University of Guelph

CIS-6050: Neural Networks

Course Project Final Report

Author: Sina Radpour **Course Professor: Dr. Neil Bruce**

December 15, 2021

1. Introduction

In this project I propose a novel model with combining Recursive Autoencoder with GAN. I created two neural networks as encoder and decoder of autoencoder part, and one additional network which works as a binary classifier for the discriminator. In this way the Autoencoder and the discriminator compete with each other in the game theory manner in order to improve the efficiency and accuracy of reconstructing input in autoencoder. Furthermore, the output of the Autoencoder will be entered to the system as an input of the Autoencoder. My goal here is to fool the discriminator by 50 percent chance for all recursive steps and update both Autoencoder and discriminator's parameters in respect of a competition between them.

2. Background

2.1 Autoencoders:

Autoencoder was first developed in late 1980s for wide range of usage like information processing and pattern recognition (Bourlard & Kamp, 1988). The early purpose of autoencoders was data analysis in term of nonlinear dimensionality reduction. It was introduced as an extension of principal component analysis (PCA) (Kramer, 1991). Autoencoders learning method is unsupervised, so it learns automatically from unlabeled data. It can help us understand underlying structure of the data by extracting relations within the input data (Lopez Pinaya et al., 2020). An autoencoder is consisting of an encoder and a decoder (Figure 1), so that each correlate with their own parameters. The job of the encoder is mapping the input into internal latent representation of the data with lower dimensionality. Then, decoder is used to produce an output with the exact dimensionality of the input. The goal is to make the output similar to the input data as much as possible.

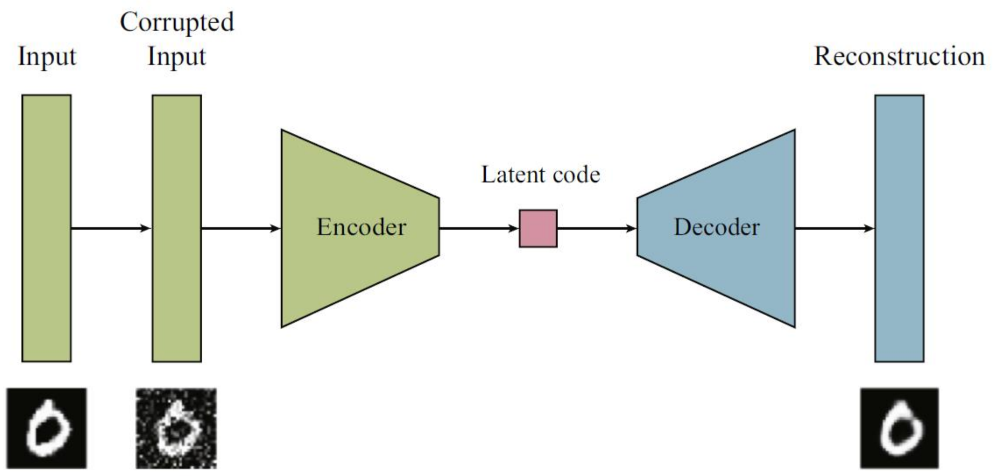


Figure 1: Autoencoder scheme

2.2 GANs:

Generative Adversarial Network is a type of Neural network which is designed for generative modeling problem. The goal of this network is to learn the probability distribution which data points in the input (training data points) can be generated from it (Goodfellow et al., 2020). GAN is consist of a pair of networks, a generator and a discriminator, which competes with each other. It uses both supervised and unsupervised learning techniques (Creswell et al., 2018). A common analogy for visualizing GAN is to presume one network as an art forger and the other one as an expert (Figure 2). On one hand, The forger which is the generator, G, trying to create perfect forgeries to fool the expert. On the other hand, The expert which is the discriminator, D, receives both real arts and forgeries and tries to identify the fake one (Creswell et al., 2018). Unlike other learning techniques which rely on optimizing, GANs use game theory to design loss function. For training this network, the generator and discriminator compete with each other (Tang et al., 2020). GAN produce promising data samples which can be used for tasks such as semantic image editing (Zhu et al., 2016), data augmentation

