Generative Adversarial Networks

COMP 551: Mini Project 4, Track 2

GROUP-9: Nikhil Podila, Shantanil Bagchi, and Mehdi Amian

***Abstract*— In the present study, the objective is to reproduce the original results for Generative Adversarial Nets (GANs) as well as to try to improve them. In addition to the original GAN, we consider other variants such as Deep Convolutional GANs (DCGAN) and Wasserstein GANs (WGAN). The algorithms are applied to some benchmark data sets i.e. MNIST, FMNIST and CIFAR10. The reproduced results are evaluated by quantitative measure, the Parzen window-based log-likelihood estimation as described in the original paper. Our results show that the proposed algorithms outperform the results of the original paper in terms of the quantitative measure and visual appearance of the generated images.**

I. INTRODUCTION

With the increasing popularity of machine learning in various fields, the need for data is ever-increasing. Availability of data in the public domain is very scarce for many niche fields like medical imaging, drug testing, etc. Hence, there arises a need for techniques which could generate more data, using existing data. To address this issue of limited data availability, generative modeling like Variational Auto-Encoders (VAEs) [1], Restricted Boltzmann Machines [2], and Pixel RNN [3] have been proposed before the arrival of Generative Adversarial Networks (GANs). However, images generated by these approaches tend to be more blurred when compared to GANs.

Generative adversarial networks (GANs) are deep neural network architectures comprised of two parts - *Generator* and *Discriminator*, pitting against each other playing the zero-sum non-cooperative game where both networks try to defeat each other till they reach Nash equilibrium, which is a stable state of a system involving the interaction of different participants, in which no participant can gain by a unilateral change of strategy if the strategies of the others remain unchanged.

During the training process, weights and biases are adjusted through backpropagation until the discriminator learns to distinguish real images from fake images which are artificially generated. The generator gets feedback from the discriminator and uses it to produce images that are more 'real'. The discriminator network is a convolutional neural network that classifies the images as either fake or real. The generator produces new images through a de-convolutional neural network. The primary goal of the generator network is to maximize the likelihood that its counterpart misclassifies its output as real whereas discriminator optimizes towards a goal distinguishing between real and generated images, untill both reach an equilibrium [4].

For this project, we have applied GANs on three datasets to see how close the artificially generated images are to the original dataset. Most of this work follows that of the original paper.

We were able to improve the quantitative measure i.e. Parzen window-based log-likelihood estimation and also generate visually better images than the original paper.

The rest of the paper is organized as follows: Section II addresses the related works in this research area. Section III introduces the datasets and setup used in the project. Section IV describes the proposed approach. Beside the original GAN, other variants are also examined in this study in a hope to improve the performance of the original one. These variants are Deep Convolutional GAN (DCGAN) and Wasserstein GAN (WGAN). Section V presents the results obtained by implementing the proposed approaches. Section VI discusses our key takeaways and proposes some possible improvement techniques. Section VII mentions the team's contributions.

II. RELATED WORK

The article developing and implementing the first Generative Adverserial Network [5] introduced various possible routes into the research of generative modelling. The article itself implements the algorithm on various standardized datasets and compares the results to other state-of-the-art generative models during the time. Beyond that, the article paves a plan for future research in GANs by proposing improvements to the existing theoretical model, performance metric and feature engineering. The authors have also consolidated and published a Theano [6] framework based code repository as a start point for other researchers. While the original libraries in that code are no longer supported by their community or on the internet, a number of attempts have been made to replicate the code in other standard frameworks like Tensorflow, Keras and Pytorch. We further utilize various components from the original author's implementation by replicating to Keras [7].

Following the first GAN article, a large community of researchers focused their attention towards GANs. Few articles contributed to major improvements in image data generation using GANs. For example, DCGAN developed and implemented by Radford et al. [8] used Convolutional Neural Networks in both Generator and Discriminator Networks in order to boost the performance of the GAN particularly for computer vision applications. This article also discusses variations of networks from CNN modeling perspective - Batch Normalization, Activation functions, Pooling layers, etc.

The article of Salimans et al. [9] focuses on improvements to the original paper and proposals by DCGAN. Here, the concepts such as minibatch discrimination and virtual batch normalization are explored to improve the stability of the GAN. The paper also highlights the performance metric and objective function problem, and proposes alternate methods. However, it also states that the proposed elements are partial solutions and could be improved further.

