Structured GANs

Irad Peleg[1] and Lior Wolf[1,2]

[1]Tel Aviv University
[2]Facebook AI Research

Abstract

We present Generative Adversarial Networks (GANs), in which the symmetric property of the generated images is controlled. This is obtained through the generator network's architecture, while the training procedure and the loss remain the same. The symmetric GANs are applied to face image synthesis in order to generate novel faces with a varying amount of symmetry. We also present an unsupervised face rotation capability, which is based on the novel notion of one-shot fine tuning.

1. Introduction

Symmetry is a prominent property that has gained attention in face recognition [9, 1, 4] and other applications [8]. Considering symmetry in the context of image generation with GANs, the core research questions that we ask in this work are: (1) how to control the symmetry of generated images and (2) how to rotate an object, when the training set did not contain the relevant supervision.

Question number one is answered by proposing two alternative architectures for generative networks. In the first architecture, the first few elements of the input vector serve as the antisymmetric component, the others serve as the symmetric component. In the second architecture, the generated image is symmetric, if the input vector has a palindrome structure, i.e., remains unchanged when flipping the order of elements. Both architectures are shown to work much better than an approach in which the loss is used in order to control the symmetry property. Fig. 1 illustrates how z is converted to $G(z)$ in the symmetric GANs.

The second question is answered by a process in which the generator network is adapted in order to generate a specific face. This powerful technique for preserving a given identity in the generated images is a general one and can be applied to many other GAN-based methods.

As far as we know, we are the first to manipulate the structure of the generator in order to enforce properties on the generated image. Due to space constraints, we have placed in the Supplementary, a different structure manipulation that creates tilable textured patches. Therefore, the core idea of our work also applies to completely different tasks.

1.1. Related Work

The task of generating realistic looking images has challenged computer vision for a long time. Recently, a major leap has been made with the development of the Generative Adversarial Network (GAN) [6]. These architectures employ two networks G and D, which provide training signals to each other. The network D tries to distinguish between the "fake" images generated by G and the real images provided as a training set. Network G tries to fool D and creates images that look as realistic as possible.

The specific architecture that we employ is based in part on the DC-GAN [10] method. This architecture uses deconvolution and batch normalization in order to create attractive outputs. Specifically, the input vector in DC-GAN is a $100D$ vector, whose elements are sampled i.i.d from a uniform distribution. In our case, we encode symmetry into this vector, and through the use of specific architectures, we enforce the generated output $G(z)$ to display the required level of symmetry.

Through our focus on symmetry, our networks are able to extract the notion of yaw. GAN-based ways to extract semantic regularities in an unsupervised manner, include InfoGAN [5] and Fader networks [7].

2. Symmetric GANs

We present two architecture-based methods and one loss-based method to serve as a baseline. The difficulty in training the loss-based method successfully,

