CSE 676: Deep Learning (Fall 2019)

Implementation of Generative Adversarial Networks (GAN)- DCGAN & SAGAN on CIFAR10 dataset

Sonali Naidu
Department of Computer Science
University at Buffalo
Buffalo, NY – 14260
sonaliye@buffalo.edu

Abstract

In this project we have worked on implementation of the two types of Generative Adversarial Networks which are as follows: Deep Convolutional GAN (DCGAN) and the Self Attention GAN (SAGAN) on CIFAR10 dataset. GANs are kind of Neural Networks that consists of 2 separate deep neural networks the generator (G) and the discriminator (D) competing each other and making each other stronger. DCGAN is one of the simple variants of GAN in Deep Convolutional GANs where G and D are both based on deep convolutional neural network. SAGANs is complementary to convolutions and helps with modeling long range, multi-level dependencies across image regions. The goal of this task is to generate data points similar to datapoints in the training set.

1. INTRODUCTION

Generative Adversarial networks (GANS) are deep neural net architectures that comprises of two nets, pitting one against the other (hence the name adversarial). GAN have a huge potential because they learn to mimic any distribution of data and can be taught to create worlds similar to our own in any domain i.e., images.

Deep Convolutional Neural Networks (DCGANs) are a class of CNNs that have certain architectural constraints which demonstrate that they are a strong candidates for unsupervised learning. It is composed of convolution layers without max pooling or fully connected layers. It uses the convolutional stride and transposed convolution for the upsampling and downsampling.

Self Attention Generative Adversarial Network (SAGAN)allows attention-driven, long-range dependency modelling for image generation tasks. In SAGAN, the details can be generated using cues from feature locations. The discriminator can check whether the highly detailed features in distant portions of the image are consistent with each other.

2. DATASET

The dataset used is CIFAR-10 which are labelled subsets of 80 million tiny images dataset. These images are much smaller than the typical photograph and are intended mostly for computer vision research. CIFAR is an acronym that stands for the Canadian Institute For Advanced Research. It consists of 60000 32x32 images in 10 classes, with 6000 images from each class. The dataset is divided into 50000 training images and 10000 test images. It is divided into five training batches and one test batch consisting of 10000 images. Thus the training batch contains exactly 5000 images from each class.

3. ARCHITECHTURE/ALGORITHM

3.1 GANs:

One of the neural networks from GAN, called the generator, generates new data instances, while the other, the discriminator, evaluates for authenticity. In other words, the discriminator takes decision on whether each instance of data that it reviews belongs to the actual training dataset or not.

Steps:

- The generator takes in random numbers and generates an image
- The generated image is fed into discriminator along with images taken from the actual, ground truth dataset.
- The discriminator takes both images real and fake and returns probabilities, 0 or 1 with 1 representing true and 0 representing fake.
  Feedback Loop
- The discriminator is in the feedback loop with the ground truth of the images
- The generator is in the feedback loop with the discriminator.

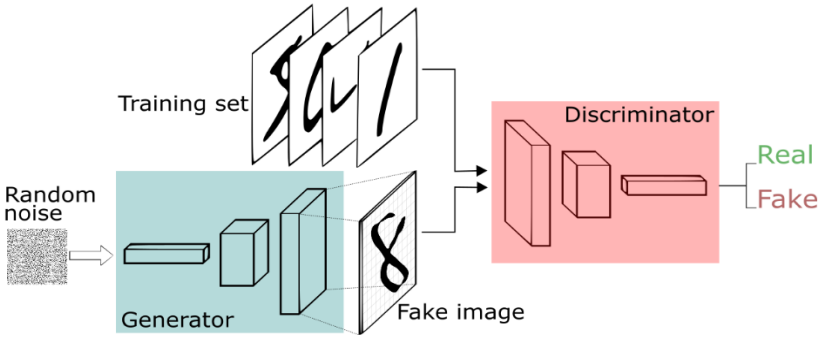


Figure: GAN Model with generator and Discriminator

3.2 DCGAN:

GANs are an architecture for training the generative models, such as the deep convolutional neural networks for generating images. Developing a GAN requires both a discriminator convolutional neural network model that classifies whether a given image is real or fake and a generator model that uses inverse convolutional layers to transform input into two dimensional image of pixel values.

