GAN - Assignment

Aryan Garg
Computer Science & Engineering, B.Tech.
Indian Institute of Technology, Mandi
b19153@students.iitmandi.ac.in

INTRODUCTION

DCGAN [3] or Deep Convolutional Generative Adversarial Networks were the first networks to deviate from Fully Connected MLPs and use Convolutional Networks for both, Generator and Discriminator Networks. Radford *et al.*test DCGAN on the following datasets: LSUN, ImageNet-1K and CIFAR-10, however, we use the Dog subset of the AFHQ dataset which contains 5000 $512x512$ high-quality images. We then go on to experiment with DCGAN's hyperparameters and infuse StyleGAN [1] & StyleGAN2's [2] improvements to DCGAN. After exploring the latent spaces (Z and W), we change the architecture slightly to improve the Generator's diversity and fidelity of synthesized images.

Note all experiments are performed at $64x64$ *to speed-up experiments*

EXPERIMENT - 1

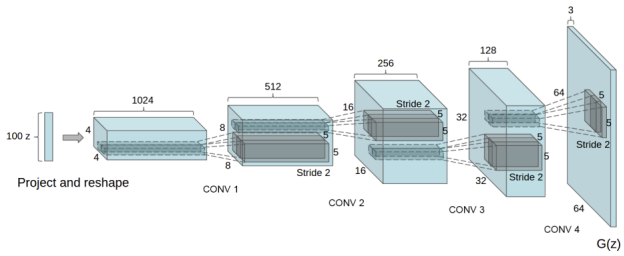
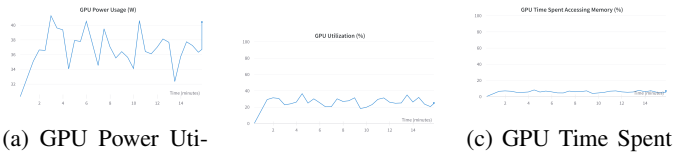


Fig. 1: The Architecture Implemented. Image Reference [3]

A. Run - 0: Sanity

All networks and data are run for the first time on a GPU session for 20 epochs. See the loss curves Figs 3, 4 and GPU statistics Fig. 2. See Fig. 1 for architecture details.

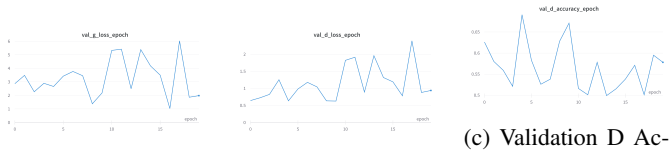


(a) GPU Power Utilization (b) GPU Utilization (c) GPU Time Spent Accessing Memory

Fig. 2: Sanity Run's GPU Statistics (Google Colab)

(a) Training Generator Loss (b) Training Discriminator Loss (c) Training D Accuracy

Fig. 3: Training Losses & Accuracy. G $\longrightarrow$ Generator. D $\longrightarrow$ Discriminator



(a) Validation G Loss (b) Validation D Loss (c) Validation D Accuracy

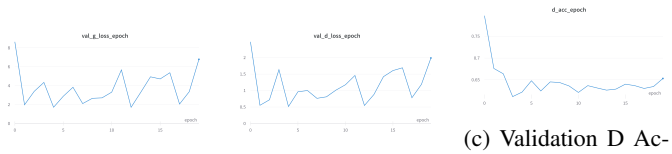
Fig. 4: Validation Losses & Accuracy

B. Run - 1: 20 Epochs

This run includes image logging to the experiment tracker as well. See Figs 5 6 for training and validation losses. Fig 7 depicts the progression of generator training over 20 epochs.

(a) Training Generator Loss (b) Training Discriminator Loss (c) Training D Accuracy

Fig. 5: Training Losses & Accuracy



(a) Validation G Loss (b) Validation D Loss (c) Validation D Accuracy

Fig. 6: Validation Losses & Accuracy

C. Run - 2: 100 Epochs

This is the main run that is for 100 epochs. See Fig **??** for generated training images and Fig 9 for images generated during validation. See Fig 10 and Fig 11 for training and validation loss & accuracy curves.