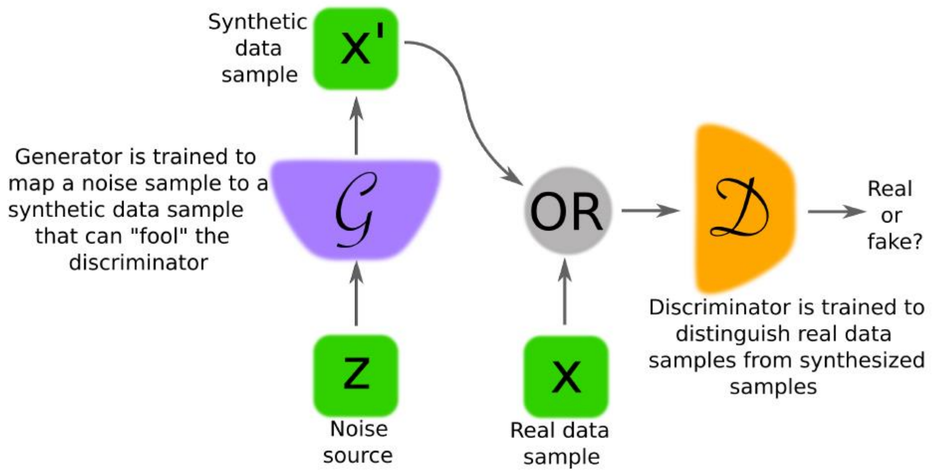

Synthetic data sample
X'
Generator is trained to map a noise sample to a synthetic data sample that can "fool" the discriminator
𝒢
OR
𝒟
Real or fake?
Discriminator is trained to distinguish real data samples from synthesized samples
Z
Noise source
X
Real data sample

Figure 2: GAN scheme

(Bousmalis et al., 2017). It can also be used for tasks such as classification (Radford et al., 2016).

Despite tremendous amount of time spent on the last two subjects, there are few articles focusing on combining these two. Here, I chose to cover the rarely touched upon topics. It will be explained with more elaboration in the following.

3. Methodology

3.1 Autoencoder:

As discussed before, autoencoders first compress the data into a latent representation with lower dimensions. Then, it tries to reproduce the input data. Suppose we have a set of input data points $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ each with many dimensions. The goal of autoencoder is to map the input to some latent representation $\{z^{(1)}, z^{(2)}, \dots, z^{(m)}\}$ which have lower dimensionality than x , and also x can be reconstructed from it (we name the reconstructed data $\bar{x}$). In order to talk about mapping more systematically, I propose z and $\bar{x}$ in the following way:

$$z^{(i)} = W_e\, x^{(i)} + b_e$$

$$\bar{x}^{(i)} = W_d\, z^{(i)} + b_d$$

Where W_e and b_e are related to the encoder part, and W_d and b_d are for decoder part of autoencoder.

Our goal here is to reconstruct $\bar{x}^{(i)}$ in order to approximate $x^{(i)}$. So the loss function is the sum of squared difference between $\bar{x}^{(i)}$ and $x^{(i)}$.

$$L(W_e, b_e, W_d, b_d) = \sum_{i=1}^{m} \left(\bar{x}^{(i)} - x^{(i)} \right)^2$$

$$= \sum_{i=1}^{m} \left(W_d\, z^{(i)} + b_d - x^{(i)} \right)^2$$

$$= \sum_{i=1}^{m} \left(W_d\, (W_e\, x^{(i)} + b_e) + b_d - x^{(i)} \right)^2$$

Therefore, minimizing this difference is the goal here, which can be done by stochastic gradient descent.

3.2 The Discriminator:

Generative Adversarial Network is made of two neural networks – the generator and the discriminator - which compete with each other in sense of game theory (Goodfellow et al., 2020). Presume training examples x has an unknown distribution $p_{input}(x)$. The goal of the generator network is to learn $p_{model}(x)$ in a way that it would be similar to $p_{input}(x)$. as much as possible. The output of the generator is defined by the generator function $G(u; W(G))$ where u is the input of the generator and $W(G)$ is a set of learnable parameters of the generator. In this project the generator is actually the decoder part of the autoencoder.

The other part is the discriminator. The discriminator takes some examples x as input and decide whether x is real (drawn from the training samples) or fake (output of the generator). The discriminator function is $D(x; W(D))$ where $W(D))$) is a set of learnable parameters of the discriminator.

Training of GANs is consist of evaluate both parameters of the discriminator which maximize its accuracy, and also parameters of the generator that can maximally fool the discriminator (Creswell et al., 2018).

The cost of training is defined by a cost function $V(G, D)$ that depends on both the generator and the discriminator. Here, I decided to use minimax GAN that is based on minimax game. This type of training solves the bellow equation,

$$\max_{D} \min_{G} V(G, D)$$

, where

$$V(G, D) = \mathbb{E}_{P_{input}(x)} \log D(x) + \mathbb{E}_{P_{model}(x)} \log(1 - D(x)) .$$

Here $\mathbb{E}_{P_{input}(x)}$ is the expectation over the input distribution, $\mathbb{E}_{P_{model}(x)}$ is the expectation over the autoencoder's output distribution, and $\log D(x)$ is the log likelihood of the discriminator. In the training process, the parameters of one network is frozen while the other network's parameters are being updated (Creswell et al., 2018).

