Analysis of GANs approaches on MNIST dataset

Harikrishnan Seetharaman, Jemuel Stanley Premkumar
hariksee@umich.edu, jemprem@umich.edu

I. INTRODUCTION

Generative Adversarial Networks (GANs) are one of the latest and most powerful areas of unsupervised learning and generative modeling. A typical GAN has two types of neural network inspired by the game theory, one network is known as the generator G that generates data based on a model it has created using samples of real data received as input. The generator learns the underlying structure by using a number of parameters significantly smaller than the amount of data it has trained on, which is a core objective of the unsupervised learning strategy. The other network known as the discriminator D discriminates between data created by the generator and data from the true distribution. The two networks are locked in a zero-sum game where the generator is trying to fool the discriminator into thinking that the synthetic data comes from the true distribution, and the discriminator is trying to call out the synthetic data as fake.

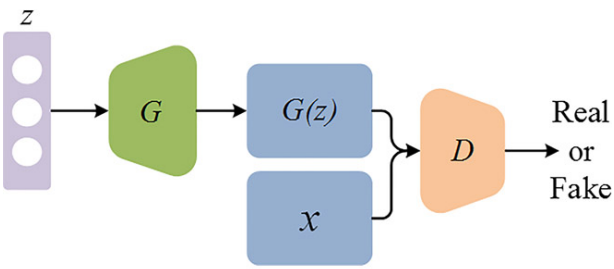

Fig. 1: Generative Adversarial Modeling

GANs provide a path to sophisticated domain-specific data augmentation and a solution to problems that require a generative solution, such as image-to-image translation. Implementing various GANs would allow us to understand the popular approaches practiced in generative modeling. From a theoretical perspective, this would not only allow one to understand different losses and evaluation metrics but also understand the underlying math corresponding to latent spaces.

Coming from a robotics background, this project would allow us to develop models for applications such as generating grasp rectangles using Pix2Pix [1] or for generating depth images from RGB images directly as most of the 6-DOF grasping policies require both the RGB image and its corresponding depth information[2].

The project lays importance on training 5 different GAN approaches practiced in the field of machine learning currently, understanding the loss curve of each and every one of them, and also producing the generative output of the MNIST handwritten dataset. The five networks trained so far were implemented using Tensorflow 2.0 framework and are listed in the order as shown.

1) Deep Convolutional Generative Adversarial Network (DCGAN)
2) Wasserstein Generative Adversarial Network (WGAN)
3) Least Squares Generative Adversarial Networks (LSGAN)
4) Information Maximizing Generative Adversarial Network (InfoGAN)
5) Style GAN

II. METHODS

A. DCGAN

DCGAN is the vanilla form of a Generative Adversarial Network. They use convolutional and convolutional-transpose layers in the generator and discriminator respectively. The architecture used for the generator and the discriminator are given in Fig 2 and Fig 3 respectively. The DCGAN's fundamental idea is to replace the fully connected layers in the generator with an upsampling convolutional layer. The other additional component the paper proposes is to use the Batch Normalization layers in both the generator and discriminator. Both the discriminator and the generator used the simple cross-entropy loss calculated on fake output for the generator and the sum of both real and fake output for the discriminator. The optimizer used for training is ADAM with a seed value of 0.0001. The latent dimension of the noise is 100 and the total number of epochs used for training is 50.