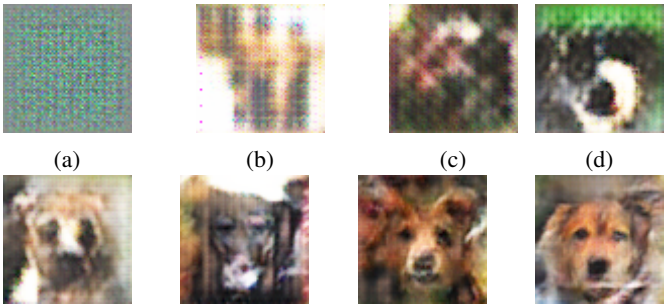


Fig. 7: Training progression of Generator. These images are chosen at random but sequentially forward steps from the validation pipeline. We start with a noise channel output to coherent dog faces that do not exist

Training progresses in the expected mini-max fashion for all the runs except the 100 epochs one. The 100 epochs run starts to diverge. The discriminator gets too strong after a certain number of steps. Early stopping, slower learning rate, or TTUR could mitigate the issue.

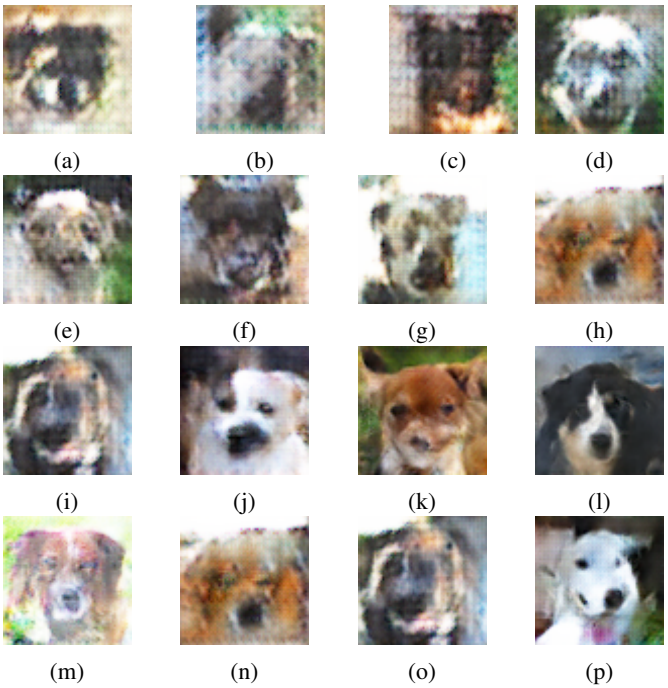


Fig. 8: 100 Epochs Run: Training progression of Generator. These images are cherry-picked. The checkered pattern artifacts of convolutions are more prominent in earlier stages represented by (a), (b), (c) and (d). The generator often deforms the face and is not aware of the global context. This immediately suggests a need for attention layers in the architecture.

See table I for best loss & accuracy values.

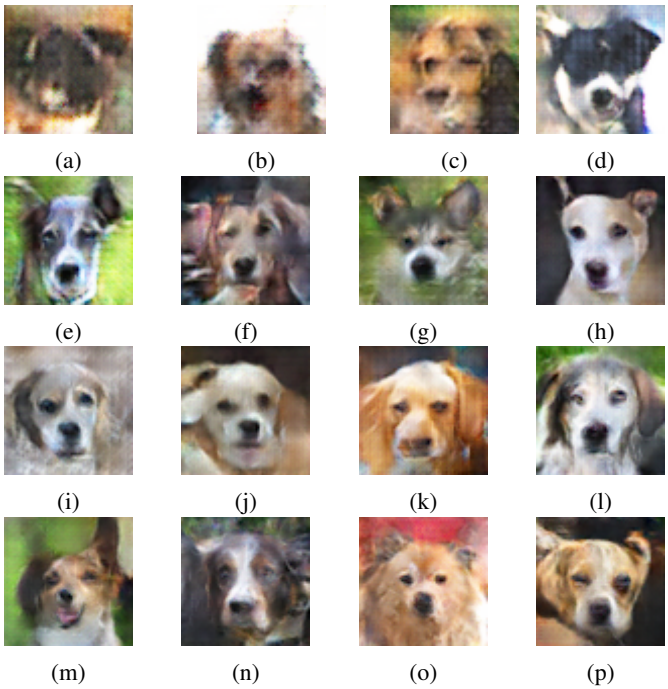
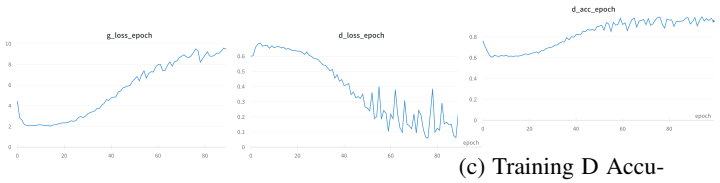
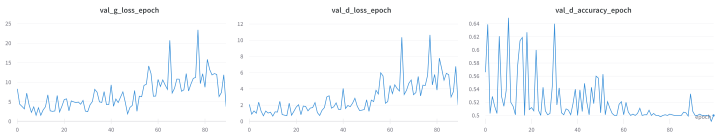


