A Study in GANs

Mentors - Aniket Das and Avik Pal

PROJECT OVERVIEW

The project runs along 2 tracks
1. Standard for most people - implement pre existing models using torchGAN, do literature review. End with models like discogan and stargan
2. NLPGAN - (difficult) Design a framework to handle gans with recurrent nets. Ideally people with some software dev experience and ml background should take this. Aim is to design something similar to torchgan but explicitly for nlp.
The aim is to strike a balance between teaching theory and practice, with the goal that upon completion, mentees are able to read, understand and easily implement a significant fraction of the current literature on GANs

ROADMAP

- Deep Learning Primer with an emphasis on Generative Models
- Introductory lecture on GANs with a Primer on PyTorch
- Writing your first GAN model using TorchGAN
- GAN Losses: Theoretical Insight and Experimentation
- Metrics of GAN performance and practical advice on evaluating and training GAN models. As an exercise at this checkpoint, mentees shall be expected to present a quantitative comparison of various GAN losses
- Semi Supervised Learning and Class Conditional Models with a quantitative comparison of various class conditional models
- Domain Application: Neural Style Transfer and CycleGAN
- Domain Application: Image Super Resolution and SRGAN

WEEKWISE PROGRESS

Week 1-2 :Going through the video lectures of CS231 , understanding Deep Convolutional Neural Networks and implementing them on basic MNIST and CIFAR-10 dataset . Implementing Resnet, VGG and DenseNet .
Week 3: Going through research papers of GAN , DCGAN and ACGAN . Understanding the basic functionality of a GAN and implementing them using Pytorch . Get familiar with Torchgan by reading its documentation and tutorials and then implementing DCGAN , ACGAN and CYCLEGAN using torchgan Trainer .
Week 4: Writing custom datasets for datasets like CelebA and CityScapes which are not in torchvision and implemating yaml config files similar to Facebook's Maskrcnn benchmark for Torchgan to make it a much better research framework for training GANs .
Week 5: Making Parser for Torchgan which would accept a yaml file , path of the dataset and start training of the GAN models. All the respective parameters and hyperparameters needs to mentioned by the user in the yaml file .
Week 6: Studied papers of CycleGAN , DiscoGAN and StarGAN and implemented them in Pytorch and got good results.
Week 7: Studied two of the special GANs , SRGAN and SAGAN .SRGAN is used to create super resolution images from low resolution images and SAGAN uses a unique model architecture which is explained in the report
Week 8: Studied stability methods for training vision based GANs and studied papers of NLP GANs and deep dived into why NLP GANs fail to produce good results .

TASKS COMPLETED

Implemented the following Classification based Convolutional neural networks:

- **RESNET**
- **DENSENET**
- **VGG**

Implemented the following GAN models

- **Basic Fully Connected GAN**
- **DCGAN**
- **CGAN (including AnimeGAN using CGAN architecture)**
- **ACGAN**
- **INFOGAN**
- **CYCLEGAN**
- **DISCOGAN**
- **STARGAN**
- **SRGAN**
- **SAGAN**

Made following things for Torchgan framework :

- **Progress bar (Merged)**
- **Dataloader (To be merged)**
- **Parser for yaml files(To be merged)**

FUTURE PROSPECTS

- Studying NLP GANs
- Generalising Parser and Dataloader for Torchgan framework .

A STUDY IN GANS

A PREPRINT

Aniket Das (Mentor)
Department of Electrical Engg.
IIT Kanpur
aniketd@iitk.ac.in

Avik Pal(Mentor)
Department of Computer Science and Engg.
IIT Kanpur
avikpal@iitk.ac.in

Naman Biyani
Department of Computer Science and Engg.
IIT Kanpur
namanb@iitk.ac.in

Nirmal Suthar
Department of Computer Science and Engg.
IIT Kanpur
nirmalps@iitk.ac.in

July 11, 2019

Introduction

Image synthesis is an important problem in computer vision. There has been remarkable progress in this direction with the emergence of Generative Adversarial Networks (GANs). GANs have been known to be unstable to train, often resulting in generators that produce nonsensical outputs/images (Figure 1). However GANs based on deep convolutional networks have been successful and especially in image generation in computer vision with the help of convolution which opens the possibility to explore multilayer GANs (DCGAN). In an unconditioned generative model, there is no control on modes of the data being generated. However, by conditioning the generator on additional information (such as classes or label) it is possible to generate more specific. Such conditioning could be based on class labels (CGAN).

However, one can observe that convolutional GANs have mush more difficulty in modeling some image classes . For example, while the state-of-the-art GANs model excels at synthesizing image classes which are distinguished more by texture than by geometric or structural patterns that occur consistently in some classes. One possible explanation for this is that GANs models rely heavily on convolution to model the dependencies across different image regions. Since the convolution operator has a local receptive field, long range dependencies can only be processed after passing through several convolutional layers. This could prevent learning about long-term dependencies for a variety of reasons. So adding self-attention mechanism into the convolutional GANs helps with modeling long-range dependencies (SAGAN).

Related Work

Until recently, most work on deep generative models focused on models that provided a parametric specification of a probability distribution function. The model can then be trained by maximiz- ing the log likelihood. In this family of model, perhaps the most succesful is the deep Boltzmann machine . Such models generally have intractable likelihood functions and therefore require numerous approximations to the likelihood gradient .

Unlike generative adversarial networks, variational autoencoders (VAE) pair a differentiable generator network with a second neural network. Unlike generative adversarial networks, the sec- ond network in a VAE is a recognition model that performs approximate inference. GANs require differentiation through the visible units, and thus cannot model discrete data, while VAEs require differentiation through the hidden units, and thus cannot have discrete latent variables. Other VAE- like approaches exist but are less closely related to our method.