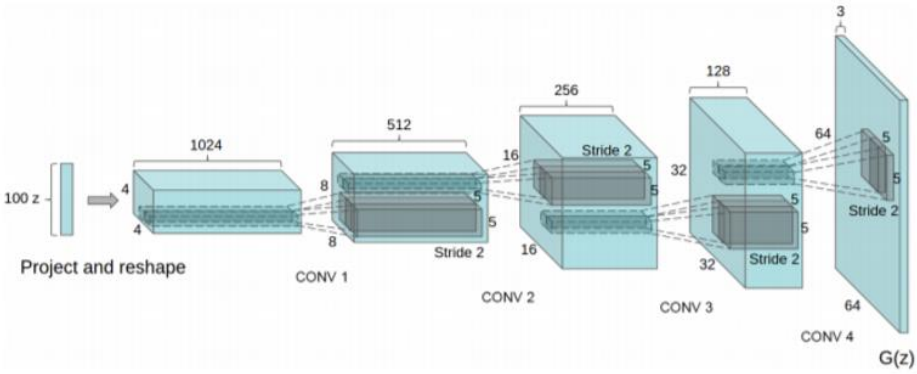


Figure: DCGAN with convolution layers

From the above figure, we see classical convolutional being applied that reshape the network with the (N+P-F)/S+1 equation taught with convolutional layers. From the diagram we see that N parameter,(Height/Weight) goes from 4 to 8 to 16 to 32 and there is no padding being applied, kernel filter parameter F is 5x5, and the stride is 2. The first layer expands random noise and is denoted as “Project and Reshape”.

Steps:

- We start with replacing all max pooling with convolutional stride.
- Use the transposed convolution for upsampling.
- Eliminate the fully connected layers.
- Use Batch Normalization for the generator and the input layer of the discriminator.
- Use ReLU activation in the generator however use the tanh for the output layer.
- Use the LeakyReLU activation for the discriminator.

3.3 SAGAN:

The convolutional filters in a GAN are good at exploring spatial locality information, the receptive fields may not be large enough to cover larger structures. With increase in filter size or the depth of the deep network makes GAN even harder to train. One of the limitations of DCGAN is that it relies heavily on the convolutions to model the dependencies across different image regions which inturn increases the computation cost. To overcome this issue Self Attention concept was introduced in DCGANS. Self Attention exhibits a better ability to model the long-range dependencies and also the computation and statistical efficiency.

SA exhibits a better balance between the ability to model long-range dependencies and the computational and statistical efficiency. The SA model calculates the response as a weighted sum of all the features at all positions, and this is computed with only small computational cost.

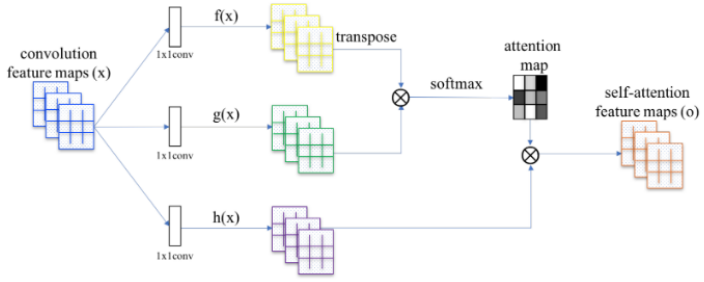


Figure: SAGAN module Architecture

3.4 DCGAN: Implementation

Loading Dataset:

The dataset used in the DCGAN implementation is CIFAR10. Keras provides access to importing and loading the dataset using the cifar10.load_dataset() function.

```
from keras.datasets.cifar10 import load_data
```

The function returns two tuples one with input and output elements for the training dataset and another with the input and output elements for the test dataset.

```
(trainX, trainy), (testX, testy) = load_data()
```

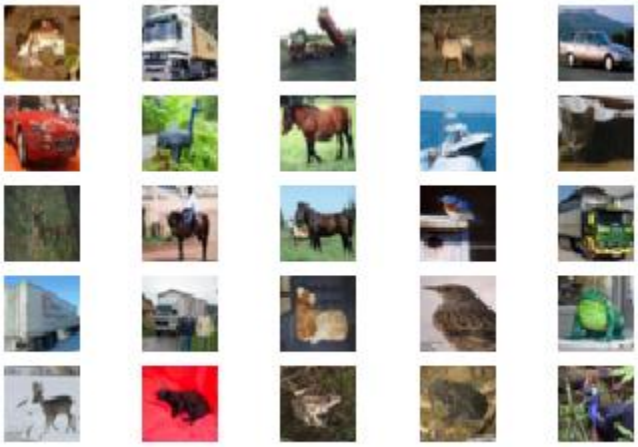
The below code snippet shows the size of the CIFAR10 dataset loaded.

```
print('Train', trainX.shape, trainy.shape)
print('Test', testX.shape, testy.shape)
```

```
Using TensorFlow backend.
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 11s 0us/step
Train (50000, 32, 32, 3) (50000, 1)
Test (10000, 32, 32, 3) (10000, 1)
```

Data Visualization:

We visualize the data by plotting a grid of 25 images from the CIFAR10 training dataset, arranged in a 5x5 square.



