Instructor: Pavlos Protopapas **TFs :** Patrick Ohiomoba, Srivatsan Srinivasan

Group Members: Ying Yi, Yunfan He, Zhilin Zhang, Shengcheng Yu, Haoyue Du

Introduction

This work is part of the homework for the online GEC Deep Learning course, offered by Dr. Pavlos Protopapas, TF Patrick Ohiomoba and Srivatsan Srinivasan. From this work, we hope to have better grasp of Autoencoders and GANs, as well as experimenting with mixing Convolutional Autoencoders, VAE and DCGANs.

The project used several neural network structures related with VAEs and GANs. Variational autoencoders (VAEs) were defined in 2013 by Kingma et al. and Rezende et al.. GANs were introduced in a paper by Ian Goodfellow and other researchers at the University of Montreal, including Yoshua Bengio, in 2014. VAE-GAN, introduced by Larsen et al., is also used in this project.

There are two parts in this project: CelebA Face Generation and VAEGANIME.

CelebA Face Generation - VAE

Rationale

When dealing with images, we usually use convolutional network. Convolutional autoencoders can extract and learn the characteristics of the images in the dataset. A convolution in the general continue case is defined as the integral of the product of two functions (signals) after one is reversed and shifted:

$$f(t) * g(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau$$

In the image domain where the signals are finite, this formula becomes:

$$O(i,j) = \sum_{u=-2k-1}^{2k+1} \sum_{v=-2k-1}^{2k+1} F(u,v) I(i-u, j-v)$$

Methods

We train our VAE and DCGAN model on CelebA (CelebFaces Attributes Dataset). CelebA is a huge dataset with big diversities which contains “10177 number of

identities, 202,599 number of face images, and 5 landmark locations, 40 binary attributes annotations per image". (cited from above) We use the preprocessed data which is composed of face images in JPEG format.

We first load the data in list_attr_celeba.txt into a Pandas data frame. The attribute data contains the characteristics of the faces, which is the basis of the training. Then we load the images in img_align_celeba into x_train, y_train, x_test, and y_test.

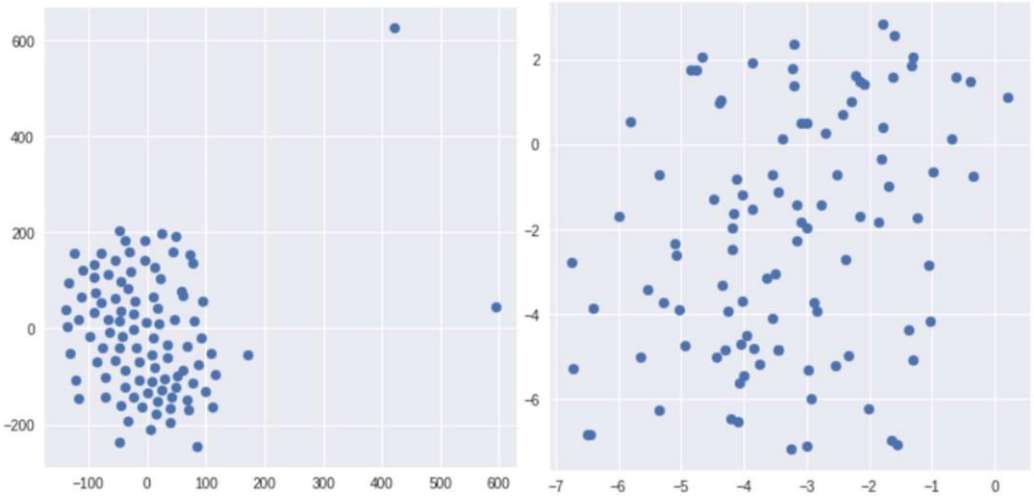
In the directory, there are 202599 pictures of different celebrities. In this experiment, we use a subset with a size of 20000 of the original data set.We randomly choose 20000 picture and load them with PIL.Image to load the image into a nparray with a shape of (20000, 218, 178, 3).

Each picture is described with a 40-dimenional vector, each dimension represents an attribute, if the corresponding dimension matches the picture, the label is 1, otherwise, it is -1. In the experiment, we load the attribute file into a pandas.DataFrame, with 202599 rows and 40 columns.

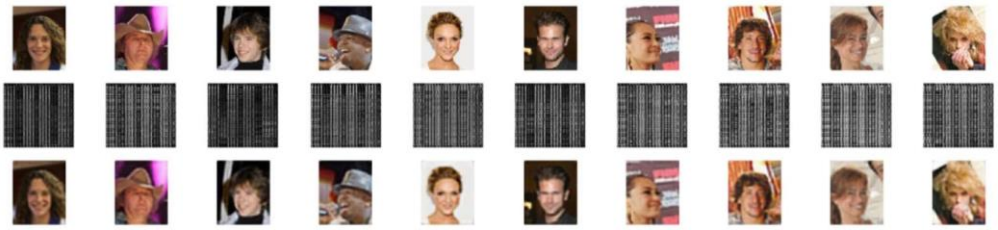
We build an encoder and a decoder, and assemble them to get a VAE (Variational Auto-Encoder). Both the encoder and the decoder are simple CNNs, consisting of Convolutional Layer, Dense Layer, Reshape Layer, and so on.

