MASC 520 Project 4 Report
Unsupervised Representation Learning with Generative Adversarial Networks (GAN) on CIFAR-10 Dataset

Ruoxi Li
5817808450

Abstract

Learning reusable feature representations from large unlabeled datasets has been an area of active research. Generative adversarial networks (GANs) provide a way to learn deep representations without extensively annotated training data. In this project, I'll explore the potential of GAN to generating realistic images based on CIFAR-10 dataset. The theoretical aspect of GAN and the architecture are fully discussed. The training and testing results prove that the trained model can generate realistic samples with high diversity.

1 Method

Generative models try to model the distribution of the data in an explicit way, in the sense that we can easily sample new data points from this model. This is in contrast to discriminative models that try to infer the output from the input. A GAN is composed of a generator that generates the images and a discriminator that distinguishes the generated images from real images. The discriminator is a binary classifier with the two classes being "taken from the real data" ("real") and "generated by the generator" ("fake"). Its objective is to minimize the classification loss. The generator's objective is to generate samples so that the discriminator misclassifies them as real.

Typically, for the discriminator we use binary cross entropy loss with label 1 being real and 0 being fake. For the generator, the input is a random vector drawn from a standard normal distribution. Denote the generator by $G_\phi(z)$, discriminator by $D_\theta(x)$, the distribution of the real samples by $p(x)$ and the input distribution to the generator by $q(z)$. The binary cross entropy loss with classifier output y and label $\hat{y}$ is

$$L(y, \hat{y}) = -\hat{y} \log y - (1 - \hat{y}) \log(1 - y)$$

For the discriminator, the objective is

$$\min_\theta \mathrm{E}_{x \sim p(x)}[L(D_\theta(x), 1)] + \mathrm{E}_{z \sim q(z)}[L(D_\theta(G_\phi(z)), 0)]$$

For the generator, the objective is

$$\max_\phi \mathrm{E}_{z \sim q(z)}[L(D_\theta(G_\phi(z)), 0)]$$

The generator's objective corresponds to maximizing the classification loss of the discriminator on the generated samples. Alternatively, we can minimize the classification loss of the discriminator on the generated samples when labelled as real:

$$\min_\phi \mathrm{E}_{z \sim q(z)}[L(D_\theta(G_\phi(z)), 1)]$$

Preprint. Under review.

And this will be used in the implementation. The strength of the two networks should be balanced, so we train the two networks alternatively, updating the parameters in both networks once in each iteration.

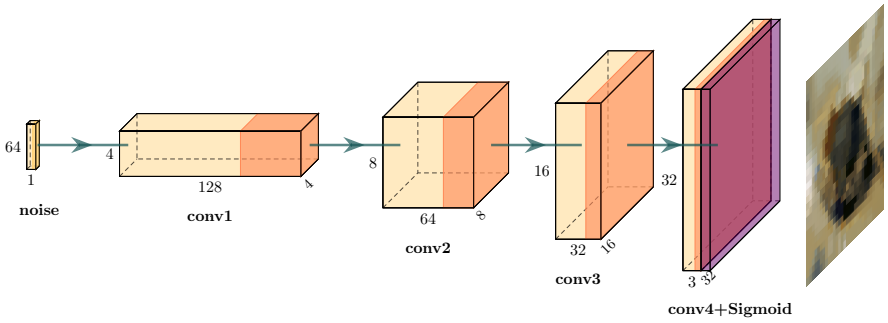

Figure 1: Schematic of the generator network.

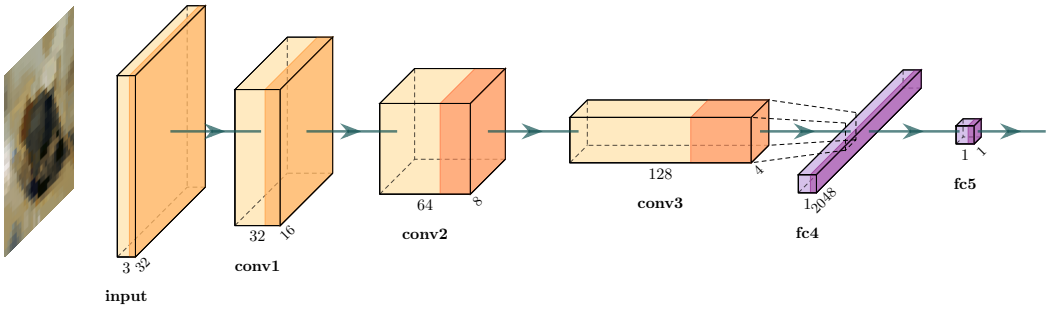

Figure 2: Schematic of the discriminator network.

Both the generator and the discriminator are based on CNN. The generator take a 1x64 noise sampled from a standard Gaussian distribution as input and tries to generate a realistic figure. It is composed of four transposed CNN layers consisting of 128, 64, 32, and 3 channel, respectively.

The discriminator evaluates the input image and determine whether it is real (1) or fake (0). It is composed of four CNN layers consisting of 3, 32, 64 and 128 channel and two fully connected layers with 2048 and 1 neutrons respectively. A sigmoid layer is used after conv4 to limit the output between 0 to 1.

The kernel size is 4×4, the stride is 2 and the padding is 1 for all the convolution layers. Leaky ReLU is used as the activation function for both networks. Compared to ReLU, the gradient of Leaky ReLU is not zero when $x < 0$. This propriety has been proved can improve the performance of GAN.

Besides, batch normalization (BN), which stabilizes learning by normalizing the input to each unit to have zero mean and unit variance, is used for each layer except the first convolution layer (conv1) in the discriminator. Since we put real samples and fake samples in separate batches, if we add a batch normalization layer after conv1, the discriminator would not be able to distinguish two distributions if one can be obtained by applying an isotropic scaling and a translation in color space to the other.