Discriminator Model:

We start with defining the Discriminator Model. The model should take a sample image as input from the dataset and output decision whether the sample is real or fake. This becomes a binary classification problem.

Input: 32x32x3 color image

Output: Binary classification, whether the sample is real or fake.

The discriminator model starts with a normal convolution layer and then three convolutional layers are added with 2x2 stride to downsample the input image. The model has a sigmoid activation function in the output layer to predict whether the image is real or fake and does not consist of any pooling. The loss function used is binary cross entropy loss function which is appropriate for binary classification. Below is the model summary for the discriminator model.

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_119 (Conv2D)          (None, 32, 32, 64)        1792
_________________________________________________________________
leaky_re_lu_37 (LeakyReLU)   (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_120 (Conv2D)          (None, 16, 16, 128)       73856
_________________________________________________________________
leaky_re_lu_38 (LeakyReLU)   (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_121 (Conv2D)          (None, 8, 8, 128)         147584
_________________________________________________________________
leaky_re_lu_39 (LeakyReLU)   (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_122 (Conv2D)          (None, 4, 4, 256)         295168
_________________________________________________________________
leaky_re_lu_40 (LeakyReLU)   (None, 4, 4, 256)         0
_________________________________________________________________
flatten_6 (Flatten)          (None, 4096)              0
_________________________________________________________________
dropout_6 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 4097
=================================================================
Total params: 522,497
Trainable params: 522,497
Non-trainable params: 0
_________________________________________________________________
```

Generate fake images:

We define the generate_fake_samples() function generates ramdom pixel values in the range [0,1], then scaled to the range [-1, 1] like our scaled real images and their associated class label of 0 for fake.

Train the Discriminator Model:

We train the model by repeatedly passing the samples of real images and samples of generated images for updating the model for fixed number of iterations. We don't pass the entire dataset to train the model, it learns to discriminate between real and fake images rapidly with smaller number of batches passed into the model. We use a batch size of 128 and iterate the model for 20 iterations. The accuracy of the model is calculated prior to updating the weights to understand how the model is performing.