Working of GANs

We define a generator which input a noise and outputs a desired result which fools the discriminator, supposed to distinguished the generator output. To learn the generator's distribution p_g over data $\boldsymbol{x}$, we define a prior on input noise

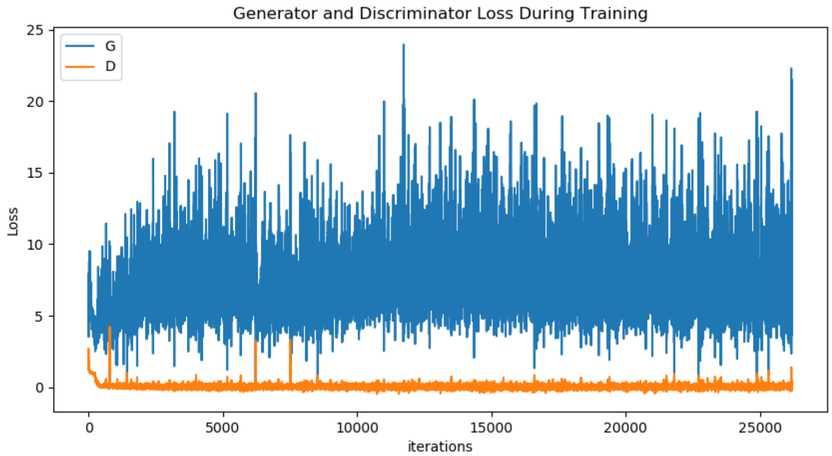


Figure 1: Loss of generator and discriminator during training DCGAN on MNIST dataset suggest that the GANs are unstable and special techniques should be used to train GANs effectively

variables $p_z(z)$, then represent a mapping to data space as $G(z; \theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters θg. We also define a second multilayer perceptron $D(\boldsymbol{x}; \theta_d)$ that outputs a single scalar. $D(\boldsymbol{x})$ represents the probability that x came from the data rather than p_g. We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize log(1 - D(G(z))) (Figure 2):

Finally, $\mathcal{L}_{GAN}$ is the generative adversarial network cost. That cost function represents the *game* between G and D. When training D, G are kept fixed and we have:

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

Figure 2: In other words, D and G play the following two-player minimax game with value function V (G, D)

For better gradient behavior of generator loss we minimize

$$\mathcal{L}_{GAN}^{G} = -log(D(G(z))) \tag{1}$$

instead of

$$\mathcal{L}_{GAN}^{G} = log[1 - D(G(z))] \tag{2}$$

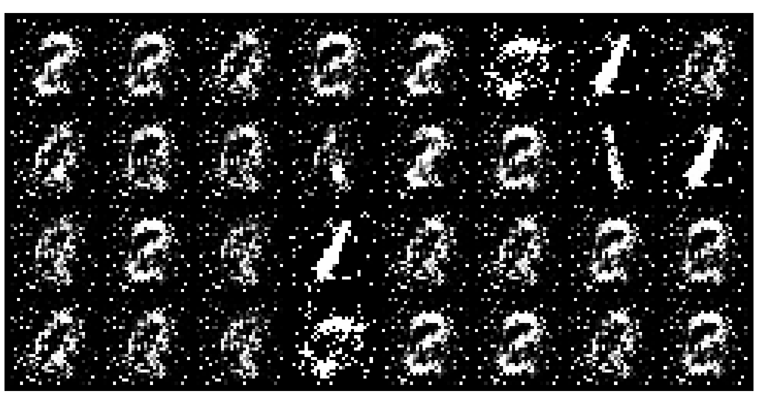


Figure 3: Hand-digit generation tranined on MNIST dataset train by basicGAN

Deep Convolutional GANs

It is worth thinking about the DCGAN as a foundational pillar for GANs research (Figure 4). The DCGAN model's fundamental component is to replace the fully connected layers in the generator with these upsampling convolutional layers. In designing this architecture, the authors cite three sources of inspiration.

- The All Convolutional Net Replacing pooling operations with spatial downsampling convolutions
- Eliminating fully connected layers after convolutions
- Batch Normalization Normalizing activations to help gradient flow

With these advancements in mind, the authors searched for a stable DC-GAN architecture and landed on the following architectural guidelines:

- Replace any pooling layers with strided convolutions in the discriminator and fractional-strided convolutions in the generator
- Use Batch Normalization in the generator and discriminator
- Remove fully connected hidden layers for deeper architectures
- Use ReLU activation in generator for all layers except for Tanh in output (These images are normalized between [-1, 1] rather than [0,1] , thus Tanh over sigmoid)
- Use LeakyReLU activation in the discriminator for all layers

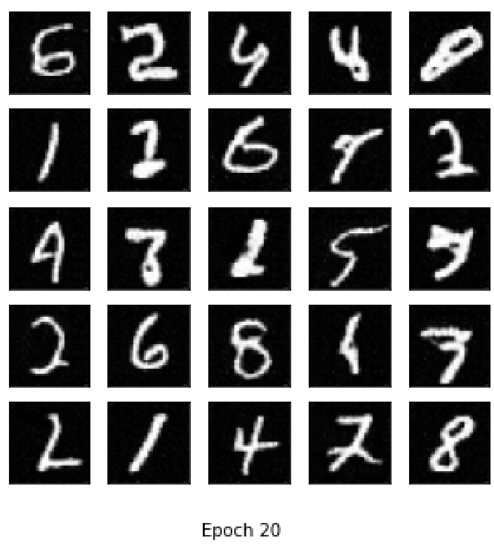


Figure 4: Hand-generated digits generated on MNIST dataset train by DCGAN is huge improvement over digits generated by original GAN paper, suggest that use of convolution operation can be beneficial in GAN.

Class Conditioned GANs

Class conditioned GANs proposes a simple extention of GANs that employs label conditioning in additional to produce high resolution and high quality generated images.

By adding an auxiliary classifier to the discriminator of a GANs, the discriminator produces not only a probability distribution over sources but also probability distribution over the class labels. This simple modification to the standard DCGAN models does not give tremendous difference but produces better results and is capable of stabilizing the whole adversarial training.

We explored different variant of class conditioned GANs in this project which are CGAN, infoGAN and ACGAN. The architecture is as shown (in Figure 5) for comparisons of several GANs.