2 Results

The DCGAN is implemented using Pytorch. Adams optimizer is used with 0.02 learning rate and 128 batch size. During the training, we will train the discriminator and the generator alternatively:

- The discriminator classifies both real data and fake data from the generator. The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real. The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.

- The generator generates figures from sampled random noise. The generator loss is calculated from discriminator classification and backpropagated through both the discriminator and generator to obtain gradients. The gradients is used to change only the generator weights.

I trained the GAN for 25 epochs. The loss for discriminator and generator are plotted in Figure 3. Since the two networks are competing for the loss curve to go different directions, so virtually anything can happen. The loss will not decrease and converge to a small value but will variate during the training.

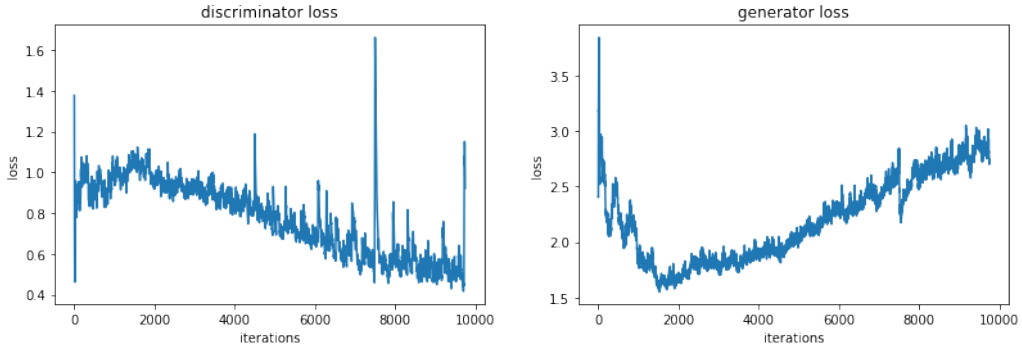

Figure 3: Discriminator loss and generator loss.

Since the loss is not a good candidate for evaluation, I use the trained model to generate 64 samples to evaluate the model. As we can see in the Figure 4, the initial model can only generate very blurred images, while the trained model can generate much clear and meaningful images. Compared to the images in the datset, as shown in Figure 5, the image generated by the trained model already have similar quality. If I increase the training epoch, the performance is almost the same. So 25 epochs is good enough for this task.

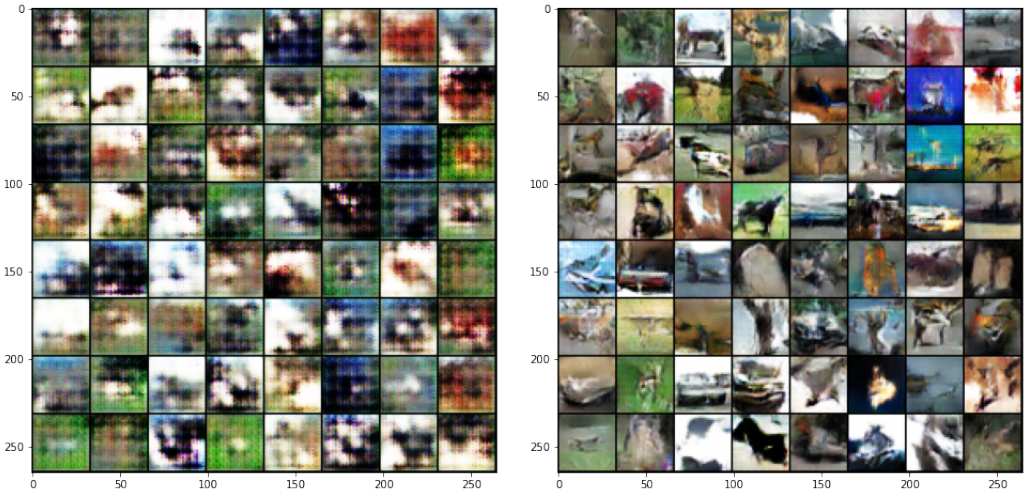

Figure 4: Figures that are generated by the generator before and after training.

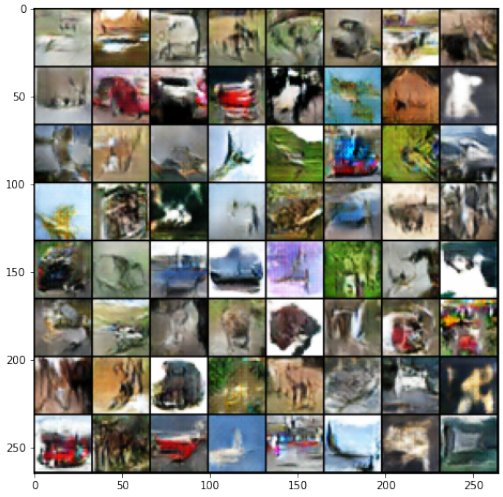

Figure 5: Figures that are sampled from dataset.

3 Conclusion

In conclusion, I demonstrated the use of GAN to generate realistic images. The two networks of generator and discriminator in GAN competitively learn to generate new images based on the CIFAR-10 dataset. Since GAN has a min-max loss function, the architecture and hyperparameters need to be chose carefully in order to make the network converge. Leaky ReLU and Batch Normalization are used for both generator and discriminator. After training, I generate 64 new samples using the generator to evaluate the performance of the GAN. I found that the new figures have high diversity and similar quality as the dataset, which means that the trained GAN can generate new figures very well.

In this project, I just trained the network on a low-resolution dataset. If we want to generate more realistic images, we can train a GAN with more convolution layers on other dataset with high quality images. It has been shown that GAN can even generate 1024×1024 images by using deeper and more complicated networks.