Results

Shown below is the original representation versus the latent representation. Left is original, right is latent representation. The latent representation is pretty spread out compared to the original one. Note that the axis are not drawn to scale.



The output of the autoencoder is shown below. The reconstruction is very close to the original one.



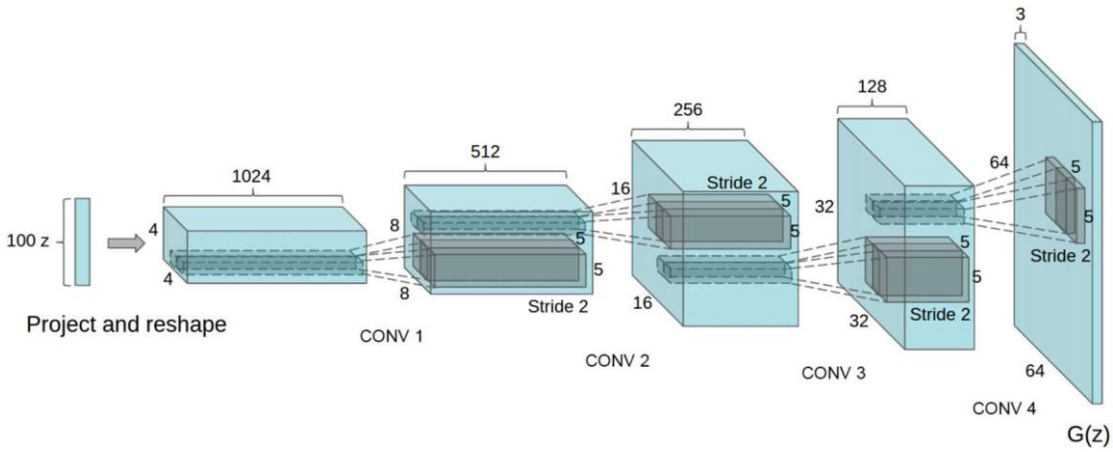
CelebA Face Generation - DCGAN

Rationale

The best model for image processing in deep learning is CNN. GANs can be well characterized without specific loss functions and supervised learning processes, but they are highly unstable in training and often result that generators produce meaningless output. DCGAN combines the deep convolutional network and the generative adversarial networks together, and after certain modifications to the structure, the advantages of them can be integrated.

The full name of DAGAN is Deep Convolutional Generative Adversarial Networks. It was put forward by Alec Radford. DAGAN is based on GAN adding depth convolutional network structure, specially generate image samples. In fact, GAN has not made any restrictions on the specific structure of D and G. The meanings and losses of D and G in DCGAN are exactly the same as those in the original GAN, but they adopt special structures in D and G in order to effectively model pictures. For discriminator D, its input is an image and its output is the probability that the image is a real image. In DCGAN, the structure of discriminator D is a convolutional neural network. After several layers of convolution, the input image gets a convolution feature, which is fed into the Logistic function. The output can be regarded as probability.

For example, z is a uniform distribution with 100 dimension. Through convolutions it finally turns into a 64*64*3 picture.



Methods

In this model, professor provides us with over 20000 pictures for training. This model is similar to GAN. The difference between DCGAN and GAN is generator and discriminator. So I define the both with Deep Convolutional Generative Adversarial Networks to create DCGAN. As a result , the model consists of four python files(main.py model.py utils.py ops.py). Model.py mainly defines generator and discriminator. Ops.py mainly defines several activations. Utils.py mainly defines how to deal with the pictures. Whlie main.py invokes all the other three files and defines most parameters.

The model is mainly composed of generator and discriminator. CNN should be applied to create both the generator and the discriminator. And both of them need the batch normalization to speed up.

Here are the tricks needing modification and attention：

-Replace all max pooling with convolutional stride.
-Use transposed convolution for upsampling.
-Eliminate fully connected layers.
-Use Batch normalization except the output layer for the generator and the input layer of the discriminator.
-Use ReLU in the generator except for the output which uses tanh.
-Use LeakyReLU in the discriminator.

Among them, batch Normalization, abbreviated BN, accelerates learning and convergence. It transforms the input of each layer into zero mean and unit standard deviation, which has been proved to be a very important method of accelerating convergence and slowing overfitting in deep learning. It can help train difficulties caused by improper initialization, and also allows gradient flow to deepen layers. Practice has shown that this is crucial to the effective learning of deep generator, and it can prevent the generator from turning all samples into a single point, which is a problem frequently encountered in GAN training. However, if BN is directly applied to all layers, the sample will oscillate and become unstable, so we only use BN for the output layer of the generator and the input layer of the discriminator.