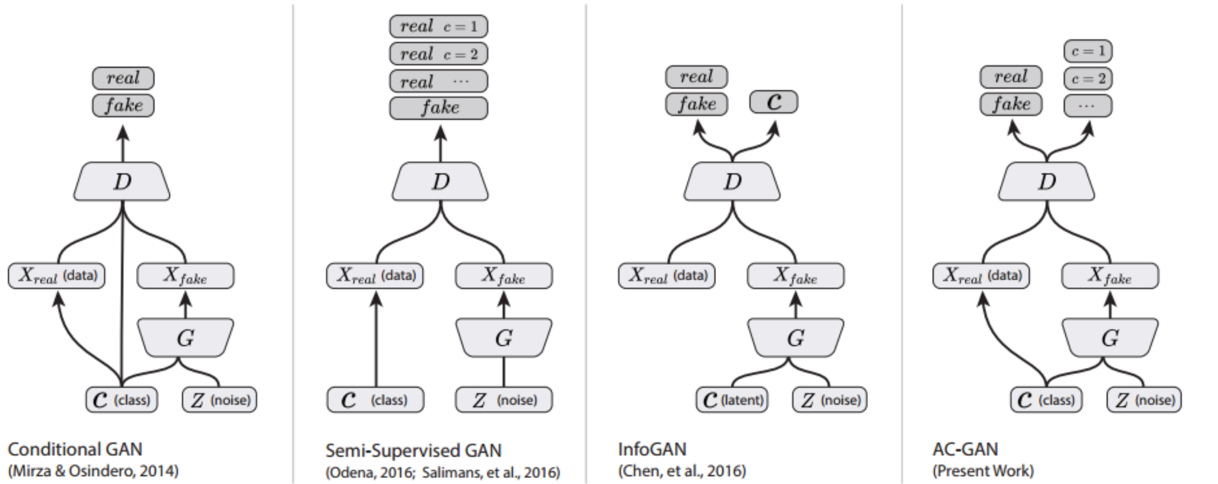


Figure 5: different architecture of variant of class conditioned GANs, in this project we implemented Conditioned GAN, infoGAN and AC-GAN. Class conditioned GANs have shown to improve the quality of generated output by explicitly providing the labels to model

Style Translational GANs

In cross domain image transfer tasks , we want to take an image from an input domain Di and then transform it into an image of target domain Dt without necessarily having a one-to-one mapping between images from input to target domain in the training set.

- Relaxation of having one-to-one mapping makes this formulation quite powerful - the same method could be used to tackle a variety of problems by varying the input-output domain pairs - performing artistic style transfer, adding bokeh effect to phone camera photos, creating outline maps from satellite images or convert horses to zebras and vice versa!!
- Style Translation tasks are broadly performed by three GAN models - CycleGAN, DiscoGAN, and StarGAN.
- In a nutshell, CycleGAN works by taking an input image from domain DA which is fed to our first generator GeneratorAB whose job is to transform a given image from domain DA to an image in target domain DB . This new generated image is then fed to another generator GeneratorBA which converts it back into an image, CyclicA , from our original domain DA (think of autoencoders, except that our latent space is Dt)
- CycleGAN and DiscoGAN use almost the same concept , losses and metrics but their they differ in their model structure . CycleGAN uses deep Residual connections whereas Model architecture of DiscoGAN is much simple and is very similar to DCGan .
- CycleGan and DiscoGan and other style translation approaches approaches had limited scalability and robustness in handling more than two domains, since different models should be built independently for every pair of image domains. To address this limitation, StarGAN, a novel and scalable approach was made which can perform image-to-image translations for multiple domains using only a single model. Such a unified model architecture of StarGAN allows simultaneous training of multiple datasets with different domains within a single network. This led to StarGAN's superior quality of translated images compared to existing models as well as the novel capability of flexibly translating an input image to any desired target domain.

Special GANs

Super-resolution GANs applies a deep network in combination with an adversary network to produce higher resolution images. As shown above, SRGAN is more appealing to a human with more details compared with the similar design without GAN (SRResNet). During the training, A high-resolution image (HR) is downsampled to a low-resolution image (LR). A GAN generator up-samples LR images to super-resolution images (SR). We use a discriminator to distinguish the HR images and back-propagate the GAN loss to train the discriminator and the generator (Figure 7).

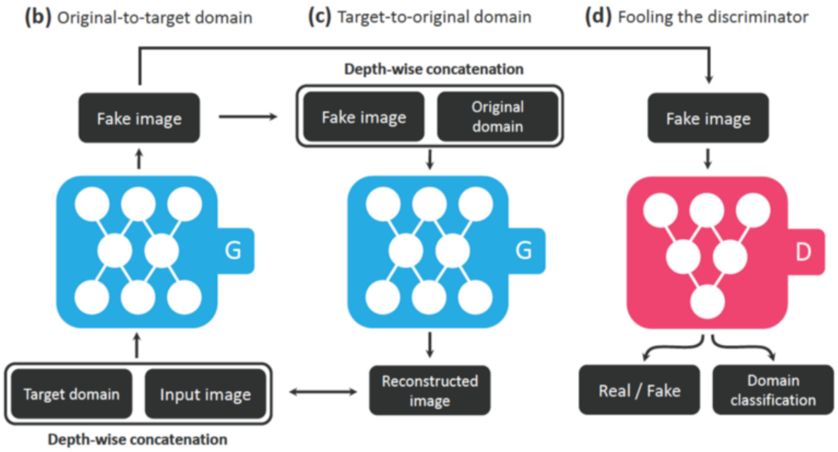


Figure 6: Overview of StarGAN, consisting of two modules, a discriminator D and a generator G. (a) D learns to distinguish between real and fake images and classify the real images to its corresponding domain. (b) G takes in as input both the image and target domain label and generates an fake image. The target domain label is spatially replicated and concatenated with the input image. (c) G tries to reconstruct the original image from the fake image given the original domain label. (d) G tries to generate images indistinguishable from real images and classifiable as target domain by D.

$$l^{SR} = \underbrace{\underbrace{l^{SR}_{\mathrm{X}}}_{\text{content loss}} + \underbrace{10^{-3} l^{SR}_{Gen}}_{\text{adversarial loss}}}_{\text{perceptual loss (for VGG based content losses)}}$$

Figure 7: perceptual loss which is used as metrics for optimmizing generator consist of weighted sum of content loss and adversial loss. Content loss ensure that the composition of output images and input images matches while adversial loss build up texture in super-resolution which other state-of-art SR model fail to generate

Techniques to Stabilize the Training of GANs

CGANs could easily generate images with a simpler geometry like Ocean, Sky etc. but failed on images with some specific geometry like dogs, horses and many more. The CGAN was able to generate the texture of furs of dog but was unable to generate distinct legs.