Fig. 9: 100 Epochs Run: Validation Pipeline images. These images are cherry-picked.



(a) Training G Loss (b) Training D Loss (c) Training D Accuracy

Fig. 10: Experiment - 1 Run-2: 100 Epochs: Training Losses & Accuracy



(a) Val G Loss (b) Val D Loss (c) Val D Accuracy

Fig. 11: Experiment - 1 Run-2: 100 Epcohs Validation Losses & Accuracy

Run	Best Val Accuracy (D)	Best Val Loss (G)
Sanity	0.691	1.028
Run - 1	0.753	1.543
Run - 2	0.649	1.543

TABLE I: Best Validation Accuracy across the three runs. Lower Validation G Loss is better while higher D accuracy denotes a strong Discriminator

EXPERIMENT - 2: HYPER-PARAMETER TUNING

A. Learning Rate

We try 3 different learning rates apart from the default ($3e-4$) for Adam and **observe** that $2e-4$ gives the best perceptual results. See Tab. I for average validation accuracy and best validation generator loss.

Learning Rate	Avg. Val (D) Accuracy	Best Val-G-Loss
0.0001	**0.701**	1.716
0.0002	0.541	**0.788**
0.0003(default)	0.517	1.714
0.0005	0.609	1.209

TABLE I: Different learning rates for Adam.

See Fig. 1 for some validation pipeline outputs.

(a) $1e-4$ (b) $2e-4$ (c) $5e-4$

Fig. 1: Random (after $> 50\%$ training had elapsed) Validation Output for different learning rates.

B. Schedulers

Note that all experiments are run for 20 epochs. The optimizer is Adam.

1) Step: Step Schedulers are piece-wise constant functions. See Figs 2 3 4 and 5.

2) Exponential Decay: See Figs 6 8 7 and 9.

3) Cosine Annealing: See Figs 11 12 10 and 13.

Quantitatively, Table II shows that Exponential LR performs better than cosine annealing and step schedulers for learning rate.

Scheduler	Avg. Val (D) Accuracy	Best Val G-Loss
StepLR	0.761	1.438
ExponentialLR	**0.878**	**0.963**
Cosine-Annealing	0.524	1.733

TABLE II: Schedulers

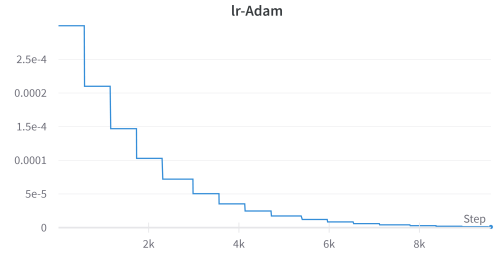
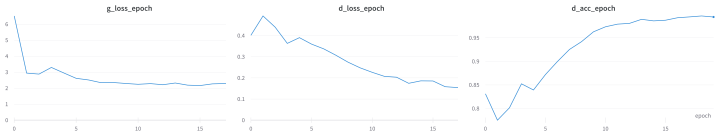
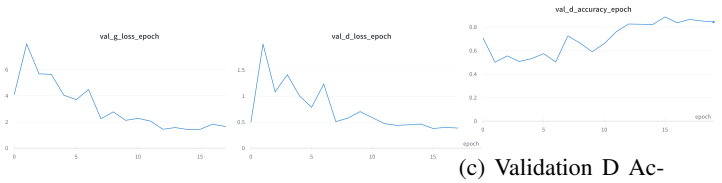


Fig. 2: Step Learning Rate over 20 epochs or 2980 steps. The multiplier used is 0.7 and the minimum LR is set to $5e-05$ beyond which the LR will not fall.



