The Thesis Committee for Tong Zhou
Certifies that this is the approved version of the following thesis:

A Study of Generative Adversarial Networks and Possible Extensions of GANs

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

____
Georgios-Alex Dimakis

____
Constantine Caramanis

A Study of Generative Adversarial Networks and Possible Extensions of GANs

by

Tong Zhou

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

August 2017

Acknowledgements

I express my sincere thanks to my respected supervisor, Dr. Georgios-Alex Dimakis for his valuable advice and guidance. I also take this opportunity to express my gratefulness towards my guide, Dr. Murat Kocaoglu, whose expert advice made me think upon and understand a number of problems. I would like to thank Dr. Constantine Caramanis to be my reader of this thesis. His valuable comments on this report guide me in the right direction. I am thankful to all other faculty and staff members of Electrical and Computer Engineering Department for their kind support and help. Last but not the least, I would like to thank my family and friends, for their unconditional encouragement and support towards the completion. Thank you.

Abstract

A Study of Generative Adversarial Networks and Possible Extensions of GANs

Tong Zhou, MSE

The University of Texas at Austin, 2017

Supervisor: Georgios-Alex Dimakis

The goal of our research is to explore the power of generative adversarial networks (GANs). We take a review of deep learning and many extended versions of GANs. We implement and empirically evaluate Deep Convolutional GANs. As a contribution, we propose an extension of GANs called Extractor GANs (EGANs). It contains a feature extractor to get extractable attributes of generated images. We first use the extractor to get some low-level features, such as brightness and sharpness. EGANs can determine the brightness and sharpness of the generated images with the help of the extractable attributes. In order to control more complicated features of face images, we consider to use convolutional neural networks (CNN) as the feature extractor. We use two ways, pre-training and without pre-training, to train CNN so we can have a well-performed feature extractor. The results show that EGANs perform well in controlling some high-level features of face images, such as gender and openness of mouth.

Table of Contents

List of Figures

Chapter 1: Introduction

The human brain neural network is a very complex nervous system. There are around 100 billion neurons in the brain of an adult. An artificial neural network (ANN) is a computational model based on the structure of biological neural networks. It mimics the way information is processed in the human brain. Typically, ANNs have an input layer, hidden layers and an output layer. In each layer, there are a set of connected units called artificial neurons.

Deep networks refer to artificial neural networks that are composed of many layers. The earliest deep learning architecture is composed of multiple layers of non-linear features and it has polynomial activation functions. It is proposed by Ivaknenko and Lapa in 1965 [28]. In 1979, Fukushima [29] proposed the Convolutional Neural Networks (CNN), which has multiple pooling and convolutional layers. CNN has the advantage of avoiding the hand-selected feature extraction, which is time-consuming and requires expert knowledge, compared to the traditional image processing algorithms. Neural networks can be trained by seeking a set of parameters for the network that minimize a loss function. Gradient descent (GD) is an optimization procedure to move along the steepest descent direction of the loss function. GD is used for the optimization of neural network loss functions. In the late 1980s [9], it was observed that calculating the derivatives for gradient descent systematically using chain rule, which is called back propagation in this context, can greatly speed up the neural networks training. During this period, training of deep network models still suffered from the vanishing gradient problem, where features in early layers could not be learned or are learned much slower than later layers by using gradient based learning methods. In 1992, Schmidhuber [47] developed the pre-training approach to solve the vanishing gradient problem in the training of deep networks. Graphics processing units

(GPUs) have parallel processing architectures with thousands of cores. Many training steps in deep neural networks can be parallelized and run on the GPUs [42, 48, 49, 50]. In 2012 [10], Krizheysky, Sutskever and Hinton introduced rectified linear activation functions (ReLU, $\max(0, x)$) in convolutional network architecture. Dropout method was used for regularization. It removed units from neural networks to improve generalization. Their work received outstanding results in the ILSVRC-2012 ImageNet competition [30]. In 2016, Google beat the top human Go player using deep neural network technology, which makes deep learning receive even more attention.

Generative Models are models that learn the natural features of a dataset and generate new samples which are indistinguishable from the existing data. They are trained using unsupervised training. The generative algorithm learns the joint probability distribution over the observations from data. Maximizing the likelihood is a common method to fit the parameters of the generative model [56]. Popular generative models include Naïve Bayes, Deep Belief Networks, Restricted Boltzmann Machines, Variational Auto-Encoder (VAE).

Generative Adversarial Networks (GAN) are one of popular generative models. GANs were introduced by Goodfellow et al. [4] in 2014. The working principle of GAN is rooted in game theory. At a high level, a GAN works by staging a battle between Generator and Discriminator networks. The task of the discriminator is to discriminate between the data from the original dataset and the data generated by the generator. The task of the generator is to fool the discriminator to think that the generated data is real. GANs have been shown to successfully approximate high dimensional data.

Related Work

There are many variants of GANs that were proposed in the last few years. Conditional GANs (cGANs) [11] are relevant extensions of GANs, where the generator and discriminator are conditioned on some external information like class labels. Deep Convolutional GAN, introduced by Radford et al. in 2015 [6], built GAN architecture using convolutional neural network components. [17] develops a novel deep architecture and GAN formulation to generate plausible images from detailed text descriptions. Bidirectional Generative Adversarial Networks (BiGANs) [14] can project data back into the latent space. The feature representation learned by BiGANs is useful for auxiliary supervised discrimination tasks. InfoGAN [13] is an information-theoretic extension of GAN. It maximizes the mutual information between a small subset of the latent variables and the observation. InfoGAN allows learning of meaningful representations, which are competitive with representations learned by some supervised methods. Christian et al. proposed SRGAN [15], which is a generative adversarial network to recover photo-realistic textures from down-sampled images. [19] uses a generative adversarial neural network to make interactive edits to the generated images and operate realistic image manipulation in shape and color of images. [22] proposed a sequence generative framework, called SeqGAN, to directly generate sequential synthetic data, and achieved significant performance in speech generation and music generation. [20] introduced the Introspective Adversarial Network to achieve accurate reconstructions without loss of feature quality, and improved generalization performance with Orthogonal Regularization. [23] pointed out the connection between GAN and Actor-Critic Methods to inspire the development of the multilevel optimization algorithms with deep networks. Invertible cGAN (IcGAN) [12] combines an encoder with a cGAN. The encoder in IcGAN is used to map a real image into a latent space and a conditional representation. IcGAN allows to reconstruct and modify real images of faces conditioning on arbitrary attributes. [18] investigates conditional adversarial networks as a general-purpose solution to image-to-image translation problems. SimGAN [16], proposed by Shrivastava Ashish et al. in 2016, uses self-regularization in adversarial loss, adds local adversarial loss, and updates discriminator using a history of refined images, which enables generation of highly realistic images. [21] applied generative adversarial network approach to generate human-like dialogue. [52] performs compressed sensing using VAEs and GANs with fewer measurements than normal compressed sensing implementations. In [51], Berthelot et al. proposes a method to measure convergence. This method can help balance adversarial networks training, and make networks converge to diverse and visually pleasing images.

Contributions

The objective of this thesis is to further explore the use of generative adversarial networks in image generation. We extend the framework of GAN so that we can control the attributes of the generated images.

- In the Chapter 2, we summarize the theory of GANs and Deep Convolutional GANs (DCGAN). We redo the simulation of the DCGAN on Cifar-10 dataset and verify the performance.
- In the Chapter 3, we extend the framework of DCGAN by adding feature extractor to learn the brightness and sharpness features of the image data. We can control the brightness and sharpness of the generated images using this extended framework.
- In the Chapter 4, we extend the DCGAN framework by adding convolutional neural networks (CNN) to extract high-level features, such as gender and openness of mouth features. We use this method to learn the

high-level features, in order to control the corresponding features of the generated images.

Chapter 2: Generative Adversarial Networks

2.1 Generative Adversarial Networks Architectures

In 2014, Goodfellow et al. proposed the Generative Adversarial Nets (GAN) [4], which is a generative modeling architecture based on differentiable generator network. GAN is used to generate new samples by learning from the existing data. GANs have a generator network G and a discriminator network D. The generator network directly produces samples. The discriminator network D attempts to distinguish between samples drawn from the training data and samples drawn from the generator. See Figure 1 for a visualization of the model architecture.

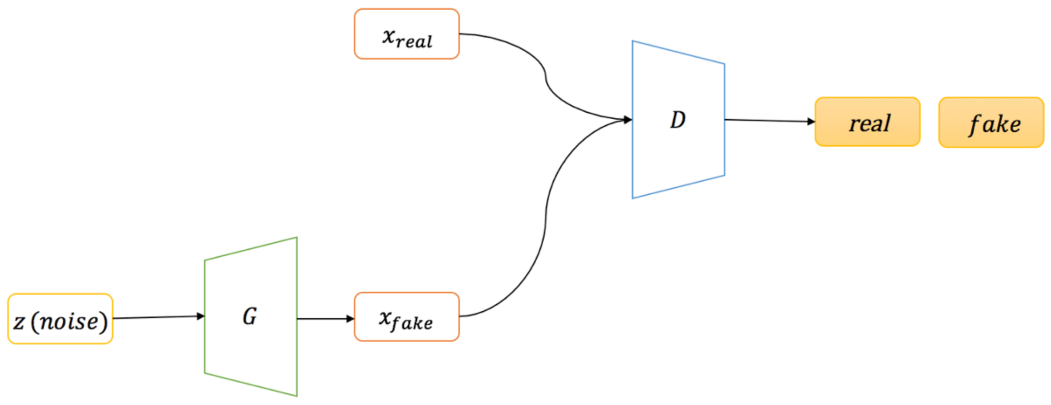


Figure 1: The architecture of GANs

Generator G takes a noise vector z as input and generates samples x_{fake}. Discriminator D receives samples from both the generator and the training data x_{real}. On a high level, generator and discriminator play a game, where generator is learning to produce more and more realistic samples, and the discriminator is learning to get better and better at distinguishing generated data from real data. Generator and discriminator are trained alternatingly, which means a gradient step firstly is taken for generator, then a step is taken for discriminator.