This problem is arising because the convolution is a local operation whose receptive field depends on the spatial size of the kernel. Making the spatial size bigger so that it captures more of the image would decrease computational efficiency achieved by smaller filters and make the operation slow. Hence the author of SAGAN introduced a self-attention module (Figure 9) into convolutional GANs which exhibits a better balance between the ability to model long-range dependencies and the computational and statistical efficiency.

Few Details from the paper (Summarized from the figure 10):

- They used this self-attention layer in both the generator and discriminator
- They applied spectral normalization to the weights in both generator and discriminator, unlike the previous paper which only normalizes the discriminator weights. They set the spectral norm to 1 to constrain the Lipschitz constant of the weights. It's just used for controlling the gradients. This spectral normalization idea was first introduced by Miyato et. al.
- They used a two-timescale update rule (TTUR) which is simply using different learning rate for both discriminator and generator.
- The metrics used in the paper are Inception Score (IS, higher is better) and Frechet-Inception Distance (FID, lower is better).

The paper explains with experiments how the Spectral Normalization and TTUR have helped the GAN to converge better. A picture of the same is shown below.

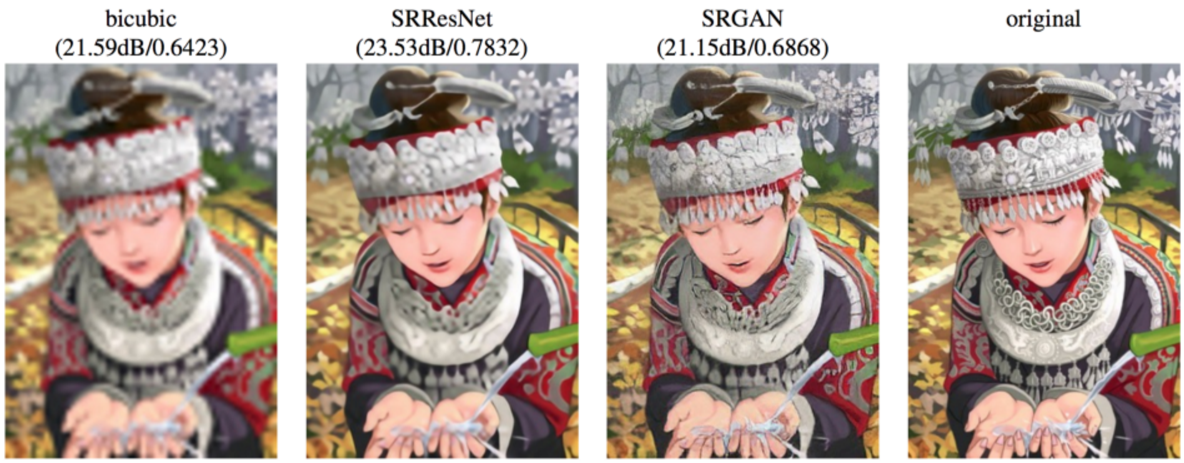


Figure 8: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4 upscaling]

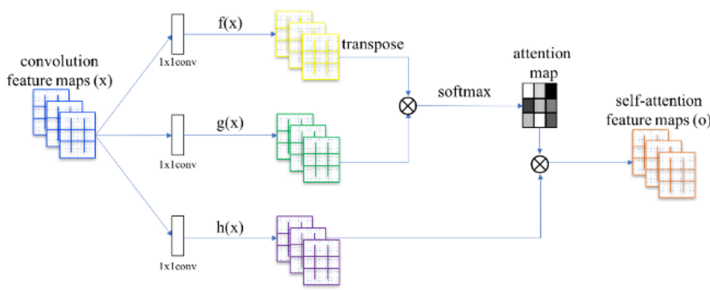


Figure 9: The proposed self-attention module for the SAGAN. The denotes matrix multiplication. The softmax operation is performed on each row.

NLP Gans

- GANs work by training a generator network that outputs synthetic data, then running a discriminator network on the synthetic data. The gradient of the output of the discriminator network with respect to the synthetic data tells you how to slightly change the synthetic data to make it more realistic.
- You can make slight changes to the synthetic data only if it is based on continuous numbers. If it is based on discrete numbers, there is no way to make a slight change.
- For example, if you output an image with a pixel value of 1.0, you can change that pixel value to 1.0001 on the next step. If you output the word "penguin", you can't change that to "penguin + .001" on the next step, because there is no such word as "penguin + .001". You have to go all the way from "penguin" to "ostrich".
- Since all NLP is based on discrete values like words, characters, or bytes, no one really knows how to apply GANs to NLP yet.
- In principle, you could use the REINFORCE algorithm, but REINFORCE doesn't work very well, and no one has made the effort to try it yet as far as I know. see other people have said that GANs don't work for RNNs. As far as I know, that's wrong; in theory, there's no reason GANs should have trouble with RNN generators or discriminators. But no one with serious neural net credentials has really tried it yet either, so maybe there is some obstacle that comes up in practice.

