Deep Learning Summative Assignment

1 Introduction

The purpose is to generate a pegasus utilizing the Cifar-10 dataset. This paper first outlines pre-processing stage, methodology and reasons for chosen architectures, the experiments performed and finally how it could be improved. Code implementation includeS Deep Convolutional Generative Adversarial Network -DCGAN- [1] and is modified with spectral normalization for smoother training phase. DCGAN was favoured over other generative models as it can produce more realistic images at the cost of unstable training phase of the model which is to be stabilized.

2 Preprocessing

Cifar-10 train set consists of 10 classes with 5000 data for each class deemed as training purposes and 1000 images with their labels for testing set. Testing data was not needed for validation so it was combined with the training data to be utilized in the training phase. Data augmentation was performed particularly random horizontal flip and normalizing to mean of 0.5 and standard deviation of 0.5 to have a better distribution and less variation in pixel intensity differences making the dataset more suitable. Transforming to grayscale was considered for efficiency but proved unnecessary as resources provided were sufficient even on three channels per image.

3 Design Architecture

The task of generating a pegasus is approached by using Spectral Normalization DCGAN. The initial architecture was taking as input the dataset images which were of size 32 but were then resized to 64 for visualization purposes as well as increasing the dimensions makes it more difficult for the generator to fool the discriminator.

The generator maps latent space vector of input size 100 to images of size 3x64x64 where 3 is the channel of the images and 64 are the dimensions of the images with a batch size of 64 images to be considered. Fractional strided convolutional two dimension transpose layers with two dimensional batch normalization and leaky rectified linear unit are used in the generator as batch normalization after convolutional transpose layer helps the flow of gradients as shown in original paper [2]. For the discriminator, initially, strided convolution layers are joined with two dimensional batch normalization and leaky rectified linear unit to take as an input the image batch provided by the generator and output a probability value between 0 and 1 to indicate whether it is fake or real respectively which is achieved with the sigmoid function. However, after adding spectral normalization [3] with code provided and modified from lectures, batch normalization was removed.

As indicated [2], strided convolution is preferred rather than pooling to down-sample as the network can learn its own pooling function. For the loss function, Binary Cross Entropy was chosen and adam optimizers are used for both generator and discriminator with learning rate 0.0002. According to one of the author of the original paper, Mr Chintala in his workshop on adversarial training [4], a modified loss function is used in practise to optimize the generator is identical to minimizing the logarithm of one minus the discriminator and it is more effective during training phase to use different batches for real and fake labels.