Deep Convolutional GANs (DCGANs) [6] are made of standard convolutional neural network components, such as convolutional layers and transposed convolutional layers. In the next section, we introduce the convolutional neural network components.

2.2 Convolutional Neural Network Components

Convolutional Neural Networks (CNN, or ConvNets) are made of neurons that have trainable weights and biases. Each neuron receives some input, performs a dot product and follows it with a non-linearity. CNN is trained by minimizing a loss function, which is defined based on the output from the last fully-connected layer. As an explicit assumption, the inputs of CNN architectures are images. CNN architectures are built using three main types of layers: Convolutional Layer, Pooling Layer, and Fully-Connected Layer, as illustrated in Figure 2. All layers are explained in the coming subsections.

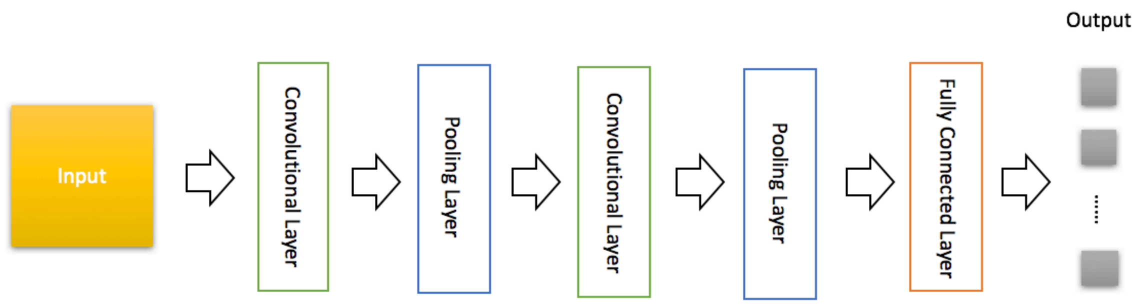


Figure 2: Convolutional neural networks architecture

2.2.1 Convolutional Layers

Convolutional neural nets do the discrete convolution computation in the convolutional layer. A discrete convolution is a linear transformation. The discrete convolution is sparse, since only few input units contribute to one output unit. The discrete convolution reuses parameters, which means the same weights are applied to multiple locations at the input. When the input is an image, the discrete convolution applies the same filter on many locations of the image.

Figure 3 provides an example of a discrete convolution. Every image is a matrix of pixel values. The 4×4 light blue grid is the input image. The dark blue grid in Figure 3 is the feature map, which is also called the kernel. Kernel slides across the input image. At each position, we multiply each element of the kernel with the input element it aligns with. The results are summed up to obtain the output in the current position. The summation is often offset by a bias. The green grid is the output feature map, which is the final output of discrete convolution calculation. In Figure 3, we can see the upper left corner 21.0 in the output feature map is coming from $1 * 1 + 2 * 2 + 3 * 2 + 2 * 0 + 0 * 0 + 2 * 1 + 2 * 2 + 1 * 2 + 2 * 1$. More details about discrete convolution can be found in [46].

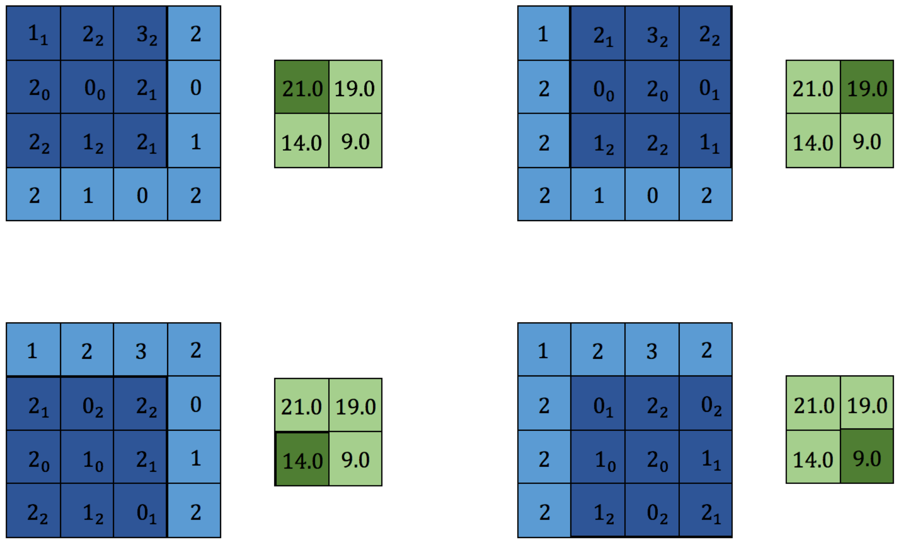


Figure 3: Compute the output values of a discrete convolution. The light blue grid is the input feature map. The dark blue grid is the kernel. The green grid is the output feature map.

2.2.2 POOLING LAYERS

Pooling layers contain the operations to reduce the size of the feature maps. It does this by taking the average or the maximum value to summarize the sub-regions of feature maps.

- Max-Pooling: use the largest element in the pooling window as the sampling value.
- Average-Pooing: use the average of all the elements in the pooling window as the sampling value.

Pooling works by sliding a window across the input feature maps. It uses pooling functions, such as Max-Pooling and Average-Pooling, to calculate the sampling value of the elements in the pooling window. Figure 4 provides an example for average pooling, and Figure 5 does the same for max pooling. More details about pooling operation are given in [46].

2.2.3 FULLY-CONNECTED LAYERS

Neurons in a fully-connected layer have full connections to all activations in the previous layer, which means every neuron in the previous layer is connected to every neuron on the next layer. The output from the convolutional and pooling layers represents high-level features of the input image. In a classification problem, the fully connected layer can use these features to classify the input image into various classes.

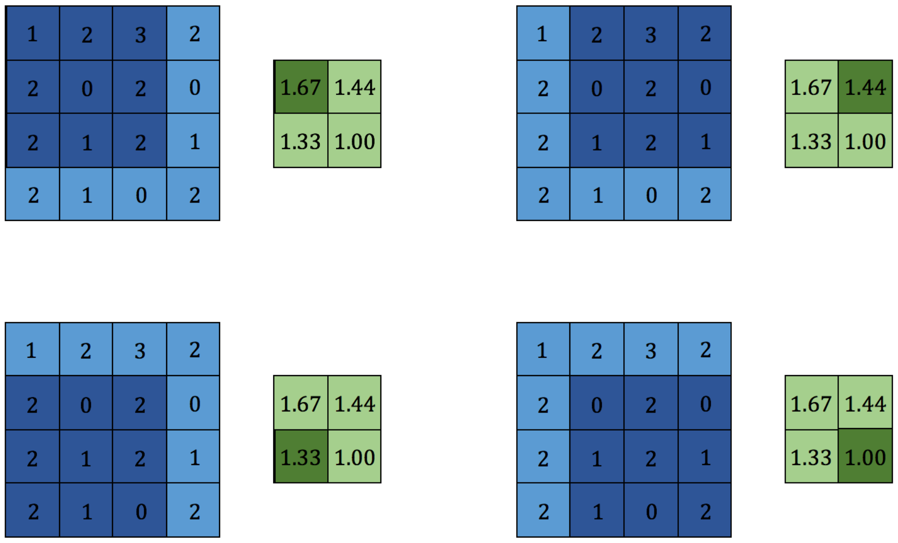


Figure 4: Compute the output values of a 3×3 average pooling operation on a 4×4 input.

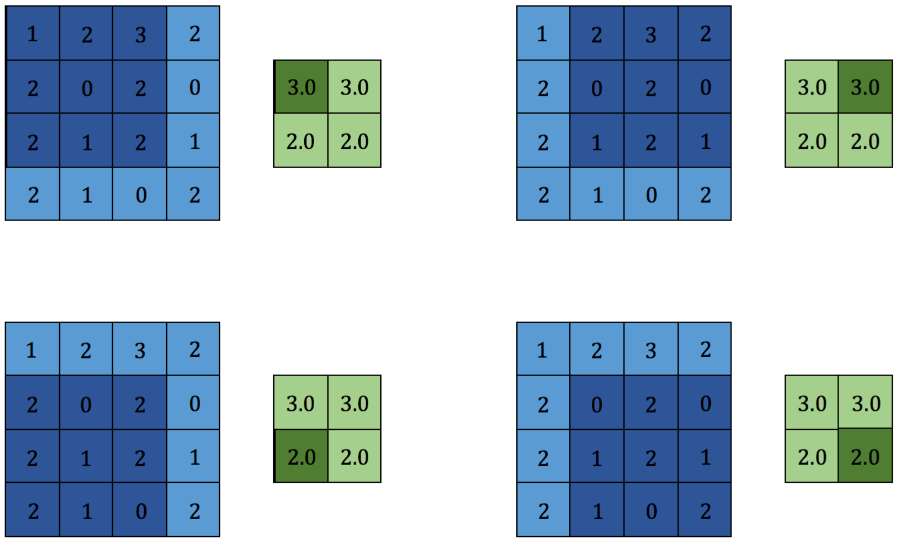


Figure 5: Compute the output values of a 3×3 max pooling operation on a 4×4 input.

2.2.4 Activation Function

In order to make neural networks have the capacity of approximating any functions, the activation functions are used to introduce non-linearity into the network. There are four main activation functions, Sigmoid, Tanh, ReLU and Leaky ReLU. Sigmoid and Tanh are often used in the fully-connected layer. ReLU and Leaky ReLU are often used in the convolutional layer.

2.2.4.1 Sigmoid

A sigmoid function produces a curve with an "S" shape. The sigmoid non-linearity is defined as

$$f(x) = \frac{1}{1 + e^{-x}}$$

The output of sigmoid falls in the interval of $(0, 1)$. The gradient of sigmoid at either tail is almost zero. The neural network using sigmoid as the activation function suffers from the vanishing gradient problem when we use gradient-based training. In the vanishing gradient problem, the gradient value in the front layers of networks decreases greatly, which makes the preceding layer train very slowly.