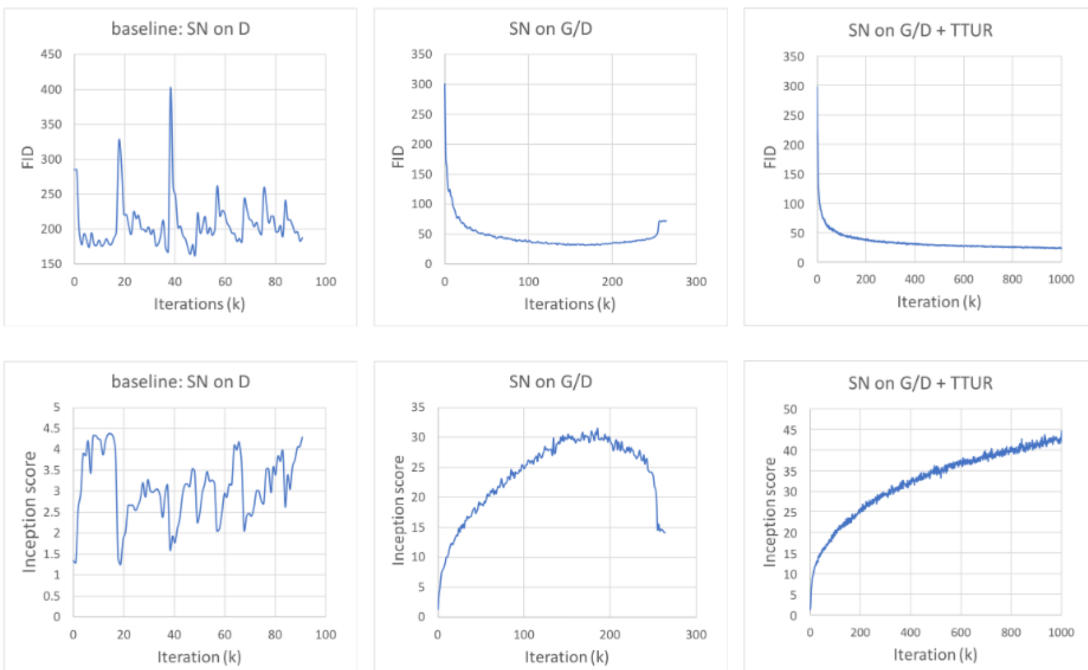
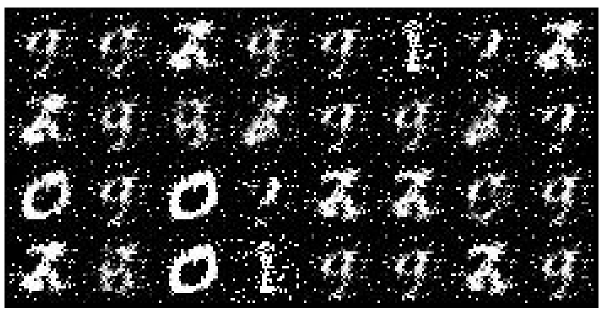


Figure 10: Training curves for the baseline model and our models with the proposed stabilization techniques, "SN on G/D" and two-timescale learning rates (TTUR). All models are trained with 1:1 balanced updates for G and D.

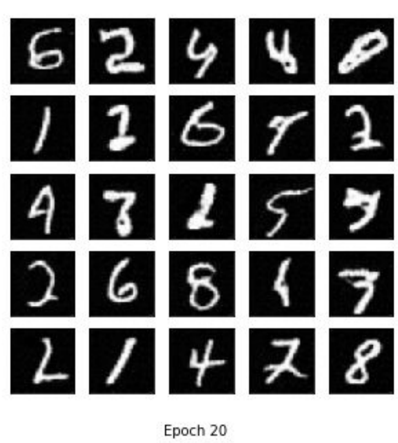
RESULTS

1. BasicGAN



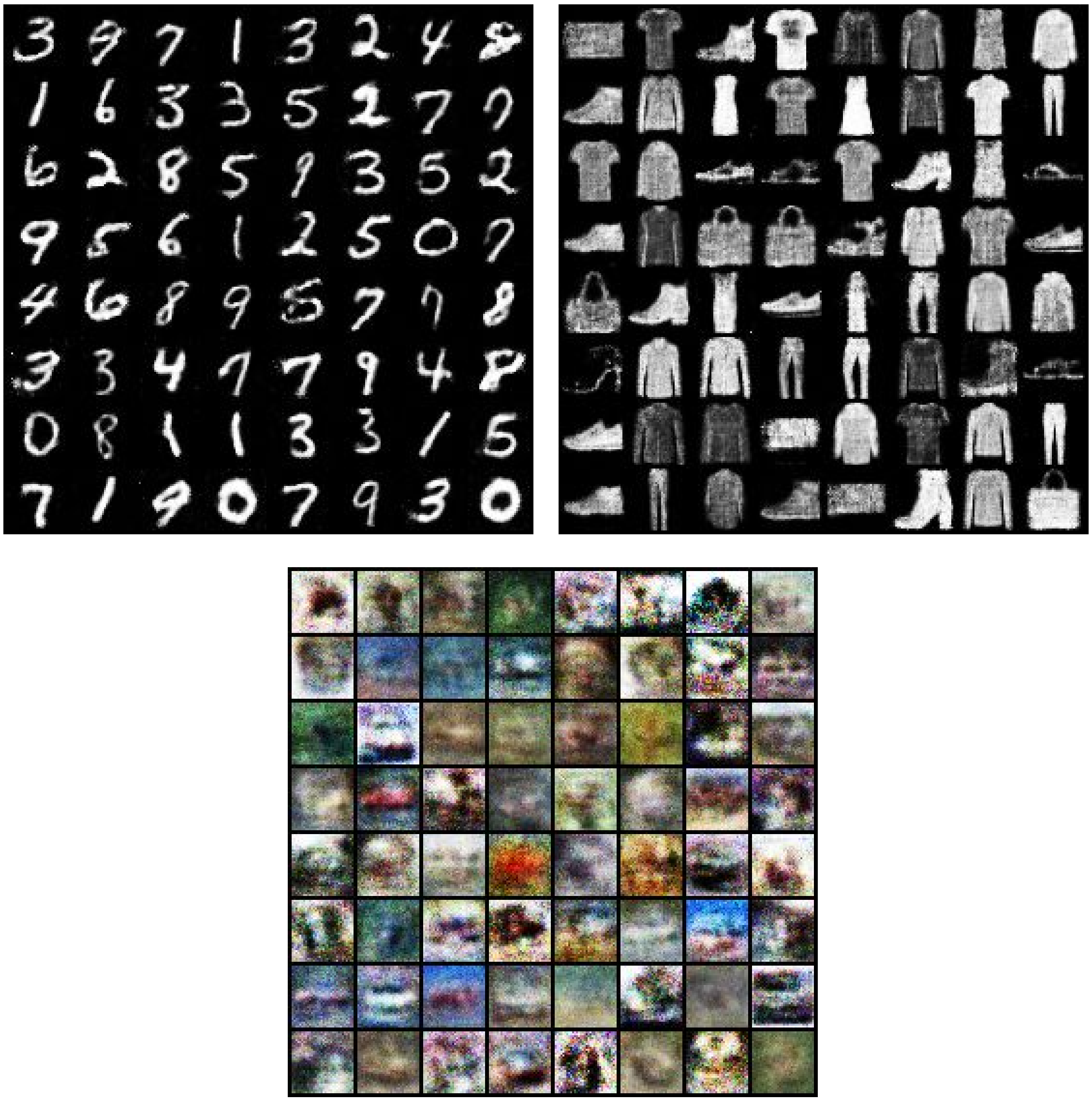
Hand-digit generation using basicGAN on MNIST dataset

