



General Adversarial Networks

01.05.2021

Dhruv Agarwal

Electronics Engineering

Objective

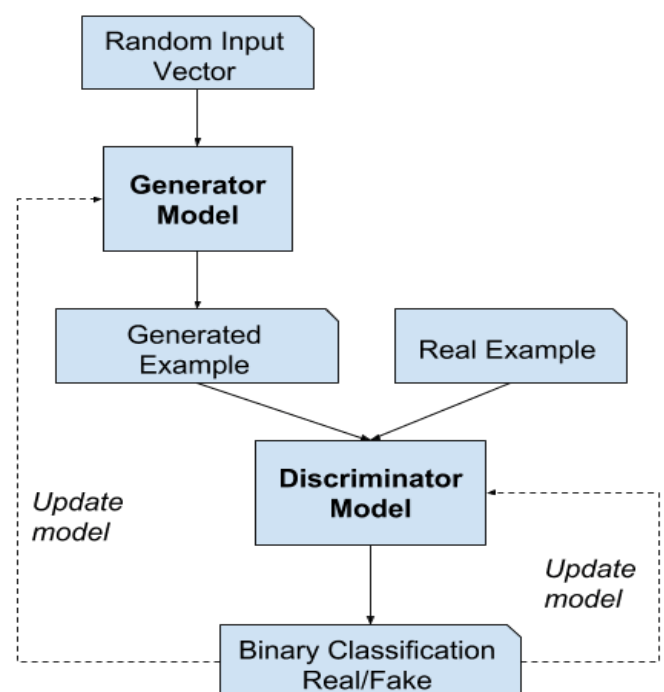
To train a generative adversarial network to generate images using the CIFAR10 dataset. There is no limitation to the kind of GAN you train. Report two results: One should be a baseline for your second result, and the second should be an optimisation or training technique which helped in stabilising your training of the GAN and trained it better.

Different models used during the project

1. Image generation using Simple-GAN
2. Image generation using DCGAN (Deep Convolutional GAN)
3. Image generation using WGAN-GP (Wasserstein GAN with gradient penalty)
4. Image generation using C-GAN (Conditional GAN)

Model description

General Adversarial Networks are used in image generation. The basic concept of GANs is that it has 2 networks playing an adversarial game against each other (where loss of one player is gain of the other). One of them is called Generator which generates images from random noise and the other is called Discriminator which distinguishes between fake and real images. The objective of the generator is to generate images that are indistinguishable from the real images, so that the discriminator can not decide which image is real and will have to guess with a probability of 0.5. By competing against each other both the networks train to get better and better.



1. SIMPLE_GAN

This model is the most basic implementation, just to get a grasp of the working of generative adversarial networks. Both the generator and the discriminator are using simple neural networks with very few layers . BCEloss function is used to compute the loss and Adam optimizer (with default parameters) is used to optimize both the networks .

Loss Function for Discriminator : **maximize : $\log(D(x)) + \log(1 - D(G(z)))$**

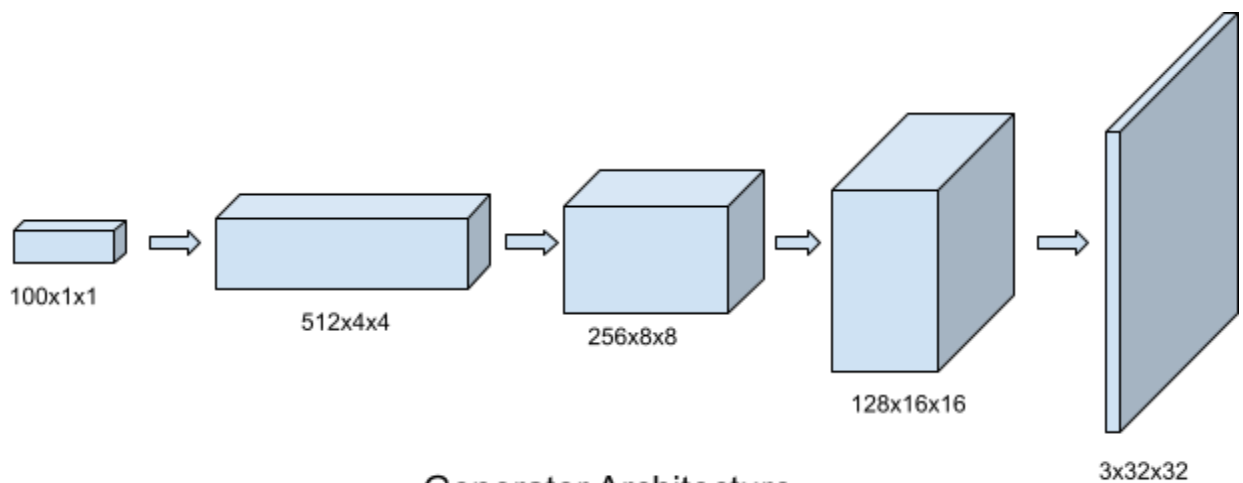
Loss Function for Generator : **maximize : $\log(D(G(z)))$**