3.1 Generator architecture

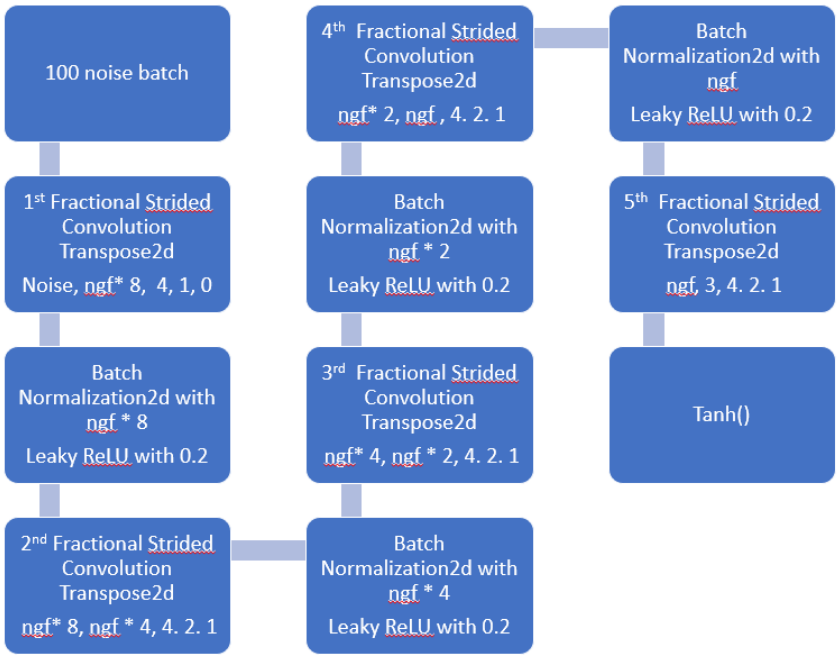

Figure 1: Generator Architecture DCGAN

3.2 Discriminator architecture

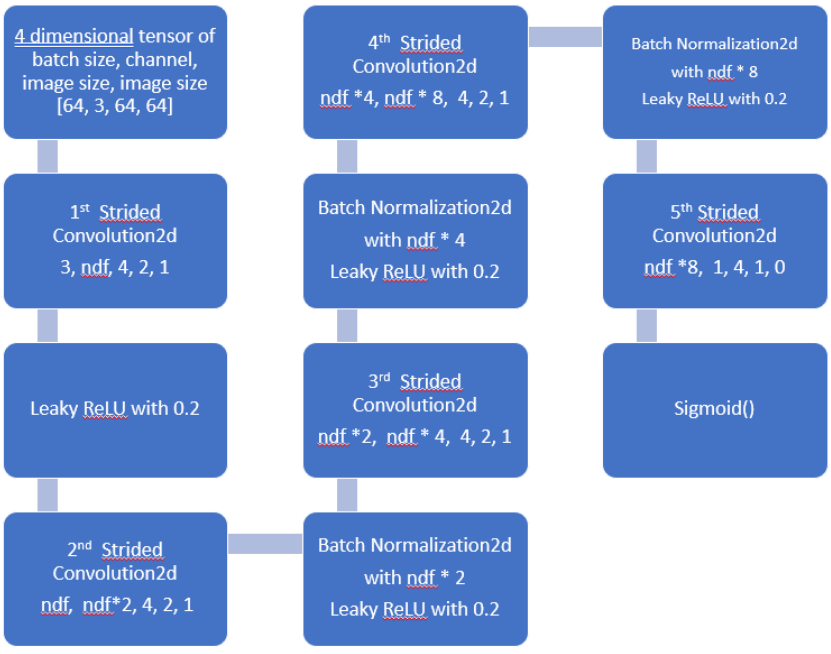

Figure 2: Discriminator Architecture DCGAN

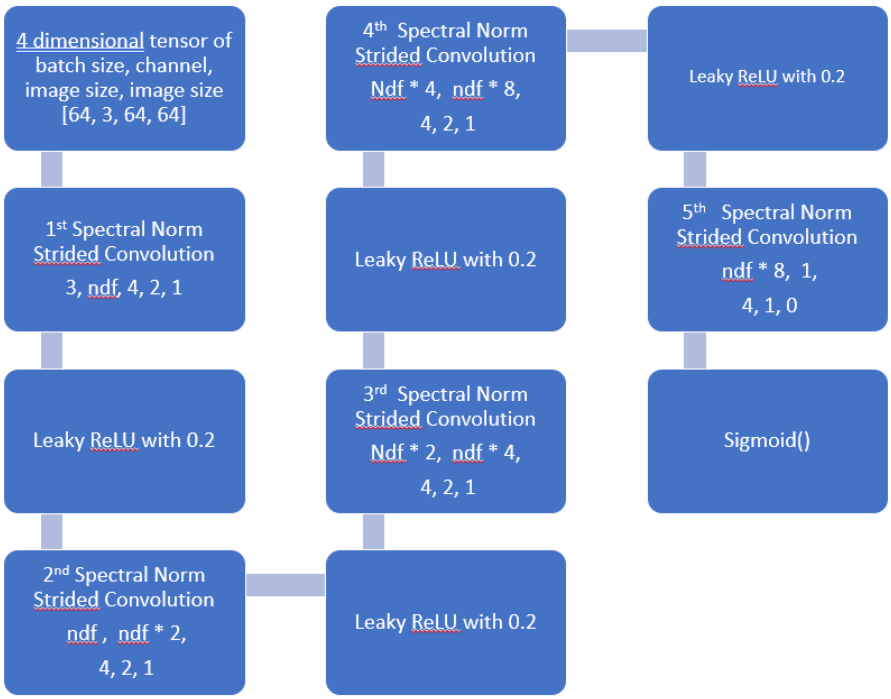

Figure 3: Discriminator Architecture SNDCGAN

3.3 Best Pegasus Image

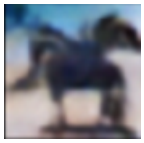

Figure 4: Horse with bird using DCGAN

3.4 Best Batch

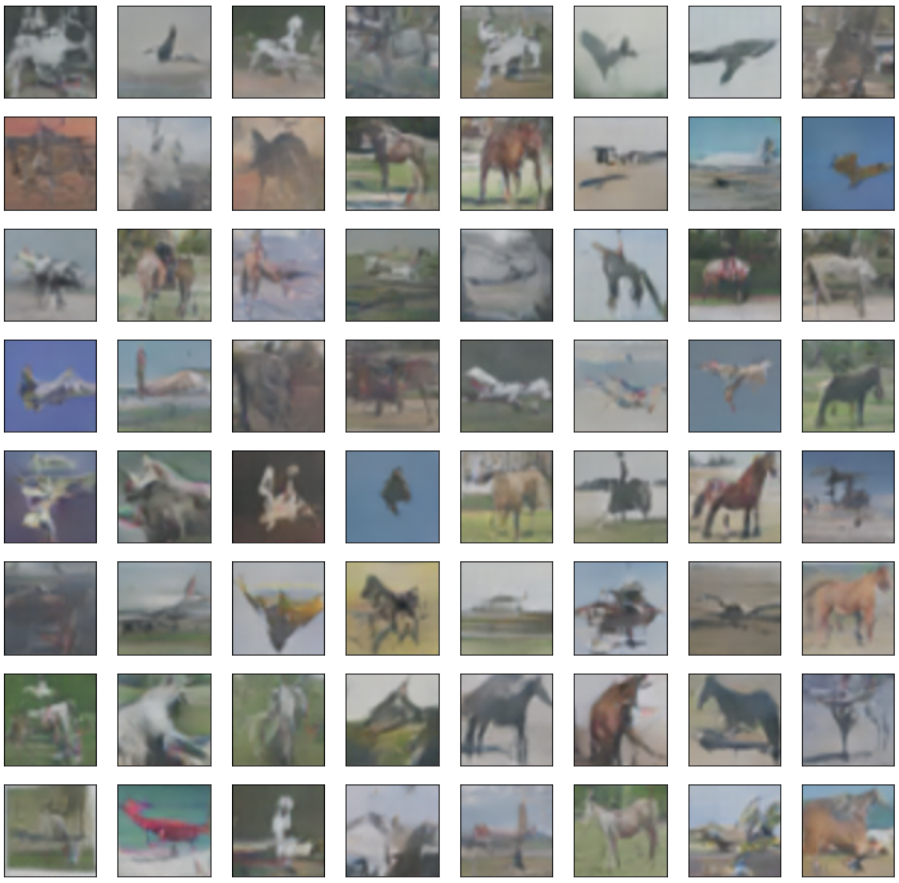

Figure 5: Pegasus made out of horses and planes using SNDCGAN

4 Experiments

First experiment occurred in the pre-processing stage where the options considered were to merge the horses with the birds or with the airplanes. Subsequent choice was to load the two classes separately and iterate over both dataloaders or combine them in