Goodfellow et al.[4] showed that for a generator with probability distribution p_{model} there is a unique optimal discriminator, $D^*(x) = \frac{p_{input}(\mathrm{x})}{p_{input}(x)+p_{model}(x)}$. They also showed that when $p_{input}(\mathrm{x})$ is equal to $p_{model}(\mathrm{x})$, the generator is optimal, which then lead them to the optimal discriminator with $D^*(x) = 0.5$. In other words, the generator is optimal when the discriminator is totally confused between the real samples or the fake ones, which in fact make the discriminator take a 50-50 chance decision.

3.3 Combining Autoencoder with Discriminator:

In this project, I combined autoencoder with GAN in the way that we have the normal autoencoder with a discriminator which defines the accuracy of our resampling (i.e., recreate the input via autoencoder). Unlike the traditional architecture in GAN, in which the generator competes with the discriminator, the whole autoencoder competes with the discriminator in this case. In this matter, after training networks individually, one of the two networks – The autoencoder or the discriminator – is frozen and the other network's parameters is being updated. So, if the discriminator can recognize fake data with more than 50% probability, the discriminator's parameters will be fixed and in order to improve autoencoder, only the autoencoder parameters will change.

As an extension to Autoencoders researchers introduced “Recurrent auto encoder” (Susik, 2021). In the Recurrent Autoencoders the generated output of the decoder will be the input of the autoencoder, and the final output of the autoencoder after k recursive process will be $x'^{(k)}$. In regard of this change, the loss function will be changed to $L(W_e, b_e, W_d, b_d) = (x'^{(k)} - x)^2$

instead of $(x' - x)^2$. The goal here is to minimize the loss function by updating encoder and decoder's parameters.

In this project I proposed a novel model with combining Recurrent Autoencoder with GAN in the way they both compete with each other in order to improve the efficiency and accuracy of reconstructing input in autoencoders. In this matter, all the outputs from decoder (from first output, $x'^{(1)}$ to the last recursive output $x'^{(k)}$) will be the input of the discriminator to decide whether it is fake or real (Figure 3). My goal here is to fool the discriminator by 50 percent chance for all recursive steps and update both Autoencoder and discriminator's parameters in respect of a competition between Recursive Autoencoder and the discriminator.

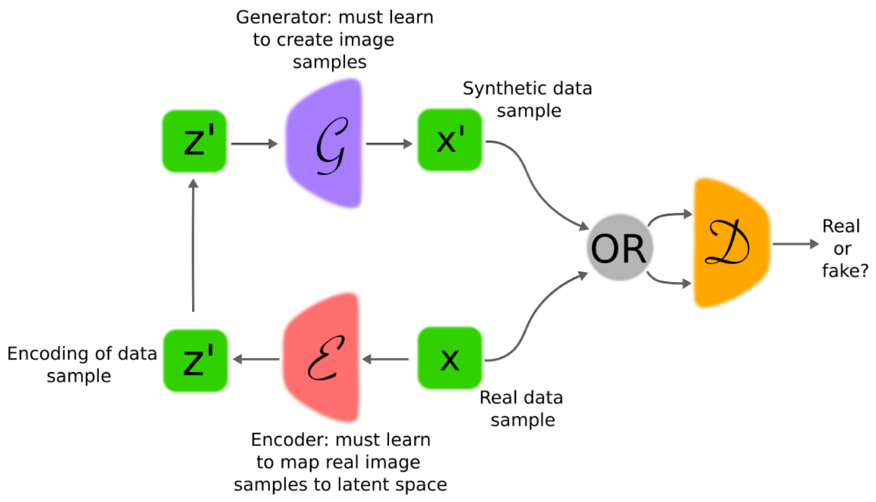


Figure 3: Combining Autoencoder with the Discriminator

4. Implementation

4.1 Dataset:

For implementing this project, I used python language and pytorch library for creating my networks. For the first step, I decided to use MNIST dataset to create a reliable autoencoder. MNIST dataset contains 28*28 pixels images which are greyscale. For the second step, I used CIFAR10 dataset to have colorful RGB images to find out about the performance of the autoencoder when working with RGB images. CIFAR10 dataset has 32*32 pixels images which have 3 channels each of them for blue, green and red respectively. In bellow, I will dive into the details of implemented networks and the results they generated.