(a) Training Generator Loss (b) Training Discriminator Loss (c) Training D Accuracy

Fig. 3: Step Learning Rate Scheduler: Training Losses & Accuracy



(a) Validation G Loss (b) Validation D Loss (c) Validation D Accuracy

Fig. 4: Step Learning Rate Scheduler: Validation Losses & Accuracy

Fig. 5: Step Learning Rate Scheduler: Generated Dogs

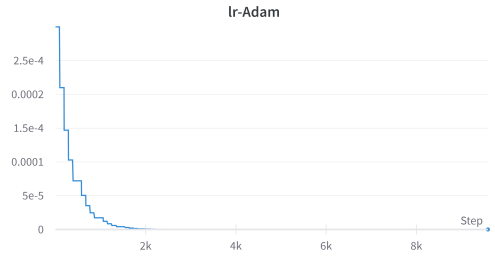
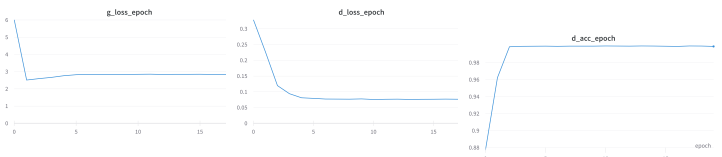
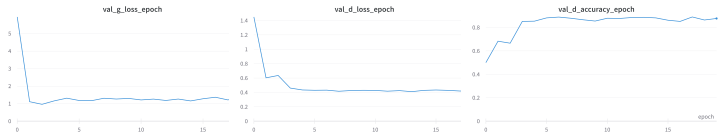


Fig. 6: Exponential Learning Rate. The multiplier used is 0.7.



(a) Training Generator Loss (b) Training Discriminator Loss (c) Training D Acc

Fig. 7: Exponential Learning Rate Scheduler: Training Losses & Accuracy



(a) Val G Loss (b) Val D Loss (c) Val D Accuracy

Fig. 8: Exponential Learning Rate Scheduler: Validation Losses & Accuracy

Fig. 9: Exponential Learning Rate Scheduler: Generated Dogs

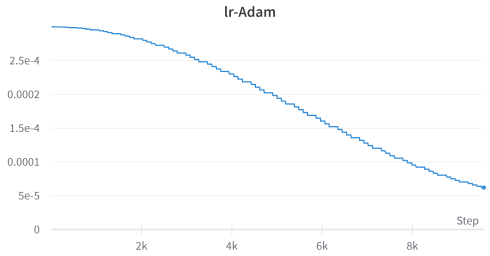
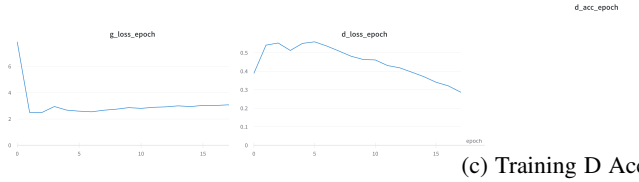


Fig. 10: Cosine Annealing Learning Rate

Fig. 13: Cosine Annealing Learning Rate Scheduler: Generated Dogs



(a) Training G Loss (b) Training D Loss (c) Training D Accuracy

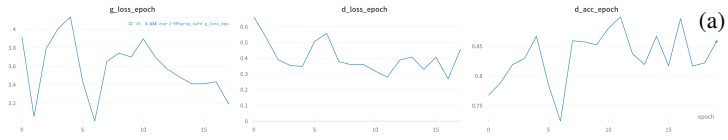
Fig. 11: Cosine Annealing Learning Rate Scheduler: Training Losses & Accuracy

(a) Val G Loss (b) Val D Loss (c) Val D Accuracy

Fig. 12: Cosine Annealing Learning Rate Scheduler: Validation Losses & Accuracy

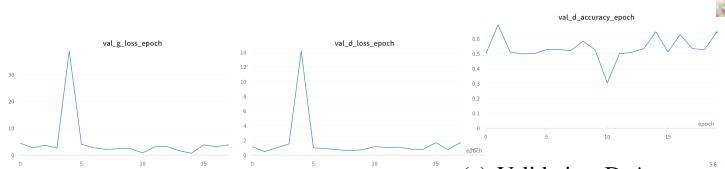
C. Optimizers

All optimizers are run at a learning rate of 0.0005. Other learning rates were producing perceptually incoherent outputs. Learning rate (hyper-parameter) search was done manually here.