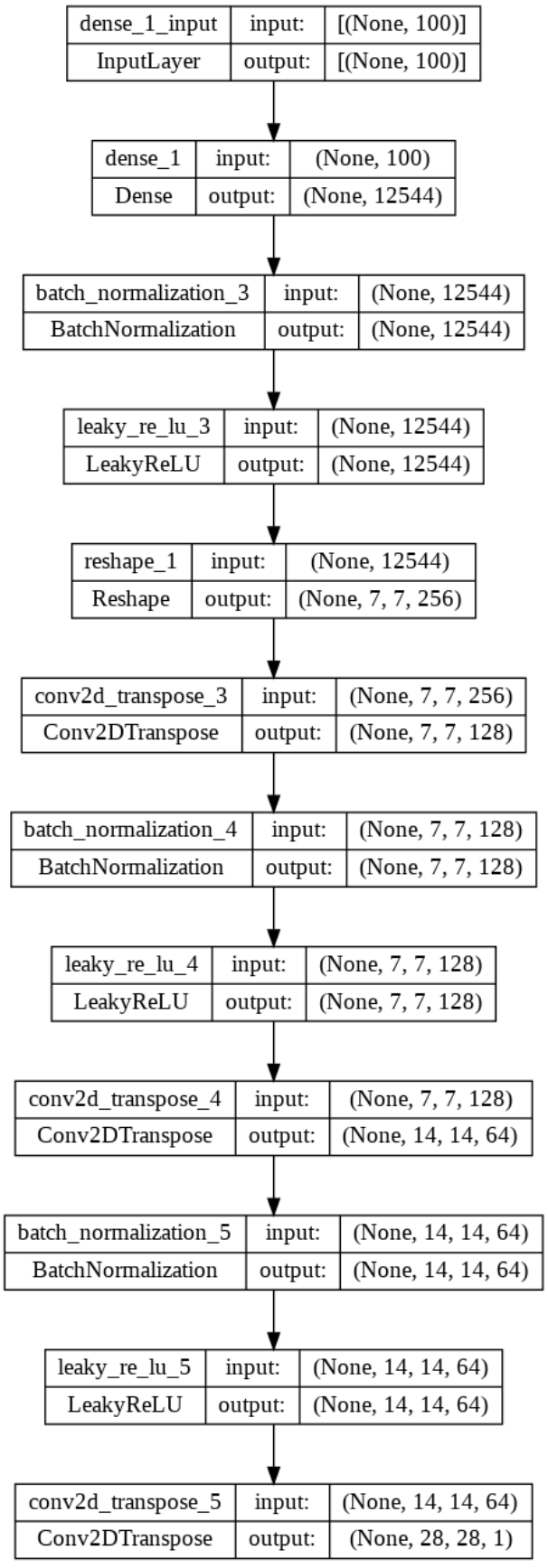

Fig. 2: DCGAN Generator Model Summary

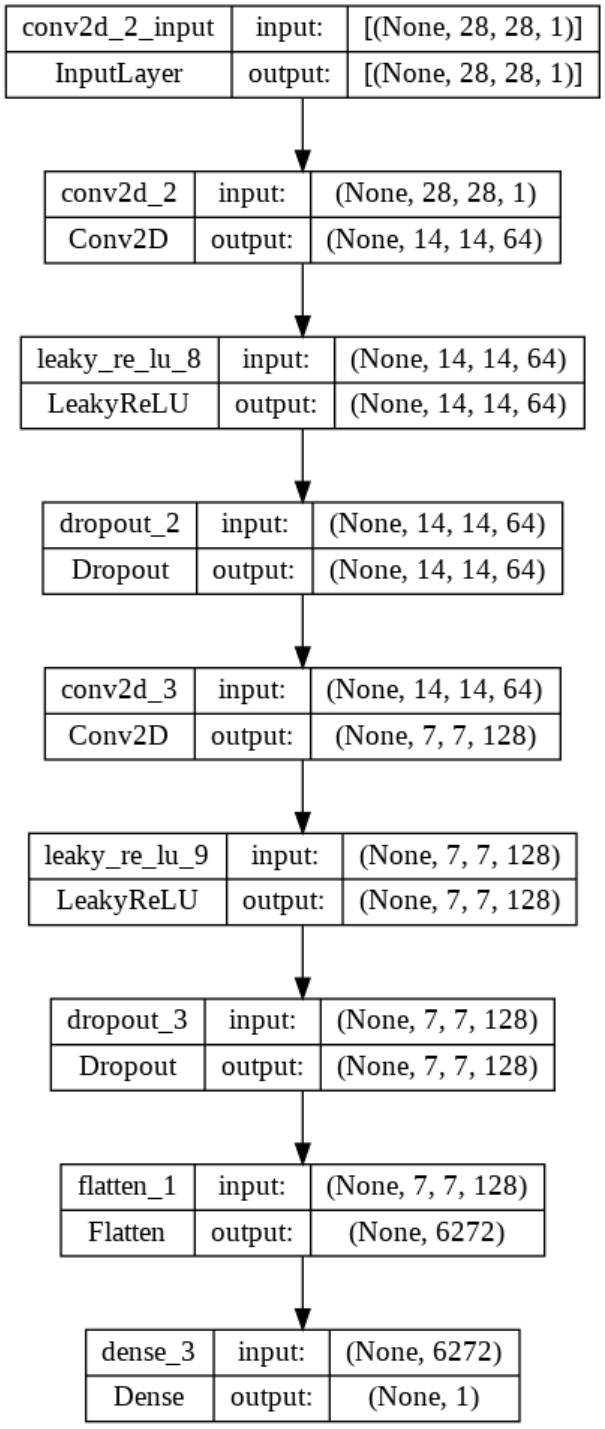

Fig. 3: DCGAN Discriminator Model Summary

The training was successful and the output generated from DCGAN is shown in Fig 4

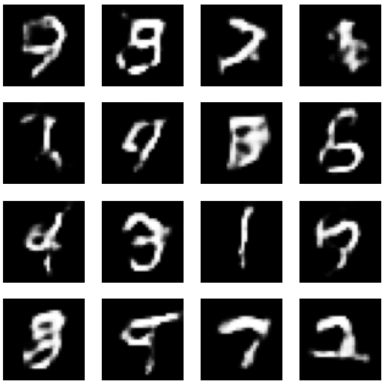

Fig. 4: DCGAN generator output sample

B. WGAN

The Wasserstein GAN proposes a new training method that works for really common GAN datasets. The paper proposes a new name for discriminator as **critic** which instead of doing a classification approach i.e, (1 or 0) gives a score for the image generated based on realness or fakeness of data. Most GAN approaches use the traditional KL divergence ref Fig 1 for doing the optimization problem but there are flaws with this approach when Q(x) = 0 at an x where P(x) > 0, the divergence approaches ∞. The WGAN instead proposes a new approach of distance metric called the **Earth Mover Distance** which can be formulated as the supremum taken over all 1-Lipschitz functions ref Fig 2.

KL Divergence is given by:

$$KL(P||Q) = \int_x P(x) \log \frac{P(x)}{Q(x)} \, dx \tag{1}$$

The wasserstein distance is given by:

$$W(P_r, P_\theta) = \sup_{f_L \leq 1} E_{x \sim P_r}[f(x)] - E_{x \sim P_\theta}[f(x)] \tag{2}$$

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.
while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Fig. 5: WGAN Algorithm

