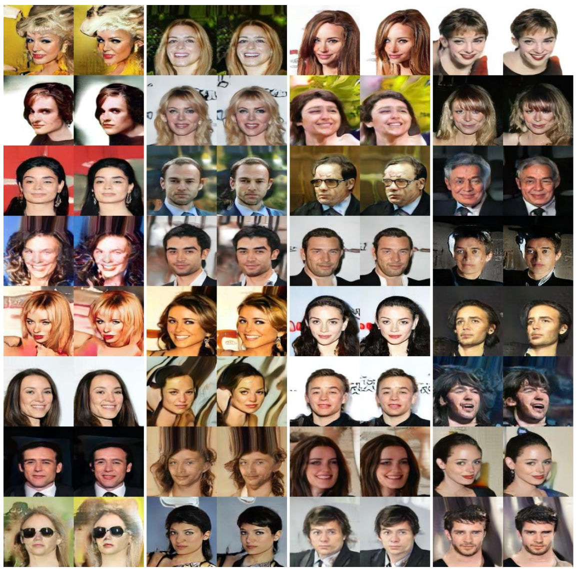


Figure 7. DLS DG generated images reconstruction examples

Figures 6 and 7 show examples of generated images reconstruction with MSE DG and DLS DG. Big size images of all figures for generated images are in the appendix.

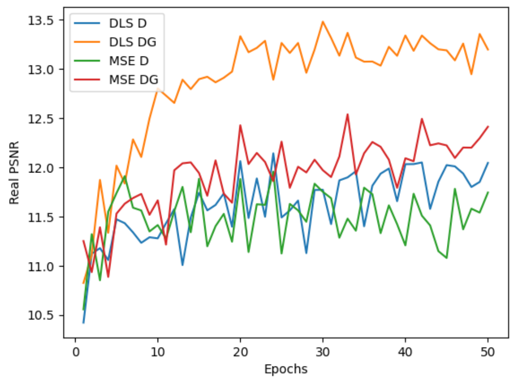


Figure 8. Average PSNR for test images according to the encoder loss L_{enc}

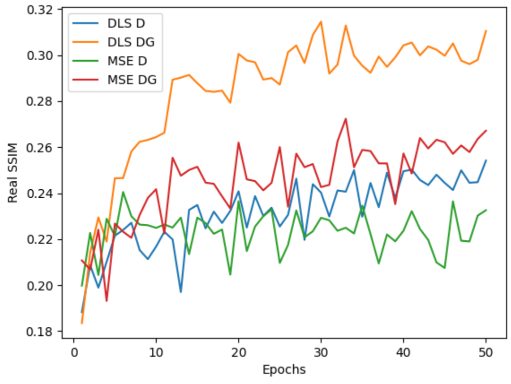


Figure 9. Average SSIM for test images according to the encoder loss L_{enc}

Figures 8 and 9 show the average PSNR and SSIM for each epoch when reconstruction is performed on the test images (real images). As reconstruction on the generated images, one can see that the performance of reconstruction on test images is much better with DLS DG.

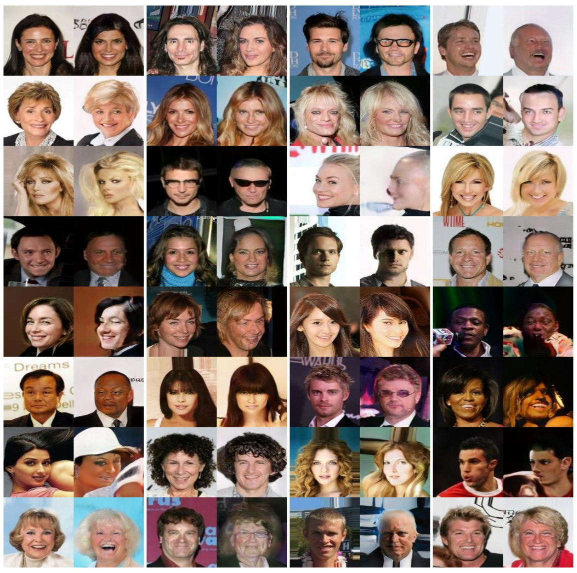


Figure 10. MSE DG Test images reconstruction examples

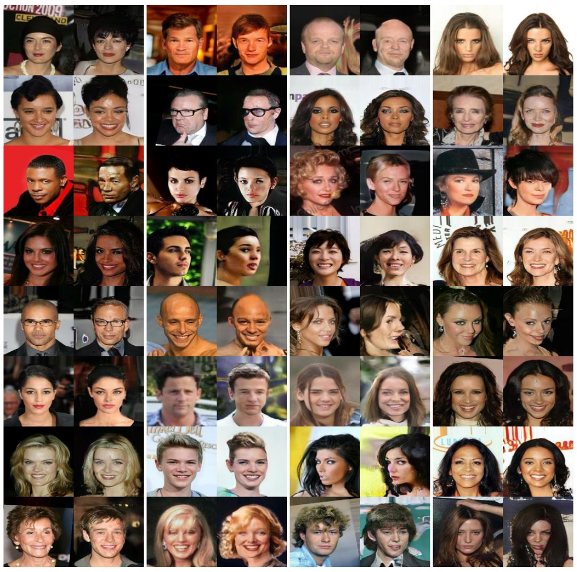


Figure 11. DLS DG Test images reconstruction examples

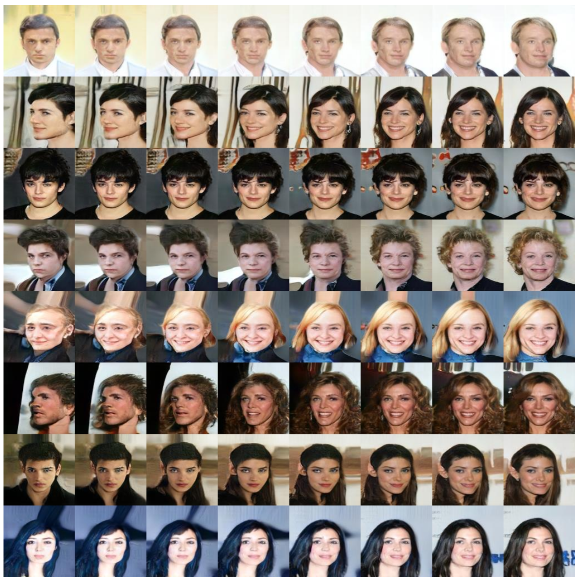


Figure 12. Latent interpolation on most important element

Figures 10 and 11 show examples of generated images reconstruction with MSE and DLS.