(a) Training Generator Loss (b) Training Discriminator Loss (c) Training D Accuracy

Fig. 1: RMSProp: Training Losses & Accuracy



(a) Validation G Loss (b) Validation D Loss (c) Validation D Accuracy

Fig. 2: RMSProp: Validation Losses & Accuracy

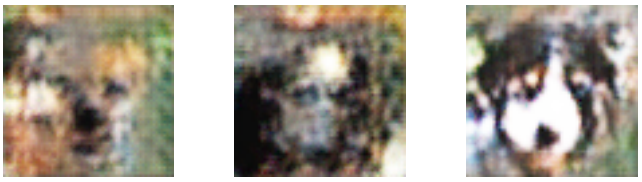
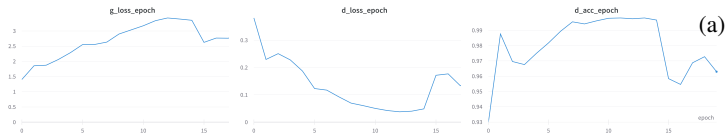


Fig. 3: RMSProp: Generated Dogs

1) RMSProp:

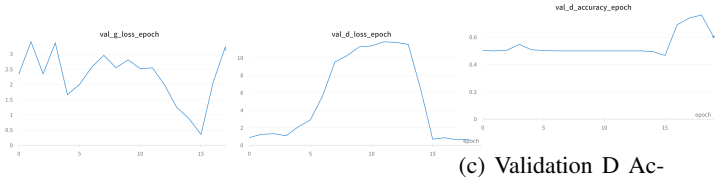


(a) Training Generator Loss (b) Training Discriminator Loss (c) Training D Accuracy

Fig. 4: SGD: Training Losses & Accuracy

2) Stochastic Gradient Descent:

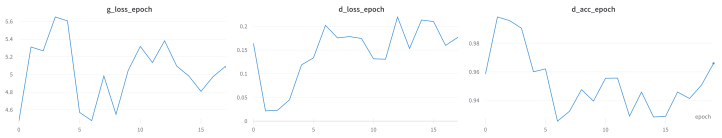
3) Nesterov Stochastic Gradient Descent: Quantitative results are in Table I.



(a) Validation G Loss (b) Validation D Loss (c) Validation D Accuracy

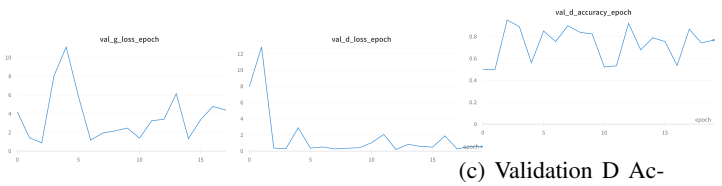
Fig. 5: SGD: Validation Losses & Accuracy

Fig. 6: SGD: Generated Dogs



(a) Training Generator Loss (b) Training Discriminator Loss (c) Training D Accuracy

Fig. 7: Nesterov-SGD: Training Plots



(a) Validation G Loss (b) Validation D Loss (c) Validation D Accuracy

Fig. 8: Nesterov-SGD: Validation Losses & Accuracy

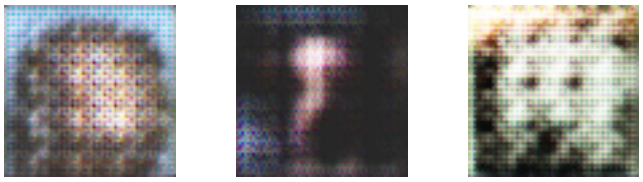


Fig. 9: Nesterov-SGD Optimizers: Generated Dogs

Optimizer	Avg. Val (D) Accuracy	Best Val G-Loss
RMSProp	0.565	0.719
SGD	0.550	**0.355**
Nesterov-SGD	**0.654**	0.899

TABLE I: Optimizer

EXPERIMENT - 3: FUSING STYLEGAN IMPROVEMENTS

A. Improvements:

1) Mapping Network: A 4-layer MLP that takes in a 100-dimensional noise vector (Z-space) and transfers it to W-space (same dimension as input dimension) which is believed to be less entangled than Z-space [1, 2].