It is worth to mention that for the autoencoder of all parts I used Adam optimizer with learning rate equals to 0.005. I also used Adam optimizer for the discriminator with the learning rate of 0.001.

4.2 Autoencoder

In the first part, I implemented an autoencoder with MNIST dataset. The encoder of this part is consist of 3 convolutional layers and 3 maxpool layers in between them which produce

3*3 images with 64 channels at the end, and also 3 fully connected layers which produce 4*4 images. The decoder is consist of 3 fully connected layers which finally produce 3*3 images with 64 channels and then 3 convolutional layers convert them to 28*28 images.

In the next part of the project I decided to change my data set to a more complex dataset than MNIST. Therefore, I changed my dataset to CIFAR10. The autoencoder of this part is consist of an encoder and a decoder, each of them has 7 convolutional layers. I used 3*3 filter with padding equals to 1 and stride equals to 1, in order to maintain the dimensions of the data. Also, 4*4 filter with padding equals to 2 and stride equals to 2 was used to half the width and height of the images. Therefore, for the encoder, after first layer the data have 8 channels and after each 2 layer the channels of the data will be doubled as the dimension of them will decrease to ${}^{1}/_{4}$ its size. For the discriminator, layers act as reverse of the encoder layers. Finally, the images dimensions will be decreased to ${}^{2}/_{3}$ its size.

4.3 Recursive Autoencoder

In this part, I used the previous autoencoder to feed it's output to the network again as the input and build the recursive autoencoder. I implemented recursive autoencoder with 2 recursive steps. For the training, in each epoch when iterating through the data, for each data I compute the loss with different recursive steps and then change the weights of the network for optimizing. Also, I start this way of training randomly with 0, 1 or 2 steps. In this way, assume that it starts training with 0 recursive step, for the first image it computes loss without any recursive steps, for the second image with 1 recursive step, for the third image with 3 steps, and this cycle will start again for the rest of the images. As a result of this way of training, the loss function will be the sum of losses with 0, 1, and 2 steps with same weights.

4.4 Autoencoder with GAN

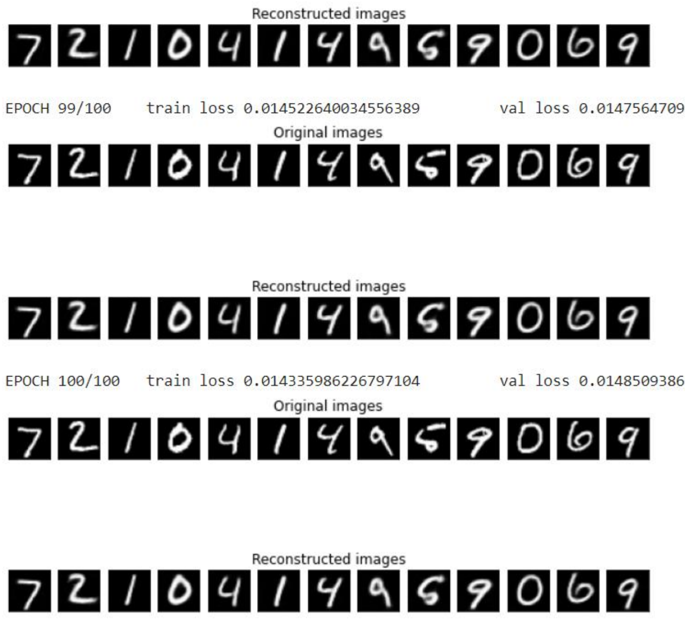
In this part, first, I tried the MNIST dataset too. For the GAN part I built a discriminator which was binary classifier and consist of four fully connected layers which convert 28*28 images into just one node in order to classify the images as fake or real. I implemented GAN in a way that discriminator competes with the whole autoencoder not just the decoder part. Training this networks is based on game theory rather than optimization The autoencoder of this part was the same autoencoder used before fore the MNIST dataset.

I also the autoencoder with GAN for the CIFAR10 dataset. Here, the discriminator is just like the discriminator for the MNIST data set and the autoencoder is the same as the autoencoder used before for CIFAR10 dataset. The way of training this network is a little different than ususal GANs. First, I train the autoencoder in order to create good reconstructed images. Then, I train the discriminator with real data which is the CIFAR10 dataset and fake data which is data that had been reconstructed with the use of autoencoder. After that, the autoencoder will be trained again in order to produce the output which fools the discriminator. Therefore, the first part is unsupervised learning for autoencoder and the second part is supervised learning for both discriminator and autoencoder.