LeakyReLU is used to speed up convergence. Unlike the activation function ReLU, LeakyReLU gives all negative values a non-zero slope. LeakyReLU activation function was first proposed in an acoustic model (2013). Mathematically, we can express it as:

$$y_i = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \frac{x_i}{a_i} & \text{if } x_i < 0, \end{cases}$$

Adam is a first-order optimization algorithm that can replace the traditional stochastic gradient descent process. It can update the weights of the neural network iteratively based on the training data. Adam was initially proposed by Diederik Kingma at OpenAI and Jimmy Ba at the university of Toronto in A 2015 ICLR paper (Adam: A Method for Stochastic Optimization).

Results

Some of the result are shown below. Note that the figures near the top of the image are not smiling, and the figures at the bottom are smiling.

VAEGANIME - DCGAN

In this section, we create and compile a DCGAN model on the basis of making clear the structure of generator and discriminator. Because there are similarities between this model and the DCGAN model used on the celeba dataset, the Ratinale section is skipped.

Methods

The model is mainly composed of generator and discriminator. CNN should be applied to create both the generator and the discriminator. And both of them need the batch normalization to speed up.

Set up loss function and optimize generator and discriminator. The discriminator learns conditional probability, while the generator learns joint probability. Finally, the generator and discriminator are optimized.

Results

The DCGAN model was successfully trained on these steps. Compared with simple CNNs and simple GANs, the products of the DCGAN are more stable and clear. Meanwhile, note the use of activation functions and remove the full connection layers. Some of the samples from the 50^{th} epoch are shown below.

The output is diversified in terms of the orientation of the faces and the hair color, however, the images are not very clear, and there is some noise.

VAEGANIME - VAEGAN

Rationale

The generator in the GAN takes a random input, and generate an output that is similar to the data in the dataset. A decoder in an autoencoder can also generate an output that is similar to the dataset. If the trained decoder can serve as a generator in a GAN, the output might be more detailed and the training speed might be faster.

Methods

The dataset contains 21551 images of anime faces. Every image is cropped so that only the face is visible in the frame. No other data is needed.

All the images were loaded into a numpy array with a size of (21551*64*64*3). Before feeding into the neural network, the data was flattened to (21551*12288) and normalized to the range of (0,1).

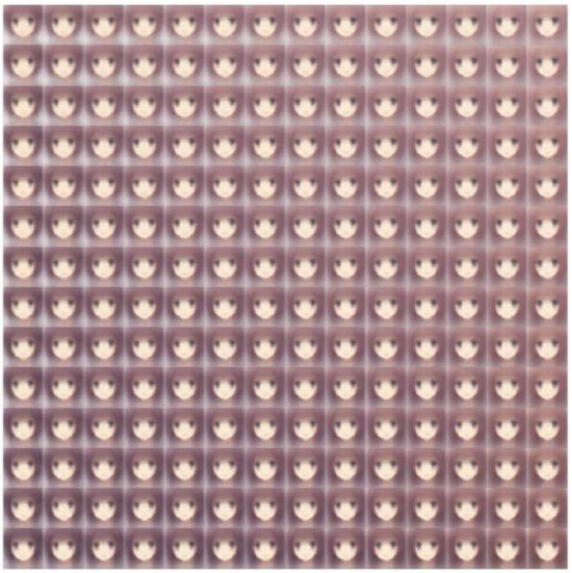
The model was built with Keras. It consists of 2 parts: the autoencoder and the GAN. The autoencoder has an encoder and a decoder, where the decoder also serves as a generator for the GAN.

When dealing with a single batch of data, the network trains in 3 steps. The first step is training the autoencoder. Both the encoder and the decoder are trained with a batch of image, where the rest of the network is not being used or trained. Next, the decoder generates some data and the discriminator is being trained. Finally, The generator is being trained on a batch of data. When training the generator, the entirety of GAN is being used, but the discriminator is freezed to prevent it from training. Thus, only the generator is being trained.

Results

The VAE-GAN model was successfully trained on these steps. Compared with ordinary autoencoders and simple GANs, the product of the VAE-GAN looks a little bit better, but not by much. When the generator is feeded with a random noise, generation of an image fails, differentiating VAE-GAN from simple GAN. Besides, training VAE-GAN consumes more time than simple GANs.

Some of the samples generated by the network is shown below. The network generalized all the inputs and did some classification on hair length and hair color.



Conclusion

The generative models did a good job at generalizing inputs and generate new samples. Networks with convolutional features did a better job than MLPs at generating images, and GANs generally outputs clearer, more different samples.

References

Doersch, C. (2016). Tutorial on variational autoencoders. arXiv preprint arXiv:1606.05908.

Larsen, A. B. L., Sønderby, S. K., Larochelle, H., & Winther, O. (2015). Autoencoding beyond pixels using a learned similarity metric. arXiv preprint arXiv:1512.09300.

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.