The WGAN paper also proposes a strategy called weight clipping in order to keep the weights in a certain range after every mini-batch update [-0.01,0.01] and also update the critic model more times than the generator each iteration. As proposed in the paper RMSProp was used as the optimizer with a really small learning rate of 000005 and no momentum. The model faced difficulty to converge when using Batch Normalization layers so they were removed as part of our implementation and the loss curve during the entire training phase with a latent dimension of 50 is as in Fig 6

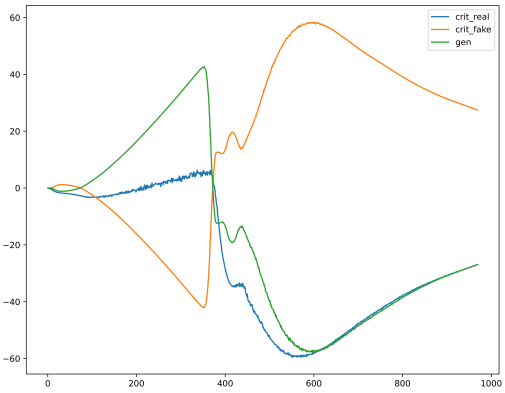

Fig. 6: Wasserstein Loss Curve

1) Wasserstein Loss Curve Inference: The main trend in the WGAN loss curve is that the loss directly correlates with the quality of the image generated. Lower loss defined better quality of generated images. There are a lot of fluctuations in the range of 300-400 epochs and after that, the generator distribution tries to almost fully follow the distribution of the Real critic. One general trend observed while training the WGAN network is the more epochs we let it run, the quality of images improves gradually as well. After say 900 epochs the WGAN produces plausible results for the numbers Fig 7 shows the output for single class label 7, although it is not perfect. The model also took multiple trials as it ran into mode collapse twice in the process of training.