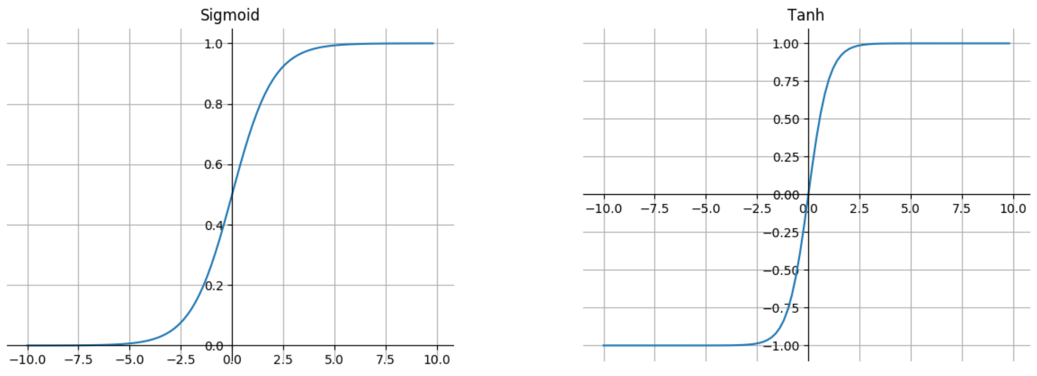


Figure 6: (left) Sigmoid. (right) Tanh

2.2.4.2 Hyperbolic Tangent

The hyperbolic tangent (TanH) non-linearity is defined by the formula

$$f(x) = tanh\ (x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The $tanh\ (x)$ function can be viewed as a rescaled version of the sigmoid, and its output falls in the interval of $(-1, 1)$, as illustrated in Figure 1.

2.2.4.3 Rectified Linear Unit

The Rectified Linear Unit (ReLU) is defined as

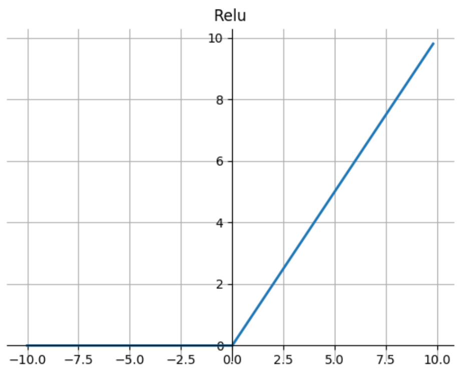
$$y = max\ (0, x)$$

The ReLU has become very popular in the last few years, because it was found to greatly shorten the learning cycle due to its linear non-saturating form. The use of ReLU introduces the sparsity [53]. In [26], it shows that ReLU has faster training speed compared to Sigmoid and Tanh. However, the ReLU units can “die” during training, where ReLU always outputs the same value for any input.

2.2.4.4 Leaky ReLU

Leaky ReLUs are one attempt to fix the dying ReLU problem. Instead of the function being zero when $x < 0$, a Leaky ReLU have a small positive gradient for negative inputs. Leaky ReLU has the following mathematical form

$$y = \begin{cases} 0.01 * x, when\ x < 0 \\ 1, when\ x \geq 0 \end{cases}$$



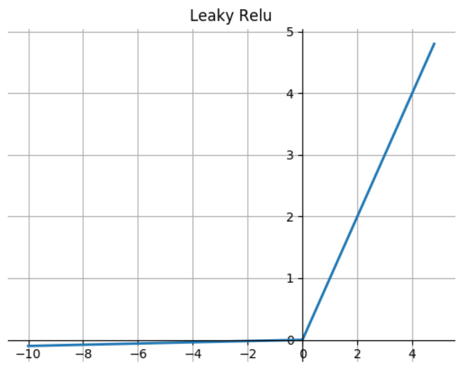


Figure 7: (left) ReLU. (right) Leaky ReLU

2.2.5 Batch Normalization

Batch normalization [25] potentially helps neural network to converge faster and get higher overall accuracy. Batch Normalization shifts inputs to zero-mean and unit variance, which makes the inputs of each trainable layer comparable across features. It allows singular functions such as TanH and Sigmoid to not get stuck in the saturation mode, where the gradient is almost 0. In practice, networks that use batch normalization are more robust to bad initialization [31].

2.2.6 Loss Functions

A loss function is minimized as the objective to measure the compatibility between a prediction and the ground truth label.

2.2.6.1 Quadratic cost

Quadratic cost is also known as mean squared error. It is defined by the formula

$$Loss(x, y) = \frac{1}{n}\sum_{i=1}^{n} |x_i - y_i|^2$$

where x represents n predictions of neural networks, and y represents the real classes of the input images.

2.2.6.2 Cross Entropy

The cross entropy of a distribution P with respect to a distribution Q measures how many bits are needed on average to encode data from P with the code that is the optimal for Q. It is defined as

$$Loss(x, y) = -\sum_{i=1}^{n} y_i * \log \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}$$

where x is a vector of n predictions of neural networks, and y is a binary vector full of 0s besides 1s representing the real classes of the input images.

2.2.7 Transposed Convolutional Layer

The concept of Transposed Convolution (also called Deconvolution) is first shown in the Zeiler's paper [24] published in 2010. The need for transposed convolutions generally arises from the desire to use a transformation going in the opposite direction of a normal convolution. Transposed convolutions work by swapping the forward and backward passes of a convolution. The transposed convolution is adopted by many works, including scene segmentation [35] and generative models [15, 34].

The deconvolution networks [32] conduct two main operations, un-pooling and deconvolution. The un-pooling operation in the deconvolution network performs the reverse operation of pooling and reconstructs the activations of the original size. The deconvolution takes in a single input and generates multiple outputs. The output of the deconvolution operation is an enlarged and dense activation map. An implementation of un-pooling operation can be found in [33].

2.3 Deep Convolutional Generative Adversarial Networks

DCGANs build the generator G with convolutional network and the discriminator D with the transposed convolutional network. The entire model can be trained with backpropagation. We train discriminator D to maximize the probability of determining correctly if a sample belongs to the training data distribution. The discriminator D and the generator G play the following two-player minimax game with value function $V(G, D)$:

$$\min_{G} \max_{D} V(D,G) = E_{x \sim P_{data}(x)}[log D(x)] + E_{z \sim P_{z}(z)}[log\ (1 - D(G(z)))]$$

We train discriminator to maximize $log\ (D(x))$ and $log\ (1 - D(G(z)))$, where x is sampled from existing dataset and noise vector z is the input to the generator G. We simultaneously train G to minimize $log\ (1 - D(G(z)))$.

In [4], it is proved that if given enough capacity and training time, the minimax game between the discriminator and generator has a global optimum for $p_g = p_{data}$, where p_g is the generator distribution and p_{data} is the training data distribution. The details of implementation of the DCGANs can be found in [6].

2.4 Training Methods

2.4.1 Stochastic Gradient Descent

Consider an objective function that we seek to optimize in the following form:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w)$$

where the parameter w is estimated to minimize $Q(w)$. Each function Q_i is typically related with the i^{th} observation in the data set. The standard gradient descent algorithm updates the parameters of the objective function over the full training set. A batch gradient descent will perform the following updates to learn the parameter w:

$$w_{t+1} = w_t - \eta \sum_{i=1}^{n} \nabla Q_i(w_t)/n$$

where η is the learning rate and $\nabla Q_i(w_t)$ is the gradient of Q_i with respect to the parameter w. Stochastic gradient descent (SGD) only use single example to update and compute the gradient of parameters, so we are able to update the weights much more frequently and therefore hope to converge more rapidly. SGD approximates the true gradient of $Q(w)$ by a gradient of single example:

$$w \coloneqq w - \eta \nabla Q_i(w)$$

In practice, SGD is typically used with a mini-batch, where the mini-batch typically contains a small number (like 100) of data-points. Momentum method [36] is often used in practice to speed the convergence of SGD. On a high level, momentum provides a way to let previous gradients incorporate into the current update. Other extensions of the SGD algorithm like Adagrad by [37], Adadelta by [38] and Adam by [39] are known to work quite well in training deep networks.

2.4.2 Backpropagation

Backpropagation is a common method of training artificial neural networks and it is used in conjunction with a gradient-based optimization method such as stochastic gradient descent. In backpropagation, we first compute the prediction and its associated loss for each training example. We sum up all the loss to compute the final error. Then we propagate the error in order to compute the partial derivatives $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ of the cost function C for all weights w and biases b in the artificial neural networks.

2.5 Simulations of Deep Convolutional GANs

We implement and empirically evaluate DCGAN [59] on the Cifar-10 dataset. Figure 1 shows examples of the simulated images by DCGAN. CIFAR-10 consists of 50,000 diverse training examples, where each image is a RGB image with size $32\times32\times3$ [54]. Each pixel value is in the range between 0 and 255. We initialize the weights of both the generator and discriminator using Gaussian distribution $\mathcal{N}(0, 0.02)$, and initialize the biases as 0. We train DCGAN using Adam [27] with a learning rate of 0.0002 for both the generator and discriminator. We used the bag of architecture guidelines for stable training suggested in DCGAN [6]. In each training cycle, the discriminator is updated once and the generator is updated twice.

In Figure 8, we get the same noise vector input to the generator. We observe that as the training goes on (from 10 training epochs to 60 training epochs), we get much more realistic samples. In Figure 9, we show 64 samples we obtained after 80 epochs training.

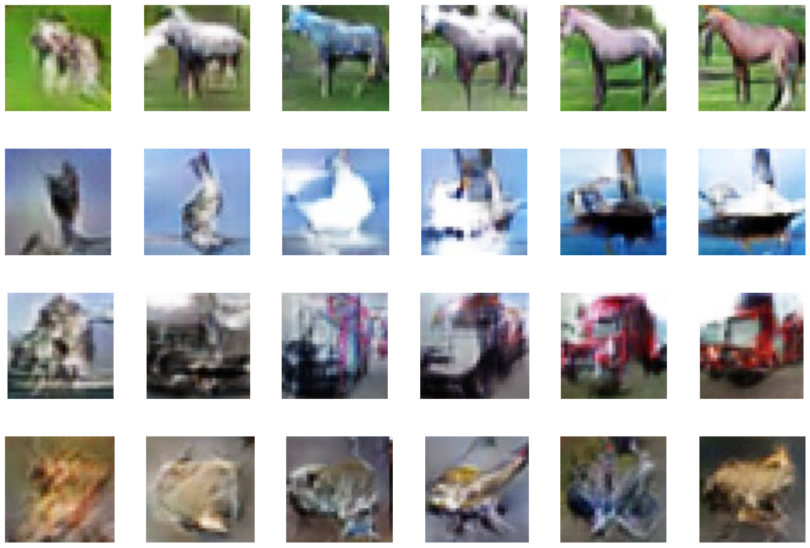


Figure 8: We show results with same noise vector inputs at epoch 10, 20, 30, 40, 50, 60.