5. Results

5.1 Autoencoder

For the MNIST dataset, I used Mean Squared Error for the loss function. The loss of the auto encoder after training it with 3000 images and 100 epochs and testing with 600 images was 0.014. The reconstructed images had great shapes and luminosity (because the images are greyscale in MNIST dataset). However, I needed to change the network architecture in order to have a better performance for CIFAR dataset. The diagram of the training loss and validation loss and reconstructed images for the last two epochs can be found in Fig 5 and Fig 6 respectively.



```
EPOCH 99/100    train loss 0.014522640034556389        val loss 0.0147564709
```

```
EPOCH 100/100   train loss 0.014335986226797104        val loss 0.0148509386
```

Figure 4

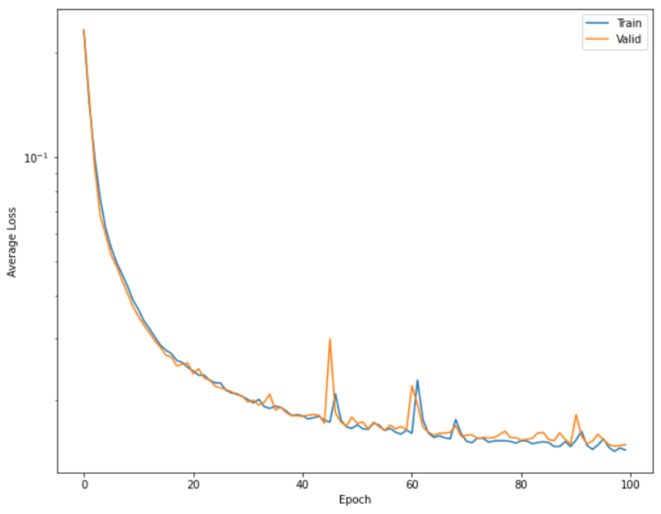
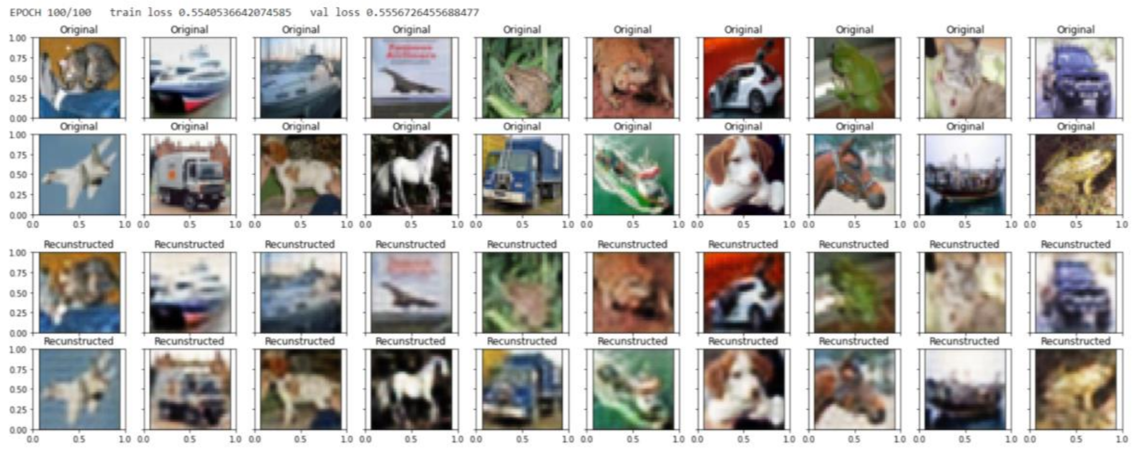


Figure 5

For CIFAR10 dataset, I used Binary Cross Entropy for the loss function. The loss of the auto encoder after training it with 10000 images and 200 epochs and testing with 2000 images was 0.553. The reconstructed images had a little shapes and the color was faded a little bit as a result of the compressive nature of the autoencoder. The reconstructed images for the last epoch and diagram of the training loss and validation loss can be found in Figure 4 and Figure 7 respectively.



```
EPOCH 100/100   train loss 0.5540536642074585   val loss 0.5556726455688477
```