Where :

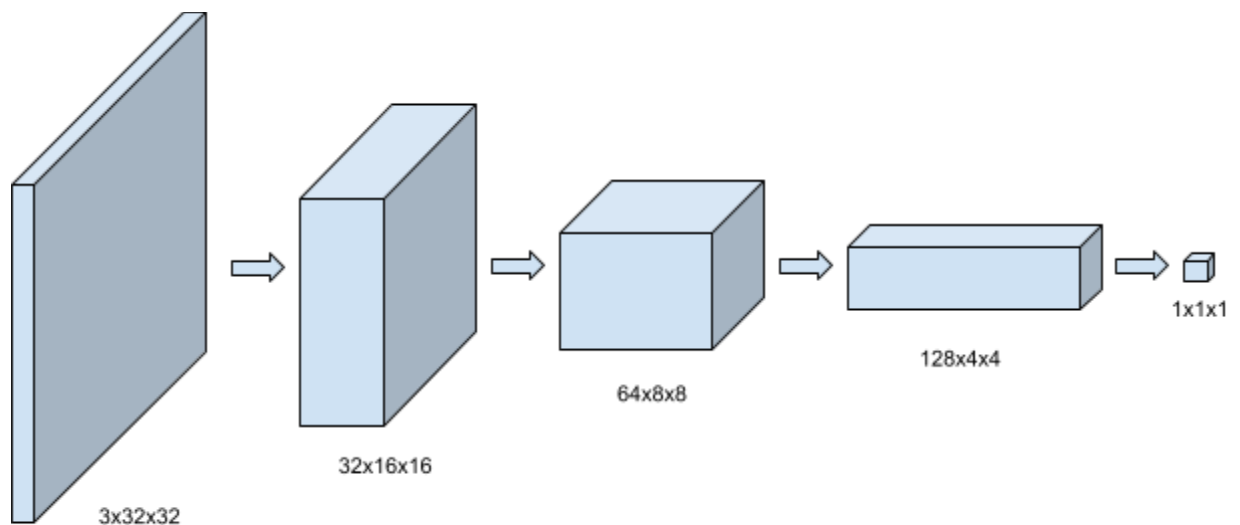
- D is the Discriminator
- G is the Generator
- X is sample from real images
- Z is random noise

2. DCGAN

Deep Convolutional GAN is a better implementation which uses a deeper convolutional neural network for both discriminator and optimizer. The architecture of the network used by me based on DCGAN is as shown in the figure.



Generator Architecture



Discriminator Architecture

At each layer in discriminator a convolution with kernel_size 4 , stride 2 and padding 1 , so at each layer the dimension of the image is reduced by half and the channels are doubled, followed by BatchNorm and LeakyReLU activation . At the last layer the convolution is set such that there is a 1x1x1 output and to ensure that it is between 0 and 1 it is passed through sigmoid . The model of the generator is just the opposite of discriminator starting with a noise of 100x1x1 and converting it into an image of 3x32x32 using transpose convolutions.

The training model is the same as that of the simple GAN with BCE loss function and Adam optimizer.

3.WGAN-GP

Wasserstein GAN with gradient penalty is an improvement of DCGAN and tries to make the training process more stable. Instead of the earlier used KL divergence and JS divergence it uses Wasserstein distance to minimize the difference between the probability distribution of the real image and the generated images as to generate real looking images.