5.3 Additional results of DLSGAN

The following figures show some additional results of DLS DG.

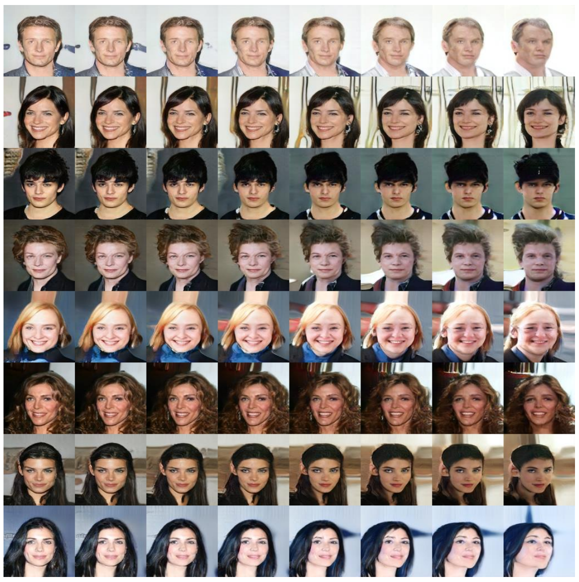


Figure 13. Latent interpolation on second most important element

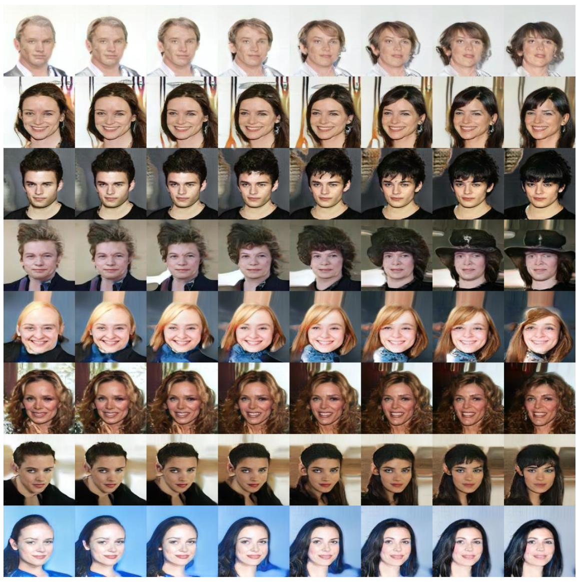


Figure 14. Latent interpolation on third most important element

Figures 12-14 show interpolating one important element of the latent random variable Z from -2 to 2 in DLSGAN. The larger the elements of the scale vector s, the more important (more informative) elements. That is, figures 12-14 are the results of interpolating the index of the first, second, and third-largest element of scale vector s, respectively.

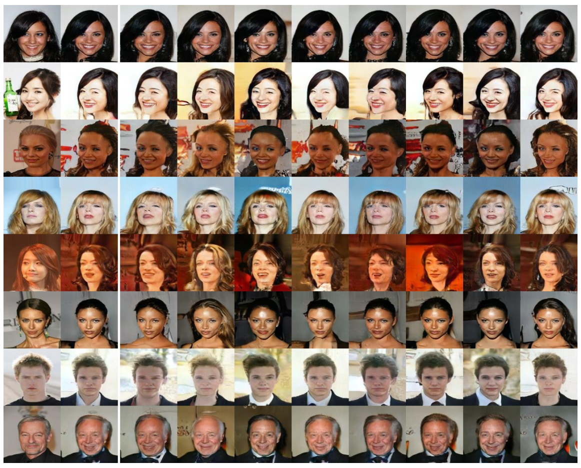


Figure 15. Stochastic noise on recovered latent codes

Figure 15 shows the result of adding stochastic noise to the predicted latent code for the test images. The first column of figure 15 shows the input images (test images), and the second column shows the reconstructed images through predicted latent codes. Based on the thick white line, the images on the right side show the images generated by adding normal noise with mean 0 and variance 0.3 for each element of the predicted latent code. Each column has the same noise.

5.4 AEDLSGAN experiment

I used 256×256 resolution resized CelebA images for AEDLSGAN experiments. Deep convolution with Wavelet [21] architecture was used for the AEDLSGAN experiment.

I trained the model with three attributes of the CelebA dataset: bangs, gender, and smile. SVM weights w are initialized with $N(0,1^2)$, and a binary step function was used for the

$to\_discrete$ function. To balance the encoder loss and adversarial loss, I used the average of each adversarial loss of CAGAN.