```
>1 real=47% fake=0%
>2 real=95% fake=6%
>3 real=98% fake=22%
>4 real=92% fake=38%
>5 real=98% fake=69%
>6 real=97% fake=91%
>7 real=98% fake=100%
>8 real=98% fake=100%
>9 real=97% fake=100%
>10 real=98% fake=100%
>11 real=98% fake=100%
>12 real=92% fake=100%
>13 real=95% fake=100%
>14 real=97% fake=100%
>15 real=100% fake=100%
>16 real=100% fake=100%
>17 real=98% fake=100%
>18 real=98% fake=100%
>19 real=100% fake=100%
>20 real=100% fake=100%
```

Generator Model:

The generator model creates fake images of plausible small size photographs of the objects. It is done by taking point from the latent space as input and outputting a square color image. The generator model assigns meaning to the latent points thus the latent space and the compressed latent space represents a compressed representation of the output space.

Input: Points in Latent space
Output: Squared color images of 32x32 size with pixel values in [-1,-1]

This is achieved by having a dense layer as the first hidden layer that has enough nodes to represent a low resolution version of the output image. We don't just need one low resolution version of the image but many parallel interpretations of it. This is a pattern in CNN where we have many parallel filters resulting in multiple parallel activation maps called the feature maps with different interpretations of the input.

The next step involves upsampling the low-resolution image to a higher resolution version of the image. The Conv2DTranspose layer is used that will quadraple the area of the input feature maps. This is repeated two more times to arrive at 32x32 output image.

The define_generator() function implements this and defines the generator model.

Now we can update the generate_fake_samples() to take generator model as argument and use it to generate desired number of samples by calling the generate_latent_points() function to generate the number of points in latent space as input to the model.

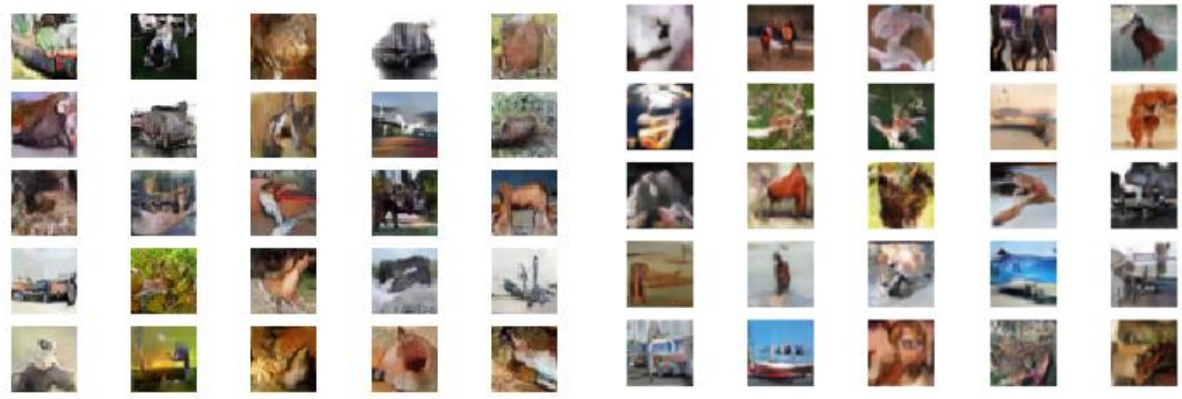
Train the Generator Model:

The generator model weights are updated based on the performance of the discriminator model.
Training the DCGAN:
We combine the train_discriminator() and the train_gan() functions which updates the discriminator and the generator model. We define the number of epochs and also number of batches within each epoch.

Result:

Below are the grid of images generated on Train set after 10, 50 and 100 epochs.

Epoch 10 **Epoch 50**



Epoch 100 **Epoch 130**

3.5 SAGAN Implementation:

GAN models for image generation are built using the convolutional layers that processes the information in local neighbourhood thus using the conv layers alone for modeling the long-range dependencies in images. Hence we introduce the self-attention to GAN framework enabling the discriminator and the generator model the relationship between separated spatial regions. We call the method as Self Attention Generative Adversarial Networks because of itself attention module.

The features from the previous hidden layer x ∈ R C×N are transformed into two feature maps f and g to calculate the attention where f(x) = Wfx, g(x) = Wgx.

$$\beta_{j,i} = \frac{\exp(s_{ij})}{\sum_{i=1}^{N} \exp(s_{ij})}, \text{ where } s_{ij} = \boldsymbol{f}(\boldsymbol{x_i})^T \boldsymbol{g}(\boldsymbol{x_j}),$$

Here, βj,I indicates the the extent to which the model attends to the ith location when synthesizing the j th region, C indicates the number of channels and N is the number of feature locations of the features from the previous layer. The output of the attention layer is o = (o1, o2, ..., oj , ..., oN) ∈ R C×N , where

$$\boldsymbol{o_j} = \boldsymbol{v}\left(\sum_{i=1}^{N} \beta_{j,i} \boldsymbol{h}(\boldsymbol{x_i})\right), \ \boldsymbol{h}(\boldsymbol{x_i}) = \boldsymbol{W_h}\boldsymbol{x_i}, \ \boldsymbol{v}(\boldsymbol{x_i}) = \boldsymbol{W_v}\boldsymbol{x_i}.$$

In the above formulation, $\boldsymbol{W_g} \in \mathbb{R}^{\bar{C}\times C}$, $\boldsymbol{W_f} \in \mathbb{R}^{\bar{C}\times C}$, $\boldsymbol{W_h} \in \mathbb{R}^{\bar{C}\times C}$, and $\boldsymbol{W_v} \in \mathbb{R}^{C\times\bar{C}}$ are the learned weight matrices, which are implemented as 1×1 convolutions. Since We did not notice any significant performance decrease when reducing the channel number of $\bar{C}$ to be C/k, where $k = 1, 2, 4, 8$ after few training epochs on ImageNet. For memory efficiency, we choose $k = 8$ (*i.e.*, $\bar{C} = C/8$) in all our experiments.

