DEEP FAKES-CAR GENERATION

BIG DATA SYSTEMS & INTELLIGENCE ANALYTICS

CSYE (7245) SEC 01

By: Nikhita Methwani [001492785]

ABSTRACT

The aim of the project was to understand and study the approach taken by the research papers on Progressive GANS and Style based GANS [1][2] which were successful in generating high resolution celeb fake images. We extend the ***understanding of Progressive GANS by studying the techniques, functioning and experimenting with the structure of the network and apply the learning on CAR dataset to produce fake care images. Dataset of 5000 real car images having different resolutions were preprocessed to an image size of 128*128 pixels which maintained the consistency of the training images. The project focused on reimplementing the Progressive GANS architecture by following the steps mentioned in the research paper [1]. Due to high training requirements of GANS, the project made use of Google cloud platform (GCP) to train the GANS, which resulted in the convergence of fake car images after 12 hours of continuous training. The results generated are low resolution car images. High resolution and real looking car images can be achieved by training the network for more hours and tuning the hyperparameters of the neural network along with considering the styling effects discussed in research paper [2] which is the future scope of the project.***

Keywords: GANS, Progressive GANS, Image Generation

INTRODUCTION

GANS belong to the field of unsupervised learning which tries to learn the given image without having specified labeled data. GANS compose of two neural networks – generator and discriminator trying to compete each other and learn the conditional probability distribution of the given data with respect to the generated data to produce fake images. GANs produce sharp images but only in small resolutions with somewhat limited variation, and the training continues to be unstable. The distribution of the generated images should follow the distribution of the input images, which is measured by various metrics such as Jensen-Shannon divergence, least squares and Wasserstein distance.

The research paper [1] introduces the problems that are faced in training the GANS with high resolution images and the improvised steps of training the network progressively starting from low resolution images and adding the layers in the network to improve and enhance the resolution of images progressively. The GAN formulation does not explicitly require the entire training data distribution to be represented by the resulting generative model. There has been a tradeoff between the quality of the generated image with the variation in the image which is evaluated by the Inspection score and multi-scale structural (MS-SSIM) described in paper [1].

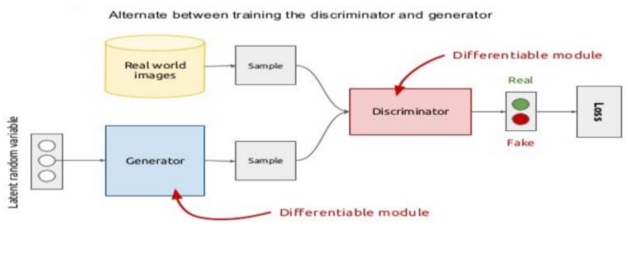
BACKGROUND

GANS:

Generative Adversarial Networks are composed of two models:

The first model is called a **Generator** and it aims to generate new data like the expected one. The generator is fed with latent vector of gaussian distribution we want the generator to samples from this probability distribution and produce images and fool the discriminator.

The second model is named the **Discriminator**. The discriminator gets the input images from the generator and the real images and tries to discriminate the difference between the probability distribution of real and fake images. The discriminator is a binary classification neural network which gives the probability of the difference. A 1 indicates that the discriminator was able to distinguish between real and fake images and less probability at the output means that the discriminator is not able to distinguish between the two inputs and the generator is learning good enough to fool the discriminator.



The **discriminator** calculates the loss for both **fake x** and **real x** and combine them as the final loss as **D loss**. The **generator** also calculates the loss from its noise as **G loss** since each network has a different objective function**.** The two losses go back to their respective networks to learn from the loss (adjusting the parameters w r t the loss). An optimization algorithm is used to minimize loss function for each network.

GANS works on the principle of min-max principle which is illustrated in the below equation:

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

$$\max_D V(D) = \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]}_{\text{recognize real images better}} + \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))]}_{\text{recognize generated images better}}$$

$$\min_G V(G) = \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))]}_{}$$

Optimize G that can fool the discriminator the most.

