Conditional image generation based on deep convolution neural networks

Kristina Milošević

September 5, 2024

Contents

1 Introduction

Generative adversarial networks, GANs, solve the problem of generating realistic data that resembles a given dataset. They aim to create new, synthetic data that is indistinguishable from the actual data, enabling applications such as image generation, data augmentation [5], style transfer [6], recovering features of images [4] and so on. In this work, we will focus on image generation using GAN based frameworks. The idea of GAN architecture is to combine two neural networks that work together in an adversarial process in order to generate realistic data. Though originally intended as unsupervised learning, some variations of GANs are considered semi-supervised.
We start off with the DCGAN architecture presented in [3]. The output of this model is an artificial image, however, we can't control the specific traits of the output. For example, if we wanted to generate images of an animal, we wouldn't be able to predict which species the result would be. In order to guide the process of producing data with specific traits, we need to expand the model. This is achieved by conditioning the input resulting in generation of specific outputs based on given labels.

2 Related work

The Generative Adversarial Network (GAN) was an initial breakthrough in tackling the challenge of producing realistic data. It introduced a generative framework based on an adversarial process, where two models are trained simultaneously: a generative model (G) that captures the data distribution, and a discriminative model (D) that estimates the probability that a given sample comes from the training data rather than from G.

The generator starts with a random noise input and, over time, learns to generate data that closely mimics the real data distribution. The discriminator, on the other hand, outputs a single scalar representing its prediction of whether the input data is real or produced. It is trained to maximize the accuracy of classifying both real and generated data correctly. Concurrently, the generator is trained to minimize the discriminator's accuracy by generating data as realistic as possible. The ultimate goal of GANs is to have the generator produce data so realistic that the discriminator can no longer reliably distinguish between real and fake data.

Deep Convolutional Generative Adversarial Networks (DCGAN) expand on the idea of GANs by integrating convolutional neural networks (CNNs) into the architecture, enhancing the generation of realistic images while maintaining the unsupervised learning framework. DCGANs are primarily used for image generation. The use of convolutional layers in DCGANs is particularly well-suited for processing and producing images, as these layers can effectively capture the spatial hierarchies and patterns present in image data. This allows DCGAN to generate more detailed and higher-resolution images compared to the original GAN, which relied on fully connected layers.

Conditional GANs [2] enhance GANs by generating data that is more controlled and specific, aligning with the given conditions. By incorporating conditional information, CGANs can produce targeted outputs, such as images corresponding to specific classes, which improves the model's ability to generate relevant and contextually accurate data compared to the original GAN.

3 Methodology

To implement the conditional image generation architecture, we integrated concepts from both DCGAN and CGAN, which build upon the foundational GAN framework described in [1].

3.1 The architecture

In order to incorporate the additional input, an embedding layer is required in both the discriminator and the generator. The embedding vectors are concatenated with the noise vector (in the generator) or the input data (in the discriminator). This allows the network to condition the generation process on the specific label information.

The architecture of both the models relies on design principles used in [3]. The discriminator is implemented as a convolutional neural network that differentiates between real and fake images. The label is passed through the embedding layer and reshaped to match the image dimensions. The embedding is then concatenated with the input image along the channel dimension. The concatenated input (image + embedding) is then passed through several convolutional layers with LeakyReLU activations and batch normalization. The final output, processed by a sigmoid function, indicates the probability of the images being real.

The Generator class creates synthetic images based on an input vector and desired class labels. The label is passed through the embedding layer and reshaped to match the input dimensions. It is then concatenated with the input noise vector along the channel dimension, forming a conditioned input for the Generator. This combined input is then processed through a series of transposed convolutional layers, each followed by batch normalization and ReLU activation, to progressively build up the image. The final output layer uses a Tanh activation function to generate images with pixel values in the range [-1, 1].

3.2 Training the model

The models' training process is based on the principles outlined in the original GAN paper. Let's start by discussing the GAN training process and then expand it with additional concepts. The training process of a GAN involves a min-max game between the models. To learn the distribution p_g over the data $\mathbf{x}$, the generator takes in noise from a random distribution $p_z(z)$ and produces data, $G(z;\theta_g)$, trying to emulate the training data. The discriminator outputs a single number representing the probability that $\mathbf{x}$ came from the real training data rather than from the generator.