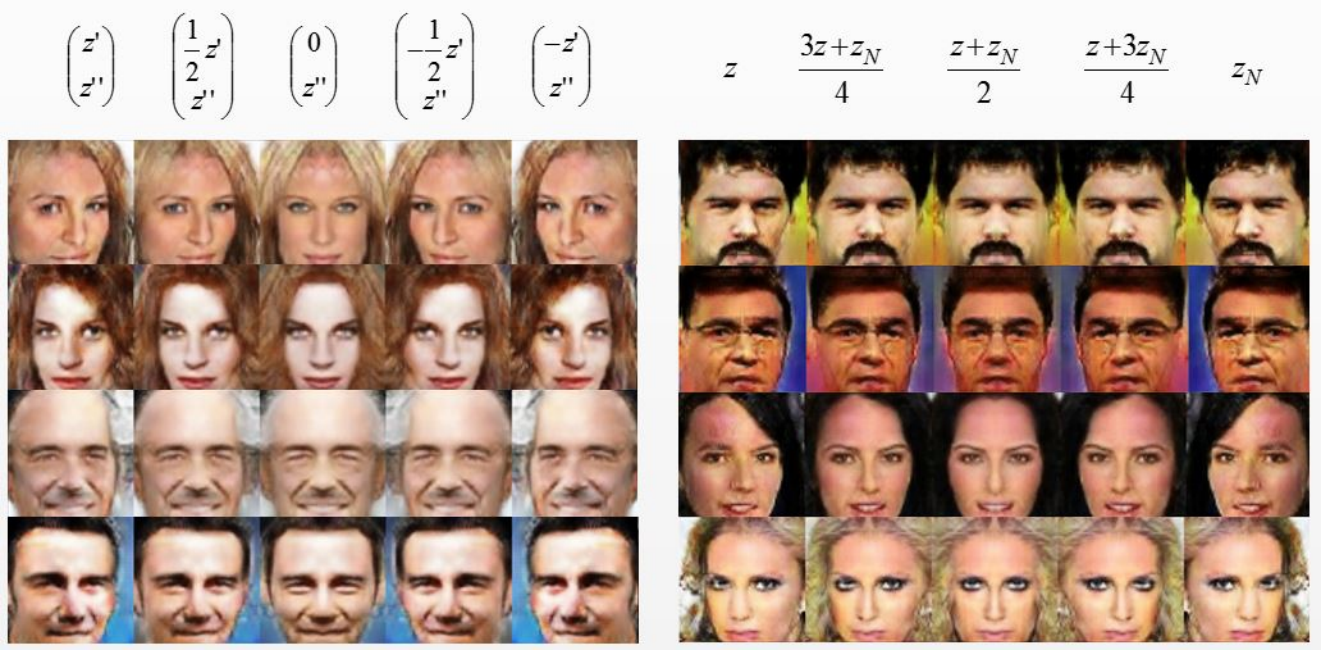

Figure 1: An illustration of the relationship between z and $G(z)$ for the two proposed methods. (Left) In the first method, called the z' architecture, part of the GAN's input encodes symmetry. When this part z' is zero, the face is symmetric, and when z' is negated, the image is mirrored along the y axis. (Right) In the flip architecture, the image is symmetric, when the flipped version of z, which we denote as $z_N = \mathbf{flip}(z)$ satisfies $z_N = z$.

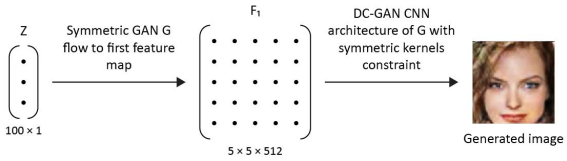

Figure 2: Flow of symmetric generator.

emphasizes the effectiveness of the architecture-based methods.

2.1. The Symmetric Architectures

Our GANs are symmetric end-to-end, including both the generator G and the discriminator D.

Both G and D contain convolutional layers, as well as fully connected ones. In order to maintain symmetry, both these layer types are augmented. The fully connected parts are handled differently in D and in G. We also present two architectures for G, which differ exactly in the way in which the fully connected layers are constructed. The convolutional layers are treated exactly the same in all variants of G and in D and in all of these cases the same symmetric kernels are introduced. The flow of the symmetric generator in presented in Fig. 2.

The two alternative architectures differ in the very first layer. The first architecture creates symmetric images for inputs $z = \begin{bmatrix} z' \\ z'' \end{bmatrix}$ in which the first part z' is zero. The second architecture produces symmetric outputs for inputs z which are themselves symmetric, i.e., $z = \mathbf{flip}(z)$, where the **flip** operator switches the first element with the last one, the second element with the one before the last and so on.

2.1.1 The Generator of the z' Architecture

The first generated architecture splits the input vector z into two parts. The first, z' is the anti-symmetric part, while the second z'' is the symmetric part. For input vectors that have $z' = 0$, the output is completely symmetric.

The architecture that ensures the symmetry and anti-symmetric property enforces this structure on the first feature map, and maintains it thereafter by employing symmetric convolutional kernels.