In this model the discriminator is generally called a critic (It does not contain a sigmoid at the end and thus its output is not constrained between 0 and 1). The loss function is given as :

$$L = \underbrace{\mathbb{E}_{\tilde{x} \sim \mathbb{P}_g} [D(\tilde{x})] - \mathbb{E}_{x \sim \mathbb{P}_r} [D(x)]}_{\text{Original critic loss}} + \underbrace{\lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}} [(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]}_{\text{Our gradient penalty}}.$$

Where :

$\tilde{x} \sim \mathbb{P}_g$: Sample generated images

$x \sim \mathbb{P}_r$: Sample real data

$\hat{x} : \epsilon x + (1 - \epsilon)\tilde{x}$ (interpolation between real image and generated image)

λ : Lambda Parameter (regularization parameter)

D : Critic Function

ϵ : random number $\sim U[0, 1]$

Where the critic wants to maximize the function and the generator wants to minimize the function. The gradient penalty term in this function is added to enforce the lipschitz condition. The critic in this model is trained more than the generator and instead of the BatchNorm , InstanceNorm or LayerNorm is used as we do not want data to be normalized through the batch. Adam optimizer is used with beta values $\beta_1 = 0.0$ and $\beta_2 = 0.9$ (data from original paper).

4.CGAN

Conditional GAN is an improvement in the WGAN-GP model, where the classification labels of the image set can help improve the training process. Here we add an embedding layer in our image channels which supplies our generator information about which class to generate and the critic also judges fake and real images for the same label class. Thus the corresponding real and fake images (in the image grid) for a particular step are of the same class.

Problems Faced

1. At the earlier stage of the project the network was not converging because of its sensitivity to hyperparameters. Each test run took a lot of time, so to completely run a few epochs for different hyperparameters took a few hours, and further conclusions could only be drawn after that.
2. The network implementation in the paper of DCGAN was for 64 x 64 images and the images in the CIFAR10 dataset were of dim 32x32.
3. I was not able to train my models on my laptop for too long as it caused overheating and could potentially cause harm to my laptop, so I went on to train my models on Google Colab where I was not able to access my results from TensorBoard at first.

Solution to those problems

1. Ran each case with different hyperparameters for a few epochs and analyzed whether the network was converging or not and if it gave satisfactory results then only considered running it for many epochs.
2. There were 2 possible solutions to this
 - a. Resize the input image to 64x64 size and implement the same network architecture as given in the original DCGAN paper.
 - b. Change the network architecture to suit a 32x32 size image, by changing the number of convolutional layers.

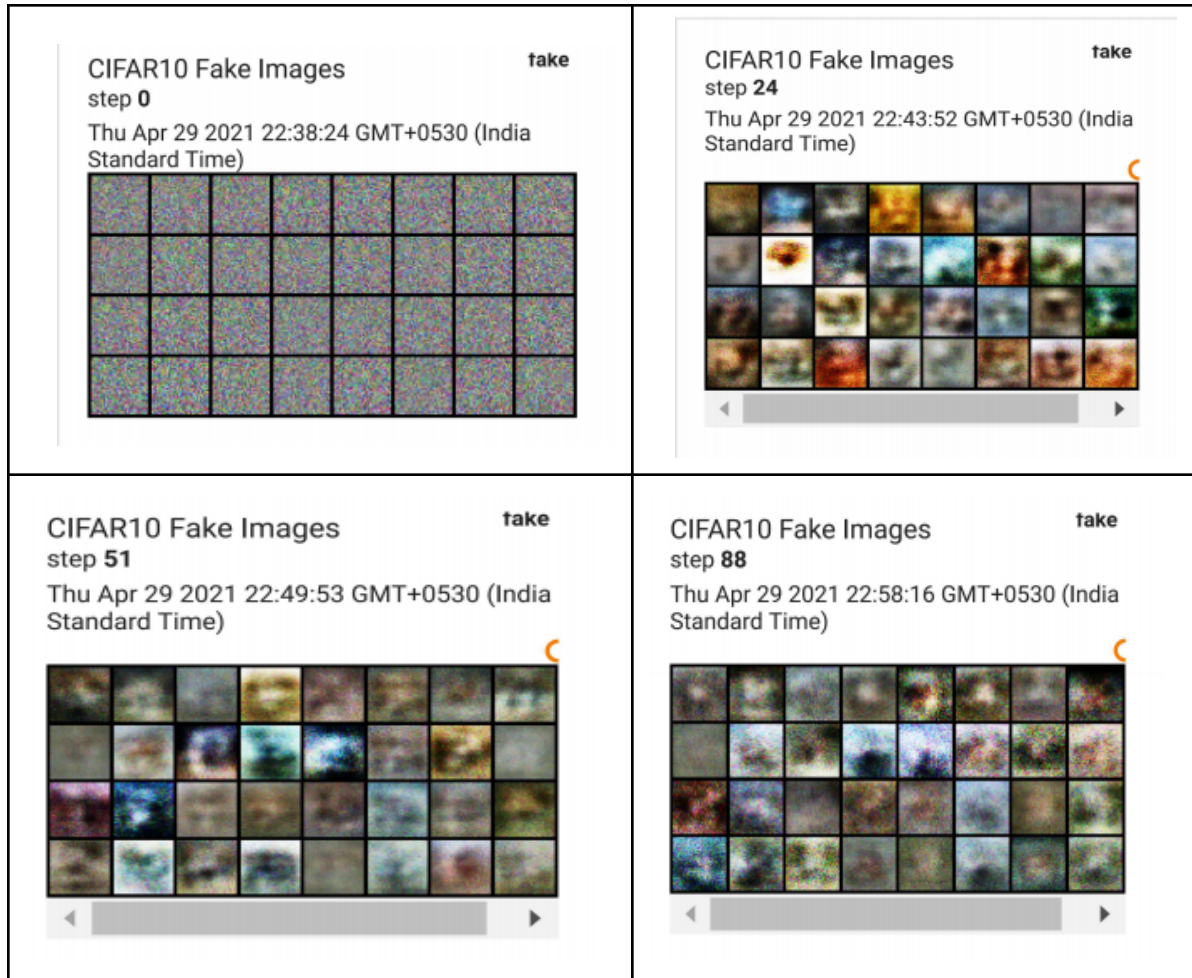
I choose the latter solution as I believe resizing the image will distort its resolution and make the training process even harder and will take more time to reach a satisfactory result.