Wasserstein GAN [10] implements traditional GANs with a different objective function derived from the Wasserstein distance. The discriminator network is modified in this article using weight clipping, which results in an objective function with non-zero gradients. This improves the stability of the training process and prevents Mode collapse in the network.

The current state-of-the-art involves implementations that are focused towards big but specific enhancements to the GAN variants proposed so far. Particularly, BigGAN [11] implements orthogonal regularization in the generator network to control the variety parameter of the images created by the Generator network. The article also performs ablation study over Batch sizes, number of units in each network layer and total number of trainable parameters. But this analysis is done for the ImageNet dataset and also involves large computational resources to generate the output. Similarly, StyleGAN [12] is state-of-the-art for GAN implementations that use intermediate latent spaces to modify the Generator network in order to improve GAN performance. The article focuses on a design from style transfer literature, and implements the network on a Human Faces dataset. This paper also requires large compute. Due to limitations in computational resources, we do not explore these papers in detail but derive the analytical ideas for in-depth ablation study of the traditional GANs.

III. DATASET AND SETUP

Three data sets are used in this study:

1) **MNIST** [13]: One of the most popular datasets used in Machine learning is MNIST dataset. It consists of handwritten digits 0-9 with a training and test sets of 50k and 10k gray scale images of size 28×28 respectively.
2) **Fashion MNIST** [14]: It is a dataset of Zalando's article images consisting of a training set of 70,000 examples. Each example is a 28 x 28 gray-scale image, associated with a label from 10 classes. Fashion-MNIST is intended to serve as a direct drop-in replacement of the original MNIST dataset for benchmarking machine learning algorithms.
3) **CIFAR-10** [15]: The CIFAR-10 dataset contains 60k color images of size 32x32 in 10 different classes. The 10 different classes represent *airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks* with 6,000 images per each class.

IV. PROPOSED APPROACH

A. GAN

The GAN architecture is inspired by Game Theory, where two Multi-Layer Perceptron (MLP) networks, generator and discriminator compete against each other while making each other strong at the same time. Thus, the discriminator requires training for a few epochs before it actually starts giving meaningful results while classifying the images generated by the Generator network. Lastly, a loss function provides a stopping criterion for the generator and discriminator training process. For the discriminator, its objective is to maximize the probability of getting a correct output, that is, classifying correctly, whether the image is fake or real.

Once the training process ends, the generator will generate images. When we talk about training GANs, it refers directly to training the generator. The Generator will take the random noise data z and tries to reconstruct the input x as can be seen in Fig. 1.

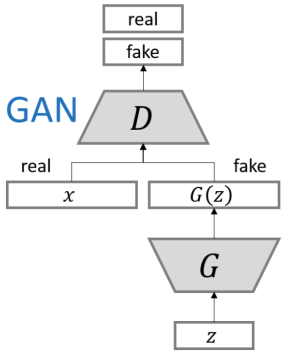

Fig. 1. The Discriminator(D) is trained on two sets of input. One input comes from the training dataset and the other input is the modeled dataset generated by Generator(G) [16]

The discriminator acts as an adaptive loss function for the GAN. Discriminator classifies input as real or fake. Based on the classification, classification error is computed. Ultimately, the discriminator is going to evaluate the output of the real image and the generated image for authenticity. The real images will get high score while the generated images will get lower score. Eventually, the discriminator will have trouble distinguishing between the generated and real images. The discriminator will rely on building a model and potentially an initial loss function.

GAN's optimization problem is in fact a two-player minimax game between discriminator D and generator G. According to [5], the problem can be summarized as

$$min_G max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))] \quad (1)$$

where, $D(x)$ is the probability that data, x come from data rather than the generator's distribution p_g. A prior is defined on input noise variables and denoted by $p_z(z).\log D(x)$ is discriminator output for real data x and $\log[1-D(G(z))]$ is discriminator output for generated fake data $G(z)$. $V(D,G)$

is in essence the cost function that is called the value function here. Discriminator is trained to maximize the probability of assigning the correct label to both training examples and samples from Generator whereas Generator is simultaneously trained to minimize $\log[1 - D(G(z))]$.