The generation of the first feature map is depicted in Fig. 3. The random input vector z, which contains 100 i.i.d uniformly $[-1, 1]$ distributed elements, is split into two parts. The first part $z' \in \mathbb{R}^5$ is mapped through a fully connected layer (affine transformation) to a vector of size 5120. This vector is then re-

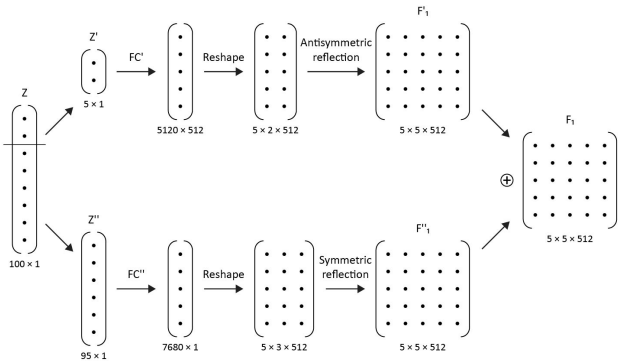

Figure 3: Symmetric GAN G flow to first feature map decomposed to symmetric and antisymmetric components

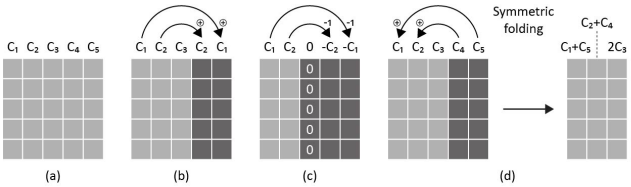

Figure 4: A comparison of four kernel types. (a) The standard kernel or feature map. (b) The symmetric kernel and feature map. (c) The anti-symmetric feature map. (d) Symmetric folding of a feature map.

shaped into 512 kernels of size 5×2, which are transformed to antisymmetric kernels of size 5×5 by taking the last column to be the negative of the first column and the column before, to be the negative of the second column, see Fig. 4(c).

The remaining 95 elements of z, denoted by z'' are mapped to a vector of size $7,680$, which is then reshaped to 512 kernels of size 5×3. By performing the symmetric reflection depicted in Fig. 4(b), i.e., copying the first column to the fifth column and the second to the fourth column, 5×5 kernels are obtained.

The rest of the network follows the DC-GAN architecture, except that the 5×5 kernels of 25 parameters each, are replaced with symmetric kernels that contain only 15 different parameters.

Fig. 4(c) shows an antisymmetric reflection of the symmetry break part on the first feature map of G.

2.1.2 The Generator of the flip Architecture

In the alternative symmetric architecture, the generator produces symmetric images, when for a given input z, it happens that $z = \mathbf{flip}(z)$. Here, too, the symmetric kernels throughout the layers make sure that a

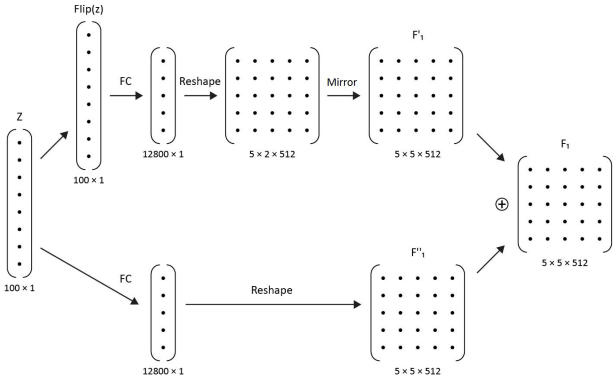

Figure 5: symmetric GAN G flow to first feature map as symmetric mapping from a vector to an image.

symmetric feature map from one layer, leads to a symmetric feature map in the next.

The architecture of the first layer is depicted in Fig. 5. The same matrix is applied as weights of a fully connected layer, to both z and $\mathbf{flip}(z)$, to obtain vectors of length 12,800. These are reshaped to 512 feature maps of size 5×5. The feature map that results from the flipped vector is being flipped left and right (the first columns is replaced with the fifth and second with the fourth). The 512 matching pairs, one from each branch of the network, are then summed.

Given that $z = \mathbf{flip}(z)$, the two branches produce identical feature maps before the left-right flipping and after flipping, thus symmetry is obtained.