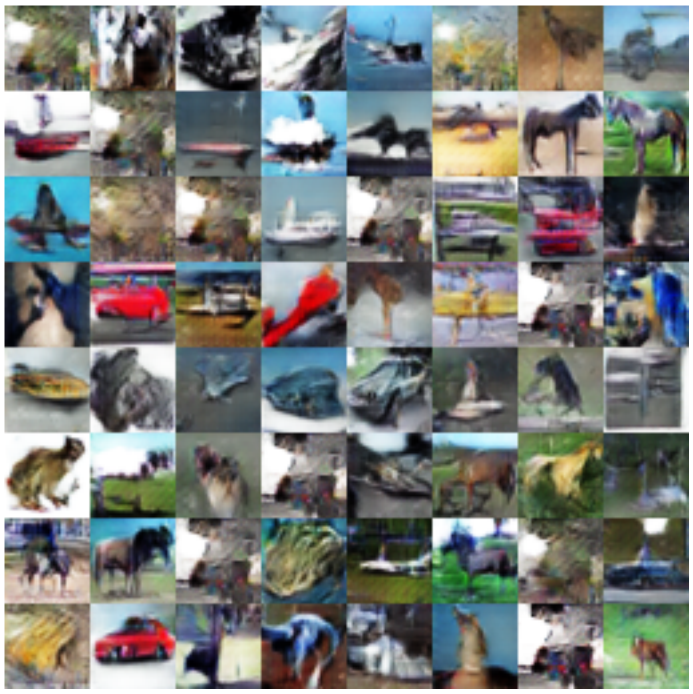


Figure 9: We show CIFAR-10 simulation results after 80 epochs training

Chapter 3: An extension of Generative Adversarial Networks using low-level features

We plan to extract attributes from image, and use the extracted attribute working as representation labels of the image. We determine representations of the generated images with the help of the extractable labels. In this chapter, we care about controlling low-level features of face images (like brightness and sharpness) using extended GANs architecture.

3.1 Related Work

Conditional GAN (CGAN) [7] is the conditional version of GAN, which feeding existing class labels both to the generator and discriminator. The results of CGAN show that it can generate MNIST digits [44] conditioned on the class labels. Auxiliary Classifier GAN (ACGAN) [43] feeds class labels to generator and modifies the discriminator to contain an auxiliary decoder network that reconstructs the class label for the training data. The class labels given by the auxiliary decoder are used to improve the quality of generated samples. Invertible Conditional GAN (IcGAN) [8] combines an encoder with a CGAN. The encoder maps a real image into a latent space and a conditional representation. It allows IcGAN to modify the attributes of generated images. In our model, we extend the architecture of DCGAN by adding a feature extractor, which can extract certain attribute labels of generated images. We use extractable labels to change generated images accordingly. In the next section, we introduce the details of the architecture.

3.2 Approach and Model Architecture

We have the noise vector z to work as the input of Generator in DCGAN, while here we concatenate a scalar representative variable c with the noise vector z. We try to control attributes of images by modifying the representative variable c provided to the Generator. We denote the concatenation of representative variable c and the noise vector

z by w. In the implementation, the noise vector z follows the uniform distribution $U\,(0,1)$. The representative variable c is either 0 or 1 with the same probability.

Generator $G_\theta(w)$ is used to generate fake images, where θ are the function parameters in the generator. Let the fake images generated by the Generator be denoted by x_{fake}, then

$$x_{fake} \coloneqq G_\theta(w)$$

In the model extension, we have feature extractor F. The inputs of F are the fake images x_{fake}. The output of feature extractor is $\hat{c}_{fake}$, which represents the attributes labels extracted from the fake images. So, we have

$$\hat{c}_{fake} = F(x_{fake})$$

For the Discriminator D_ϕ, where ϕ are the parameters of the discriminator network, the inputs are the fake images x_{fake} and the real images x_{real} sampled from the existing data. The motivation of Discriminator is to classify if an input images is real or generated. See Fig. 10 for a visualization of the model architecture. We call it *Extractor GANs* (EGANs).

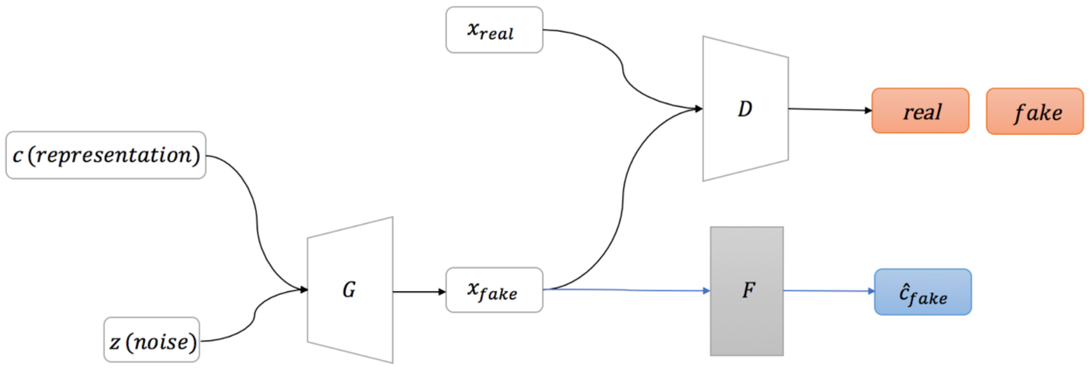


Figure 10: An extension of GANs with a feature extractor at the end of Generator

$D_\phi(\cdot)$ represents the probability that inputs are sampled from the data rather than the generator. Thus, $D_\phi(G_\theta(w))$ represents the probability that $G_\theta(w)$ came from the data rather than the generator. One task of generator in the extension model is to fool the discriminator to make wrong discrimination, which means $D_\phi(G_\theta(w))$ should be large. Alternatively, $1 - D_\phi(G_\theta(w))$ should be small. So, we have the following additional loss in the generator

$$\ell_{fool}(\theta) = -\sum_i \log\left(1 - D_\phi(G_\theta(w))\right)$$

By minimizing this loss function, the generator forces the discriminator to fail classifying the fake images as real. The second task of the generator is to let representative variable c learn the representative attributes of the images. We calculate the cross entropy between representative variable c and $\hat{c}_{fake}$, the output of feature extractor, as a representation loss term

$$\ell_{rep}(\theta) = \sum_i cross\_entropy(c_i, \hat{c}_{fake_i})$$

where c_i is the i^{th} representative variable, and $\hat{c}_{fake_i}$ is the corresponding output of feature extractor. We propose to learn the function parameters θ of the generator $G_\theta(w)$ by minimizing a combination of two losses:

$$\mathcal{L}_G(\theta) = \ell_{fool}(\theta) + \ell_{rep}(\theta)$$

For the Discriminator, the input can be the fake images generated from Generator and the real image sampled from the dataset. Train the discriminator D to maximize the probability of assigning the correct label to both training examples and samples from the generator G. We denote the loss of the discriminator as

$$\mathcal{L}_D(\phi) = \sum_i \log\left(D_\phi(x_{fake_i})\right) + \sum_j \log\left(1 - D_\phi(x_{real_j})\right)$$

where $\mathrm{D}_\phi(\cdot)$ represents the probability of the input being a fake image, and $1 - \mathrm{D}_\phi(\cdot)$ is the probability of the input being a real image.

For training this network, each mini-batch consists of randomly sampled fake images x_{fake} and real images x_{real}. The θ and ϕ for a mini-batch are updated by taking a stochastic gradient descent step on the mini-batch loss gradient. We learn the generator and discriminator parameters by minimizing $\mathcal{L}_{\mathrm{G}}(\theta)$ and $\mathcal{L}_{\mathrm{D}}(\phi)$ alternately, which is similar to the GAN training method.

To extend DCGANs with feature extractor at the end of the generator network, one experiment we perform is to control the brightness of the images, and another one is to control the sharpness of the images.

In the experiment, we first use a feature extractor to extract the brightness of the images. The brightness of an image is related to the pixel values. A RGB format image has 3 channels. Each channel has several pixels. Each pixel has a value between 0 and 255. In the RGB color space, brightness can be computed as a weighted sum of red, green, and blue coordinates [45], as illustrated in Figure 11.

$$\mathrm{y} = 0.2126 * \mathrm{r} + 0.7152 * \mathrm{g} + 0.0722 * \mathrm{b}$$

In the experiment, we need to calculate the brightness value of the fake images. We denote the width of the image as W, and the height of the images as H. The fake images are in the RGB color format. The shape of the fake image is $W \times H \times 3$. We use the above equation to get Y, which has a shape of $W * H$. We sum up the pixel values in Y. We do a normalization on the sum S to make the final value in the range between 0 and 1. The normalization operation is defined as sum S divided by $255 * W * H$. We use the final normalized value as the brightness of the fake image. Hence, we have

$$\mathrm{F}_{brightness} = reduce\_sum(Y)/(255 * W * H)$$

where the $\mathrm{F}_{brightness}$ is the brightness of an image.