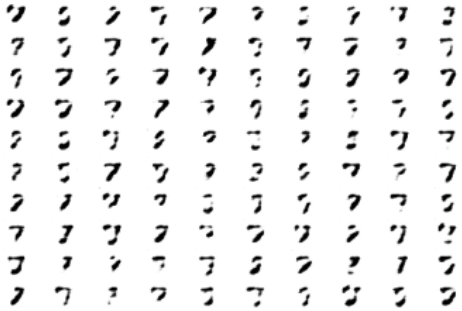

Fig. 7: Wasserstein Generator Output

The architecture used for the generator and the critic are described in 8 and 9 respectively.

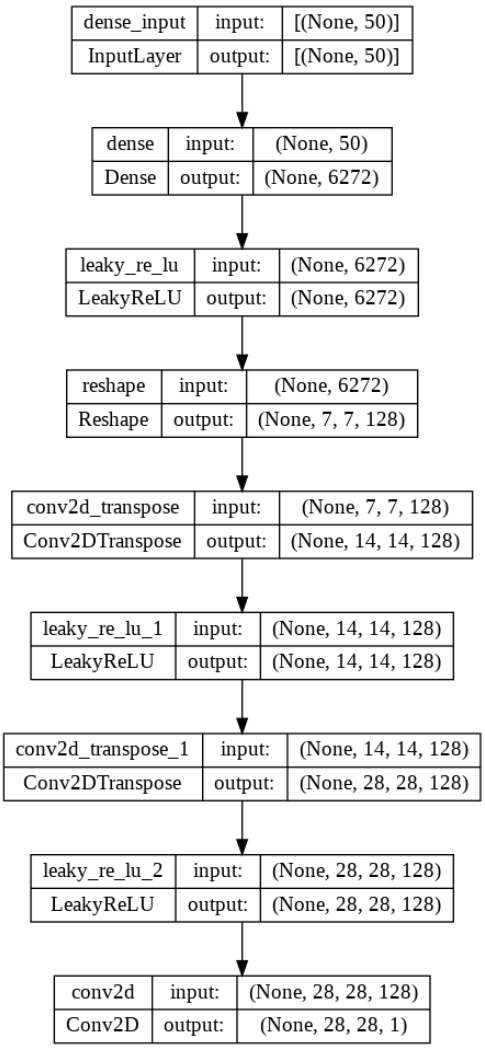

Fig. 8: Wasserstein Generator Architecture

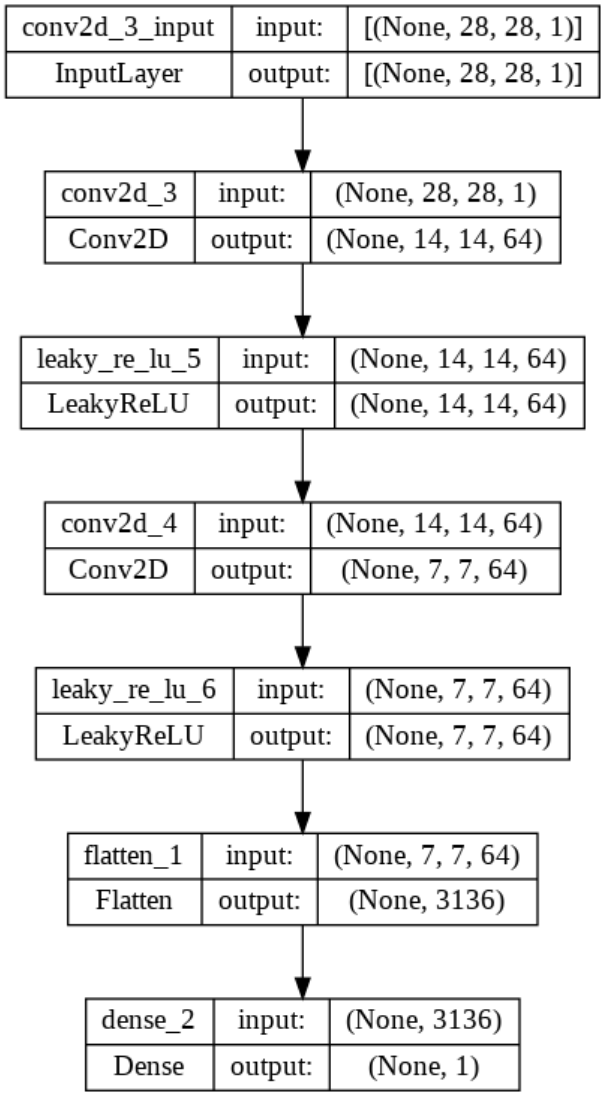

Fig. 9: Wasserstein Critic Architecture

C. LSGAN

Typically, minimizing the objective function of regular GANs suffer from vanishing gradients for the samples that are on the correct side of the decision boundary, but are still far from the real data., which makes it hard to update the generator. Since LSGANs adopt least squares loss function for the discriminator to remedy this problem it is able to move the fake samples toward the decision boundary, because the least squares loss function penalizes samples that lie in a long way on the correct side of the decision boundary. Another benefit of LSGANs is the improved stability of learning process. If a and b are the labels for fake data and real data, then the objective functions for LSGANs can be defined as follows:

$$\min V_{LSGAN}(D) = \frac{1}{2}\mathbb{E}_{x\sim p_{data}(x)}[(D(x)-b^2)] + \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[(D(G(z))-a^2)] \quad (3)$$

$$\min V_{LSGAN}(G) = \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[(D(G(z))-c^2)] \quad (4)$$

Here c denotes the value that G wants D to believe for fake data. One method to select these parameters a,b,c is to satisfy $b-c=1$ and $b-a=2$, such that minimizing Eqn.(4) yields minimizing the Pearson divergence.