2) Adaptive Instance Normalization: A scaled and biased standardization technique that takes the style vector as input for it's scaling (S) and bias (B) parameter. AdaIN layers are formulated using the equation II-A2.

$$y = S(w) * \frac{x - \mu}{\sigma} + B(w)$$

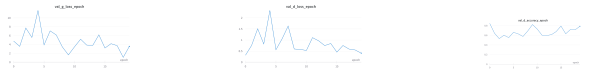
3) Weight Demodulation: Weight demodulation is an improved alternative to AdaIN. It is a weight regularization technique where all weights are first scaled by the style vector (modulation operation). Then demodulation operation includes scaling the respective weights with their $L2$-norm.

B. Results

1) Mapping Network Only: See Fig 3 for generated results. See Fig 1, 2 for training and validation curves.

(a) Training G Loss (b) Training D Loss (c) Training D Acc

Fig. 1: Mapping Network Only: Training Losses & Accuracy



(a) Val G Loss (b) Val D Loss (c) Val D Accuracy

Fig. 2: Mapping Network Only: Validation Losses & Accuracy

2) Mapping Network + AdaIN: See Fig 6 for generated results. See Fig 4, 5 for training & validation curves.

EXPERIMENT - 4: LATENT SPACE EXPLORATION

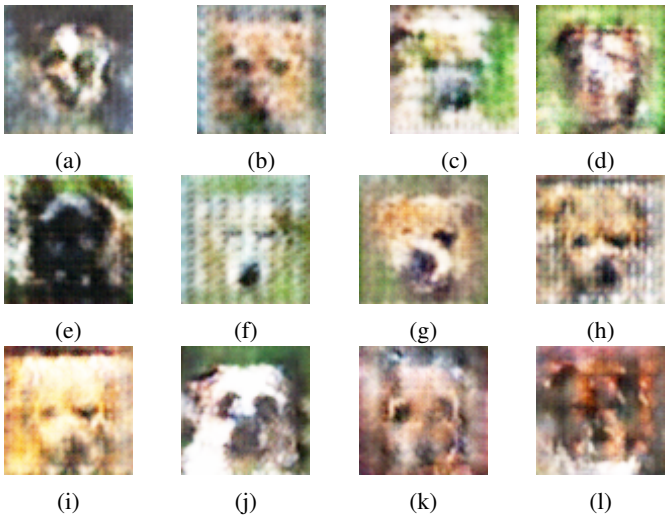
Corresponding GIFs are included alongside the code.

C. Z-Space Interpolation

We sample two random noise vectors and interpolate uniformly between them, generating 1000 vectors in between. Some of the outputs are shown in Fig. 7

D. W-Space Interpolation

For W-space interpolation, we detach the mapping network from the Generator by converting the MLP to an Identity layer. So, effectively $z = w$. However, we do use the trained mapping network first to sample two w vectors from the W-space. We verify that w-space is disentangled due to the outputs we see on interpolation in Fig 8's description for further explanation.

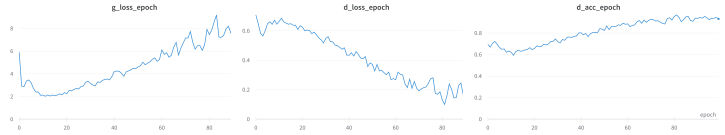


(a) (b) (c) (d)

(e) (f) (g) (h)

(i) (j) (k) (l)