We train the discriminator D to maximize its ability to correctly label both real data and generated samples. Simultaneously, we train the generator G to minimize the probability that the discriminator can identify its fake samples, specifically aiming to reduce $\log(1 - D(G(z)))$. Mathematically, this is formulated as a min-max optimization problem with the value function

$$\min_G \max_D V(D,G) = E_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + E_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{1}$$

In order to define the training process in our project, we expand the previous formula using the idea presented in the conditional GAN paper, [2], by simply taking into

account the additional information being fed to the models. The generator combines the input noise $p_z(z)$ and label y. In the discriminator the data and the labels are presented as combined input as well. The objective function of a two-player minimax game would then look like the following:

$$\min_G \max_D V(D,G) = E_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x},\mathbf{y})] + E_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z},\mathbf{y})))]. \quad (2)$$

The loss function used in this project that corresponds to this objective function is the binary Cross Entropy loss.

The training loop iteratively improves the generator and discriminator through an adversarial process. The discriminator firstly tries to minimize the difference between its predictions and the real images. After the generator has produced a fake image, the discriminator tries to learn to recognize the fake images. Whilst training, the generator learns to produce images that can fool the discriminator by trying to minimize the difference between the real images and the artificial ones. When calculating the generator loss, we also take into account that the generator should be penalized if its generated images deviate too much from the real images. The following pseudocode provides an overview of the training process for a CDCGAN:

Algorithm 1 CDCGAN Training Loop

```
for each epoch do
    for each batch (images, labels) do
        predicted_labels_real = D(images, labels)
        discriminator_real_loss = BCE([1, 1, ... 1], predicted_labels_real)

        fake_images = G(noise, labels)
        predicted_labels_fake = D(fake_images, labels)
        discriminator_fake_loss = BCE([0, 0, ... 0], predicted_labels_fake)
        discriminator_loss = discriminator_real_loss + discriminator_fake_loss
        discriminator_loss.backward_and_update()

        predicted_labels_fake = D(fake_images, labels)
        generator_loss = BCE([1, 1, ... 1], predicted_labels_fake)
              +λ · L1(images, fake_images)
        generator_loss.backward_and_update()
```

4 Experiments

We tested the CDCGAN on the MNIST, Animal-faces, and CelebA datasets. The CDCGAN can work with both labeled and unlabeled data. Below is a table of the parameters used for each dataset.

Parameter	MNIST	Animal-faces	Celeba
Number of Classes	10	3	1
Embedding Dimension	5	5	1
Number of Channels	1	3	3
Input Vector Dimension	20	100	100
Reconstruction Weight	5	5	0
Batch Size	256	64	256
Feature Map Dimension	64	96	64

Table 1: Parameters for each dataset

For the MNIST dataset, the transformations included resizing the images to 64x64 pixels, applying a random rotation of up to 30 degrees, and performing random cropping with a padding of 4 pixels. These changes help to augment the data, providing more diverse training examples and improving the model's generalization. For both the Animal-faces and Celeba dataset, the transformations involved resizing the images to a fixed size, center cropping to ensure consistent input dimensions, converting the images to tensors for model processing, and normalizing them with a mean and standard deviation of 0.5 for each channel. These steps ensure uniformity in the data and stabilize the training process by maintaining consistent input scales.

The images displayed below represent the best results from qualitative analysis.

The results on the MNIST dataset demonstrate that the training produces a rich set of images that match their predefined labels.