2.1.3 The Symmetric Discriminator Architecture

A desired property in our framework is that the discriminator D would return the same probability of "real" for an image and its mirrored version. This is not the case in conventional architectures, and, for example, during training the conventional discriminator would overfit on the training sample, while fails on its mirror image, see Fig. 6.

We, therefore, enforce symmetry by using a specific architecture. First, we replace the discriminator of the DCGAN with one that has symmetric left-right kernels, similar to what we have used in the generators. For the last layer, we then obtain a feature map of size $5 \times 5 \times 512$. This feature map undergoes a symmetric folding, as depicted in Fig. 4(d). Namely, the first (second) column is replaced by the sum of this column and the fifth (fourth) column. In addition, the third column is doubled. The result can be seen as a vector the size of 7680. Through a fully connected layer, a single output is then obtained. The succession of symmetric kernels and the symmetric folding, en-

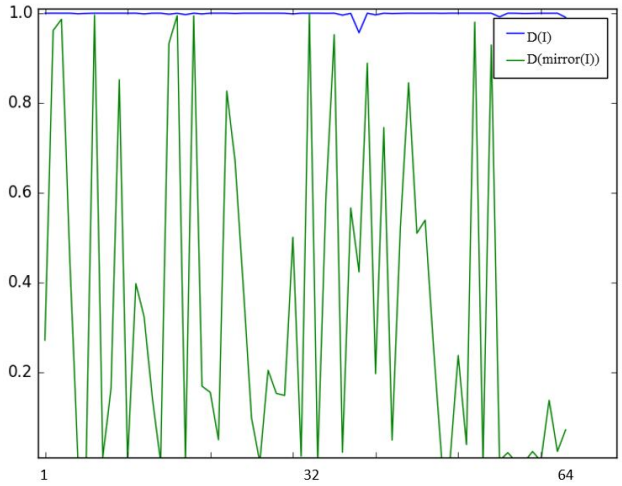

Figure 6: The output probability of the "real" label of the classifier D for the training samples and their mirrored image, when the classifier does not use the symmetric invariant architecture of Sec. 2.1.3. There is a clear overfitting on the training data. However, there is also a great uncertainty about the flipped version.

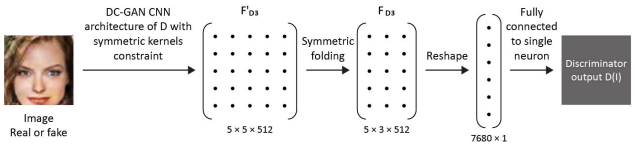

Figure 7: Flow of symmetric discriminator.

sure that mirror images obtain the same score by D. The flow of the symmetric discriminator is shown in Fig. 7.

2.2. Alternative Loss Based Method

In addition to the architecture based approaches, we evaluated baseline methods for which the symmetry is enforced by adding a loss term. Multiple experiments were conducted, each with varying emphasis (weight) of the added loss term. The overall conclusion is that such training is highly unstable and that the model tends to collapse and either ignore the term (when the weight is low), or result in symmetric images for every input (when the weight is high).

As above, the vector z encodes symmetry either by having the first five elements z' as zero, or by having the vector invariant to element flipping.

The loss term we add is based on pairs of inputs z. For the z' based symmetry encoding, we create pairs (z, z_N), where z_N is identical to z, except that the first elements z' are replaced by $-z'$. For the flipping based symmetry encoding, we take $z_N = \mathbf{flip}(z)$.

Figure 8: Optimizing vector z to fit the top input image in BEGAN [2] (used with permission). The generated images look realistic and somewhat similar. However, identity is not preserved.

The loss term used aggregates over all such pairs $\alpha\|G(z) - \mathbf{mirror}(G(z_N))\|^2$, for a weight α and where **mirror** flips the order of the columns of the image.

3. Processing an Existing Image

In order to manipulate an existing image I, one needs to first recover the "underlying" z vector by employing a reconstruction loss. Employing this vector in order to recover I and rotated versions of it, suffers from noticeable reconstruction errors. Most notably, the reconstructed face $G(z)$ does not maintain the identity of the image I, despite their similarity. The same phenomenon is also apparent when observing the approximation results of the most recent generation schemes, such as BEGAN [2] (see Fig. 8).