We further multiply the output of the attention layer by scale parameter and add the input feature map. The final output is given by

$$\boldsymbol{y_i} = \gamma \boldsymbol{o_i} + \boldsymbol{x_i},$$

where γ is a learnable scalar and it is initialized as 0.

Learnable scalar allows the network to rely on cues of the local neighbourhood and then gradually learn to assign more weight to non local evidence. We do this to learn the easy task first and progressively increase the complexity of the task. IN SAGAN, the attention module is applied both to the generator and the discriminator that is trained in altering fashion by minimizing the hinge version of the adversarial loss.

$$
\begin{aligned}
L_D = & -\mathbb{E}_{(x,y)\sim p_{data}}[\min(0, -1 + D(x,y))] \\
& -\mathbb{E}_{z\sim p_z, y\sim p_{data}}[\min(0, -1 - D(G(z),y))], \\
L_G = & -\mathbb{E}_{z\sim p_z, y\sim p_{data}} D(G(z),y),
\end{aligned}
$$

Techniques to stabilize the training of GAN

We have two techniques to stabilize the training of the GANs. We first use Spectral normalization in the generator as well as the discriminator. Secondly, we confirm the two timescale update rule (TTUR) is effective, and we specifically use it to address the slow learning in regularized discriminators.

Spectral Normalization:

GAN faces a challenge of inaccurately estimating the density ratio of the discriminator when exposed to high dimensional spaces and thus results in poor training of the generator that results in termination of training as a derivative of the produced discriminator with the input as zero. To stabilize such training spectral normalization is introduced. The spectral normalization does not require extra hyper parameters tuning as it sets spectral norm of all weight layers to 1 which performs well. Also, the computational cost is also low.

In Spectral Normalization, the Lipschitz constant is the only hyper-parameter that is to be tuned, and the algorithm does not require intensive tuning of the only hyper-parameter for satisfactory performance. By applying Spectral normalization to the GANs, the generated examples are more diverse than the conventional weight normalization and hence achieve better or comparative inception scores.

3.6 Difference between DCGAN and SAGAN

The traditional convolution GAN produce high resolution details as a function of only spatially local points in lower dimension feature maps where as in SAGAN the details are generated using the cues from all the feature locations. Also the discriminator in SAGAN can check highliy detailed features in distant portions of the image are consistent with each other. Also, we observe that the FID scores for SAGAN is better when compared to DCGAN. The convolutional GANs have difficulty in modelling soe image classes than others when trained on multiclass datasets.

In order to process long range dependencies in DCGAN can be attained by passing it through several convolution layers. Increasing the size of the convolution kernel increases the representational capacity of the network but this reduces the computation and statistical efficiency obtained by using local convolutional structure. The Self Attention has better balance between long term dependencies and the computation and statistical efficiencies and also has less computational cost.

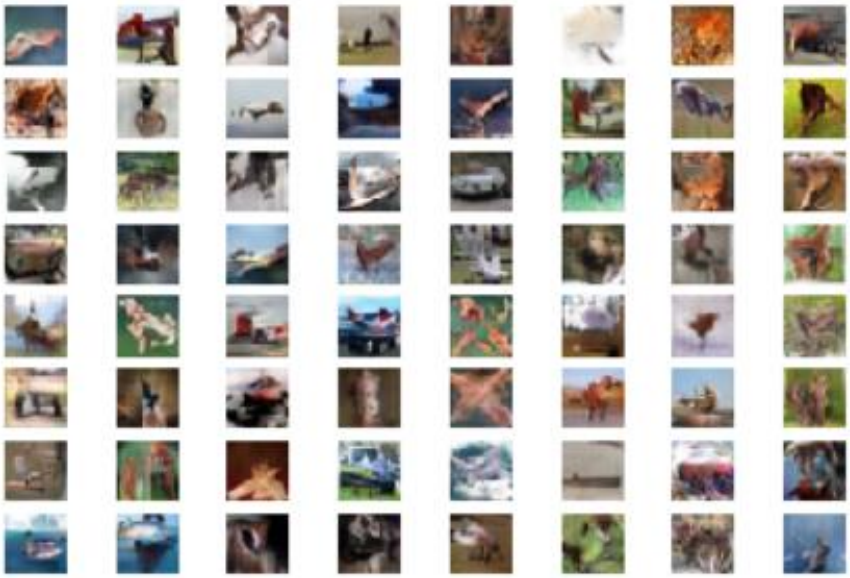
Importance of Attention:

Any neural network with attention mechanism can actually understand what ‘it’ is referring to. It knows how to disregard the noise and focus on what is relevant and how to connect relations between two features which do not canny any markers among them pointing to one another. Hence attention allows us to look at the totality of the input and extract the most relevant features that might have connections to other features in the image.

Example: In a sentence, attention allows us to travel through the wormhole of syntax to identify relationships with other words that are far away ignoring other words that don’t have much bearing on whatever word we are trying to make a prediction on.

4. RESULTS:

8x8 grid images for DCGAN after 130 epochs



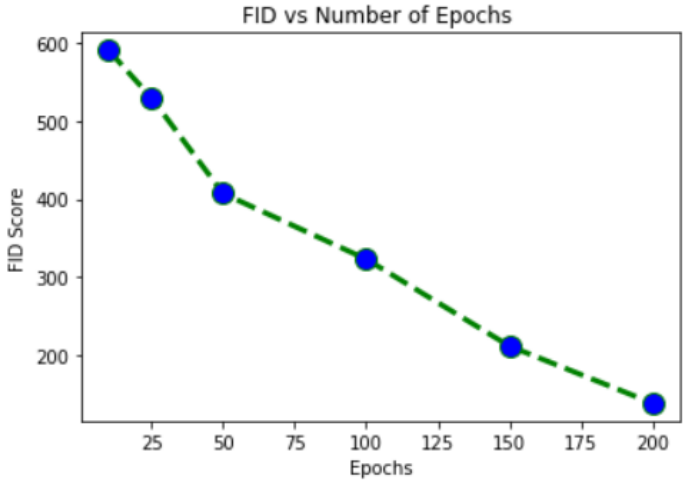
FID Score run on the Test set after 10 epochs