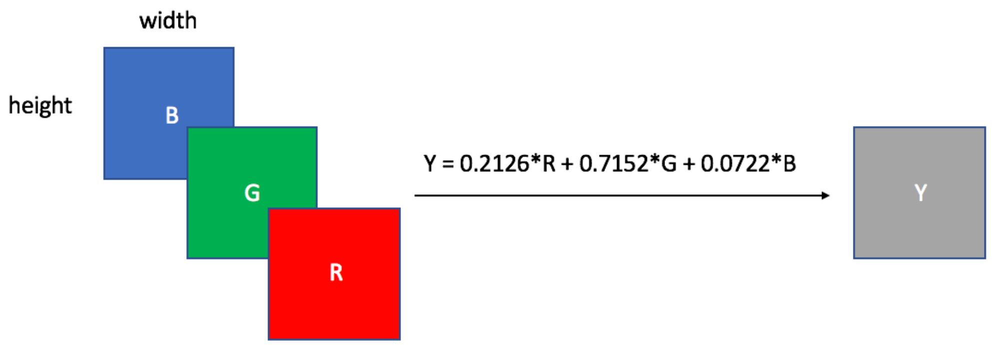


Figure 11: Brightness calculation

In the second experiment, we build a sharpness feature extractor. A sharp image can be characterized by the image contrast at the object edges. We estimate the sharpness of an image by calculating the average gradient of the image [41]. The input of the sharpness extractor is the fake images, which are RGB color images. We first convert the input to grayscale to get gray scaled images GSI. Each gray scaled image has a shape of $W \times H$.

To compute the gradient, we calculate the first difference at the boundaries and calculate the central differences in the interior. For example, we have an array A:

$$A = array(a_0, a_1, a_2, a_3, a_4)$$

The gradient of A is

$$gradient(A) = array(\frac{a_1 - a_0}{1}, \frac{a_2 - a_0}{2}, \frac{a_3 - a_1}{2}, \frac{a_4 - a_2}{2}, \frac{a_4 - a_3}{1})$$

We calculate the gradient of the GSI in two dimensions of the gray scaled images. We denote the two gradients as GX and GY, respectively. Both of GX and GY have the shape $W \times H$. We calculate the normalization of the gradients (denoted as $GNORM$), whose value is the geometric mean of each of the values in GX and GY. To be specific, $gx_{i,j}$ ($gy_{i,j}$ or $gnorm_{i,j}$)is the i-th row and j-th column value in GX (GY or $GNORM$). So, we have

$$gnorm_{i,j} = sqrt(gx_{i,j}^2 + gy_{i,j}^2)$$

Next, we sum up all values in $GNORM$ and divide the sum by $(W * H * max\,(GNORM))$, where $max\,(GNORM)$ is the maximum in $GNORM$ to get the estimation of sharpness. Hence, the sharpness value is calculated as

$$\mathrm{F}_{sharpness} = reduce\_sum(GNORM)/(W * H * max\,(GNORM))$$

where the $\mathrm{F}_{sharpness}$ is a sharpness estimation of an image with a value ranging between 0 and 1.

3.3 Results and discussion

The work of the experiments is based on the Tensorflow [58] implementation of the DCGAN [59]. We use CelebA dataset [55] in our experiments. CelebA has 202,599 RGB format face images. Each pixel value is in the range between 0 and 255. In the experiment, the pixel value is scaled to [-1, 1]. Our experiments are made with a batch size of 64, Adam as the optimizer with 0.0002 learning rate and 0.5 momentum term.

As shown in the figure 13, the brightness of the generated images is changed when we change the brightness representative variable from 0 to 1. In the sampling image step, we keep the noise vector z the same and only change the brightness representative variable c. We observe that when the noise vector is kept the same, the generated image shows the same person's face. The images become brighter when the value of the brightness representative variable goes up (from 0 to 1). In figure 14, we interpolate the representative variables from 0 to 2. We observed that it gave us better performance at changing the brightness of the images.

In the figure 15, we find that the sharpness of the images is changed when we change the sharpness representative variable from 0 to 1. The edges of the face become more obvious when the sharpness representative variable has 1 value. In the figure 16, we interpolate the sharpness representative variables from 0 to 2.

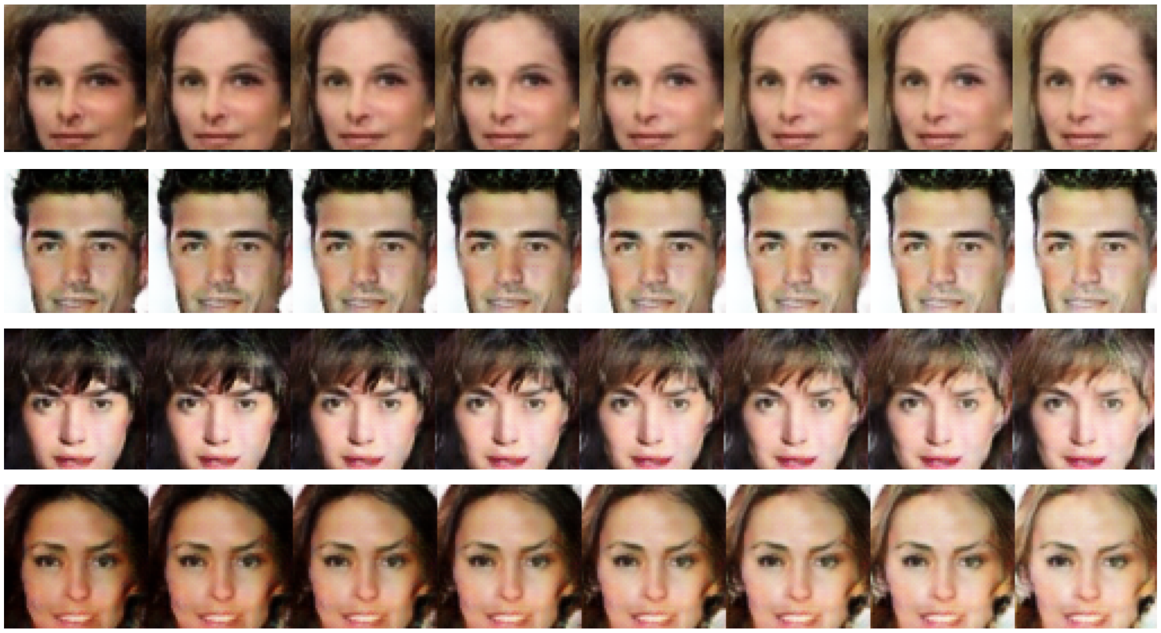


Figure 12: Results of the brightness experiments: Representative variable is interpolated from 0 to 1.

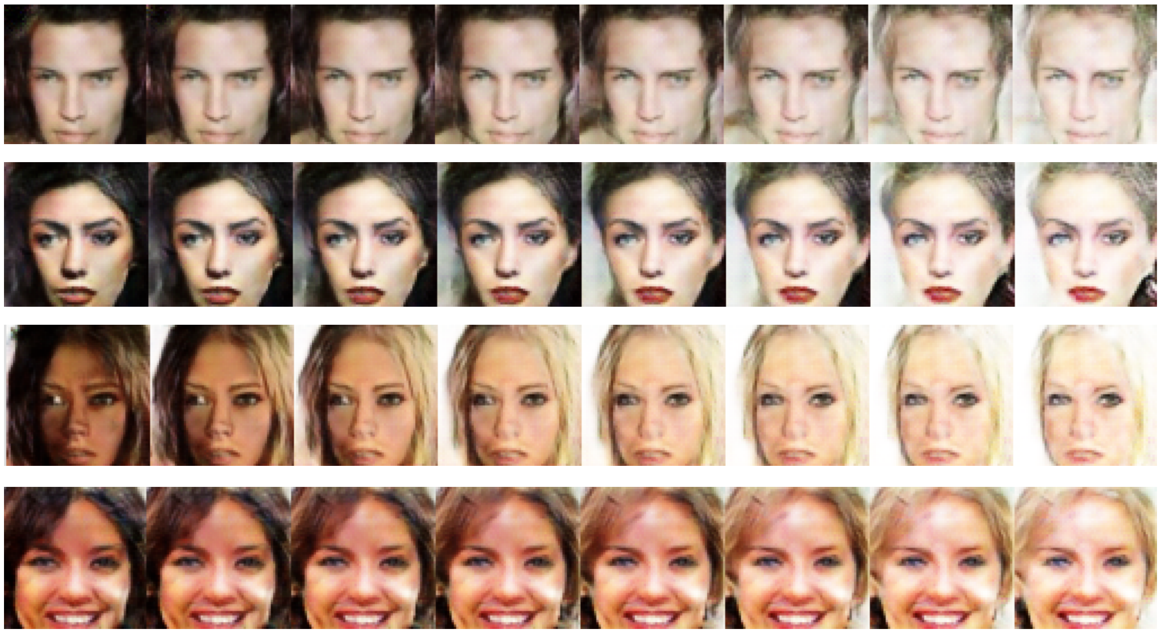


Figure 13: Results of the brightness experiments: Representative variable is interpolated from 0 to 2.

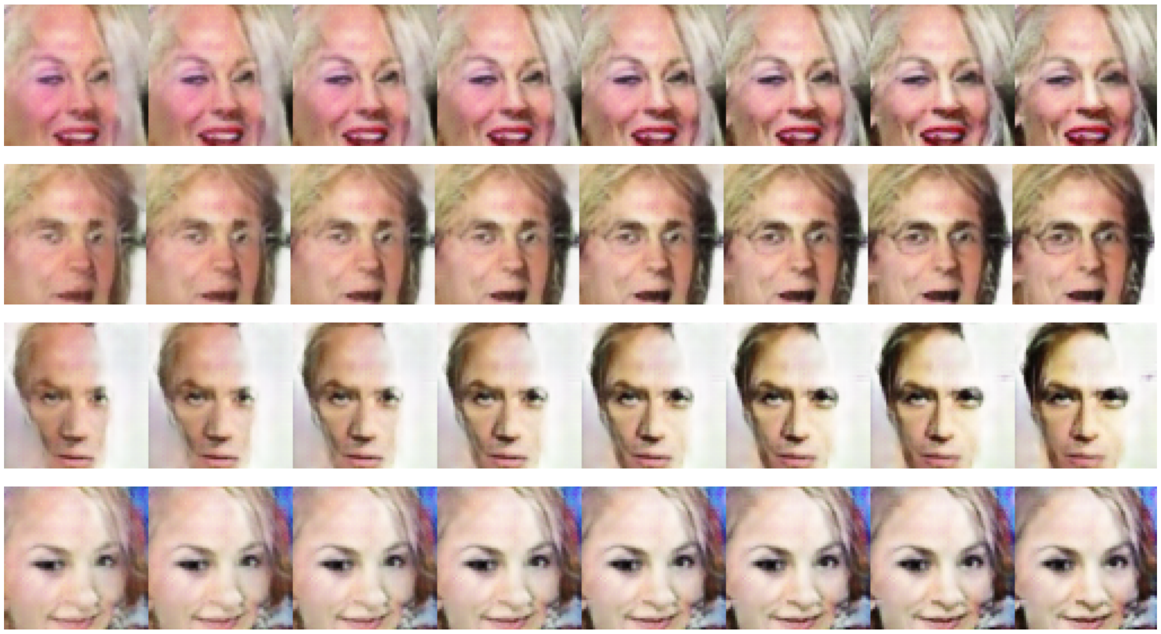


Figure 14: Results of the sharpness experiments: Representative variable is interpolated from 0 to 1.