3. After reading a few articles on Google I was able to print my TensorBoards after each iteration using command `%load_ext tensorboard` and `%tensorboard --logdir logs`.

Comparison

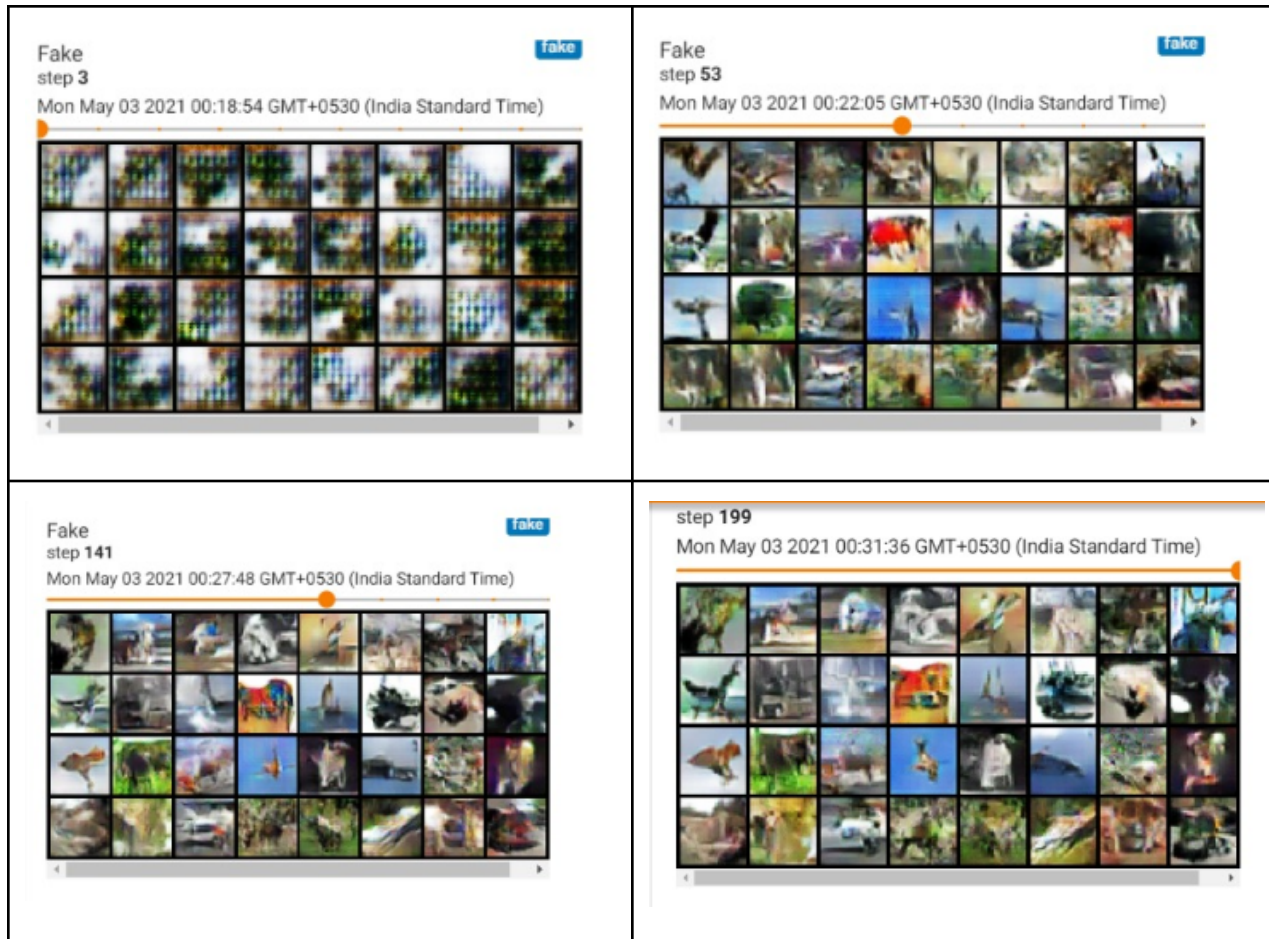
In this section I will attach the results of all the models for different steps for comparison .