```
>Accuracy real: 61%, fake: 89%
Loaded (2000, 32, 32, 3) (2000, 32, 32, 3)
Scaled (2000, 299, 299, 3) (2000, 299, 299, 3)
FID: 576.8080890859787
```

No of Epochs	10	25	50	100	150	200
DCGAN	576	529	407	323	212	139
SAGAN						

The SAGAN couldn't produce proper results hence FID scores couldn't be produced.

Graph for FID versus Epoch for DCGAN



The losses of the discriminator and the generator for the Training Set.

```
>130, 369/390, d1=0.571, d2=0.599 g=1.186
>130, 370/390, d1=0.650, d2=0.521 g=1.131
>130, 371/390, d1=0.477, d2=0.476 g=1.179
>130, 372/390, d1=0.670, d2=0.569 g=1.108
>130, 373/390, d1=0.601, d2=0.573 g=1.094
>130, 374/390, d1=0.616, d2=0.536 g=1.115
>130, 375/390, d1=0.649, d2=0.590 g=1.062
>130, 376/390, d1=0.620, d2=0.524 g=1.119
>130, 377/390, d1=0.532, d2=0.764 g=1.005
>130, 378/390, d1=0.478, d2=0.696 g=0.994
>130, 379/390, d1=0.493, d2=0.688 g=1.151
>130, 380/390, d1=0.608, d2=0.487 g=1.151
>130, 381/390, d1=0.651, d2=0.512 g=1.118
>130, 382/390, d1=0.676, d2=0.592 g=1.174
>130, 383/390, d1=0.658, d2=0.594 g=1.125
>130, 384/390, d1=0.555, d2=0.605 g=1.123
>130, 385/390, d1=0.666, d2=0.447 g=1.172
>130, 386/390, d1=0.770, d2=0.572 g=1.166
>130, 387/390, d1=0.555, d2=0.544 g=1.170
>130, 388/390, d1=0.546, d2=0.571 g=1.142
>130, 389/390, d1=0.579, d2=0.602 g=1.103
>130, 390/390, d1=0.575, d2=0.584 g=1.105
```

Graph for Iterations vs Loss for the 13oth epoch

References:

https://arxiv.org/pdf/1511.06434.pdf
https://arxiv.org/pdf/1805.08318.pdfs
https://skymind.ai/wiki/generative-adversarial-network-gan
https://medium.com/@jonathan_hui/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c438f
https://medium.com/@jonathan_hui/gan-self-attention-generative-adversarial-networks-sagan-923fccde790c

https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-a-cifar-10-small-object-photographs-from-scratch/

https://github.com/taki0112/Self-Attention-GAN-Tensorflow

https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/