Parzen window density function estimation is actually another name for the kernel density estimation that is at first published in 1962 by Emanuel Parzen [17].

$$f_n(x) = \frac{1}{nh} \sum_{j=1}^{n} K(\frac{x - x_i}{n}) \tag{2}$$

where $f_n(x)$ is an estimate of the density function $f(x)$. The main idea is that we approximate f by a mixture of continuous distributions K called kernels that is centered at x_i data points and have scale(bandwidth) h. In our case, the kernel function is Gaussian. Probability of the test set data under p_g is estimated by fitting a Gaussian Parzen window to the samples generated with Generator and the log-likelihood under this distribution is reported. The σ parameter of the Gaussians as reported in the next section has been obtained by cross validation on the validation set.

B. DCGAN

One of the most popular variants of GAN's is the Deep Convolutional GAN (DCGAN). It's functioning is very similar to regular GANs, but specifically focuses on using deep convolutional networks instead of fully connected networks. Convolutional Nets, in general, find areas of correlation within an image, that is they look for spatial connections. Thus, for image or video data, DCGAN would be more fitting, whereas the general implementation of GAN (also referred to as Vanilla GAN), can be applied to wider domains. The DCGAN architecture that is used in this project, is shown in Fig. 2.

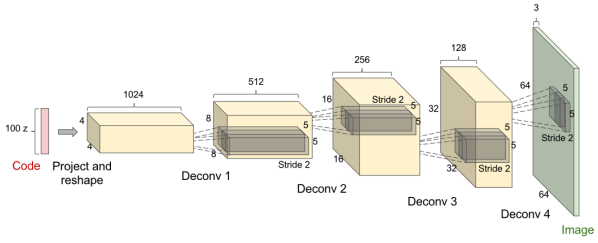

Fig. 2. DCGAN network

DCGAN introduces several constraints and modifications to the original GAN architecture for improved stability and performance. In DCGAN, the generator and discriminator are composed of multiple layers of convolutional computational units, as opposed to the multilayer perceptron networks proposed in the original GAN paper [5].

First, the network structure in DCGANs replaces pooling layers with strided convolutions, which allows the sub-networks to adjust the spatial down-sampling and up-sampling based on the input data. Second, it eliminates fully connected layers that are commonly used after convolutional layers in deep Neural Networks, and it relies solely on convolutional layers. Third, batch normalization is applied to all layers, except to the output layer of the generator and the input layer of the discriminator. Fourth, ReLU activation function is used for all layers in the generator, except for the last layer where a Tanh activation function is applied. For the discriminator, leaky ReLU activation function is adopted for all layers. These modifications resulted in generating complex and visually realistic images. We were able to generate better images than the original paper for CIFAR-10 dataset as reported in the next section.

C. WGAN

Wasserstein GANs (WGANs) [18] introduce a new loss function for training the generator and discriminator sub-networks. The Wasserstein distance is the minimum cost of transporting mass in converting the data distribution q to the data distribution p. The loss function is based on this Wasserstein distance between the real data distribution P_r and the model distribution P_g learned by the generator.

$$W(P_r, P_g) = inf_{\gamma \in \pi(P_r, P_g} E_{(x,y) \simeq \gamma}[x - y] \tag{3}$$

In the above equation, $\pi(P_r, P_g)$ denotes the set of joint distributions $\gamma(x, y)$ whose marginals are P_r and P_g. In simpler terms, $\gamma(x, y)$ defines the amount of earth mass that needs to be moved from a point x to a point y in order P_r and P_g to be identical.

In the regular GAN, we want to maximize the score of classification. A score of 0 for fake image and 1 for real image will be assigned. In WGAN, it changes the task of discriminator as regression problem, and it is renamed as $critic$ as shown in Fig. 3. The critic should measure the EM-distance that how many works should spend, and find the maximum case. WGAN can avoid the gradient vanishing problem.

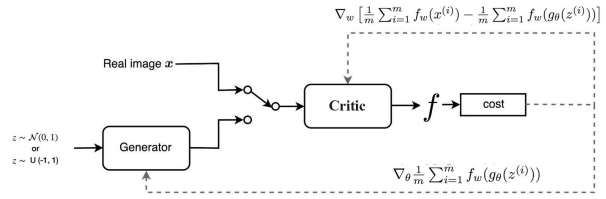

Fig. 3. WGAN network

In GAN, the loss measures how well it fools the discriminator rather than a measure of the image quality. On the contrary, WGAN loss function reflects the image quality which is more desirable and it will be justified in results.