1) Model Architecture:: LSGAN has the same architecture as that of WGAN and the same has been described in Fig. 9,8).

2) Implementation Details:: The optimizer used for training is ADAM with a seed value of 0.0002. The latent dimension of the noise is 100 and the total number of epochs used for training is 588.

D. Least Squares Loss Curve Inference:

It was observed that the generator was producing convincing results after 400 epochs. This can also be inferred from the loss curve; the generator loss decreased after 400 epochs and subsequent epochs resulted in the generator converging toward the distribution of the real data.

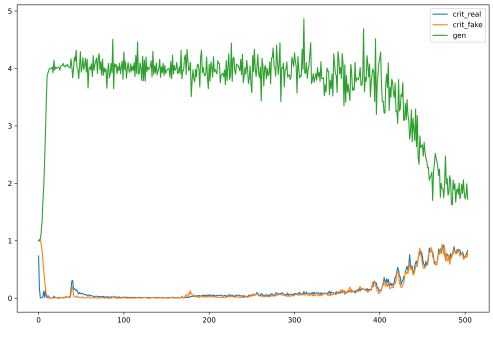

Fig. 10: LSGAN Loss vs Epochs curve

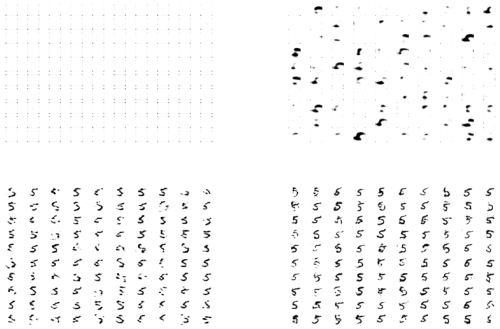

Fig. 11: LSGAN Generator results through multiple epochs for the class 5

E. StyleGAN

The Style Generative Adversarial Network is an extension to the GAN architecture that proposes significant changes to the generator model, including the use of a mapping network to map points in latent space to an intermediate latent space. It also uses an intermediate latent space to control style at each point in the generator model while introducing noise as a source of variation at each point in the generator model. The resultant model is capable of not only creating remarkably photo-realistic high-quality images of faces, but also of providing control over the style of the created image at different degrees of detail by adjusting the style vectors and noise.