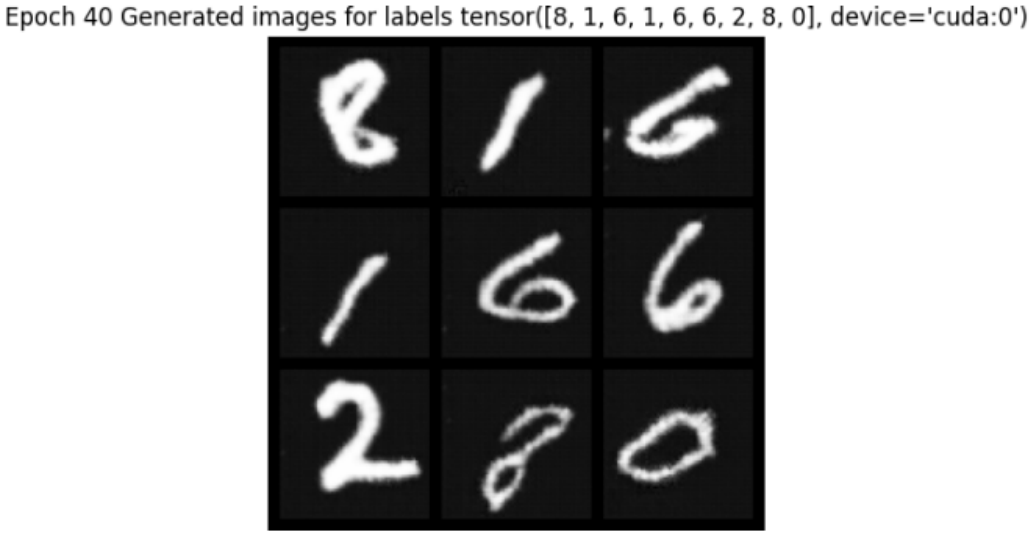

Figure 1: Generated MNIST digits after 40 epochs

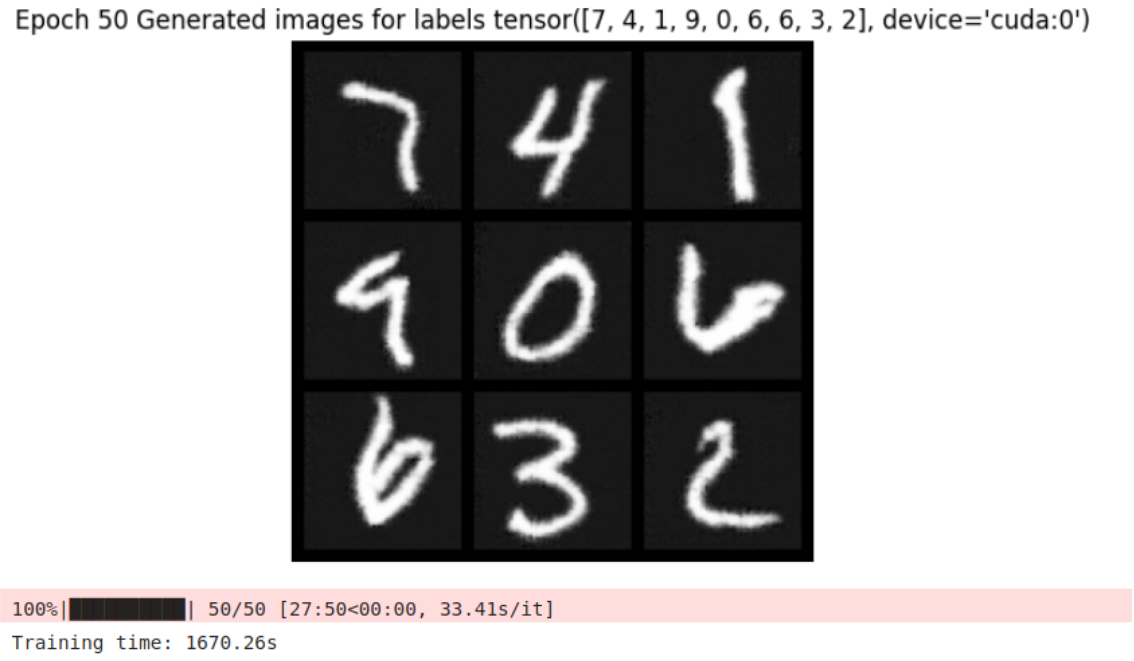

Figure 2: Generated MNIST digits after 50 epochs

The following image displays the graph of both the discriminator and generator loss across epochs.