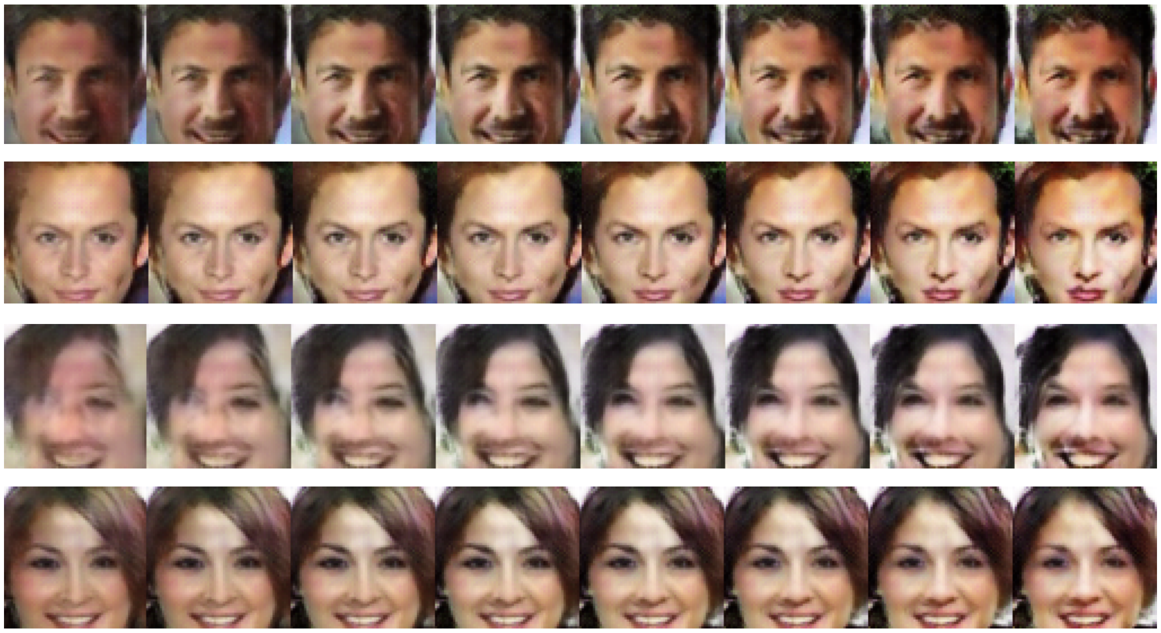


Figure 15: Results of the sharpness experiments: Representative variable is interpolated from 0 to 2.

Chapter 4: An extension of Generative Adversarial Networks using high-level features

4.1 Approach and Model Architecture

In the previous chapter, we introduce the brightness/sharpness extractor function to let *EGANs* learn the brightness/sharpness feature of the data. The extended model can control the brightness/sharpness of the generated images with the help of extractable labels. We are further interested in controlling some high-level features of the face images. High-level features are used to detect and classify objects in real life, such as gender and openness of mouth of face images.

We need to build a model to work as the feature extractor. This model can classify images into different classes. Convolutional Neural Networks commonly are used to classify images. Naturally, we think we can pre-train the CNNs using the training data. Then, we can use the well-trained CNNs to work as the feature extractor to extract the high-level features of the face images. The extractable labels given by the CNNs are used to add an extra loss function to the original DCGAN. See Fig. 17 for a visualization of the model architecture.

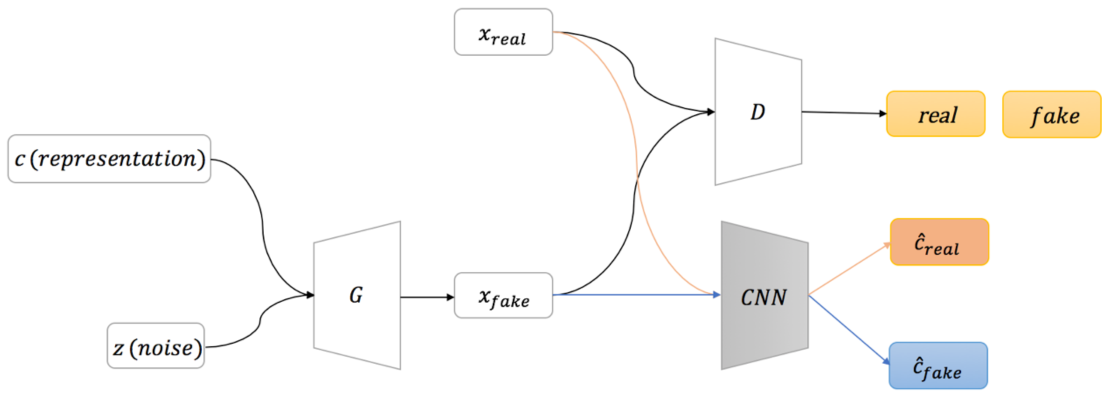


Figure 16: An extension of GANs with CNNs as the feature extractor.

The architecture is almost same as the one in the Chapter 3. The only difference is that we use CNNs to work as the feature extractors. The input of the generator is the noise vector z concatenated with representative variable c. Each number in the noise vector z is sampled from the uniform distribution $U\ (0, 1)$. The representative variable c is either 0 or 1 with same probability. The fake images x_{fake} are the output of the generator $G_{\theta}(w)$, where θ are the function parameters of the generator.

In the model extension, we have a convolutional neural network CNN. When the fake images x_{fake} are the inputs of the CNN, the output of the CNN is $\hat{c}_{fake}$, which represents the labels extracted from the fake images. Also, $\hat{c}_{fake}$ represents the probability that an image has a certain attribute. Hence, we have

$$\hat{c}_{fake} = CNN(x_{fake})$$

The $\hat{c}_{real}$ is the output of the CNNs when the input is real images, so

$$\hat{c}_{real} = CNN(x_{real})$$

The fake images x_{fake} and the real images x_{real} sampled from the dataset are the inputs of the discriminator. *EGANs* learn the function parameters θ of the generator $G_{\theta}(\cdot)$ and ϕ of the discriminator $D_{\phi}(\cdot)$ by minimizing $\mathcal{L}_G(\theta)$ and $\mathcal{L}_D(\phi)$ respectively, which have been defined in the Chapter 3.

Two training modes

Because we do not know the parameters of the CNNs in advance, we need to find a way to train the CNNs to get the CNNs working as the feature extractors with decent performance. We consider two training modes. One is to use pre-trained convolutional neural networks to work as the feature extractor in the architecture we show above. Another one is to train the CNN, the discriminator and generator together, which means we train them alternatingly in each step.

1. *Pre-training* mode

We build a simple CNN with 3 layers. The last layer is a full-connected layer with sigmoid as the activation function. We view the output of the sigmoid function as the probabilities that certain images have certain attributes, such as how much percent the gender of face is female. We train the CNN with the CelebA training data and its real attributes labels for 25 epochs. We save the checkpoint [57] at the end of each epoch. After the pre-training, we can use the extended DCGAN to reload the checkpoint for the CNN, and use the CNN to work as the feature extractor.

2. *Without pre-training* mode

In non-pre-training mode, we train the CNN, discriminator and generator alternatingly in each step. The input of the CNN are the real images or the fake images output by the generator. We minimize the loss function $\mathcal{L}_{cnn}(\varphi)$ to optimize the CNNs, where φ are the parameters of the CNNs. We only update the parameters in the CNNs with the real images and real labels, so

$$\mathcal{L}_{cnn}(\varphi) = \sum_{i} cross\_entropy(l_{real_i}, \hat{c}_{real_i})$$

where l_{real_i} is the real labels of the i^{th} real image, and $\hat{c}_{real\,i}$ is the output of the CNNs when the input is the i^{th} real image. We update the generator for both the $\mathcal{L}_G(\theta)$ and the discriminator with the $\mathcal{L}_D(\phi)$. $\mathcal{L}_G(\theta)$ and $\mathcal{L}_D(\phi)$ are defined in Chapter 3.

4.2 Results and discussion

Our experiments are performed on the CelebA dataset. It has 40 binary attributes annotations per image, which we use as the real attribute labels of each image. When we train the CNN, we need to use the real labels from the CelebA. The output of CNN is the probability that an image has a certain attribute. This is why $\hat{c}_{fake}$ and $\hat{c}_{real}$ fall in the interval of $(0,1)$. For the real attribute labels, we use 0.9 to represent that an image has certain attribute and use 0.1 to represent that an image does not have certain attribute. Only when the input for the sigmoid is positive infinity or negative infinity, the output of the sigmoid is able to be 1 or 0. Using 0.9 and 0.1 instead of 1 and 0 can help the *EGANs* achieve expected results, which we observed from our experiments.

1. *Pre-training* mode

   (a) Mouth slightly open

   We first pre-trained the CNN for 25 epochs. Then, we use the extended DCGAN to reload the parameters in the checkpoint for the CNN. We trained the extended model for 25 epochs. When we train the EGANs, representative variables c, output of the CNNs $\hat{c}_{fake}$ and $\hat{c}_{real}$ all range from 0 to 1. Naturally, we sweep the representative variable from 0 to 1 to see the change of openness of mouth, the change is not obvious as illustrated in Figure 17. We found that when we interpolate the representative variable from 0 to 2, it gave us good performance in changing the "Mouth slightly open" attribute of generated samples, as illustrated in Figure 18.