1) Architecture:: The StyleGAN is described as a progressive growing GAN architecture with five modifications:

1) Baseline Progressive GAN
2) Addition of tuning and bilinear upsampling
3) Addition of mapping network and AdaIN (styles)
4) Removal of latent vector input to generator
5) Addition of noise to each block
6) Addition Mixing regularization

The StyleGAN generator no longer accepts a latent space point as input; instead, it employs two new randomness sources to create a synthetic image: an independent mapping network and noise layers. The output of the mapping network is a vector defining the styles that are combined at each point in the generator model via a new layer called adaptive instance normalization. This style vector gives control over the style of the resultant picture. Stochastic variation is introduced through noise added at each point in the generator model. The noise is added to entire feature maps that allow the model to interpret the style in a fine-grained, per-pixel manner. Fig. 13

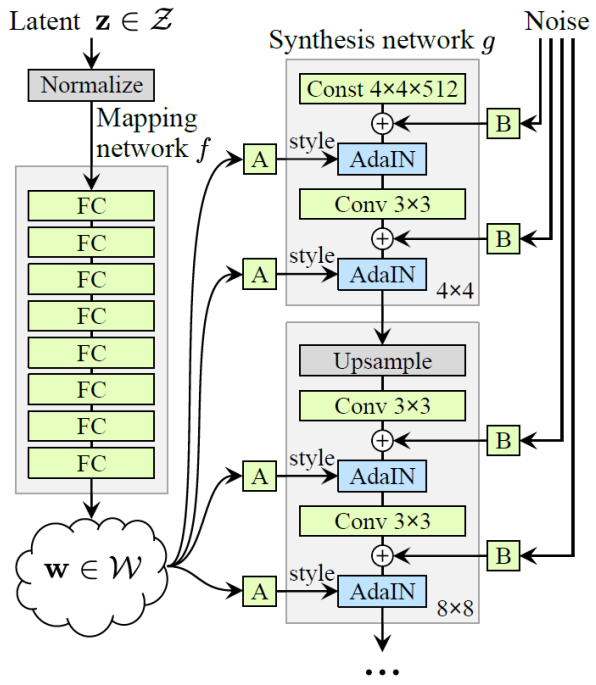

Fig. 12: Architecture of the generator in StyleGAN

This per-block incorporation of style vector and noise allows each block to localize both the interpretation of style and the stochastic variation to a given level of detail. While a traditional generator feeds the latent space through the input layer only, in style GAN the input is mapped to an intermediate latent space W, which then controls the generator through adaptive instance normalization ($AdaIN$) at each convolution layer. Gaussian noise is added after each convolution, before evaluating the nonlinearity. Here "A" stands for a learned affine transform, and "B" applies learned per-channel scaling factors to the noise input. The AdaIN operation is given by:

$$AdaIN(\mathcal{X}_i, y) = y_{s,i} \frac{\mathcal{X}_i - \mu(\mathcal{X}_i)}{\sigma(\mathcal{X}_i)} + y_{b,i} \qquad (5)$$

where each feature map $\mathcal{X}_i$ is normalized separately, and then scaled and biased using the corresponding scalar components from style y.

In order for the styles to localize, a mixing regularization is implemented, where a given percentage of images are generated using two random latent codes instead of one during training. When generating an image, the latent codes are switched to another, which is referred to as style mixing. To be specific, two latent codes, $z1$ and $z2$ are run through the mapping network, and have the corresponding $w1$, $w2$ to control the styles so that w1 applies before the crossover point and w2 after it. This regularization technique prevents the network from assuming that adjacent styles are correlated.