We therefore, propose to fine-tune G using the same loss, while focusing on the recovered z. By doing so, we obtain an image specific network G_I that is able to generate the input image I, but is no longer as general as G.

We assume a symmetric generator that employs the z' method. Having G_I and z, we are then able to alter the amount of symmetry by modifying z. First, the mirror image is generated by employing z_N (as above). The spectrum in between the image and the mirror image, which provides a virtual yaw effect, is then spanned by interpolating between z and z_N. The results are shown in Fig 13(d). As shown in the experiment, the results are superior to those obtained using the unmodified generator G.

The process of recovering z of G_I is illustrated in Algo. 1. First, we iteratively optimize z to minimize the following term: $argmin_z \|G(z) - I\| + \alpha\|z\|^2 + \beta(|z| - 1)_+$. We found it necessary to employ weight decay on z and also to use a hinge-loss to encourage the values to remain in the range $[-1, 1]$.

In the second phase, we allow G to be optimized as well, creating a version G_I that is tailored to the specific sample I. In order to prevent G_I from becoming degenerate and too specific to the problem of reconstructing I, we alternated between iterations that

Algorithm 1 Recovering the underlying vector z for an input image I and modifying the generator G in order to obtain a version G_I such that $I = G_I(z)$

procedure $[z, G_I] = \text{FitAndTune}(G, I, \alpha, \beta)$
I. Solve the following optimization problem:
  $z \leftarrow argmin_z \|G(z) - I\| + \alpha\|z\|^2 + \beta(|z| - 1)_+$
II. Alternate between the following:
  1. Symmetric GAN training for G_I and D
  2. $z, G_I \leftarrow argmin_{z,G} \|G(z) - I\| + \alpha\|z\|^2 + \beta(|z| - 1)_+$, where G, z are initialized as the current G_I, z

are performed on the training data that was used for training G and between optimizing z and G_I to minimize the reconstruction risk.

4. Experiments

We present empirical evaluation results for both types of structured GANs studied: z' and flip.

4.1. Applying Symmetric GANs to Face Images

For evaluating the symmetric GAN methods, we have compared the following methods:

1. A simple DC-GAN with no symmetric properties.
2. DC-GAN with soft symmetric loss term: $\alpha = 40$.
3. DC-GAN with strong symmetric loss term: $\alpha = 100$.
4. Our Symmetric GAN using the z' Architecture.
5. Our Symmetric GAN using the flip Architecture.

In each method, we generated a series of nine images while attempting to enforce symmetry on it, so that the first image will be a mirror of the last, the second of the one before the last and so on. Since the number of images is odd, the middle image is expected to be symmetric to itself. The way of obtaining the symmetry is determined by the method and follows the description in Fig. 1.

Sample results are presented in Fig. 9. As can be seen, DC-GAN creates high quality images but had no symmetric effect as expected. DC-GAN with soft symmetric loss was not strong enough to enforce symmetry over the generated image. However, on the other hand, it created light deformation to the image caused by unstable training. DC-GAN with a strong symmetric loss was very unstable during training. The generated images were mostly symmetric to themselves and with bad quality. The results of both of our symmetric

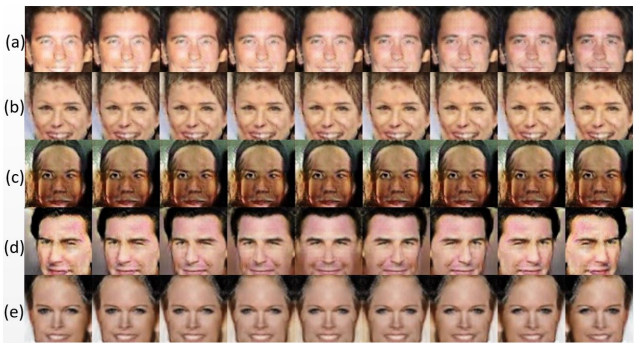

Figure 9: Series of images generated by various GAN methods, trying to obtain a mirroring effect between images on the left and right side of each series. (a) DC-GAN. The first 5 elements of the 100D vector z change, but this has little effect on the output image. (b) DC-GAN with a soft symmetric loss term, using the z' encoding of symmetry.(c) DC-GAN with a strong symmetric loss term and using the z' encoding of symmetry. (d) Symmetric-GAN using the z' architecture.(e) Symmetric-GAN using the flip architecture.

training techniques were much more convincing and presented the desired effect.