V. RESULTS

Following the GAN paper, we first examined it on the MNIST dataset. We followed the default hyperparameter values, as specified in the original paper to create our first model. To understand how the latent variable dimensions (denoted by z here) affect the GAN's performance, we used

different values of z which are 1, 10, 100, 1000. In the GAN paper, the algorithm is applied to another dataset i.e. TFD (Toronto Face Dataset) too, however, due to nonavailability of TFD, we chose Fashion MNIST and CIFAR-10 instead, with the value of z as 100.

In this section, we present the results of our experiments as mentioned below:

(A) Replicating the work in the original paper. (Experiment A)
(B) Investigating effect of the latent dimension on generated images. (Experiment B)
(C) Applying GAN variants to new datasets. (Experiment C)
(D) Stabilizing the loss of networks using WGANs. (Experiment D)
(E) Studying effect of activation functions on generated images. (Experiment E)
(F) Learning from GAN implementation
(G) Learning from DCGAN implementation

A. *Experiment A*

We were able to replicate the work of the authors in the original GANs paper . Figures 4 and 5 show samples of the various output generated by the original GAN . The main hurdles that we faced are mode collapse, non convergence and/or slow convergence. The values mentioned in Table I is generated with 15k images and is the standard value followed for all the other experiments in our paper. We were successful in obtaining the best Parzen value of 269 which exceeds the value 225 that is reported in the original paper for MNIST dataset with 50k generated images. We have not been able to hypothesize the reason for the value to depend on number of images as has been discussed in detail in section F along with other hurdles.

Dataset	LL	Std	Batch Size
MNIST	222.759	2.128	1000
FMNIST	285.2	2.066	128
CIFAR-10	114.083	1.358	16

TABLE I
BEST PARZEN WINDOW-BASED LOG-LIKELIHOOD ESTIMATE VALUES ON DIFFERENT DATASETS

B. *Experiment B*

The effect of the latent dimension in GAN architecture is investigated on two data sets and the results are presented in Table 1. As can be seen, the best results occur with a latent dimension of 1000 for MNIST and 200 for CIFAR-10. In Table 1, "LL" and "Std" stand for log-likelihood and standard deviation respectively. What we have learnt is if we use too few, like 10, as latent dimension, we may not have enough dimension to model the data. If we use too many e.g. 1000, it would take a long time for the network to learn how to map these to the data. Hence, we stick to a safe number 100 for all the other experiments. However, we feel that the size will depend on the complexity of the distribution too.

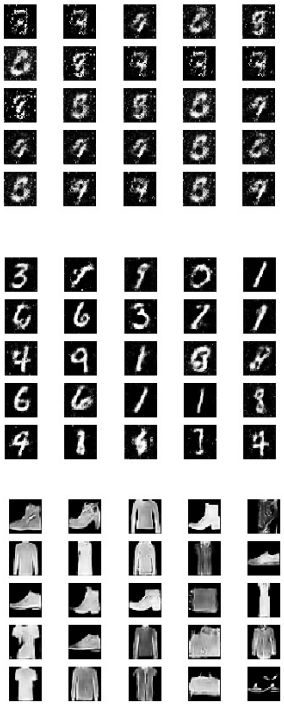

Fig. 4. GAN outputs. Top: Mode Collapsed. Middle: MNIST dataset. Bottom: FMNIST dataset