Fig. 3: Mapping Network Only Generated Dogs During *Validation*



(a) Training G Loss (b) Training D Loss (c) Training D Acc

Fig. 4: Mapping + AdaIN Network: Training Losses & Accuracy

(a) Validation G Loss (b) Val D Loss (c) Val D Accuracy

Fig. 5: Mapping + AdaIN Network: Validation Losses & Accuracy

Fig. 6: Mapping Network + AdaIN Network: Generated Dogs During *Validation*

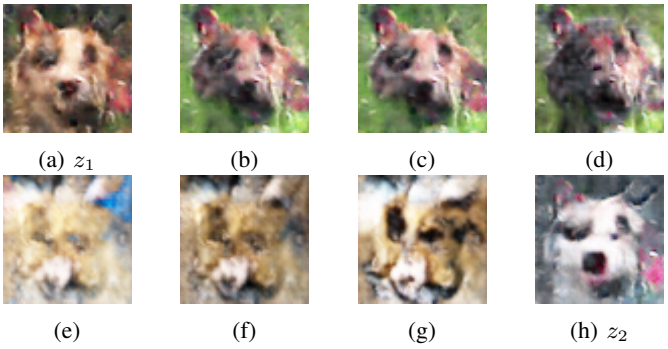


Fig. 7: Z-space Interpolation. Notice that we interpolate in a direction that does not induce any changes in the number or shape of the dogs' ears. All dogs have just the left ear standing tall. (a), (b), (c) and (d) are close interpolations, (e), (f), and (g) are close intermediate interpolations in the middle (index 600) while (h) is the last vector z_2

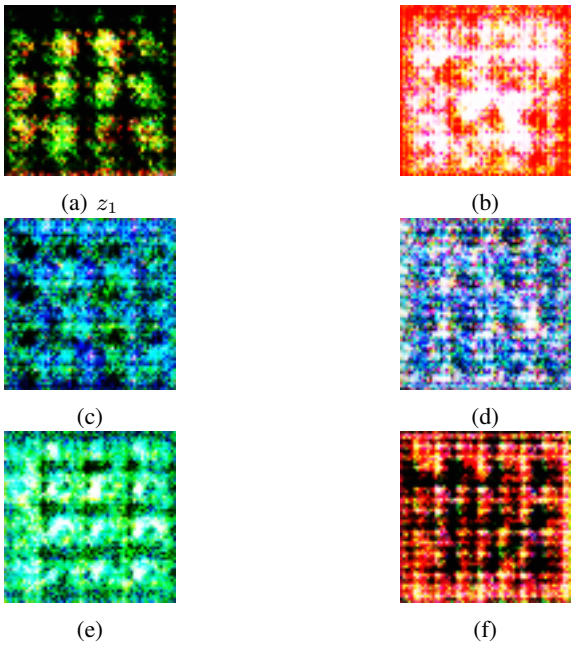


Fig. 8: W-space Interpolation's Generator's Output. We interpolate in the W-space and get incoherent output. This implies that the feature space had been disentangled properly and there are dimensions still left that produce noise. Possibly, $z \longrightarrow w$ mapping is sparse

EXPERIMENT - 5 (BONUS): ARCHITECTURAL CHANGES FOR DIVERSITY & QUALITY

A. *Spectral Normalization*

Due to the instability in training any GAN, spectral normalization can help significantly [3] by mitigating exploding or vanishing gradients. This is ensured by restricting the gradient norm of weights within a strict bound. Precisely, when the spectral norm of weights is 1 (1-Lipschitz). It is important to note that spectral normalization does not eliminate the need for batch normalization. Spectral normalization affects the weights of each layer, while batch normalization affects the activations of each layer.

We run two experiments using the spectral discriminator (SpD). First, we use our vanilla-DCGAN Generator with SpD then we use the Mapping+AdaIN Generator with SpD.

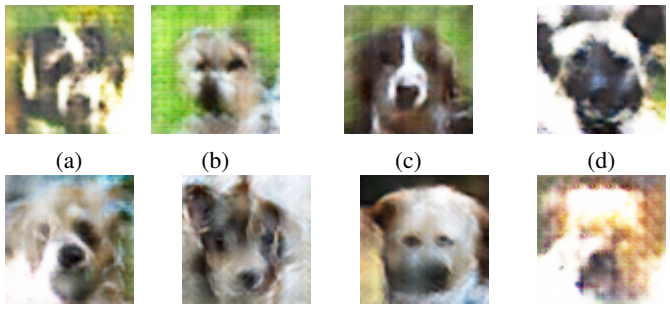
1) Vanilla G + SpD: See Figures 1, 2 & 3.