one dataloader while in training phase. Experimentation was based in an effort to bring the latent space interpolation in the training phase. Firstly, separate dataloaders were used for birds and horses and for a maximum of 250 epochs, interchange dataloaders during each epoch ended up producing one excellent pegasus as indicated by the above figure. Further experimentation and upon considering the bird class depicting very inconsistent results, airplanes were chosen to create cyborg-pegasus. However, in all scenarios considered the training was unstable with both network loss functions to be unpredictable which was fixed by applying spectral normalization.

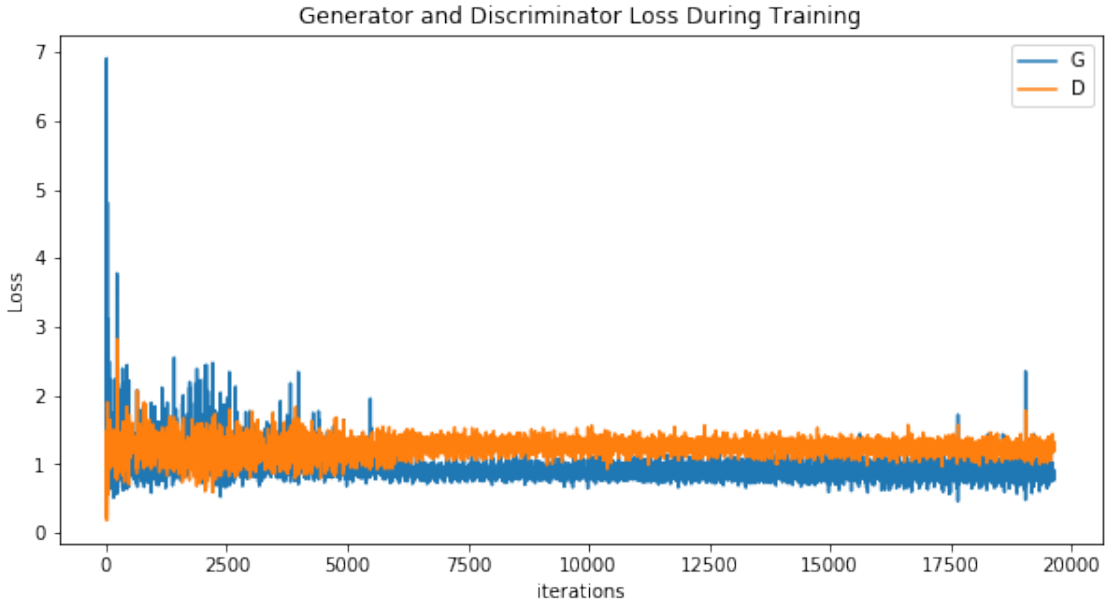

Figure 6: Training prior spectral normalization discriminator loss is the sum of fake and real labels