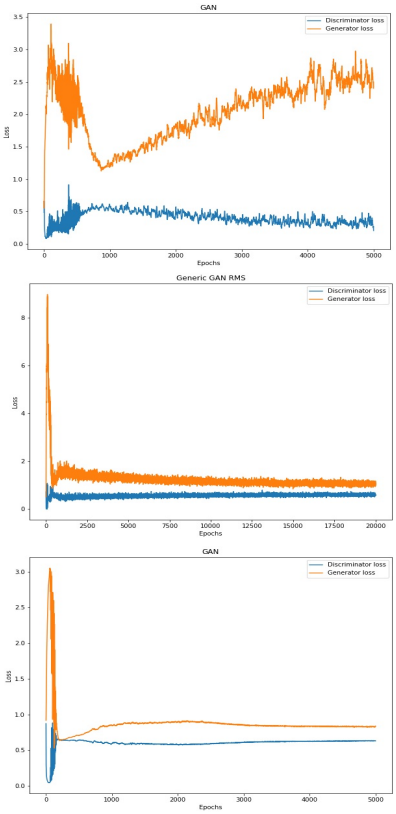

Fig. 5. Loss of discriminator (blue) and generator (orange) of GAN. Top: Mode Collapse. Middle: MNIST dataset with batch size of 128. Bottom: MNIST dataset with batch size of 5000

Latent Dimension	LL (MNIST)	Std (MNIST)	Batch Size (MNIST)	LL (CIFAR-10)	Std (CIFAR-10)	Batch Size (CIFAR-10)
1	-153.302	2.061	5000	-259.309	1.111	100
10	201.59	2.121	5000	82.458	1.272	100
100	217.029	2.015	5000	104.775	1.273	100
200	217.108	2.035	5000	**107.690**	1.280	100
1000	**217.247**	2.066	5000	103.437	1.279	100

TABLE II
EFFECT OF THE LATENT DIMENSION ON THE PARZEN WINDOW-BASED LOG-LIKELIHOOD ESTIMATION FOR VANILLA GAN

C. *Experiment C*

We have implemented DCGAN on MNIST and CIFAR-10 dataset (Figs. 6-8). This implementation was done with the default hyperparameters as used by their original authors. The modifications implemented for DCGAN really worked, as can be seen in Fig. 7, where the generated images on CIFAR-10, after just 200 epochs, have better visualization than those of the original GAN paper.

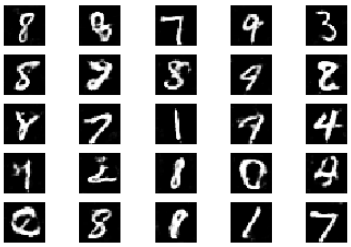

Fig. 6. DCGAN output for MNIST dataset

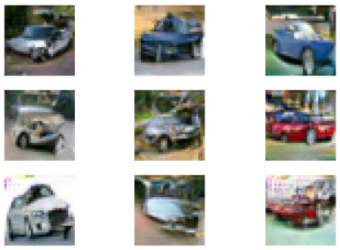

Fig. 7. DCGAN output for CIFAR-10 Dataset

D. *Experiment D*

We implemented Wasserstein GAN on MNIST dataset. The generated images can be seen in Fig. 9. On plotting the loss function as shown in Fig. 10, we find that the loss axis is scaled quite differently than the one for DCGAN and GAN. The reason is that WGAN loss function reflects the image quality which is more desirable.

E. *Experiment E*

In this experiment, we evaluated the performance of vanilla GAN on the MNIST and CIFAR-10 dataset by varying the various activation functions in the hidden and

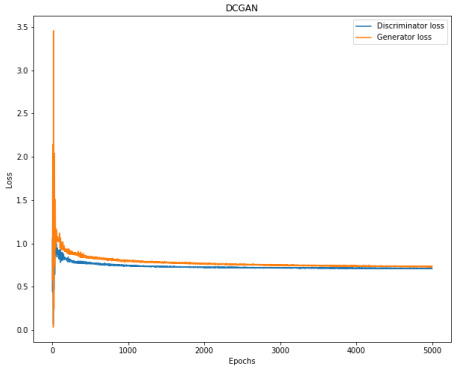

Fig. 8. Loss of discriminator (blue) and generator (orange) for DCGAN model on MNIST dataset with batch size of 5000