We then averaged each generated image with the corresponding image on the other side of the series, after the 2nd image was mirrored. If the mirroring effect is exact, no artifacts are expected. The results can be seen in Fig. 10. This visualization clearly demonstrates that both our methods (z' and flip) create mirror images when the input dictates this.

Finally, we measure the MSE between each image and the mirror version of it. The results are shown in Fig. 7 in the Supplementary. As can be seen, the proposed methods drop to nearly 0 in the middle image, indicating that those images are symmetric. We can see that the MSE of the other methods is relatively constant and does not drop to zero. The loss-based method, with the strong symmetric constraints creates images that are symmetric throughout the range of z' values. An even stronger symmetry loss would lead to an MSE close to zero along the entire curve, with an image that is barely recognizable as a face.

Manipulating a Face Image In order to manipulate a given image I, we have recovered the z vector that best matches the image and then manipulated it, as explained in Sec. 3. Fig. 12 depicts the results obtained by recovering this vector and then generating images from manipulated versions. There are noticeable artifacts. These artifacts are largely reduced when performing the per image tuning of G in order

Figure 10: Same as Fig 9 where symmetry is further evaluated by averaging the image $G(z)$ with the mirror image of $G(z_N)$, where z is manipulated to generate the mirror image, i.e., in each location we present $(G(z) + \textbf{mirror}(G(z_N)))/2$. (a) DC-GAN. (b) DC-GAN with a soft symmetric loss term, z' encoding of symmetry.(c) DC-GAN with a strong symmetric loss term, z' encoding. (d) symmetric-GAN using the z' Architecture. (e) Symmetric-GAN using the flip Architecture.

to obtain G_I, as can be seen in Fig. 13.

4.2. Symmetrical Views of Man-Made Scenes

To show that our method is general, the network was also trained on the LSUN bedrooms dataset [12]. Unlike a face, a bedroom is not symmetric. However, since mirror images of rooms belong to the same class, the method fits this kind of data well.

In our experiments, we focused on the z' architecture. The first experiment shows how a generated image is affected when setting its z' component closer or further away from zero. The results, depicted in Fig. 11, show that the closer to zero, the more symmetric the generated image is, and that the image associated with z' and that associated with $-z'$ are mirror images. The second experiment is similar, with the single change that we fix $z' = 0$ everywhere except for one coordinate at a time that is changed. This way, we can study the effect of each individual dimension on the output. The results are shown in Fig. 14. It is clear that each dimension controls a different mode of variability. However, the dimensions are not independent and the same objects emerge by using different coordinates.

5. Conclusions

DC-GANs are being used today for a wide range of applications, such as domain transfer networks [11], photo editing [3], denoising, data creation and more. We demonstrate how by manipulating the structure of the generator, we can directly control the symmetry of the output. A second, completely different, application to tiling, which is presented in the Supplementary, shows that a similar structure modifying design provides a solution for a completely different application.

Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant ERC CoG 725974).

We are grateful to Barak Itkin for proposing the tiling application.

References

[1] Yael Adini, Yael Moses, and Shimon Ullman. Face recognition: The problem of compensating for changes in illumination direction. *TPAMI*, 19(7):721–732, 1997.

[2] David Berthelot, Tom Schumm, and Luke Metz. BEGAN: boundary equilibrium generative adversarial networks. *arXiv preprint arxiv:1703.10717*, 2017.

[3] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Neural photo editing with introspective adversarial networks. *arXiv preprint arXiv:1609.07093*, 2016.