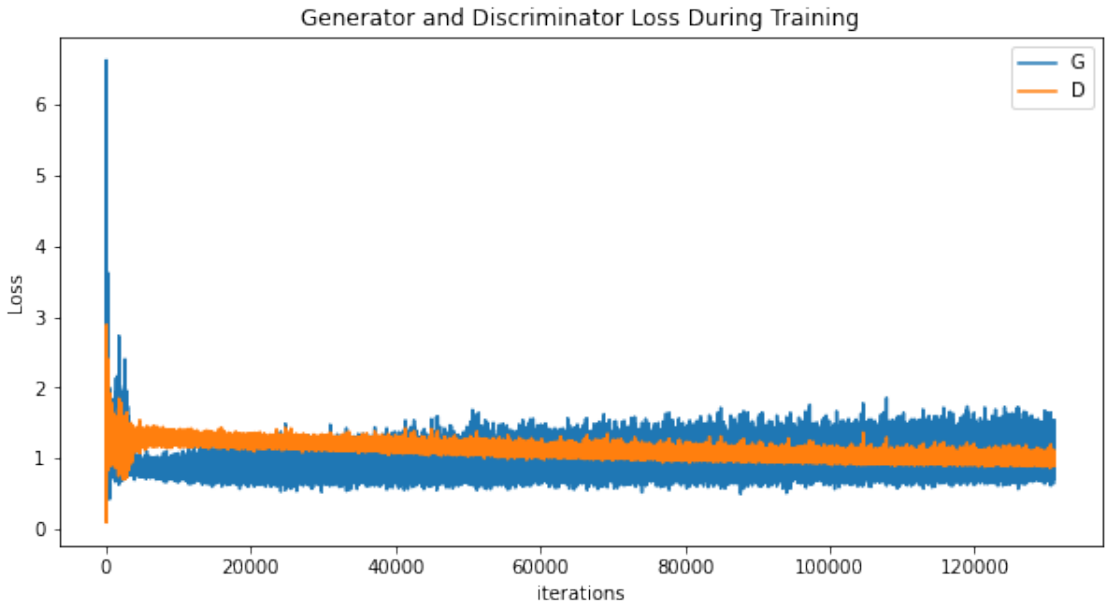

Figure 7: Training post spectral normalization and discriminator loss is the sum of fake and real labels

Once spectral normalization was implemented, loading different dataloaders in each epoch was not an option as it would re-initialize the Lipschitz constant which would not be useful as they mitigate the exploding gradient problem and the mode collapse problem. A dictionary was then created with phases of value 1 to 5 and corresponding key the number of iterations in each phase and odd phases would represent the horse class and even the wing class in an effort to phase between the two classes while learning to produce good quality images of both classes.Then as number the value of phase key reached 5, the number of iterations decreased but it did not yield satisfying results. Thus a new dataloader is created combining both classes rather than having disjoint dataloaders which is the method used to generate the diverse batch indicated and proved to be most effective indicated by batch figure.

5 Further work

An improvement in the training stability of the gan would be to use soft and noisy label instead of hard labels as it would be making stochastic value for the fake label to be between 0 and 0.3 and 0.7 until 1 for the real

label. Noisy labels for discriminator means with some probability p to flip the real with fake labels [3]. Even though SNGAN is successful in generating better results than dcgan, there is still no guidance in the latent space to interpolate between the two classes. An attempt was performed using a conditional dcgan with the class of the images to be fed in the generator and discriminator. As two classes were considered, the extra information was 1 bit and was added to the latent space vector and post training phase, by modifying that bit, interpolation could be achieved. However, because of time constraint, this idea was not fully developed but the source code is provided.

Last improvement would be to feed to the architecture horses so that it can generate very good horses as the dataset is contains blurry and zoomed in images making a diverse batch impossible and then freeze the discriminator and feed in the generator good images of birds to generate a diverse batch of pegasus.

```
Generator(
  (emb): Linear(in_features=101, out_features=1, bias=True)
  (main): Sequential(
    (0): ConvTranspose2d(101, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): LeakyReLU(negative_slope=0.2, inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.2, inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): LeakyReLU(negative_slope=0.2, inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

Figure 8: Generator architecture of cdcgan

```
Discriminator(
  (emb): Linear(in_features=101, out_features=1, bias=True)
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Conv2d(512, 100, kernel_size=(4, 4), stride=(1, 1), bias=False)
  )
)
```

Figure 9: Discriminator architecture of cdcgan

References

[1] DCGAN Face Tutorial

[2] Radford, A.; Metz, L. and Chintala, S. (2015), 'Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks' , cite arxiv:1511.06434

[3] Miyato, Takeru et al. (2018) "Spectral Normalization for Generative Adversarial Networks." , cite ArXiv abs/1802.05957

[4] NIPS 2016 Workshop on Adversarial Training - Soumith Chintala - How to train a GAN