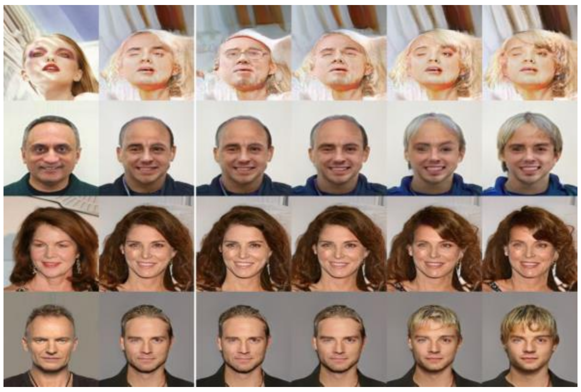


Figure 16. Attribute 'Bangs' transfer of test images

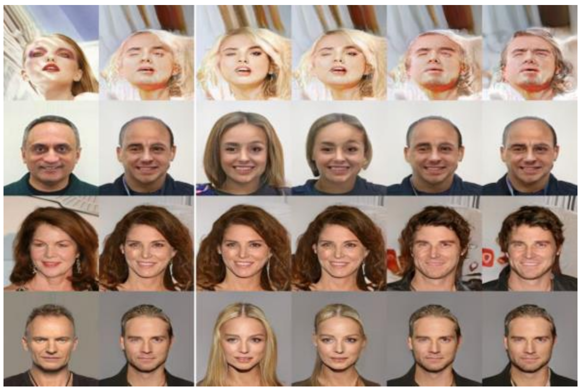


Figure 17. Attribute 'Gender' transfer of test images

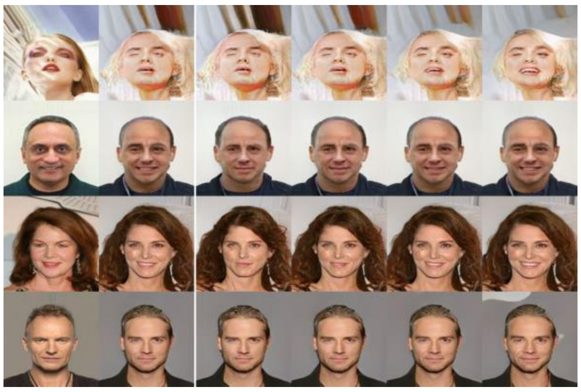


Figure 18. Attribute 'Smile' transfer of test images

Figs. 16-18 shows continuous attributes transfer of test images. The first column of figures shows the input images, and the second column shows the reconstructed images through predicted latent codes. The images on the right side show results of continuous attribute transfer. SVM score $(z' \circ s)w/d_z$ changes from -0.1 to 0.1. If the score is positive, the image has the attribute 'has bangs,' 'male,' and 'smile,' respectively. One can see that AEDLSGAN can edit the attribute of the input image continuously.

6. Conclusion

In this paper, I proposed a DLSGAN, a method for training a generator without information loss on the latent random variable, and an encoder that inverts the generator. Dynamic latent scale GAN dynamically adjusts the scale of the i.i.d. latent random variable to have the optimal entropy to express the data random variable. This ensures that there is no information loss on the latent random variable in the generator so that the encoder can converge to invert the generator. The encoder of DLSGAN showed much better performance than simply using MSE loss. Also, DLSGAN can be used for continuous attribute editing with InterFaceGAN.

7. References

[1] GAN Inversion: A Survey

https://arxiv.org/abs/2101.05278

[2] Generative Adversarial Networks

https://arxiv.org/abs/1406.2661

[3] Which Training Methods for GANs do actually Converge?

https://arxiv.org/abs/1801.04406v4

[4] Are GANs Created Equal? A Large-Scale Study

https://papers.nips.cc/paper/2018/file/e46de7e1bcaaced9a54f1e9d0d2f800d-Paper.pdf

[5] A Style-Based Generator Architecture for Generative Adversarial Networks

https://arxiv.org/abs/1812.04948

[6] Encoding in Style: a StyleGAN Encoder for Image-to-Image Translation

https://openaccess.thecvf.com/content/CVPR2021/papers/Richardson_Encoding_in_Style_A_StyleGAN_Encoder_for_Image-to-Image_Translation_CVPR_2021_paper.pdf

[7] It Takes (Only) Two: Adversarial Generator-Encoder Networks

https://ojs.aaai.org/index.php/AAAI/article/view/11449

[8] Celeb A dataset

https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

[9] Analyzing and Improving the Image Quality of StyleGAN

https://arxiv.org/abs/1912.04958

[10] GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium

https://arxiv.org/abs/1706.08500

[11] Adversarial learned inference

https://openreview.net/forum?id=B1ElR4cgg

[12] Adversarial feature learning

https://openeview.net/forum?id=BJtNZAFgg

[13] Large scale adversarial representation learning

https://papers.nips.cc/paper/2019/hash/18cdf49ea54eec029238fcc95f76ce41-Abstract.html

[14] Conditional GAN

https://arxiv.org/abs/1411.1784

[15] InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets

https://papers.nips.cc/paper/2016/hash/7c9d0b1f96aebd7b5eca8c3edaa19ebb-Abstract.html

[16] Invertible Conditional GANs for image editing

https://arxiv.org/abs/1611.06355

[17] Interpreting the Latent Space of GANs for Semantic Face Editing

https://ieeexplore.ieee.org/document/9157070

[18] In-Domain GAN Inversion for Real Image Editing

https://www.ecva.net/papers/eccv_2020/papers_ECCV/papers/123620579.pdf

[19] Exploiting Spatial Dimensions of Latent in GAN for Real-time Image Editing

https://openaccess.thecvf.com/content/CVPR2021/papers/Kim_Exploiting_Spatial_Dimensions_of_Latent_in_GAN_for_Real-Time_Image_CVPR_2021_paper.pdf