[4] Roberto Brunelli and Tomaso Poggio. Face recognition: Features versus templates. *TPAMI*, 15(10):1042–1052, 1993.

[5] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*, 2016.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[7] Guillaume Lample, Neil Zeghidour, Nicolas Usunier, Antoine Bordes, Ludovic Denoyer, and Marc'Aurelio Ranzato. Fader networks: Manipulating images by sliding attributes. *arXiv preprint*, 1706.00409, 2017.

[8] Shaoheng Liang, Jiansheng Chen, Zhengqin Li, and Gaocheng Bai. Linear time symmetric axis

Figure 11: Using the z' architecture, z' is multiplied by a scalar, shifted from [-0.5,+0.5] with leaps of 0.1 (left to right). For each row, z'' is held constant. Each row is a different experiment, producing 11 images.

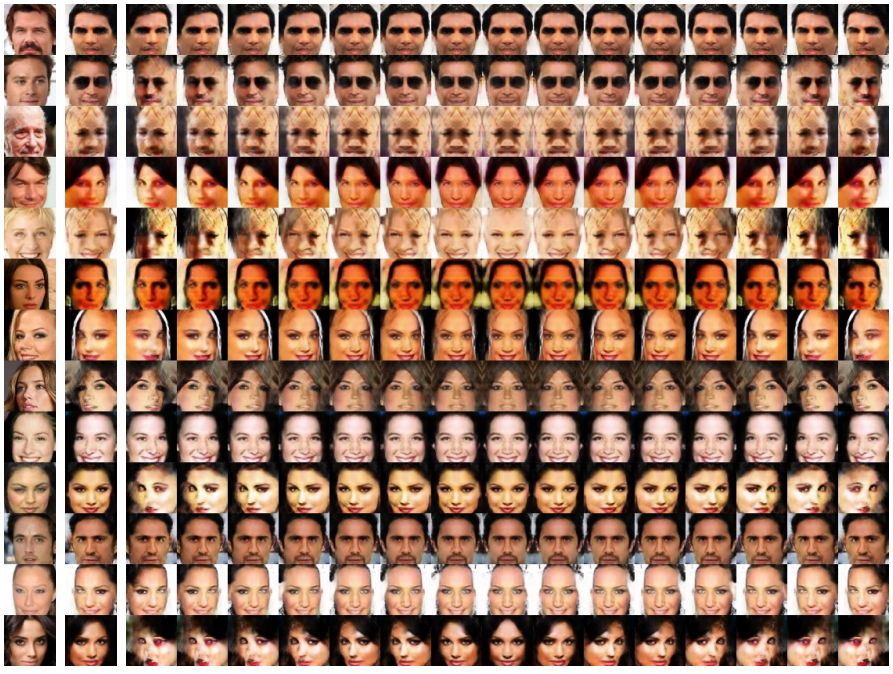

Figure 12: The results obtained without G transfer. (a) Sample dataset images I. (b) $G(z)$ for the vector z that was optimized to minimize the reconstruction loss. Results are shown for a symmetric generator that is based on the z' architecture. (c) Rotations performed using G and manipulated versions of z.

Figure 13: (a) Dataset samples I. (b) $G(z)$ for the vector z that was optimized to minimize the reconstruction loss. Results are shown for a symmetric generator that is based on the z' architecture. (c) $G_I(z)$, where G_I is finetuned from G in order to further minimize the reconstruction error. (d) Rotations performed using G_I and manipulated versions of z.