2. DCGAN



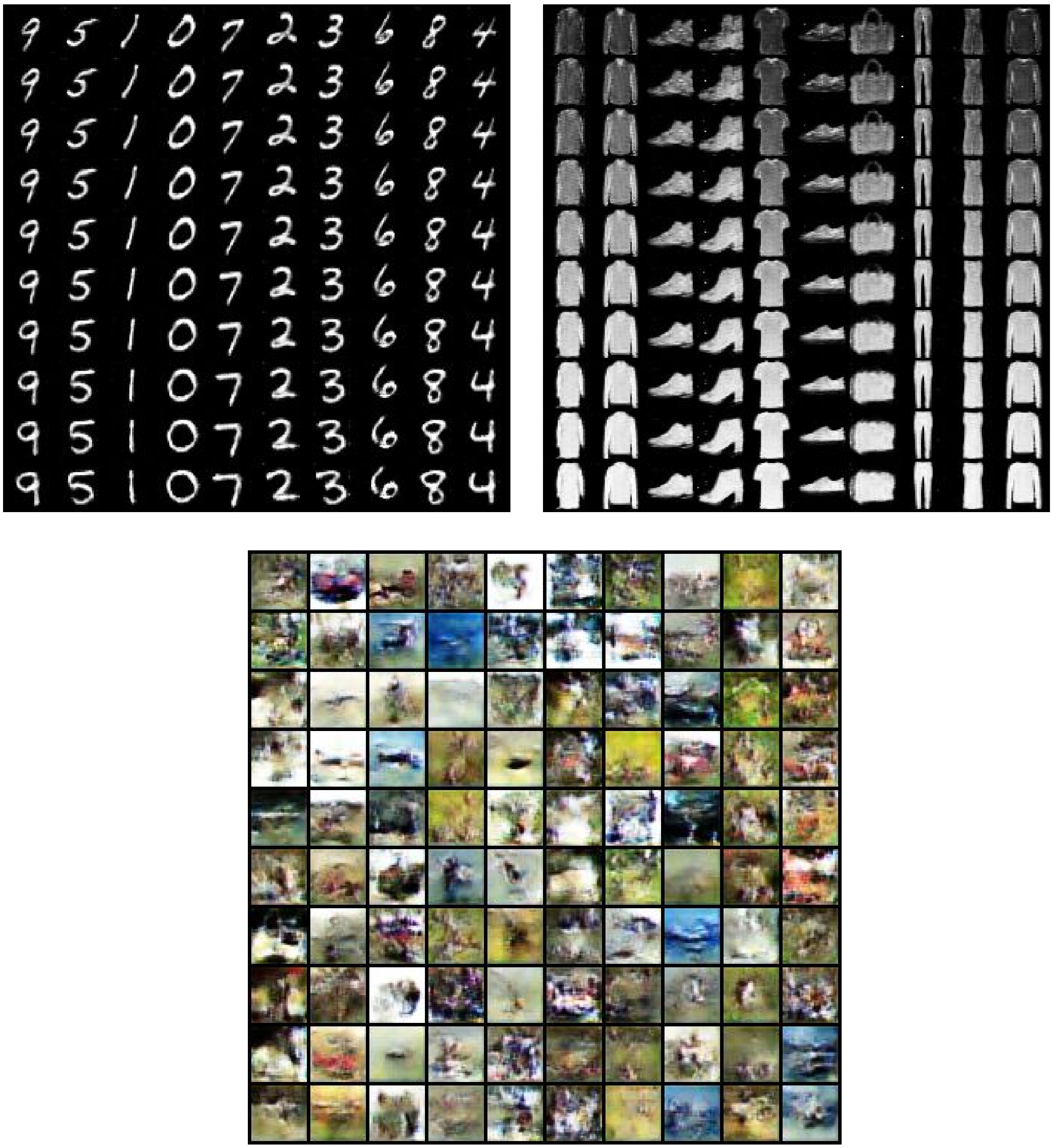
Results on DCGAN on MNIST and CelebA dataset