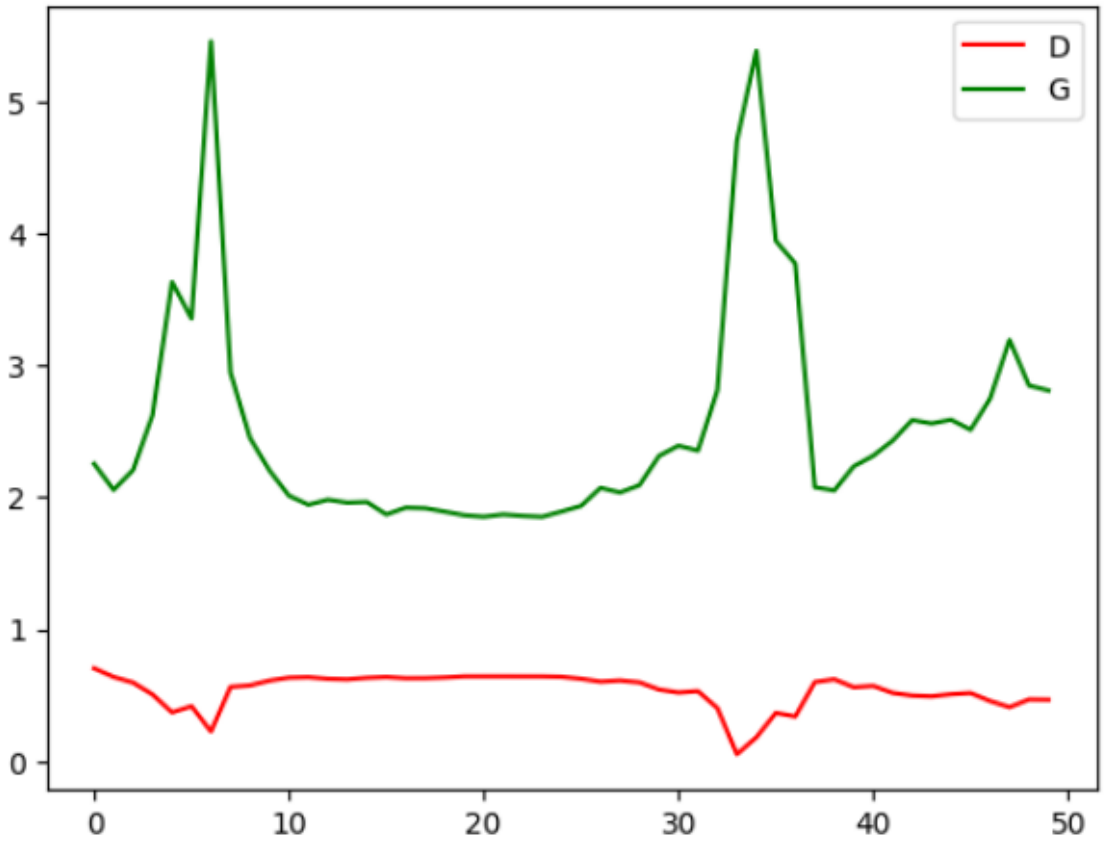

Figure 3: Generator and discriminator loss for MNIST digits through epochs

The discriminator loss converges to around 0.6 in terms of cross entropy(i.e. around 0.5 probability), indicating that it has reached a point where it is making random guesses between real and generated images, which is the intended outcome of the training process.

Regarding the Animal-faces dataset, it would take longer to achieve realistic results, but the images remain diverse and accurately correspond to their labels.

Figure 4: Generated Animal-faces images after 47 epochs

Figure 5: Generated Animal-faces images after 74 epochs

The graph below shows the change in loss for both the discriminator and the generator.