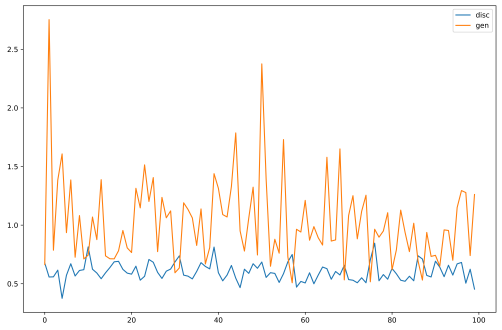

Fig. 13: Loss vs Epoch curve for StyleGAN

Fig. 14: Generated images by StyleGAN

F. InfoGAN

Information Maximizing GAN is another important development in the field of Deep Generative modeling. In general, the generator model takes random noise from latent space that typically limits gaussian variables and they apply during the training phase to generate the images from the real distribution P_r. The Info GAN was proposed to give some control variable as input to the generator along with the latent space noise in order to constrain the generator to learn the relation between the latent space and these constraint variables as well. Consequently, the idea is to use this set of control variables to influence specific properties of the generated images. Mutual information refers to the amount of information learned about one variable and shared with another variable, specifically in this problem the question of interest is the information about the control variables given the image generated using noise and the control variables.

$$MI = Entropy(c) - Entropy(c; G(z, c)) \qquad (6)$$

The paper uses a separate model called Q (Auxilliary model) to train the generator via mutual information, but the auxiliary model predicts the control codes than working with real or fake images. The loss function should have the component of mutual information on the Gaussian control codes. The Q model can

implement the prediction of continuous control variables with a Gaussian distribution, where the output layer is configured to have one node, mean, and one node for standard deviation. The paper uses a separate model called Q (Auxilliary model) to train the generator via mutual information, but the auxiliary model predicts the control codes than working with real or fake images. The loss function should have the component of mutual information on the Gaussian control codes. The Q model can implement the prediction of continuous control variables with a Gaussian The control variables used in Info GAN are basically of two types.

- Categorical control variable
- Continuous control variable

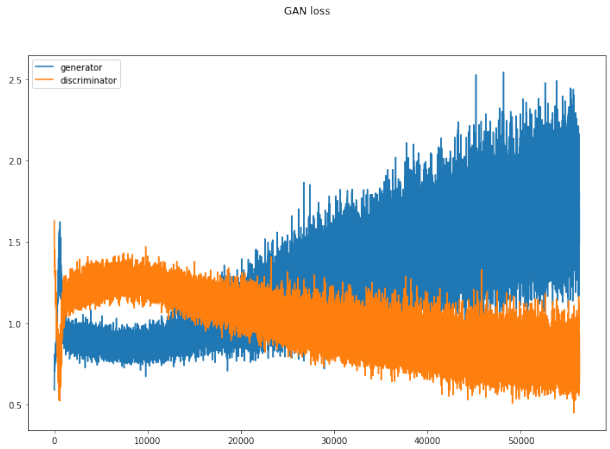

Fig. 15: Info GAN Loss Curve

The categorical variables are encoded as one hot vector say in the case of the MNIST handwritten dataset the network could be tuned such that one categorical variable is responsible for generating one digit from the generator when given as the input. On the other hand, the continuous control variable is used to control the style of the image produced this could be like producing images of different thicknesses and orientations. Ref Fig 15 for the loss curve for Info GAN trained for 60 epochs. Fig 16 shows the effect of varying the categorical variable, Fig 17 and Fig 18 shows the effect of varying the continuous control input c1 and c2 left to right. It can be seen that each categorical one hot input corresponds to producing one number's image from the generator and the continuous variables control the rotation and thickness of the image produced by the generator.