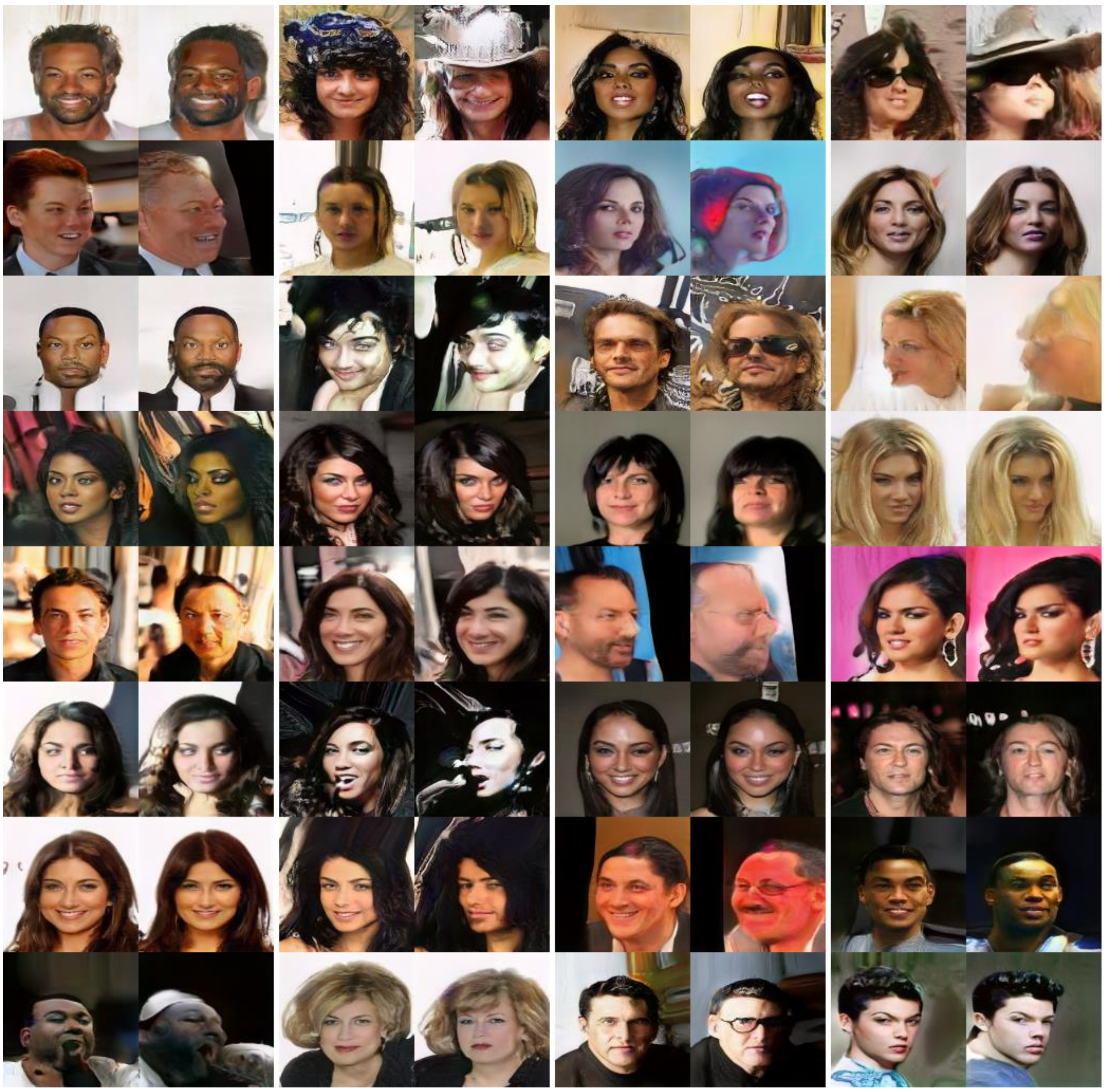
[20] Collaborative Learning for Faster StyleGAN Embedding

https://arxiv.org/abs/2007.01758

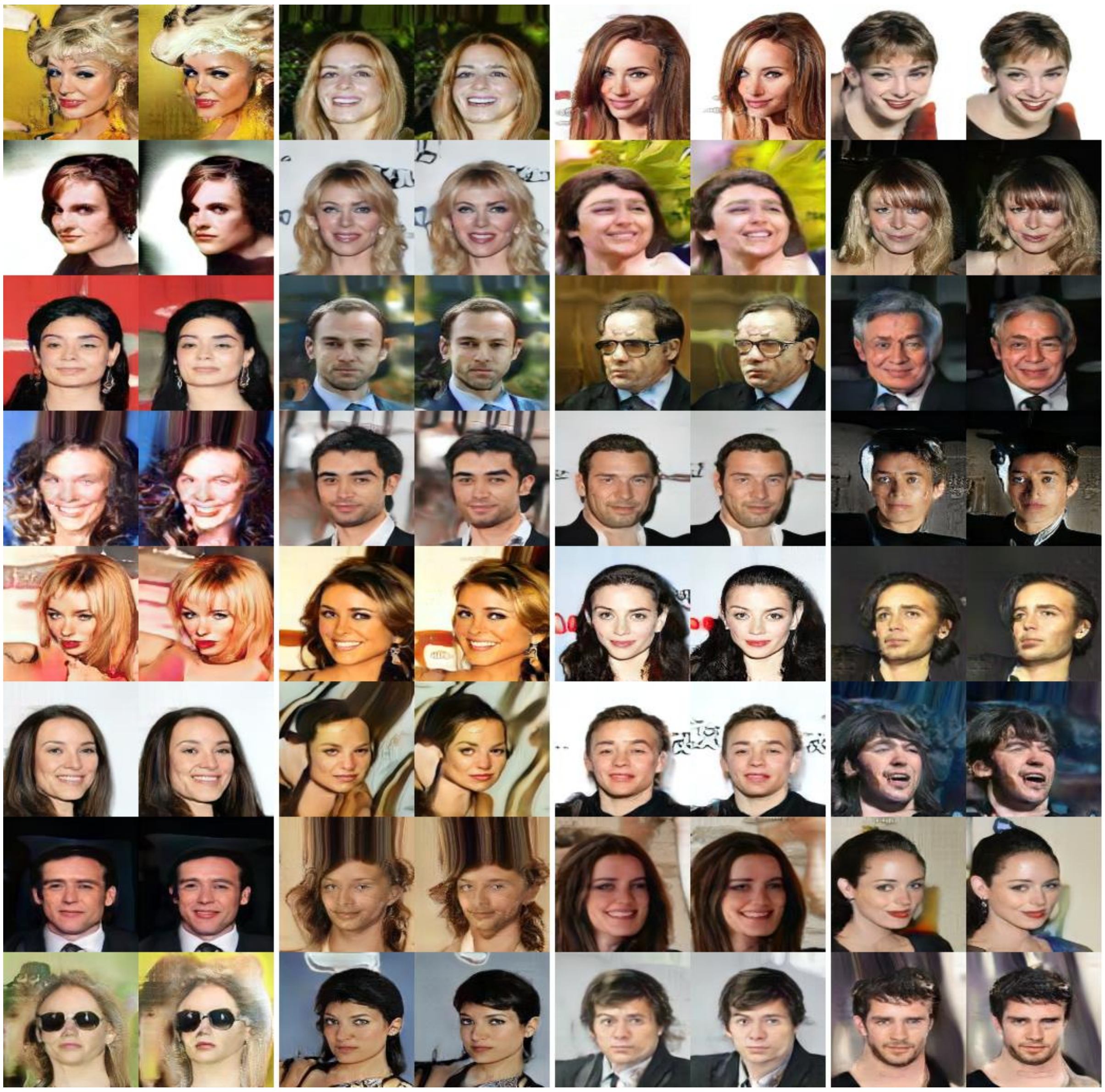
[21] Conditional Activation GAN: Improved Auxiliary Classifier GAN https://ieeexplore.ieee.org/document/9274378