Figure 6

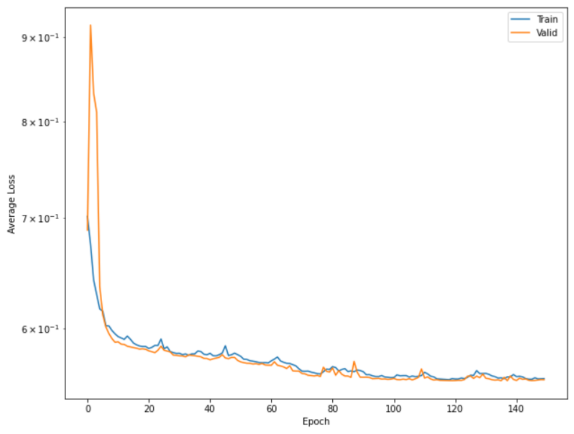


Figure 8

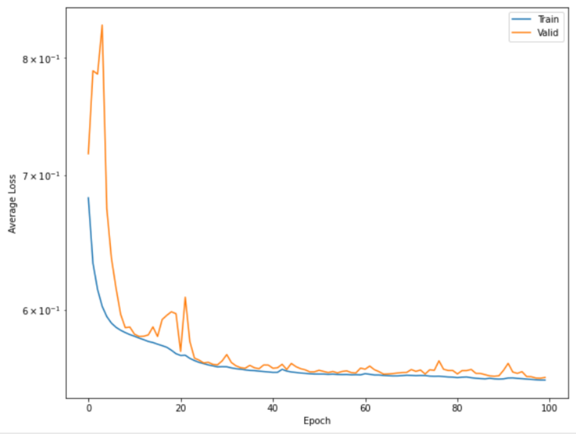
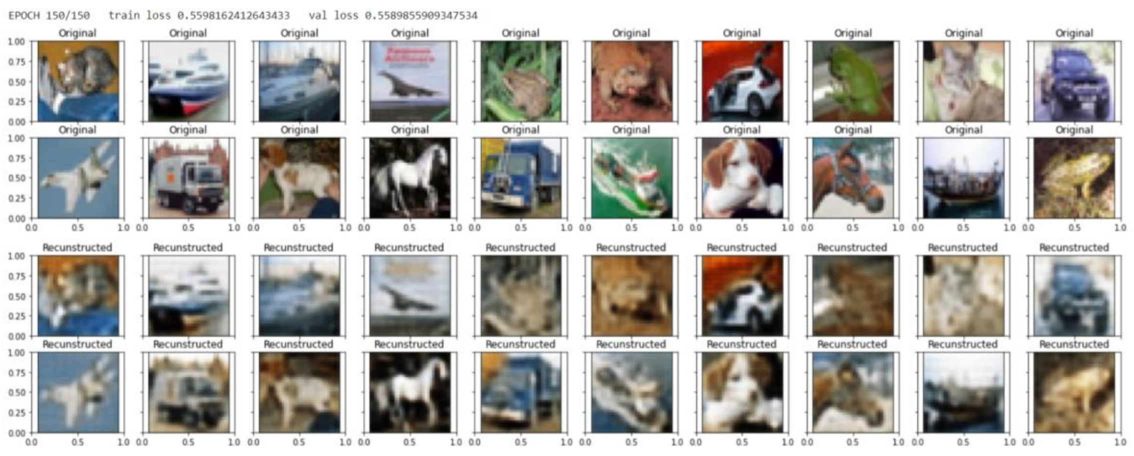


Figure 7

5.2 Recursive Autoencoder

For the recursive part, I wanted to observe how this way of training the autoencoder will affect the performance of it. I used BCE for the output of the autoencoder without any recursive steps. After training the recursive autoencoder with 10000 images and 150 epochs, the loss was 0.559. The reconstructed images have better shapes than autoencoder without any recursive steps, but the color of the images became faded a little bit. The diagram of the training loss and validation loss and reconstructed images for the last epoch can be found in Figure 8 and Figure 9 respectively.



```
EPOCH 150/150   train loss 0.5598162412643433   val loss 0.5589855909347534
```

Figure 9

5.3 Autoencoder with GAN

For the last part, I implemented it with both MNIST dataset and CIFAR10 dataset. For MNIST dataset, I used BCE to for the loss function of the discriminator and MSE for the autoencoder. The loss of the auto encoder after training it with 7000 images and 200 epochs and validating it with 2000 images was 0.509. The reconstructed images got a little blurry and the loss is higher than the regular autoencoder with MNIST dataset. It may be because of the perfect discriminator I had for the MNIST dataset. The discriminator loss here was 0.02, and the autoencoder could not fool the discriminator. This may be because of the fact that MNIST dataset images are not complex. The reconstructed images for the last epoch can be found in Figure 11.