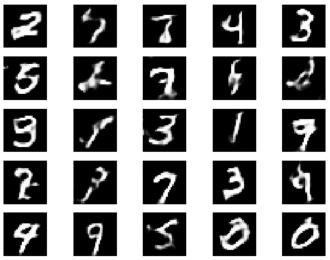

Fig. 9. WGAN output for MNIST dataset

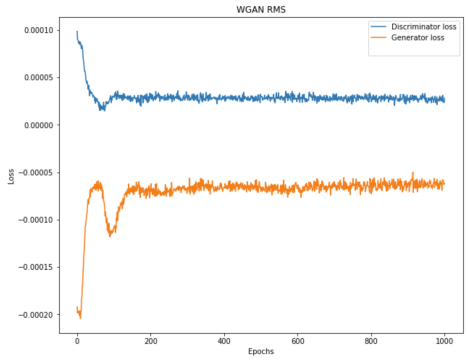

Fig. 10. Loss of discriminator (blue) and generator (orange) for WGAN model on MNIST dataset

output layers. The results obtained from this experiment are displayed in Table III. Vanilla GAN with all other combinations of activation functions apart from ones specified in the table performed poorly in terms of the Gaussian Parzen log-likelihood metric.

The generator network used for this experiment consists of two hidden layers of 256 and 512 units each. The discriminator network also consists of 2 hidden layers with 512 and 256 units each. Batch normalization is used in both the networks. We assume a constant batch size of 64 for CIFAR-10 Dataset and 5000 for MNIST dataset. The variations were performed using Rectified Linear Units (ReLU), Leaky ReLU, Sigmoid, tanh and softplus activations.

We obtained the best results using Leaky ReLU as hidden

layer activations for both the generator and discriminator networks, $\tanh$ as output activation for generator, and sigmoid as output activation for discriminator.

Dataset	Activation functions				Latent Dimensions	Gaussian Parzen Value
	Generator Network		Discriminator Network			
	Hidden layer	Output layer	Hidden layer	Output layer		
MNIST	Leaky ReLU	tanh	Leaky ReLU	Sigmoid	1000	**217.3**
	ReLU		Leaky ReLU			206.8
			ReLU			182.1
			Leaky ReLU		100	205.8
			ReLU			179.3
	softplus		Leaky ReLU			201.6

TABLE III
GAUSSIAN PARZEN LOG-LIKELIHOOD VALUES FOR VANILLA GAN WITH VARIOUS ACTIVATION FUNCTIONS

F. *Learning from original GAN implementation*

*1) **Loss function selection**:* The authors of the main paper mentioned that loss function other than the one they used can perform better at output generation. So we were wondering what is the most appropriate loss function to use in training the GAN. But according to a recent research [19] in which several loss functions were studied and the results were compared as shown in Fig. 11. Their results imply that no function absolutely prevailed over the others, and the GAN was able to learn in all different cases. Thus, it was reasonable for us to use the simplest loss function mentioned in the GAN paper.

Dataset	GANs	FID Scores						
		$k_{SN}=0.2$	0.25	0.5	1.0	5.0	10.0	50.0
MNIST	NS-GAN-SN	5.41	3.99	4.20	3.90	144.28	156.60	155.41
	LS-GAN-SN	5.14	**3.96**	**3.90**	4.42	36.26	59.04	309.35
	WGAN-SN	6.35	6.06	4.44	4.70	**3.58**	**3.50**	**3.71**
	COS-GAN-SN	5.41	4.83	4.05	3.86	291.44	426.62	287.23
	EXP-GAN-SN	**4.38**	4.93	4.25	**3.69**	286.96	286.96	286.96
CIFAR10	NS-GAN-SN	29.23	**24.37**	23.29	**15.81**	41.04	49.67	48.03
	LS-GAN-SN	**28.04**	26.85	23.14	17.30	33.53	39.90	349.35
	WGAN-SN	29.20	25.07	26.61	21.75	**21.63**	**21.45**	**23.36**
	COS-GAN-SN	29.45	25.31	**20.73**	15.88	309.96	327.20	370.13
	EXP-GAN-SN	30.89	24.74	20.90	16.66	401.24	401.24	401.24

Fig. 11. Fréchet Inception Distance (FID scores) vs k_sn. Lower FID scores are better [19]

*2) **Mode Collapse**:* A mode collapse refers to a generator model that is only capable of generating one or a small subset of different outcomes, or modes i.e. the vast number of points in the input latent space result in one or a small subset of generated images. Here, mode refers to an output distribution.

Based on our experiments, we found that tuning the learning rate resolves this issue. We found other solutions that were present online like Feature Matching and Minibatch Discrimination, but weren't able to implement them due to time constraints.