[22] SWAGAN: a style-based wavelet-driven generative model https://dl.acm.org/doi/abs/10.1145/3450626.3459836

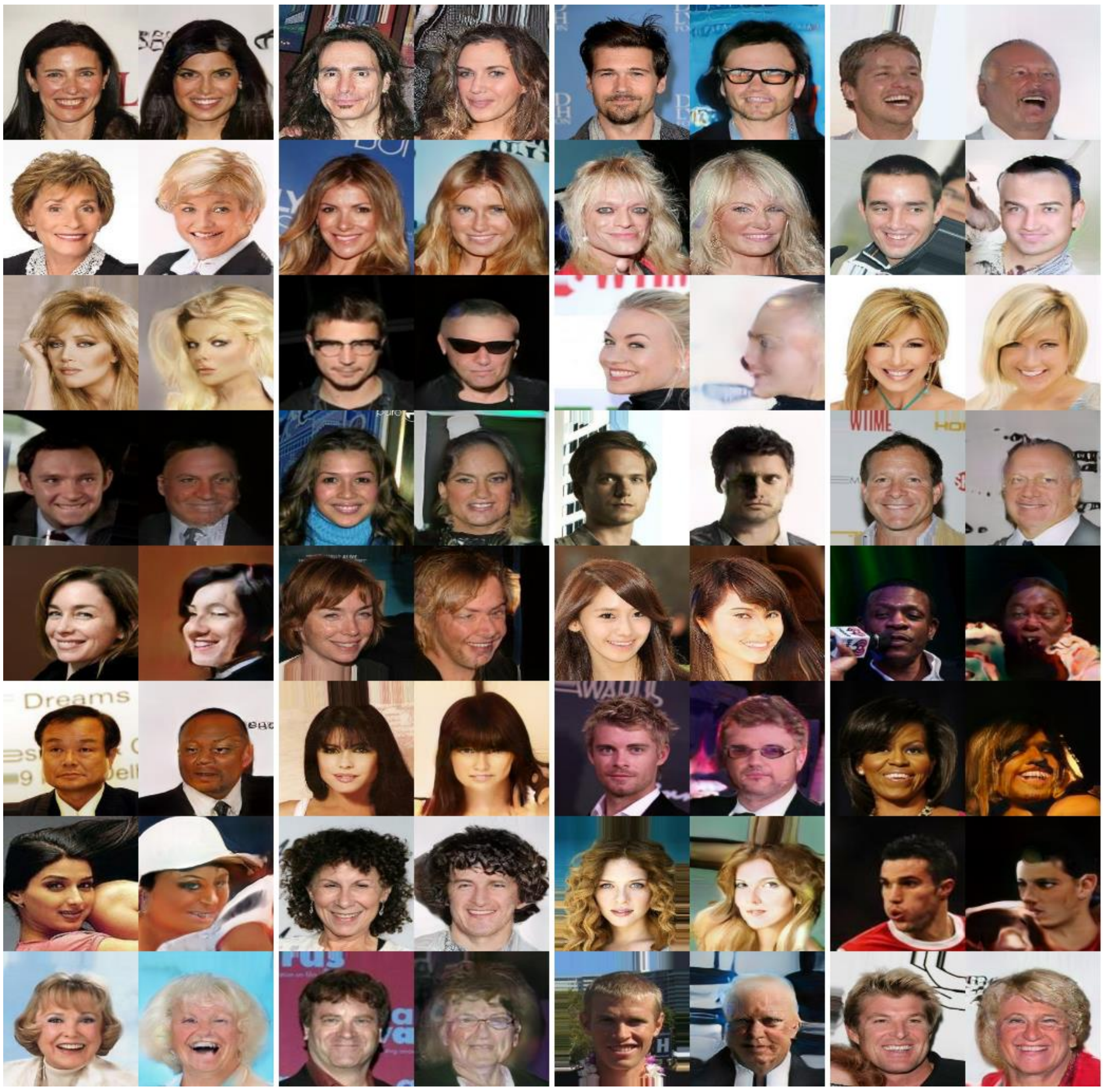
Appendix

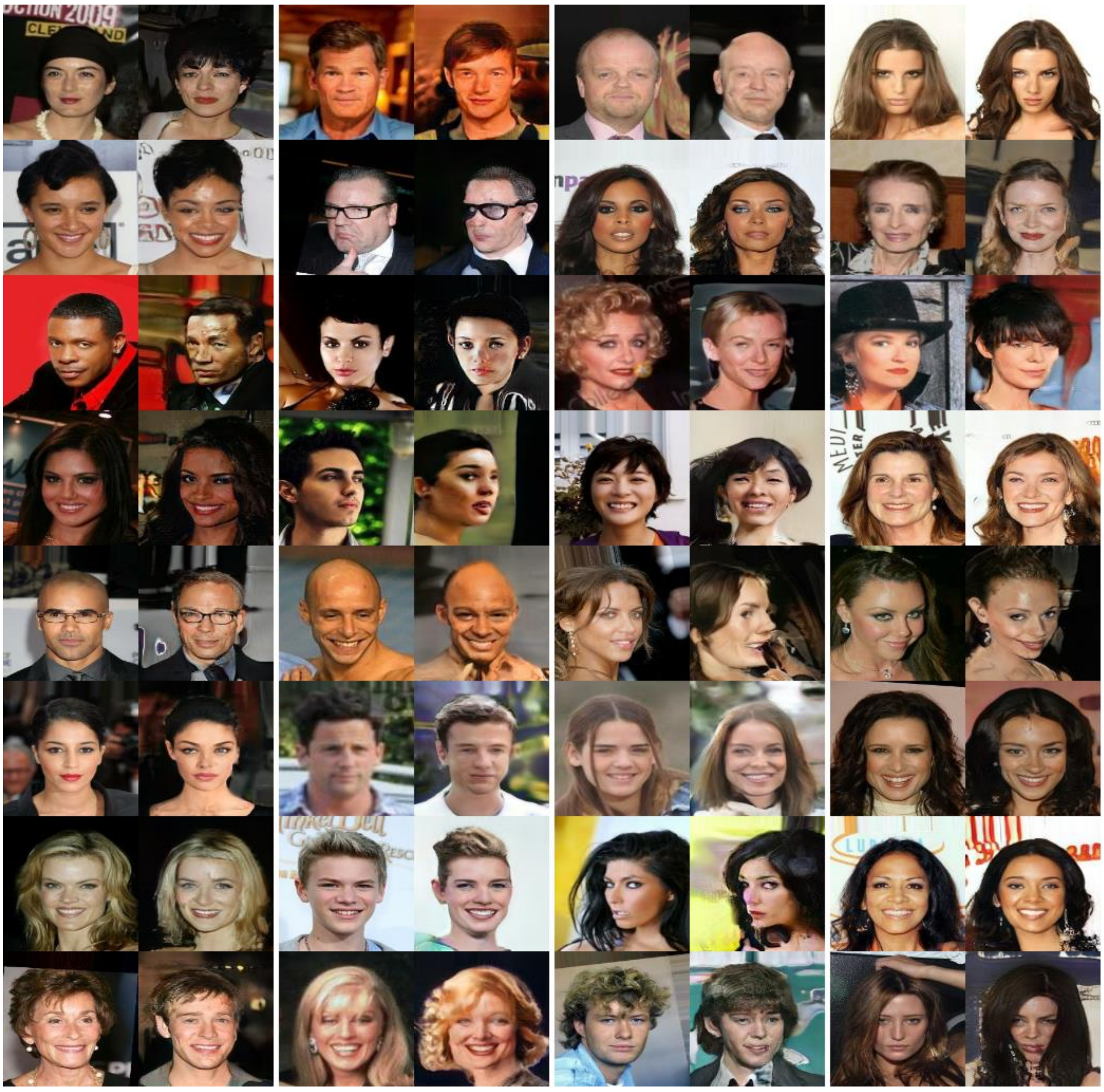


Big size image of figure 6. MSE DG generated images reconstruction example