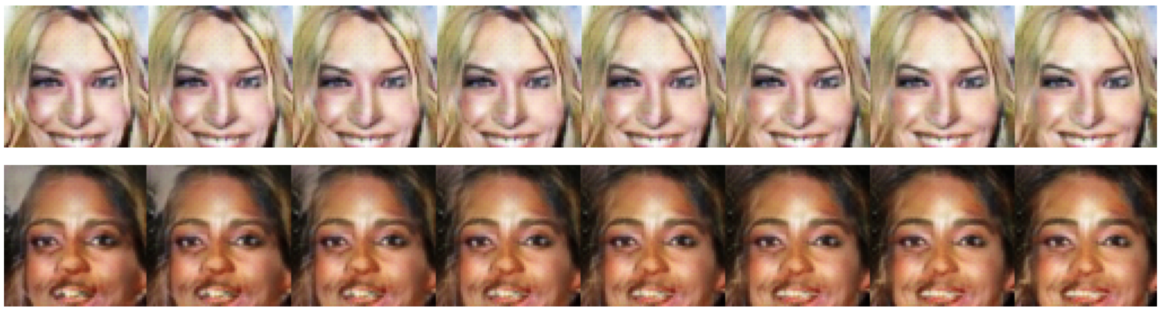


Figure 17: *Pre-training* mode: We pre-train CNN for 25 epochs. We run extension model for 25 epochs. We use mouth slightly open representative variable. The range of representative variable is (0, 1).

2. *Without pre-training* mode

   (a) Gender

   In the Figure 19, we want to change the gender attribute of the generated faces. We use a convolutional network to work as a classifier to classify the gender of the images. We use the *without pre-training* mode, which means we train the convolutional neural networks, the discriminator and generator alternatingly. We find that when we change the representative variable from 0 to 5, the gender of the images changes gradually from female to male.

   (b) Mouth slightly open

   We follow the same steps to learn the "mouth slightly open" attribute of the face images. As we alter the representative variable from 0 to 5, the model can change the state of the mouth in the images from close to open, as illustrated in Figure 20.

   (c) Wearing lipstick

   When we use the extended architecture to learn the "wearing lipstick" attribute of the face images, we alter the representative variable from 0 to 2. We observed that the person in the image wears lipstick gradually as the value of representative variable goes up, as illustrated in Figure 21.

The experiments suggest that both pre-training and non-pre-training modes work. We see that our extended model is able to learn the representative attributes of the images. Note that the results shown in Figure 18, 19, 20, 21 have been selected to highlight the success of the model. The approach in EGANs is preliminary and it does not work for all labels, such as smiling and wearing eyeglasses.

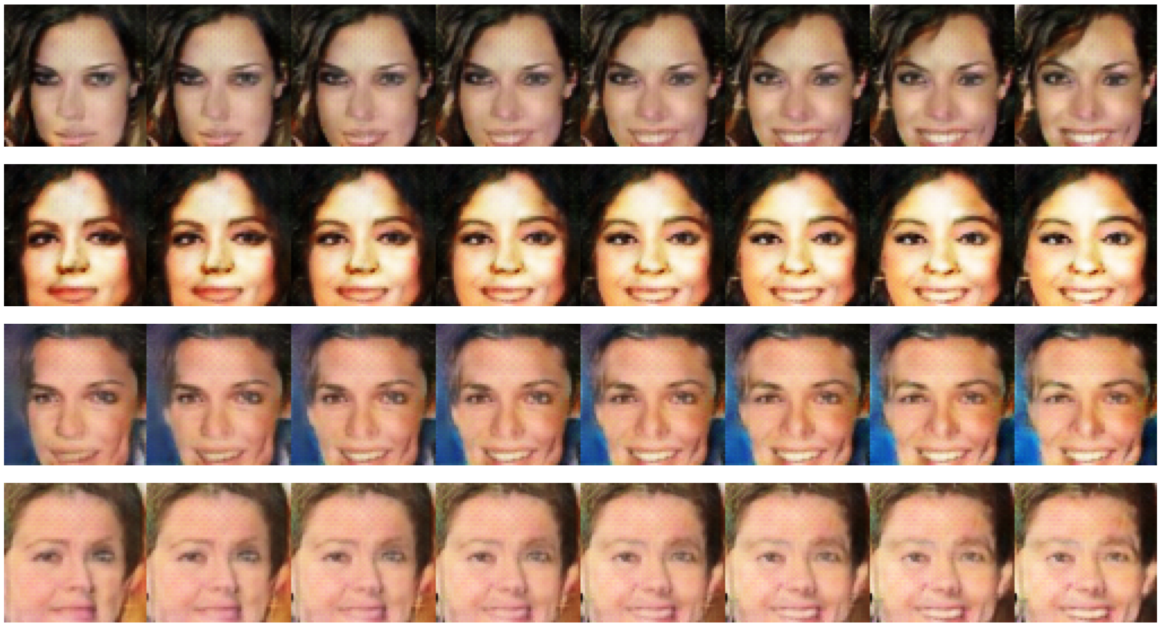


Figure 18: *Pre-training* mode: We pre-train CNN for 25 epochs. We run extension model for 25 epochs. We use mouth slightly open representative variable. The range of representative variable is $(0, 2)$.

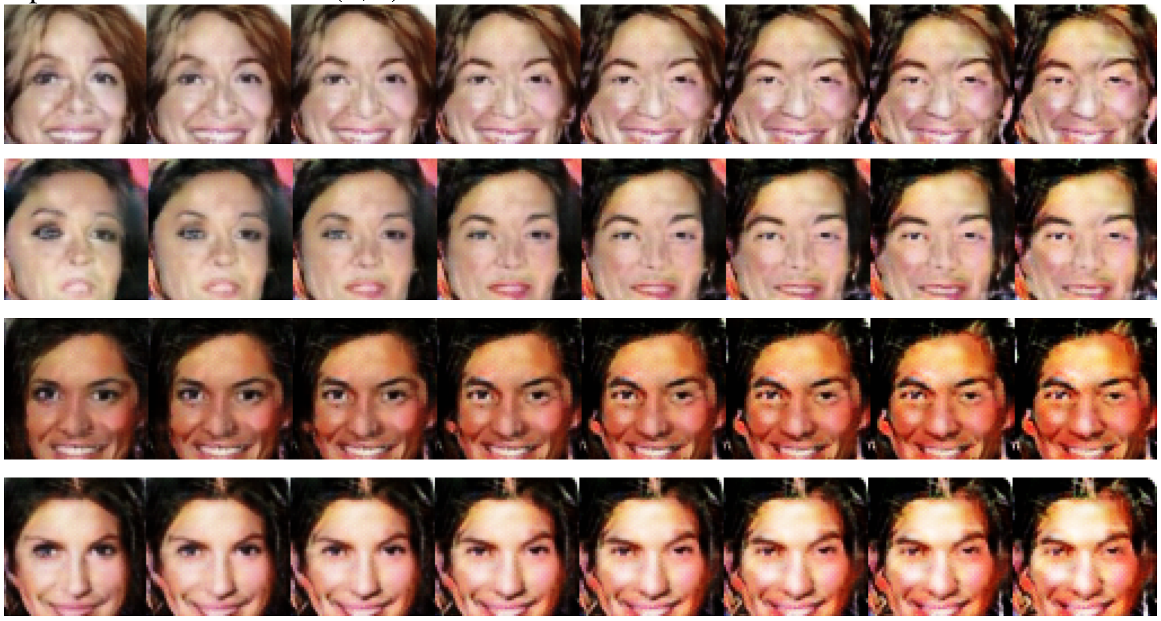


Figure 19: *Without pre-training* mode: We run extension model for 13 epochs. We use gender representative variable. The range of representative variable is $(0, 5)$.

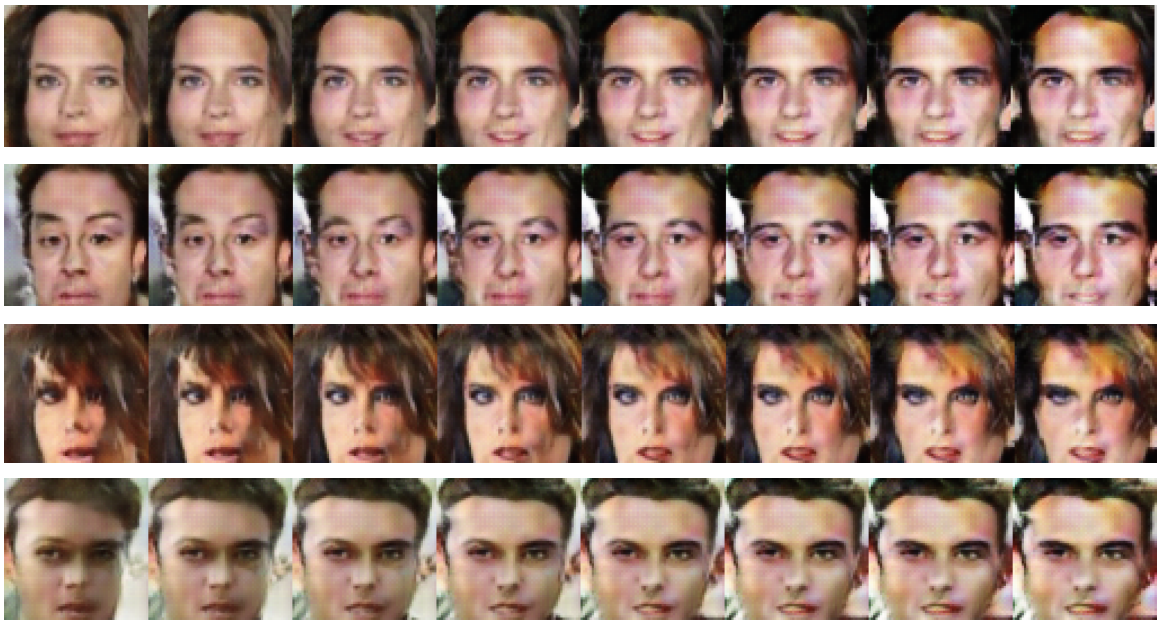


Figure 20: *Without pre-training* mode: We run extension model for 13 epochs. We use mouth slightly open representative variable. The range of representative variable is $(0, 5)$.