*3) **Non-convergence**:* Sometimes our models never converged and worse they became unstable. In this case, our model failed to find the Nash equilibrium between generator and discriminator. We observed our loss graph of discriminator to reach either zero or near zero values. The output generated were meaningless images. We hypothesized that our models had insufficient capacity and/or our optimization algorithm was too aggressive. Changing the values, helped resolve the issue.

*4) **Parzen window estimate**:* One major point that we pondered on was the parzen window log likelihood estimates. Although no specific number of output images used for this estimation is mentioned in the original paper, majority of the online community followed 10000 images as standard. We experimented with this number and our result varied highly. We have not been able to explain this observation but our results are as follows:

Output Images Size	LL (MNIST)	Std (MNIST)	Batch Size (MNIST)
10k	204.865	2.199	1000
15k	222.759	2.128	1000
30k	249.878	2.042	1000
50k	269.2332	1.9694	1000

TABLE IV
EFFECT OF THE NUMBER OF OUTPUT IMAGES ON THE PARZEN WINDOW-BASED LOG-LIKELIHOOD ESTIMATION

G. *Learning from DCGAN implementation*

*1) **Kernel and filter size implication**:* We know that larger kernels can cover more pixels in the previous layer image and hence, can look at more information. In our case, 5x5 kernels worked well with CIFAR-10. When we used used 3x3 kernels in discriminator, its loss rapidly approached 0. We used 128 filters in our final model as using less filters made the final generated images blurry.

*2) **Playing with labels**:* We experimented with flipping the labels of real and fake images as mentioned online. Although the output images in the end turned out to be the same but according to the blog, this trick supposedly helps with the gradient flow in the early iterations.

One other trick mentioned was to use soft labels i.e. instead of using 0 and 1 for fake and real images, we can use a random number between 0 and 0.1 to represent 0 labels (real images) and a random number between 0.9 and 1.0 to represent 1 labels (generated images) for the discriminator. We were not able to implement this experiment due to time constraints but as mentioned in the blog [20], it helped in the training phase.

VI. DISCUSSION AND CONCLUSION

In this project, we have first implemented the GAN architecture on MNIST and Fashion-MNIST dataset. One shortcoming of the original GAN paper is the quantitative measure introduced for evaluating the network. The Parzen window-based log-likelihood estimation has been proposed as the evaluation metric. As a comparison means, it performs well, however, its absolute value does not reflect the true performance of the model. In other words, as the number of sample for calculating the metric increases, so does the metric. More precisely, we hypothesize that higher value can be attained provided that a sufficient number of samples are used. By using 50k samples on MNIST dataset, we obtained

a Parzen value of 269 by using the GAN and actually surpassed the value of 225 that is reported in the original paper.

Experiments carried out by varying the latent variable size z didn't affect the performance of GANs as long as there is enough dimension for the model to learn.

For DCGANs, we have used discriminator network very similar to the usual CNN classification network whereas the generator looks very similar to the reverse of the discriminator except the convolution layer is replaced with the transposed convolution layer. We evaluated DCGANs with MNIST and CIFAR 10 datasets. We experimented by increasing the depth of the network which turns out to generate better results than the default network but took more time. The network was able to learn more details and generated clearer images. As expected, the generated images for CIFAR-10 dataset have a better visualization than those of the original paper.

Finally, we use WGAN which relies upon Wasserstein distance with MNIST dataset with default parameters. As indicated by figure10, we obtained loss function which converged faster as compared to original GAN network.

There were several hurdles that we faced implementing the models which have been reported in the results. We have learnt a lot while performing the experiments and about the dependence on hyperparameters for getting decent image output. Although there were other loss graphs and generated images, but we are not able to include all those due to space limitations. For further results, the reader is referred to our notebook. In the future, we would like to attempt to experiment with various GAN architectures and hope to improve our model.

Some advancements to GAN architectures that can be further implemented and studied are :

1. Improved WGAN [21], where clipping of weights is eliminated by adding a penalization term to the norm of the gradient of the critic function. This improvement would also address the issue of mode-collapse, which is often encountered in the images generated by vanilla GAN, where the generator generates the same image in every iteration.

2. Least Squares GAN [22], where the loss function is used instead of a critic and weight decay regularization is used to bound the loss function.