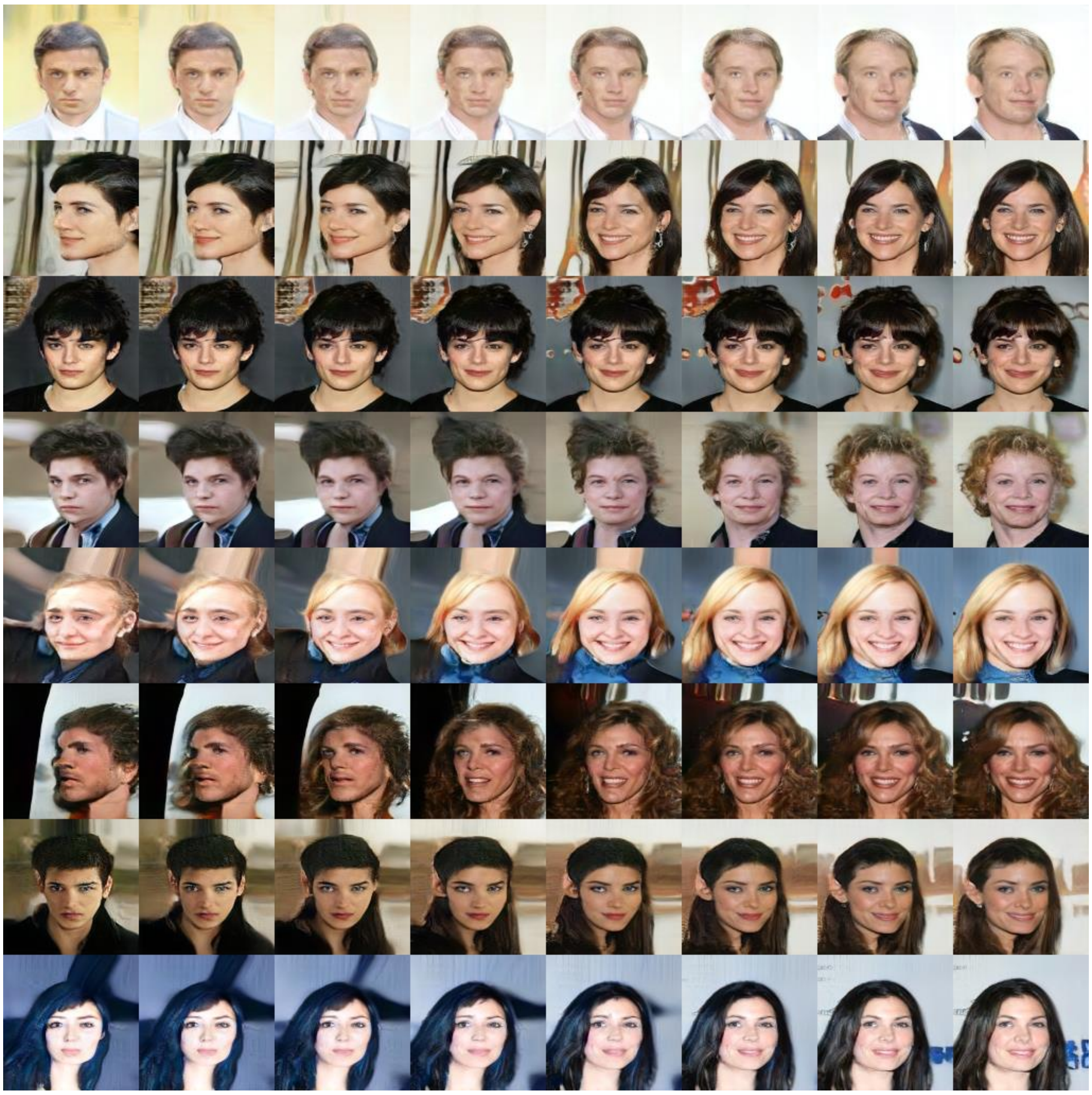


Big size image of figure 7. DLS DG generated images reconstruction examples

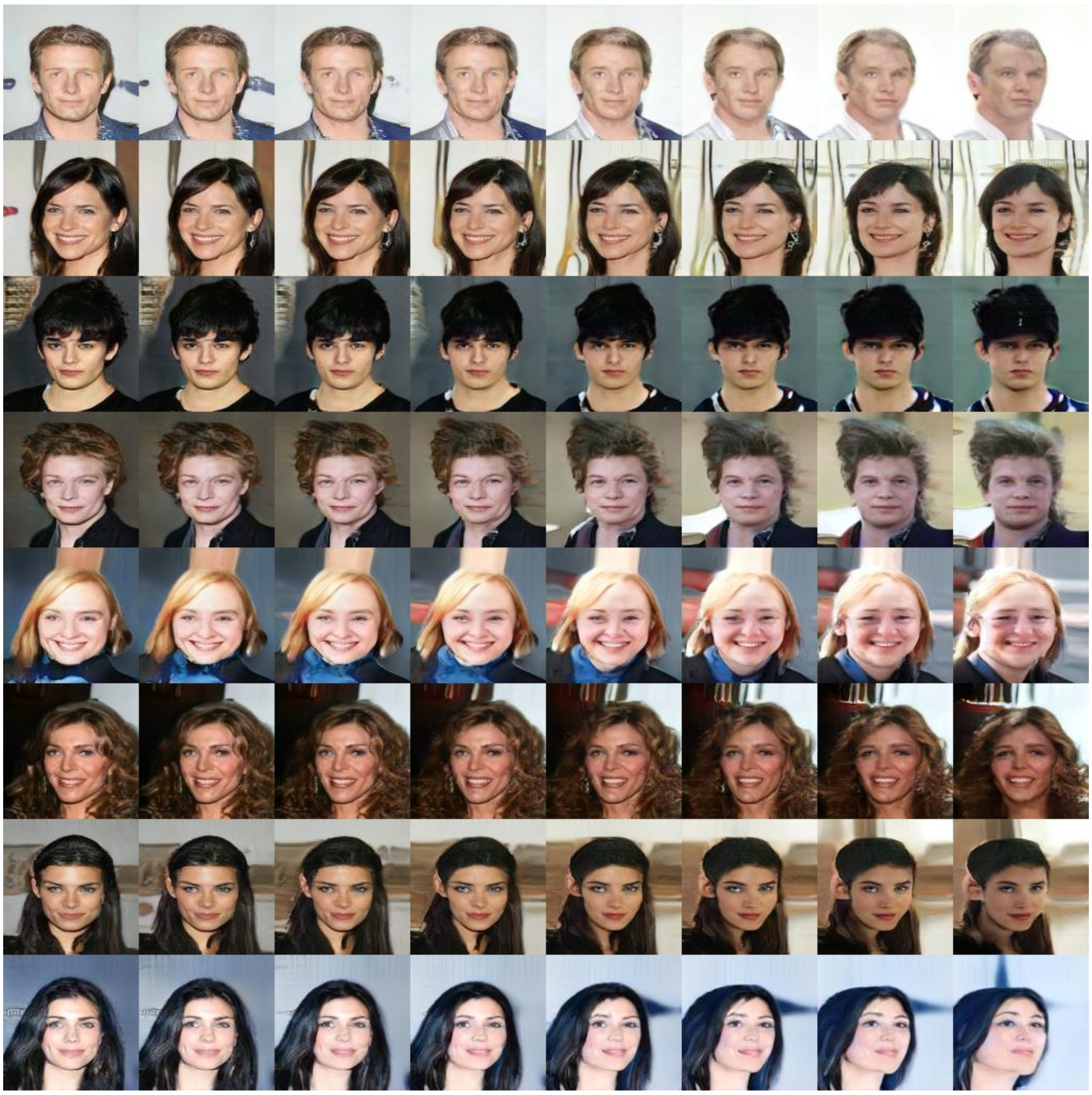

Big size image of figure 10. MSE DG Test images reconstruction examples