SIMPLE_GAN



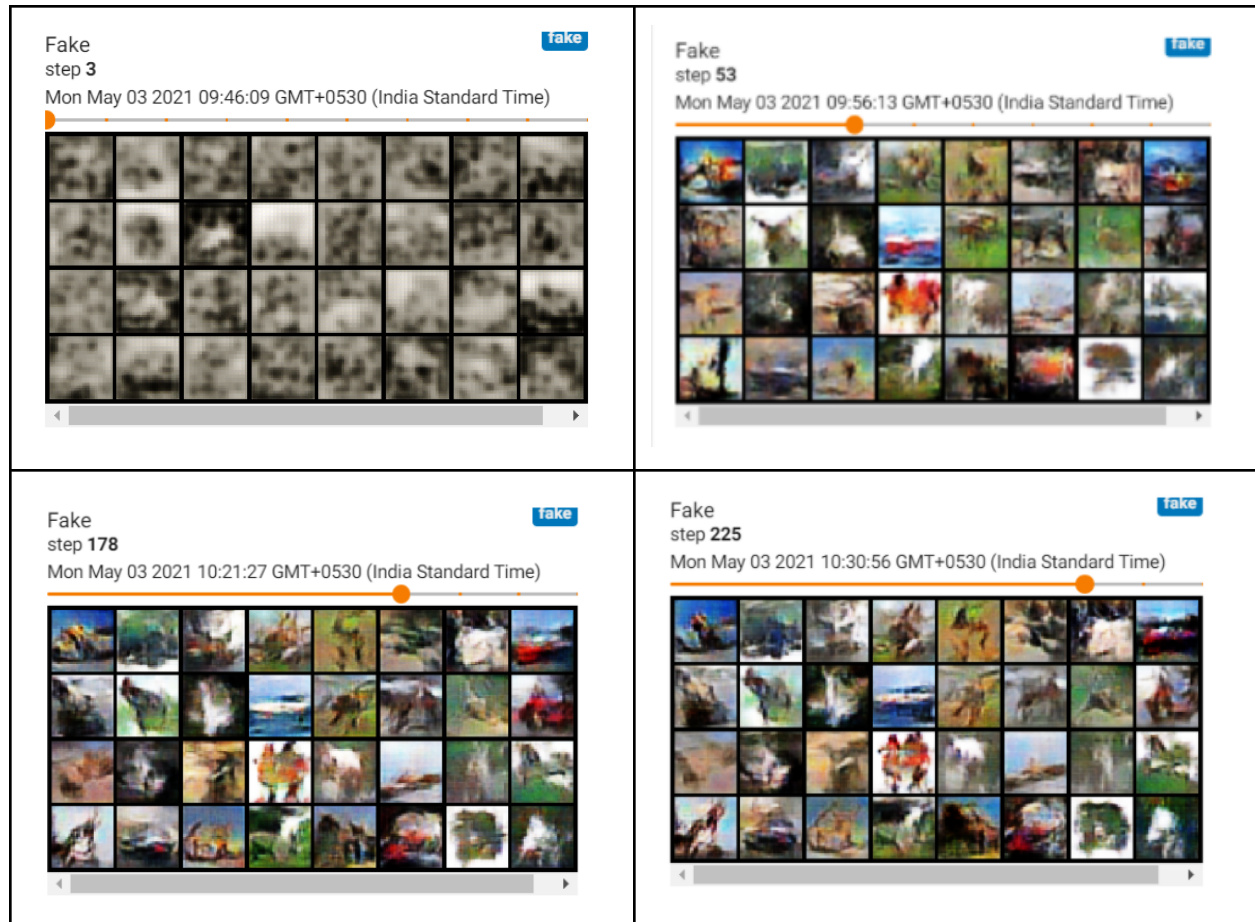
The above images shown are images generated by SimpleGAN. SimpleGAN is highly sensitive with respect to the hyperparameters. The hyperparameters chosen while generating above images were finalised after try and error, which gave satisfactory convergence after a few steps. The above images show significant signs of convergence as blur faces and automobile outlines are visible. There might be better hyperparameters for which the images could have better convergence but with due consideration to time I had while training these, I finalized these images to be added in my report.