3. StyleGAN [23] model is allegedly the state-of-the-art in its way, especially in the Latent Space control. This model borrows a mechanism from Neural Style Transfer known as Adaptive Instance Normalization,(AdaIN), to control the latent space vector z and improves the state-of-the-art in terms of traditional distribution quality metrics, leads to demonstrably better interpolation properties variation.

Overall, we are able to reproduce the original results which are decent considering the constraints of time and resources. Despite the limitations, GANs prove to be an excellent generative modelling technique as confirmed by our results, which will provide several breakthroughs in many important fields.

VII. STATEMENT OF CONTRIBUTIONS

The collaborative work of this team involved everyone contributing to the project. Shantanil worked on visualisation and hyperparameter tuning for the MNIST and FMNIST datasets. Nikhil worked on activation layer effect and parzen window implementation. Mehdi worked on latent dimension effect and improving DCGAN output for CIFAR-10. The team collaboratively worked on the report.

REFERENCES

[1] Carl Doersch. Tutorial on Variational Autoencoders. arXiv:1606.05908, 2016.

[2] Asja FischerChristian Igel. An Introduction to Restricted Boltzmann Machines. In Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. CIARP 2012.

[3] Aaron van den Oord, Nal Kalchbrenner, Koray Kavukcuoglu. Pixel Recurrent Neural Networks. arXiv:1601.06759, 2016

[4] https://heartbeat.fritz.ai/introduction-to-generative-adversarial-networks-gans-35ef44f21193

[5] Mehdi Mirza Bing Xu David Warde-Farley Sherjil Ozair Aaron Courville Yoshua Bengio Ian J. Goodfellow, Jean Pouget-Abadie. Generative adversarial networks. arXiv preprint arXiv:1406.2661, 2014.

[6] Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A., Belopolsky, A. and Bengio, Y., 2016. Theano: A Python framework for fast computation of mathematical expressions. arXiv preprint arXiv:1605.02688.

[7] Francois Chollet. Keras: Deep learning library for theano and tensorflow. https://keras.io, 2016.

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

[9] Salimans, Tim, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training gans." In Advances in neural information processing systems, pp. 2234-2242. 2016.

[10] Arjovsky, Martin, Soumith Chintala, and Léon Bottou. "Wasserstein gan." arXiv preprint arXiv:1701.07875 (2017).

[11] Brock, Andrew, Jeff Donahue, and Karen Simonyan. "Large scale gan training for high fidelity natural image synthesis." arXiv preprint arXiv:1809.11096 (2018).

[12] Karras, Tero, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4401-4410. 2019.

[13] LeCun, Yann and Cortes, Corinna. "MNIST handwritten digit database." (2010)

[14] Xiao, Han, Rasul, Kashif and Vollgraf, Roland. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.." CoRR abs/1708.07747 (2017)

[15] Krizhevsky, Alex, and Geoffrey Hinton. Learning multiple layers of features from tiny images. Vol. 1, no. 4. Technical report, University of Toronto, 2009.

[16] https://github.com/hwalsuklee/tensorflow-generative-model-collections

[17] Emanuel Parzen. On estimation of a probability density function and mode. Annals of Mathematical Statistics. 1065-1076, 1962

[18] Soumith Chintala Martin Arjovsky and L eon Bottou. Wasserstein gan. arXiv preprint arXiv:1701.07875, 2017.

[19] Yipeng Qin and Niloy Mitra and Peter Wonka. "How does Lipschitz Regularization Influence GAN Training?" arXiv:1811.09567 2018.

[20] https://medium.com/@utk.is.here/keep-calm-and-train-a-gan-pitfalls-and-tips-on-training-generative-adversarial-networks-edd529764aa9

[21] 1349723655104342. An intuitive introduction to generative adversarial networks (gans), Jan 2018.

[22] Haoran Xie Raymond Y.K. Lau Zhen Wang Stephen Paul Smolley Xudong Mao, Qing Li. Least squares generative adversarial networks. arXiv:1611.04076, 2016.

[23] Othman Sbai, Mohamed Elhoseiny, Antoine Bordes, Yann LeCun, and Camille Couprie. Design: Design inspiration from generative networks. In Proceedings of the European Conference on Computer Vision (ECCV), pages 0–0, 2018.