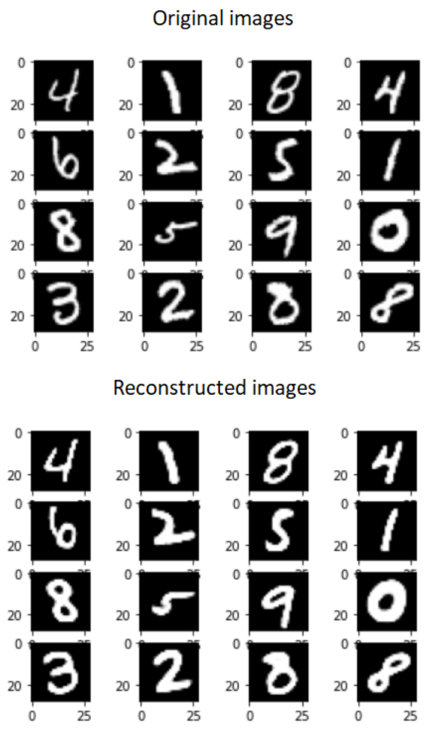


Figure 11

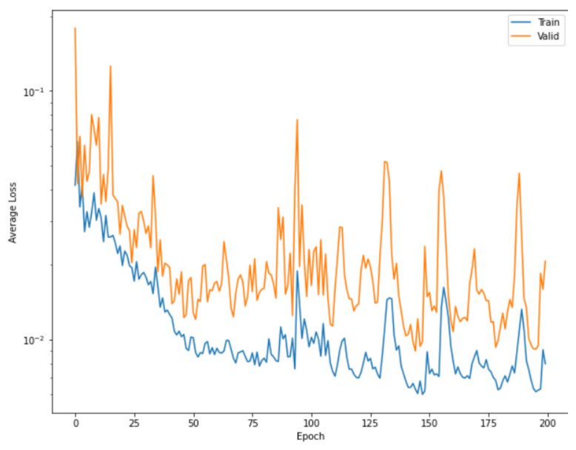


Figure 10

When using CIFAR dataset, I used BCE for loss too. In each epoch, first I train autoencoder just like before, then train the discriminator and autoencoder with making them compete with each other. After training was over with 20000 images and 20 epochs, I tested just the autoencoder to evaluatre its performance. The loss of the autoencoder was 0.567. The reconstructed images lost color in some cases than autoencoder without the discriminator, and the images became blurrier. The reconstructed images for the last epoch and diagram of the training loss and validation loss can be found in Figure 11 and Figure 12 respectively.