DCGAN



The above images shown are generated by DCGAN. These show significant improvement from the Simple_GAN. The images have converged really well. Some ducks, birds, ships and frogs can be well recognised. This model is also very sensitive towards hyperparameters and I have used the hyperparameters suggested in the original paper.

WGAN-GP

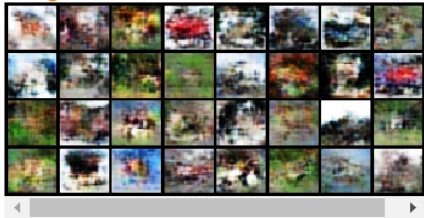


The above images shown are generated from WGAN-GP. Some automobiles, birds, frogs and ships can be easily recognised. This network is less sensitive towards hyperparameters. This model takes a large number of steps to generate good fake images , but if I were able to train this network for a larger number of epochs (like for around 500 steps) , I believe this model would have given great results.

CGAN

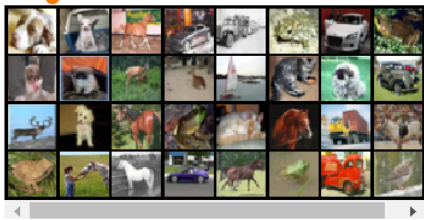
Fake

Fake
step 20
Mon May 03 2021 09:59:48 GMT+0530 (India Standard Time)



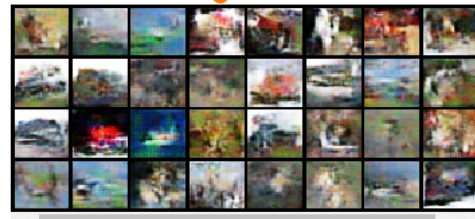
Real

Real
step 20
Mon May 03 2021 09:59:48 GMT+0530 (India Standard Time)



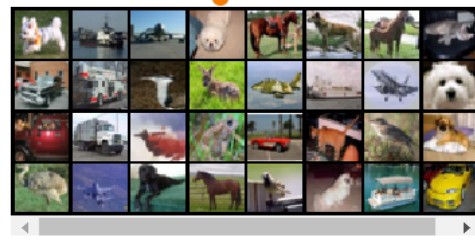
Fake
step 53

Mon May 03 2021 10:09:53 GMT+0530 (India Standard Time)



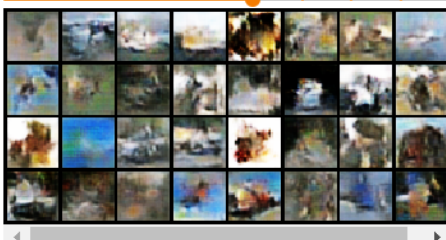
Real

Real
step 53
Mon May 03 2021 10:09:53 GMT+0530 (India Standard Time)