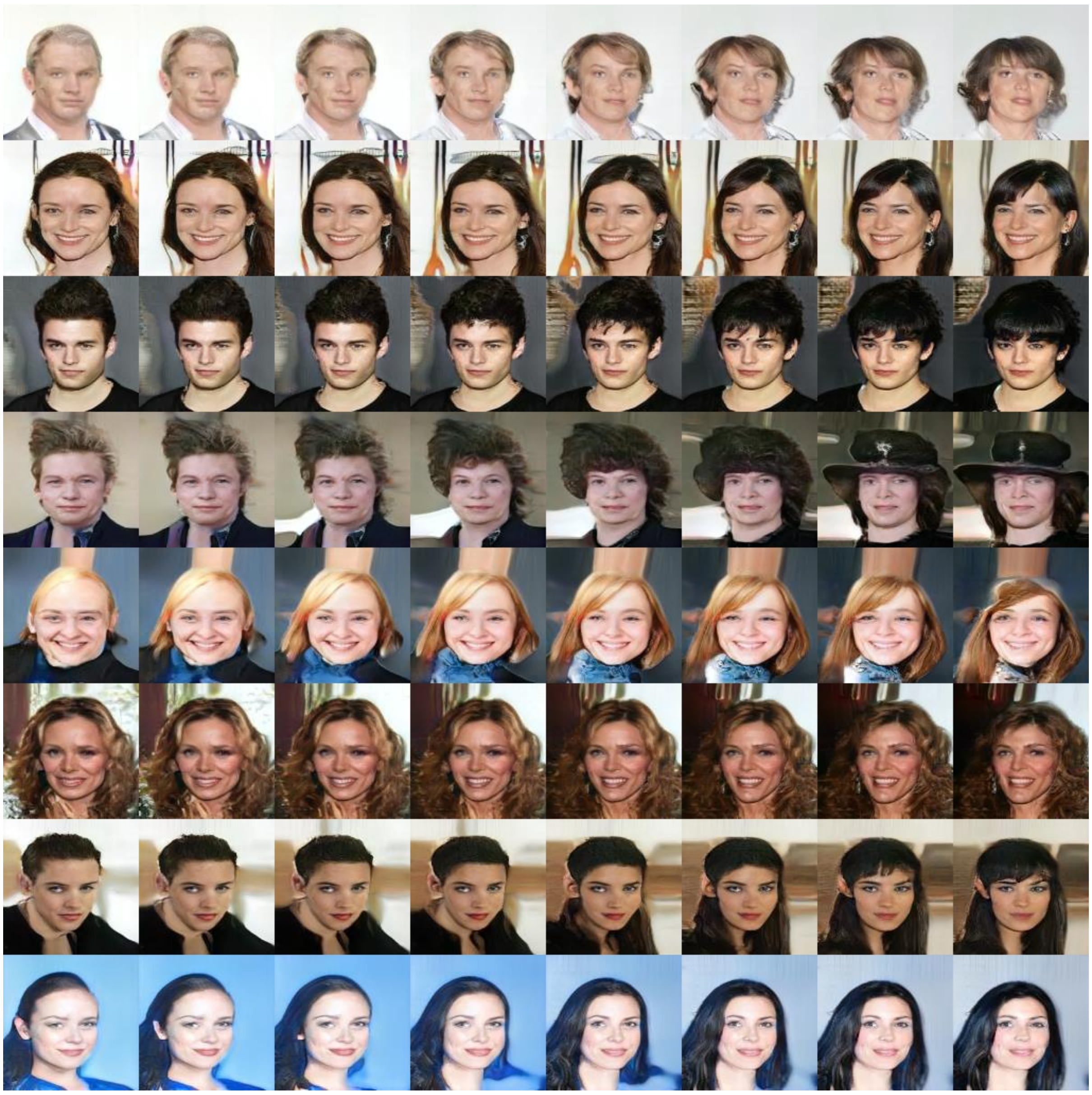
Big size image of figure 11. DLS DG Test images reconstruction examples



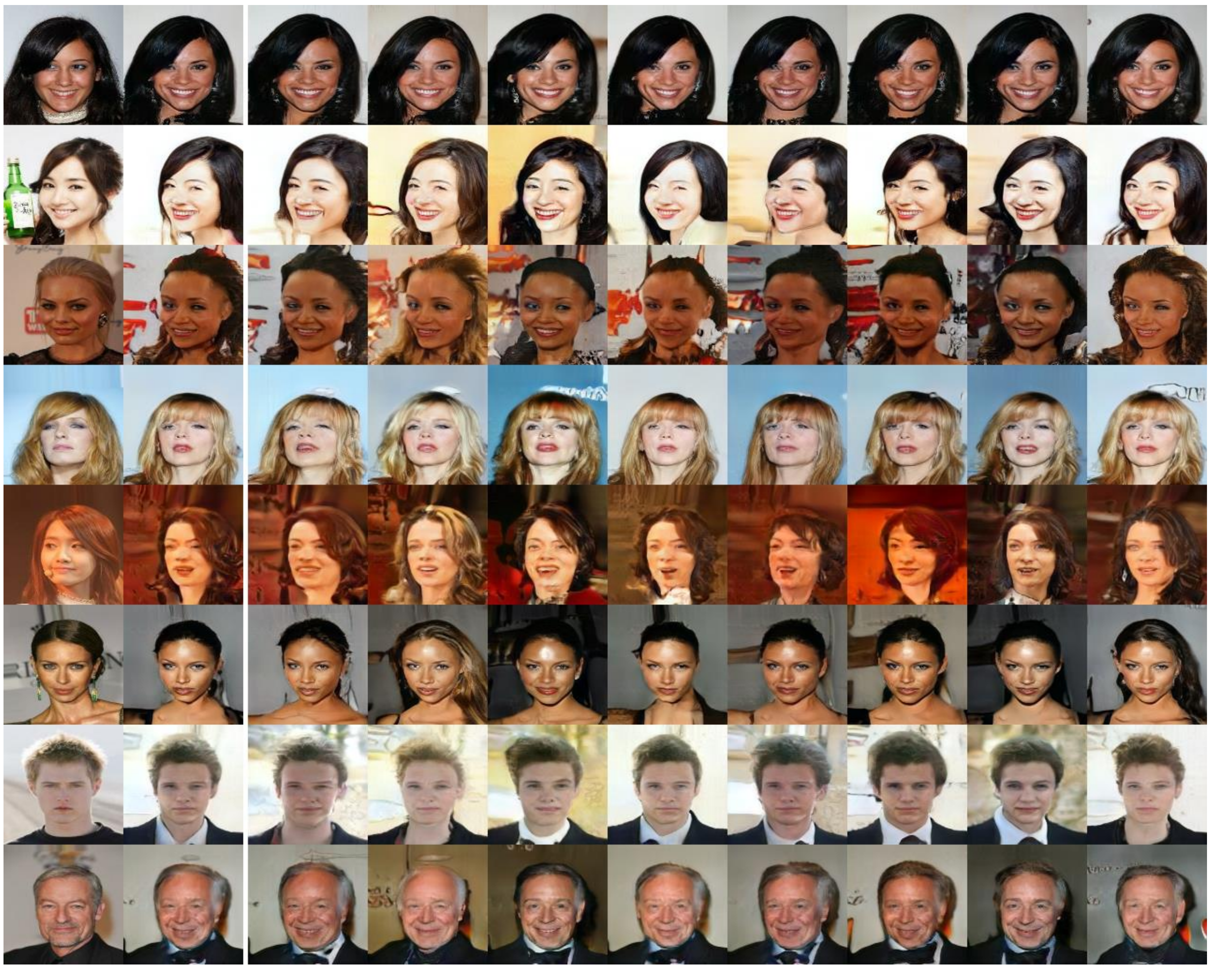
Big size image of figure 12. Latent interpolation on most important element



Big size image of figure 13. Latent interpolation on second most important element



Big size image of figure 14. Latent interpolation on third most important element



Big size image of figure 15. Stochastic noise on recovered latent codes