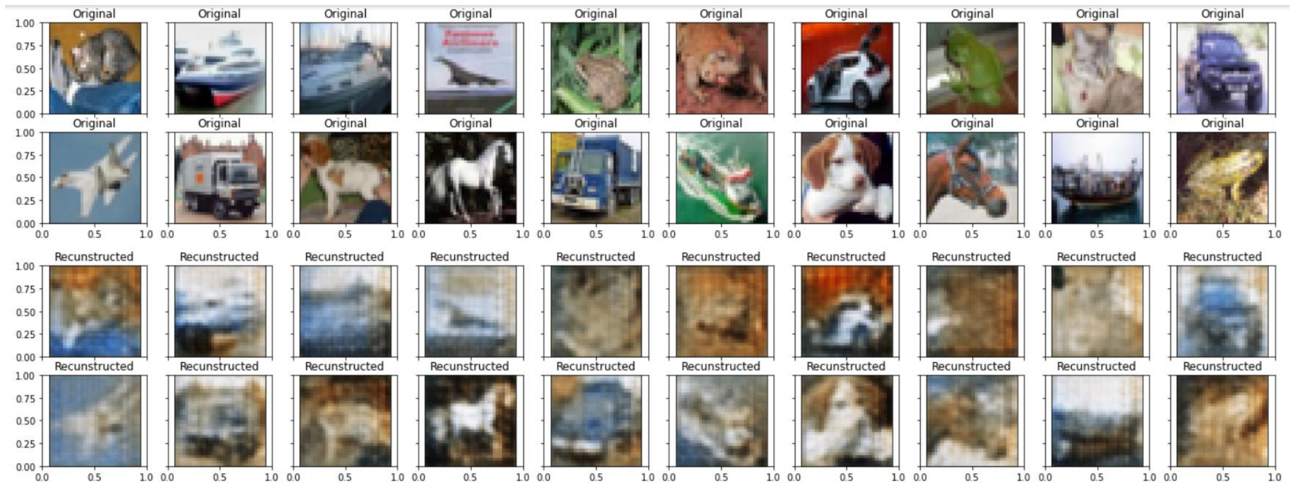


Figure 12

5.4 Comparing three networks

In conclusion to the results, three networks should be compared to each other. As mentioned above, losses for three networks is close to each other and no major differences could be found. The reconstruction of images for all three networks are look the same, however, autoencoder without GAN and recursive steps looks like to perform a little better than the other with reconstruction of shapes. Autoencoder with recursive steps looks like to generate colors better than the others but the shapes got blurrier. Finally, the combination of autoencoder with GAN does not outperform any of other networks. It could be because of the fact that recursive autoencoder and autoencoder with GAN need more data and epochs to learn the reconstruction better, but I used Google colab and had a limited amount of RAM and computational power. The tables comparing three networks for CIFAR dataset can be found in Table 1.

Table 1

Neural Network Type	Average Loss (BCE)
Autoencoder	0.553
Recursive Autoencoder	0.559
Autoencoder with GAN	0.593

6. Conclusion

In this project, I combined autoencoder with GAN in the way that we have the normal autoencoder with a discriminator which defines the accuracy of our resampling. Unlike the traditional architecture in GAN, in which the generator competes with the discriminator, here the whole autoencoder competes with the discriminator in the game theory manner. First I implemented autoencoder, then recursive autoencoder with the use of that particular autoencoder. After those two steps, I built a discriminator and make it compete with the autoencoder. For the first attempt I used MNIST dataset and then switch to CIFAR dataset to observe the network's performance for RGB images. As a result of above mentioned implementation, we can see that images has better shape when reconstructing images with recursive autoencoder, but their color becomes faded a little bit. I believe that by working in a better programming environment and have more RAM and better performance GPUs, deeper and more complex network can be used to have a better performance. As we can see the recursive autoencoder reconstruct shapes better, so there is high hope to have a better performance with recursive autoencoder.

7. Limitation and Future Works

I used Google Colab to run my codes. One of the important limitation was limited and low amount of RAM that Google Colab allow you to use. Because of that, I could not train my networks with larger datasets or more epochs. Perhaps using a better device which has more RAM and more GPU cores will improve the accuracy of the networks. With using a better environment for running codes, deeper and more complex architecture can be used as well as larger dataset. In future works, I might use larger more complex image datasets like ImageNet. Having a deeper network with more convolutional layers is another work that could be done to help improving the autoencoder. Also, it is worth to mention that fully connected layers could be add to this network to see the results of adding fully connected layers to the CNN that has been built before.

References

Radford, A., Metz, L., Chintala, S. (2016). Unsupervised Respresentation Learning with Deep Concolutional Generative Adversarial Network, http://arxiv.org/abs/1511.06434

Bourlard, H., & Kamp, Y. (1988). Auto-association by multilayer perceptrons and Singular Value Decomposition. *Biological Cybernetics*, *59*(4-5), 291–294. https://doi.org/10.1007/bf00332918

Bousmalis, K., Silberman, N., Dohan, D., Erhan, D., & Krishnan, D. (2017). Unsupervised pixel-level domain adaptation with generative Adversarial Networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr.2017.18

Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., & Bharath, A. A. (2018). Generative Adversarial Networks: An overview. *IEEE Signal Processing Magazine*, *35*(1), 53–65. https://doi.org/10.1109/msp.2017.2765202

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2020). Generative Adversarial Networks. *Communications of the ACM*, *63*(11), 139–144. https://doi.org/10.1145/3422622

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative Neural Networks. *AIChE Journal*, *37*(2), 233–243. https://doi.org/10.1002/aic.690370209

Lopez Pinaya, W. H., Vieira, S., Garcia-Dias, R., & Mechelli, A. (2020). Autoencoders. *Machine Learning*, 193–208. https://doi.org/10.1016/b978-0-12-815739-8.00011-0

Susik, R. (2021). Recurrent autoencoder with sequence-aware encoding. *Computational Science – ICCS 2021*, 47–57. https://doi.org/10.1007/978-3-030-77964-1_4

Tang, T.-W., Kuo, W.-H., Lan, J.-H., Ding, C.-F., Hsu, H., & Young, H.-T. (2020). Anomaly detection neural network with dual auto-encoders gan and its industrial inspection applications. *Sensors*, *20*(12), 3336. https://doi.org/10.3390/s20123336

Zhu, J.-Y., Krähenbühl, P., Shechtman, E., & Efros, A. A. (2016). Generative visual manipulation on the natural image manifold. *Computer Vision – ECCV 2016*, 597–613. https://doi.org/10.1007/978-3-319-46454-1_36