(a) Training G Loss (b) Training D Loss (c) Training D Acc

Fig. 1: Vanilla G + Spectral Discriminator: Training Losses & Accuracy

(a) Validation G Loss (b) Val D Loss (c) Val D Accuracy

Fig. 2: Vanilla G + Spectral Normalization Discriminator Run: Validation Losses & Accuracy



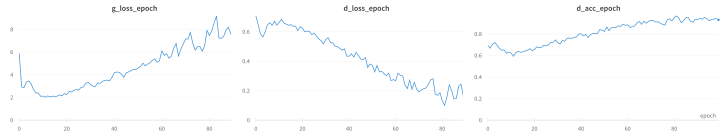
(a) (b) (c) (d)

(e) (f) (g) (h)

Fig. 3: Spectral Normalized Discriminator: Generated Dogs During *Validation*

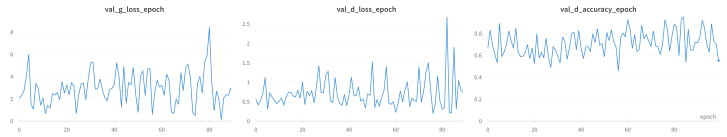
2) Mapping+AdaIN G + SpD: See Figures 4 5 6.

Analysis: Why does it work for Vanilla Generator and not work for the Mapping+AdaIN Generator?



(a) Training G Loss (b) Training D Loss (c) Training D Acc

Fig. 4: Mapping + AdaIN Network: Training Losses & Accuracy



(a) Validation G Loss (b) Val D Loss (c) Val D Accuracy

Fig. 5: Mapping+AdaIn + Spectral Normalization Discriminator Run: Validation Losses & Accuracy

1. We see that we get smooth and soft outputs with the vanilla generator. We can also notice that many color artifacts are gone. Thus spectral normalization worked to improve quality in this case.

2. Although the training curves are stabilized, there is minimal information to the weights about the style information. The Generator is not able to get down to an acceptable loss value. In addition to this, the weights are normalized thus reducing an information-rich signal for backpropagation. Hence, we see **distorted outputs with many aberrations.**

B. *Other possible improvements:*

1) Residual connections: To counter the *degradation problem*, skip or residual connections were implemented in the Generator network. (The performance of the model drops down with the increase in depth of the architecture. This is known as the)

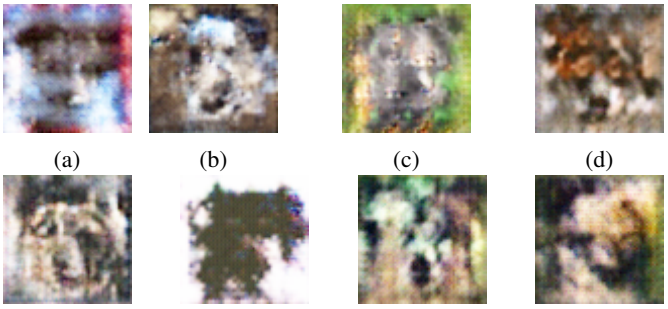
2) TTUR: Although not an architectural change, two Timescale Update Rule proposes two different learning rates for the two networks, D and G. The main objective is to keep D at an optimal place to pass good feedback signals to the generator for learning. Very fast generators drive the discriminator to useless information-returning states and hence, in practice, the values observed are 0.004 and 0.001 for D and G respectively.

3) Self-Attention Feature Maps: Inspired from SAGAN: Attention layers will impose facial symmetry and color consistency across the output image.

REFERENCES

[1] Tero Karras, Samuli Laine, and Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019. arXiv: 1812.04948 [cs.NE].

[2] Tero Karras et al. *Analyzing and Improving the Image Quality of StyleGAN*. 2020. arXiv: 1912.04958 [cs.CV].



(a) (b) (c) (d)

(e) (f) (g) (h)

Fig. 6: Mapping+AdaIN G + Spectrally Normalized Discriminator: Generated Dogs During *Validation*

[3] Takeru Miyato et al. *Spectral Normalization for Generative Adversarial Networks*. 2018. arXiv: 1802.05957 [cs.LG].

[4] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].