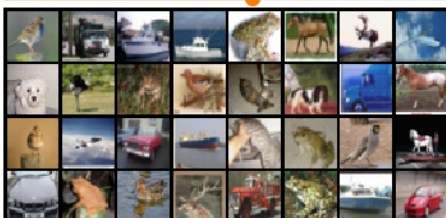
Fake
step 141

Mon May 03 2021 10:36:39 GMT+0530 (India Standard Time)



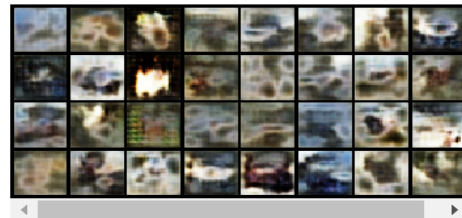
Real

Real
step 141
Mon May 03 2021 10:36:39 GMT+0530 (India Standard Time)



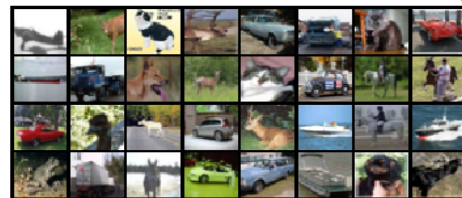
Fake


Fake
step 285
Mon May 03 2021 11:20:11 GMT+0530 (India Standard Time)



Real

Real
step 285
Mon May 03 2021 11:20:11 GMT+0530 (India Standard Time)





The fake images shown above are generated by CGAN. The speciality of this model is that the fake images generated are of the same label class as the real image in the corresponding cell of table (corresponding images of a batch). This model is better than other models as it creates a new image every single time. The other models just try to improve the quality of the images produced in the first batch (as can be seen in the images from other models). Though this was working really well , I am not yet able to understand why in the later (after around 200) steps white colour blobs start appearing. I suspect that it is because of overfitting and by producing such blobs the network is still able to reduce loss and it is escaping producing new images each time and just report these images with blobs with similar probability distribution as real images. This is a topic I would like to explore more upon and try to find which part of my code needs to be improved.

Insight

I came across a lot of models while studying for this project and was really amazed by the development in this field in a span of a few years. Right now I have only understood a few of the models developed by other people and was able to modify them as per my requirements to train it on CIFAR10. I believe studying and reading about different models have helped me understand what is happening in this field and what are the problems and solutions, and by practicing more I believe I would also be able to help in the development of this field. The concept of Image generation and labeling it would further help in increasing our dataset for supervised learning and in training models to more improved accuracy. The concept of SRGAN - Super Resolution GANs is very intriguing and can be used for restoring old images and I believe if it is further developed and clubbed with Natural Language Processing , it can be used to restore old damaged documents.

Conclusion

I have used 4 different architectures for this project and with each model I have tried to improve the fake image and make it indistinguishable from the real image. Due to the resources available I was only able to train these models for around 50 epochs and I believe that if these models are further trained they should be able to generate even better results. For training of GANs it is highly necessary to select the correct hyperparameters and when doing in real practice a loop for great hyperparameter search would be really helpful as with correct hyperparameters the model will converge really fast and produce great fake images.

Future Work

I would like to further work on GANs and I am really excited to implement the model for SRGAN - Super Resolution GANs and recreate some images from really old movies or some of my own photo albums. I also want to rectify the 'blob' problem for my code for CGAN and modify the CGAN in ACGAN (Auxiliary Classifier GAN). I would also love to experiment with style transfer and make some interesting images.

In this short span of time I was only able to look into the Computer Vision section of machine learning, now I would like to go study Natural Language Processing and Reinforcement Learning in depth and get a fair idea about what they are and what are their future applications.

Bibliography

- <https://youtube.com/playlist?list=PLhhyoLH6lJfwlp8bZnzX8QR30TRcHO8Va> playlist on GANs by Aladdin Persson -- understood and studied different GANs and their implementation.
- <https://youtube.com/playlist?list=PLhhyoLH6lJfxeoooqP9rhU3HjIAVAJ3Vz> playlist on PyTorch basics by Aladdin Persson -- learned basics of PyTorch
- <https://pytorch.org/tutorials/beginner/basics/intro.html> understood basic of pytorch
- <https://youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLqRF3EO8sYv> course on computer vision by Stanford -- learned all about computer vision and machine learning