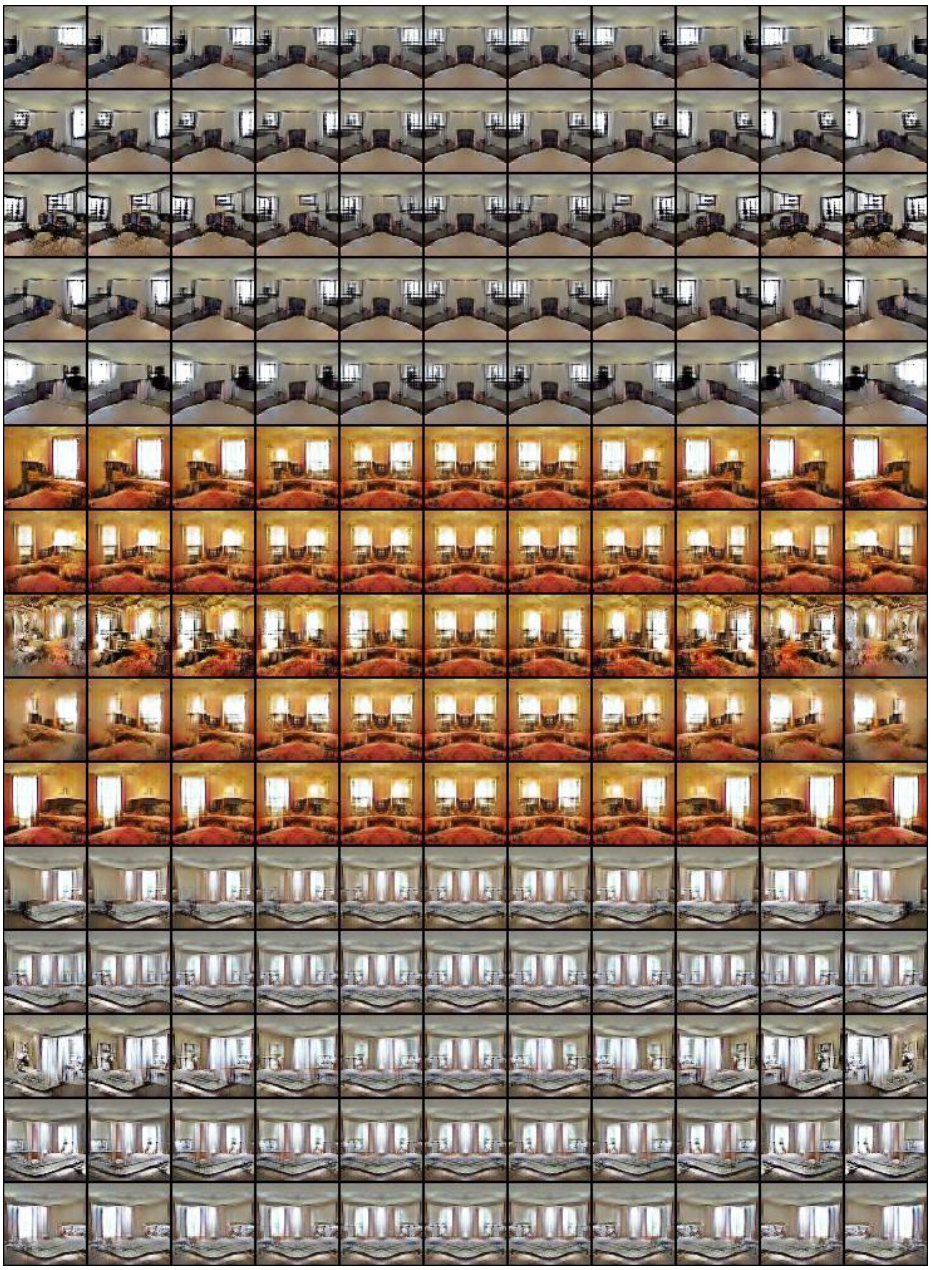

Figure 14: Using the z' architecture, all five z' fields, except one, are set to zero. The remaining field is shifted from [-1,+1] with leaps of 0.2 (left to right). In each row, a different field is changed. All 95 z'' fields are unchanged during the experiment. Each experiment produce 5x11 images. The following experiment has been repeated three times.

search based on palindrome detection. In *ICIP*, pages 1799–1803. IEEE, 2016.

[9] Alex Pentland, Baback Moghaddam, Thad Starner, et al. View-based and modular eigenspaces for face recognition. In *CVPR*, volume 94, pages 84–91, 1994.

[10] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint*, 1511.06434, 2015.

[11] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. *arXiv preprint arXiv:1611.02200*, 2016.

[12] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.