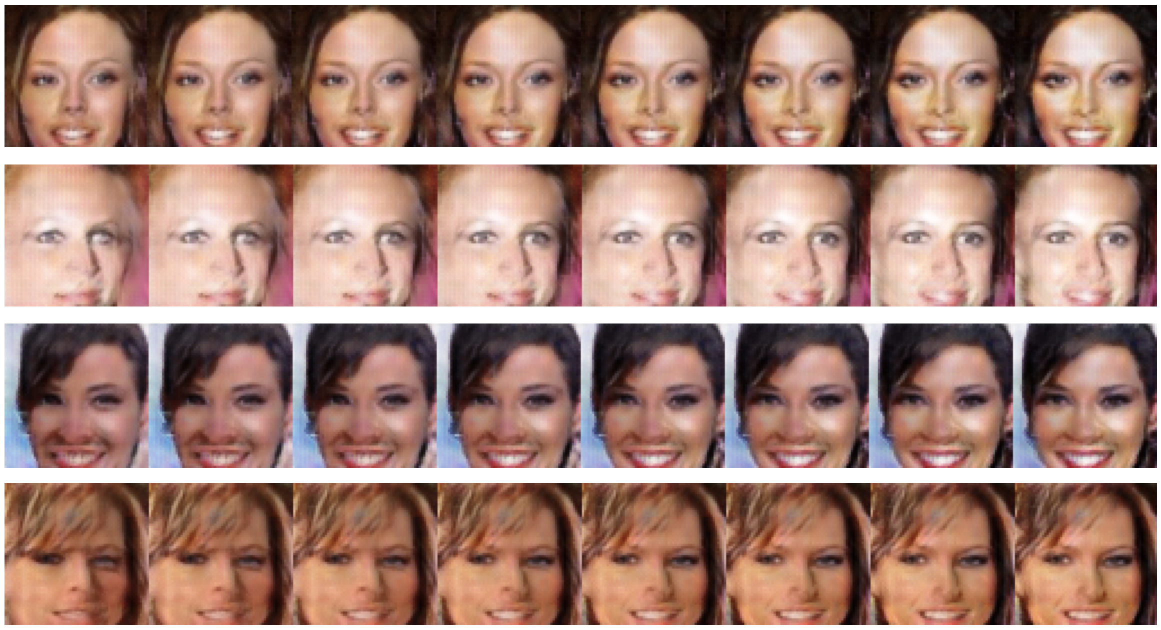


Figure 21: *Without pre-training* mode: We run extension model for 16 epochs. We use lipstick representative variable. The range of representative variable is $(0, 2)$.

Chapter 5: Conclusion

In this M.S. thesis, we study GAN, in particular DCGAN, and its extensions. We do the simulations to explore the power of DCGAN to generate new samples. As a contribution, we extend the DCGAN architecture by taking advantage of the extractable labels. We first extract two low-level labels, brightness and sharpness in our experiments. Next, we try to control some high-level attributes of the generated images, such as gender, openness of the mouth, lipstick. To achieve this, we use convolutional neural networks as feature extractors. The function of CNN is to classify if an image has a certain attribute. We propose two ways to train the CNN to help us get a good feature extractor with high performance. One way is to pre-train the CNN by using the real attribute labels provided by the training data. Another way is to train CNN together with the generator and discriminator. By observing the results of experiments, we found that our extended GAN architecture works well to determine brightness, sharpness and even some high-level representations of the generated images.

References

[1] Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." *Neural computation* 18.7 (2006): 1527-1554.

[2] Kingma, Diederik P., and Max Illing. "Auto-encoding variational bayes." *arXiv preprint arXiv:1312.6114* (2013).

[3] Gregor, Karol, et al. "DRAW: A recurrent neural network for image generation." *arXiv preprint arXiv:1502.04623* (2015).

[4] Goodfellow, Ian, et al. "Generative adversarial nets." *Advances in Neural Information Processing Systems*. 2014.

[5] Salimans, Tim, et al. "Improved techniques for training gans." *Advances in Neural Information Processing Systems*. 2016.

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

[7] Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." *arXiv preprint arXiv:1411.1784* (2014).

[8] Perarnau, Guim, et al. "Invertible Conditional GANs for image editing." *arXiv preprint arXiv:1611.06355* (2016).

[9] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. Cognitive modeling, 5(3):1, 1988.

[10] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

[11] Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." *arXiv preprint arXiv:1411.1784* (2014).

[12] Perarnau, Guim, et al. "Invertible Conditional GANs for image editing." *arXiv preprint arXiv:1611.06355* (2016).

[13] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets.

[14] Donahue, Jeff, Philipp Krähenbühl, and Trevor Darrell. "Adversarial feature learning." *arXiv preprint arXiv:1605.09782* (2016).

[15] Ledig, Christian, et al. "Photo-realistic single image super-resolution using a generative adversarial network." *arXiv preprint arXiv:1609.04802* (2016).

[16] Shrivastava, Ashish, et al. "Learning from Simulated and Unsupervised Images through Adversarial Training." *arXiv preprint arXiv:1612.07828* (2016).

[17] Reed, Scott, et al. "Generative adversarial text to image synthesis." *Proceedings of the 33rd International Conference on Machine Learning*. Vol. 3. 2016.

[18] Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." *arXiv preprint arXiv:1611.07004* (2016).

[19] Zhu, Jun-Yan, et al. "Generative visual manipulation on the natural image manifold." *European Conference on Computer Vision*. Springer International Publishing, 2016.

[20] Brock, Andrew, et al. "Neural photo editing with introspective adversarial networks." *arXiv preprint arXiv:1609.07093* (2016).

[21] Li, Jiwei, et al. "Adversarial learning for neural dialogue generation." *arXiv preprint arXiv:1701.06547* (2017).

[22] Yu, Lantao, et al. "Seqgan: sequence generative adversarial nets with policy gradient." *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[23] Pfau, David, and Oriol Vinyals. "Connecting generative adversarial networks and actor-critic methods." *arXiv preprint arXiv:1610.01945* (2016).

[24] Zeiler, Matthew D., et al. "Deconvolutional networks." *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010.

[25] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. JMLR, abs/1502.03167, 2015.

[26] Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks." *Aistats*. Vol. 15. No. 106. 2011.

[27] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).

[28] Ivakhnenko, A. G. and Lapa, V. G. (1965). Cybernetic Predicting Devices. CCM Information Corporation.

[29] Fukushima, K. (1979). Neural network model for a mechanism of pattern recognition unaffected by shift in position - Neocognitron. Trans. IECE, J62-A (10):658–665.

[30] http://www.image-net.org/challenges/LSVRC/2012/

[31] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv preprint arXiv:1502.03167* (2015).

[32] Noh, Hyeonwoo, Seunghoon Hong, and Bohyung Han. "Learning deconvolution network for semantic segmentation." *Proceedings of the IEEE International Conference on Computer Vision*. 2015.

[33] Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." *European conference on computer vision*. Springer International Publishing, 2014.

[34] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum, “Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling,” in Advances in Neural Information Processing Systems, 2016, pp. 82–90.

[35] Badrinarayanan, Vijay, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation." *arXiv preprint arXiv:1511.00561* (2015).

[36] Polyak, Boris T. "Some methods of speeding up the convergence of iteration methods." *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964): 1-17.

[37] Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of Machine Learning Research* 12. Jul (2011): 2121-2159.

[38] Zeiler, Matthew D. "ADADELTA: an adaptive learning rate method." *arXiv preprint arXiv:1212.5701* (2012).

[39] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).

[40] Roth, Kevin, et al. "Stabilizing Training of Generative Adversarial Networks through Regularization." *arXiv preprint arXiv:1705.09367* (2017).

[41] Batten, Christopher F. "Autofocusing and astigmatism correction in the scanning electron microscope." *Mphill thesis, University of Cambridge* (2000).

[42] Chellapilla, Kumar, Sidd Puri, and Patrice Simard. "High performance convolutional neural networks for document processing." *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

[43] Odena, Augustus, Christopher Olah, and Jonathon Shlens. "Conditional image synthesis with auxiliary classifier gans." *arXiv preprint arXiv:1610.09585* (2016).

[44] http://yann.lecun.com/exdb/mnist/

[45] https://en.wikipedia.org/wiki/Relative_luminance#cite_note-2

[46] Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).

[47] Schmidhuber, Jürgen. "Learning complex, extended sequences using the principle of history compression." *Neural Computation* 4.2 (1992): 234-242.

[48] Oh, Kyoung-Su, and Keechul Jung. "GPU implementation of neural networks." *Pattern Recognition* 37.6 (2004): 1311-1314.

[49] Raina, Rajat, Anand Madhavan, and Andrew Y. Ng. "Large-scale deep unsupervised learning using graphics processors." *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009.

[50] Ciresan, Dan Claudiu, et al. "Flexible, high performance convolutional neural networks for image classification." *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[51] Berthelot, David, Tom Schumm, and Luke Metz. "BEGAN: Boundary Equilibrium Generative Adversarial Networks." *arXiv preprint arXiv:1703.10717* (2017).

[52] Bora, Ashish, et al. "Compressed Sensing using Generative Models." *arXiv preprint arXiv:1703.03208* (2017).

[53] Shi, Shaohuai, and Xiaowen Chu. "Speeding up Convolutional Neural Networks by Exploiting the Sparsity of Rectifier Units." *arXiv preprint arXiv:1704.07724* (2017).

[54] https://www.cs.toronto.edu/~kriz/cifar.html

[55] http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

[56] https://en.wikipedia.org/wiki/Generative_model

[57] https://www.tensorflow.org/programmers_guide/variables

[58] https://www.tensorflow.org/

[59] https://github.com/carpedm20/DCGAN-tensorflow