3. Conditional GAN



Result of cGAN on MNIST, FMNIST and CIFAR10

4. infoGAN

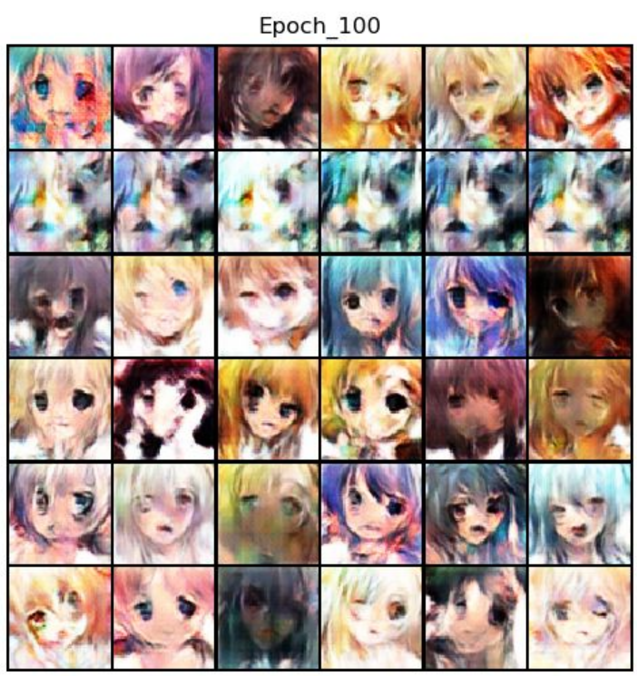


Result of infoGAN on MNIST, FMNIST and CIFAR10

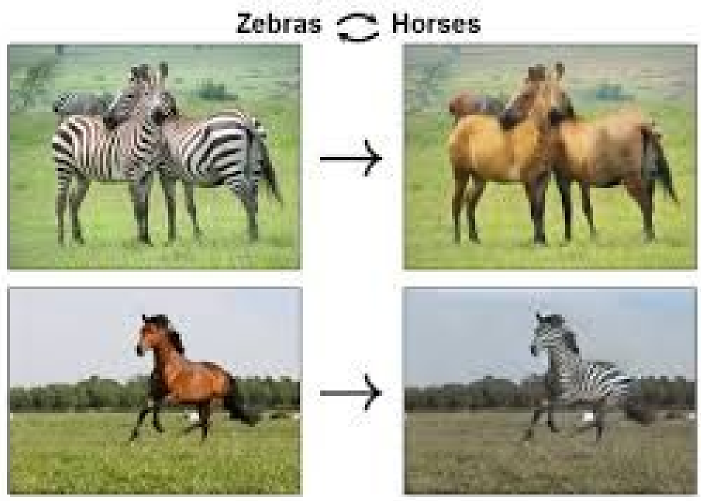
5. AC-GAN

Result of infoGAN on MNIST, FMNIST and CIFAR10

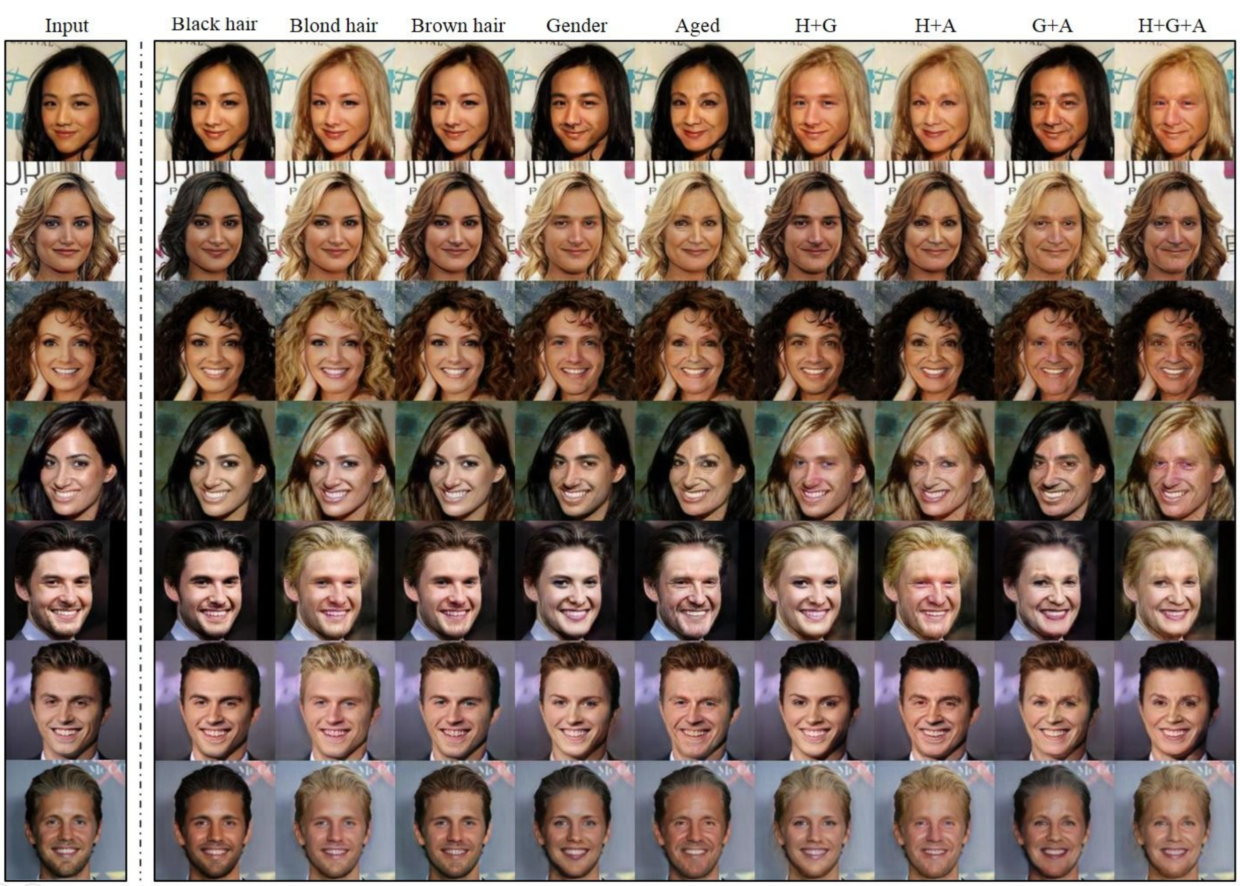
6. Amine GAN



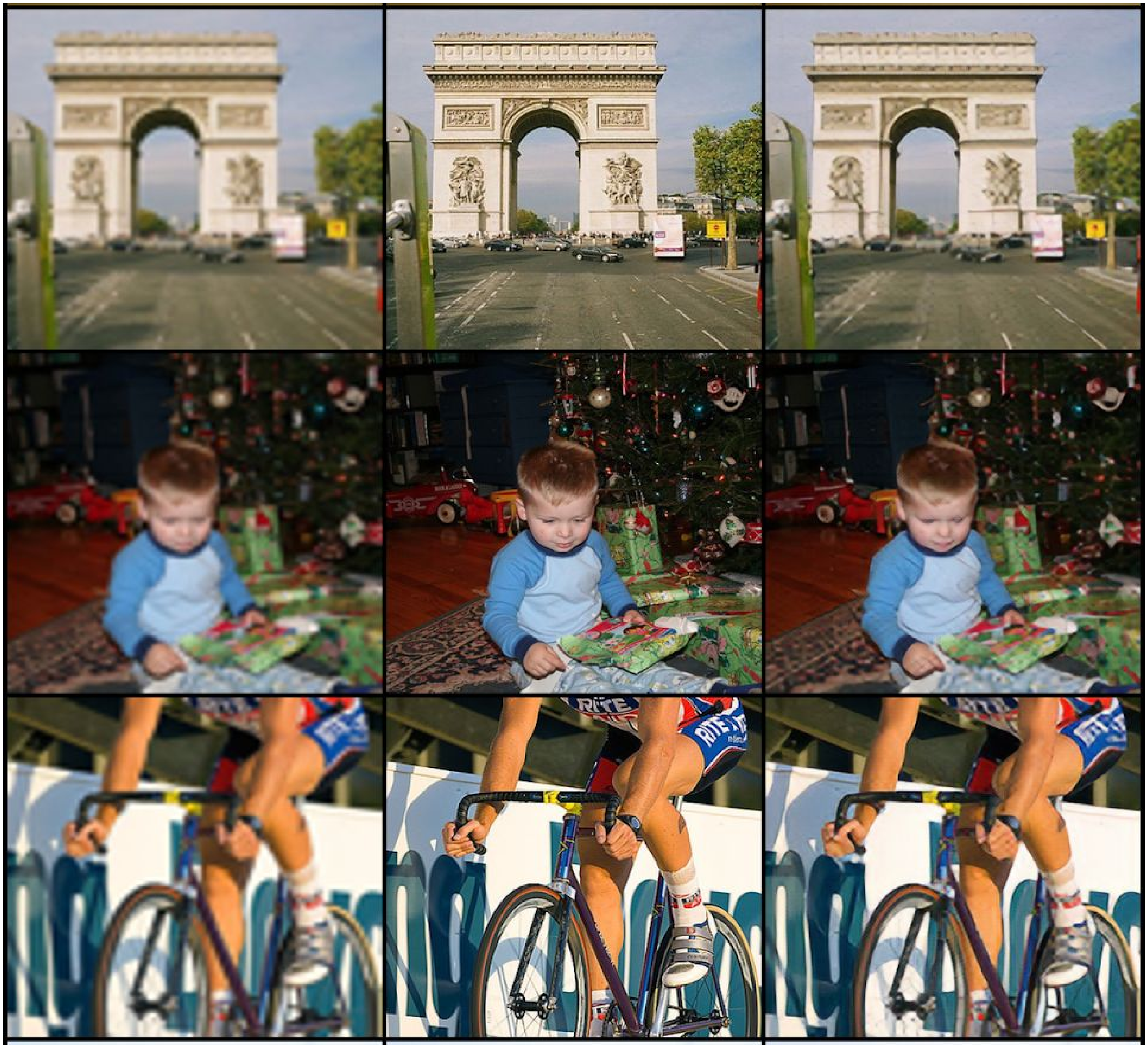
7. Cycle GAN



8. Star GAN



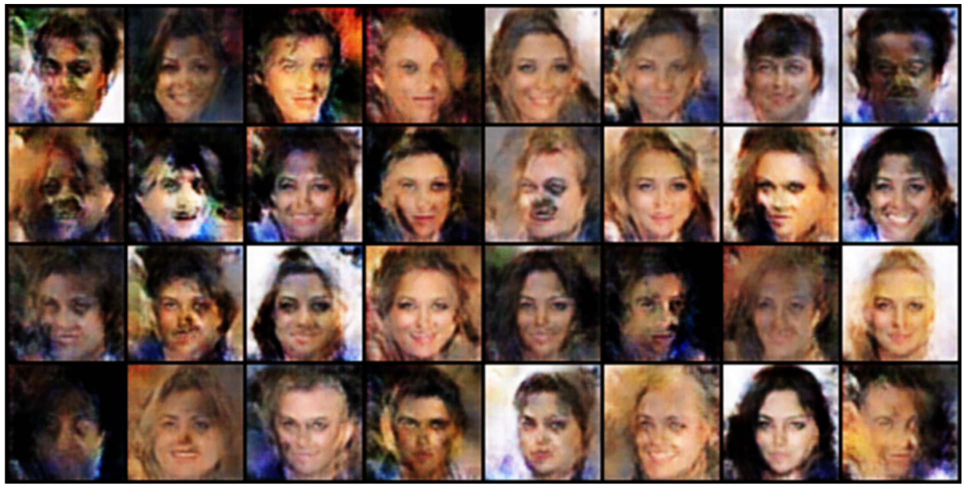
9. SRGAN



Left to Right: (1) Low resolution image (LR) which is also the inputs to our model (2) High Resolution image (HR) which is ground truth image for evaluating image [4x resolution of LR] (3) Super resolution image (SR) which is the generated image from the model

10. SAGAN

Epoch 0.5



Epoch 8

Self Attention module and other stabilizing method applied on DCGAN as proposed in the paper SAGAN. Above results suggest that our model has overcome the problem of long range dependencies which can clearly be seen in initial epoch’s result.