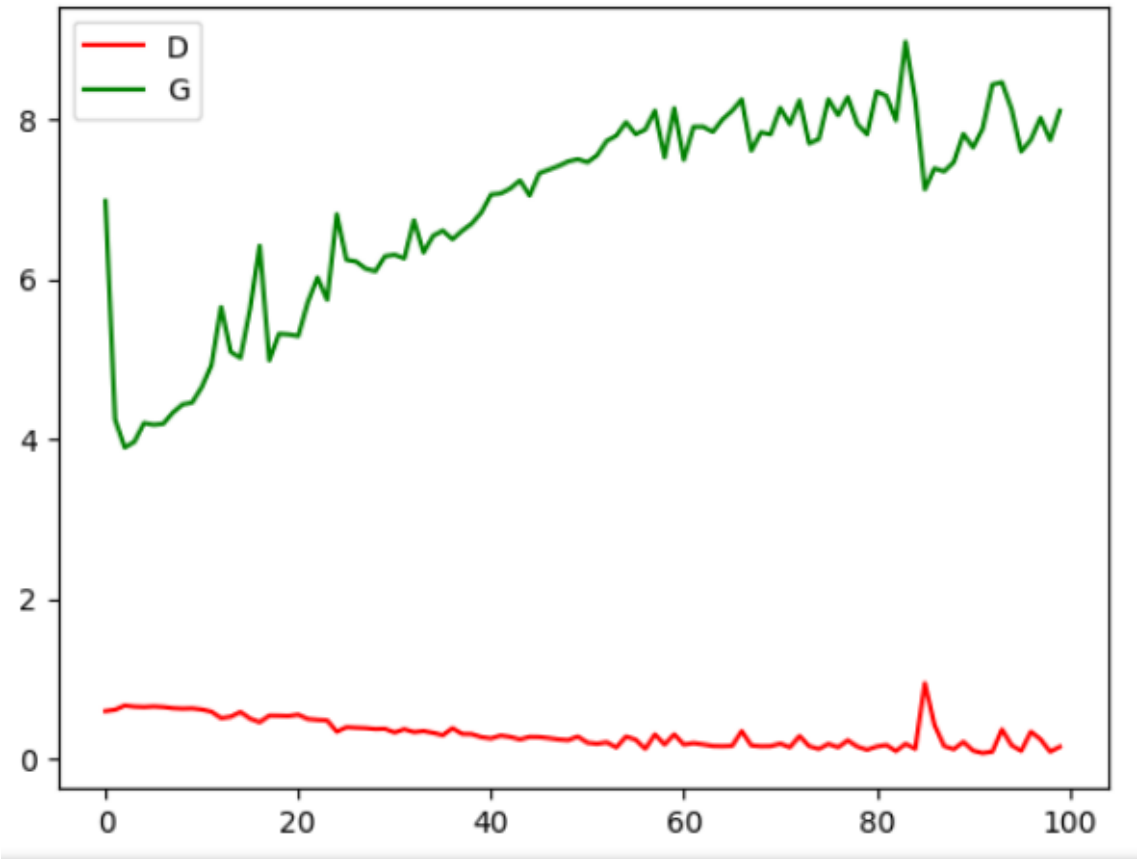

Figure 6: Animal-faces dataset : change in loss through the epochs

The images below show the results of the model on the Celeba dataset. It would take longer for proper training due to the size of the dataset. At the end of the training the loss grows exponentially — we speculate that is due to exploding gradients.