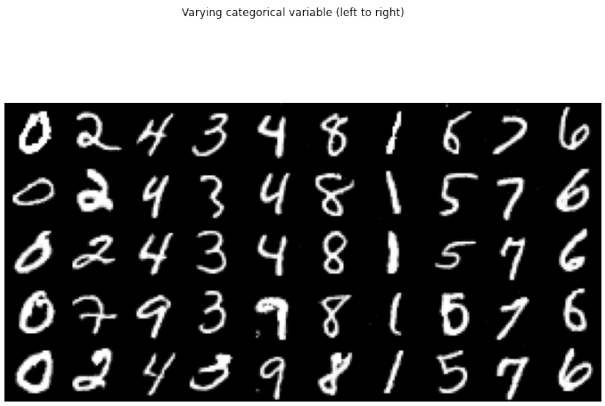

Fig. 16: Categorical Control Input Variation

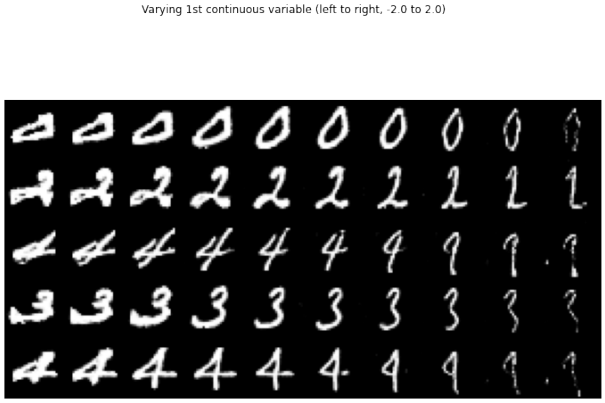

Fig. 17: First Continuous Input Variation - Rotation

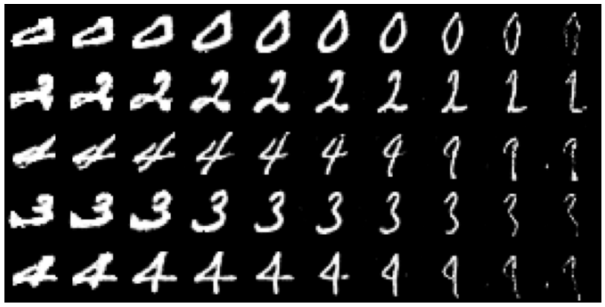

Fig. 18: Second Continuous Input Variation - Thickness

III. CONCLUSION

Hence, the 5 different famous GAN approaches have been implemented and the output of the generators of the MNIST handwritten dataset has been reported.
LINK: https://gitlab.eecs.umich.edu/hariksee/gans-eecs-498

REFERENCES

[1] V. Kushwaha, P. Shukla, and G. C. Nandi, "Generating quality grasp rectangle using pix2pix gan for intelligent robot grasping," 2022. [Online]. Available: https://arxiv.org/abs/2202.09821

[2] L. Chen, S. Lin, Y. Xie, Y. Lin, W. Fan, and X. Xie, "DGGAN: depth-image guided generative adversarial networks for disentangling RGB and depth images in 3d hand pose estimation," *CoRR*, vol. abs/2012.03197, 2020. [Online]. Available: https://arxiv.org/abs/2012.03197

[3] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[4] J. Brownlee, *Generative Adversarial Networks with Python: Deep Learning Generative Models for Image Synthesis and Image Translation*. Machine Learning Mastery, 2019. [Online]. Available: https://books.google.com/books?id=YBimDwAAQBAJ

[5] Q. Xu, G. Huang, Y. Yuan, C. Guo, Y. Sun, F. Wu, and K. Q. Weinberger, "An empirical study on evaluation metrics of generative adversarial networks," *CoRR*, vol. abs/1806.07755, 2018. [Online]. Available: http://arxiv.org/abs/1806.07755

[6] A. Borji, "Pros and cons of GAN evaluation measures: New developments," *CoRR*, vol. abs/2103.09396, 2021. [Online]. Available: https://arxiv.org/abs/2103.09396

[7] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, and Z. Wang, "Multi-class generative adversarial networks with the L2 loss function," *CoRR*, vol. abs/1611.04076, 2016. [Online]. Available: http://arxiv.org/abs/1611.04076

[8] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," 2017. [Online]. Available: https://arxiv.org/abs/1701.07875

[9] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015. [Online]. Available: https://arxiv.org/abs/1511.06434