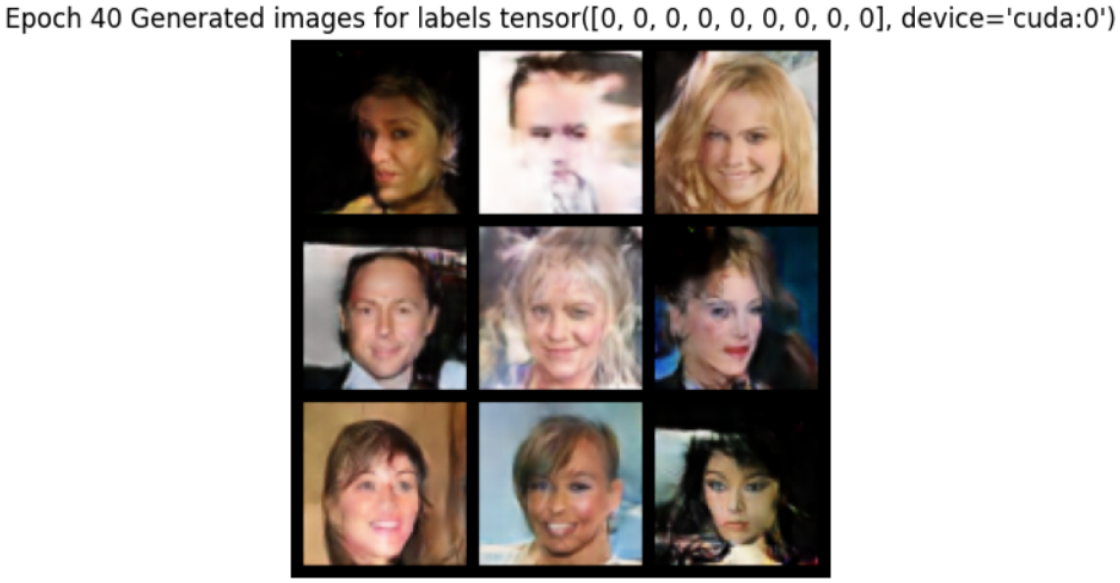

Figure 7: Generated Celeba images after 40 epochs

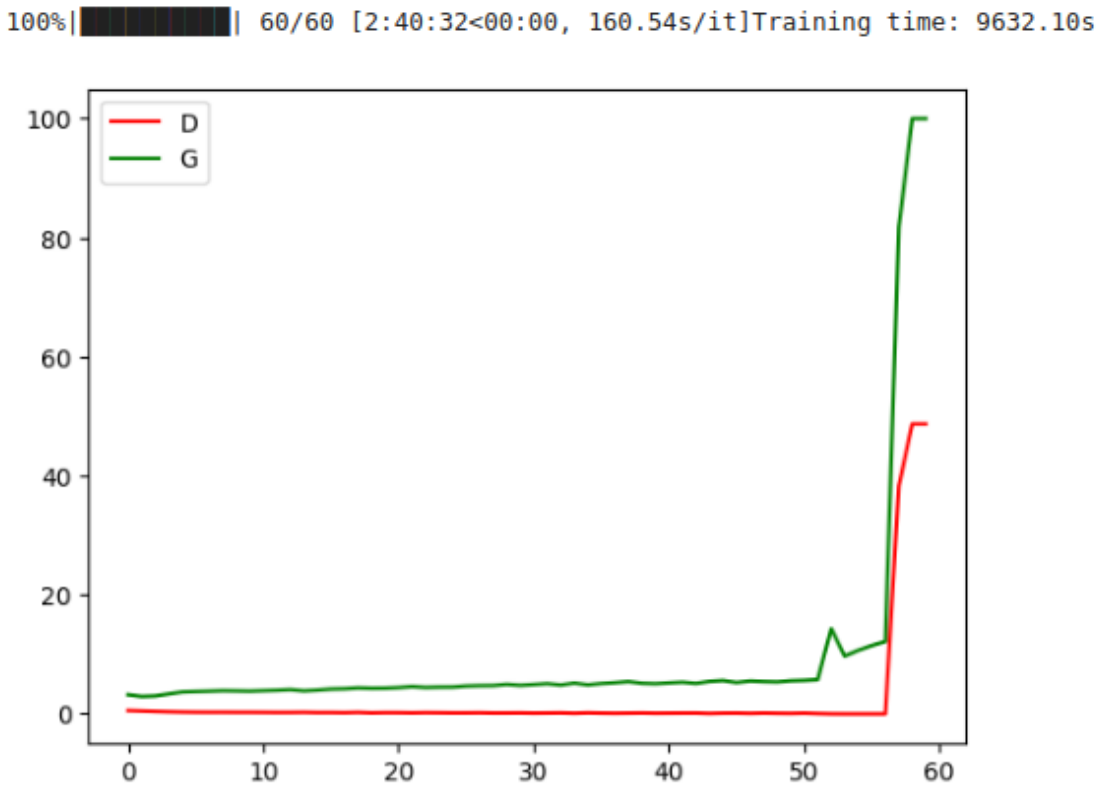

Figure 8: Celeba dataset: change in loss through epochs

5 Conclusions

The CDCGAN architecture effectively generates realistic and diverse images, successfully modeling complex data distributions. It demonstrates versatility across different datasets, from simple ones like MNIST to more complex ones like Animal-faces and Celeba. Additionally, the use of label conditioning allows for controlled image generation, ensuring that the outputs are both realistic and aligned with the intended labels.

References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[2] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

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

[4] Kevin Schawinski, Ce Zhang, Hantian Zhang, Lucas Fowler, and Gokula Krishnan Santhanam. Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit. *Monthly Notices of the Royal Astronomical Society: Letters*, 467(1):L110–L114, 01 2017.

[5] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, and Xiaoou Tang. ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks. *arXiv e-prints*, page arXiv:1809.00219, September 2018.

[6] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.