The discriminator tries to maximize the probability by detecting the fake images where the generator tries to minimize the output generated by the discriminator for the fake images

z → Noise vector G(z) → Generator's output → x_{fake}

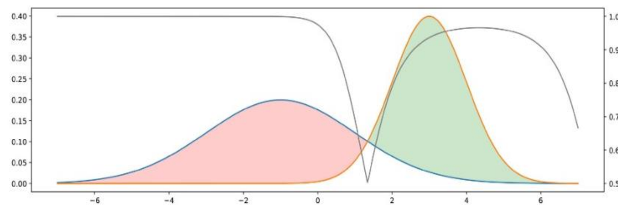
x → training sample → x_{real}

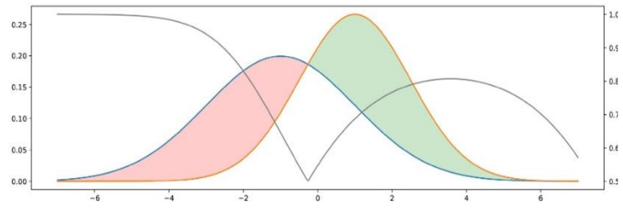
D (x) → Discriminator's output for x_{real} → P(y | x_{real}) → {0,1}

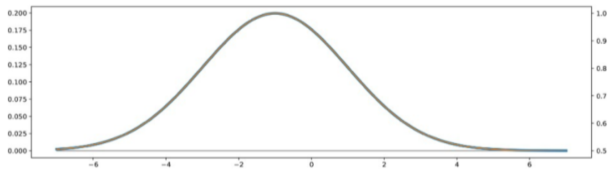
D (G(z)) → Discriminator's output for x_{fake} → P(y | x_{fake}) → {0,1}

at Discriminator D	at Generator G
D (x) → should be maximized D (G(z)) → should be minimized	D (G(z)) → should be maximized

Below is the illustration for GANS.





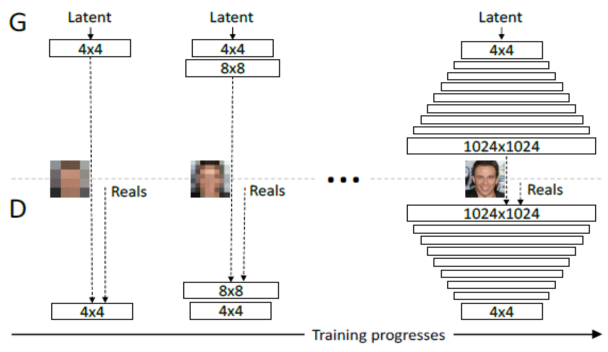


The blue distribution is of the true images, the orange is the generated one. The grey, with corresponding y-axis on the right, indicates the probability to be true for the discriminator if it chooses the class with the higher density in each point (assuming "true" and "generated" data are in equal proportions). The closer the two distributions are, the more often the discriminator is wrong. When training, the goal is to "move the green area" (generated distribution is too high) towards the red area (generated distribution is too low) [3]

More details about GANS: https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

Progressive GANS:

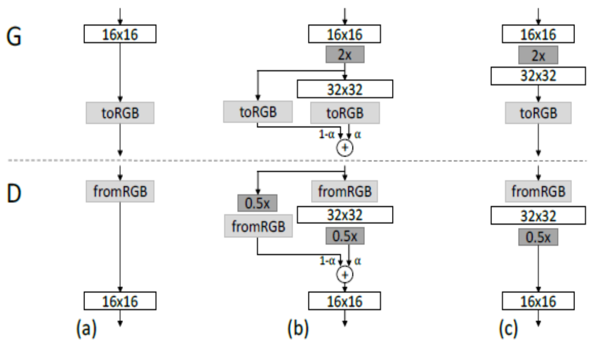
The paper illustrates the problem faced if the network is trained with the high-resolution images directly, which results in discriminator to easily classify between real and fake images and makes the training complex. In order to overcome this, the training of the network starts with low resolution generated images while smoothly adding the layers in the network to improve the resolution of the generated images.



The training starts with 4*4 resolution and comparable result quality is often obtained up to 2–6 times faster, depending on the final output resolution.

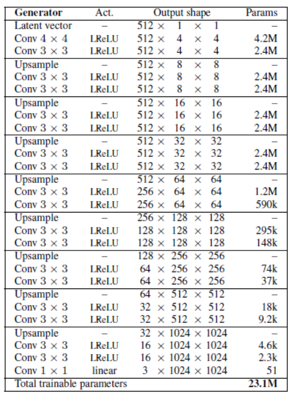
The paper also illustrates the problems in training GANS, as the generator has the tendency to capture a single spot or a single variation in training data which results in the issue of mode collapse. Here the generator starts to produce the same looking images and if the discriminator is not well trained it doesn't classify the images correctly and the training of the network stops with no proper images at the output.

To overcome this and increasing the variation of the generated images, minibatch discrimination is used. They compute feature statistics not only from individual images but also across the minibatch, thus encouraging the minibatches of generated and training images to show similar statistics. This is implemented by adding a minibatch layer towards the end of the discriminator, where the layer learns a large tensor that projects the input activation to an array of statistics. A separate set of statistics is produced for each example in a minibatch and it is concatenated to the layer's output, so that the discriminator can use the statistics internally.

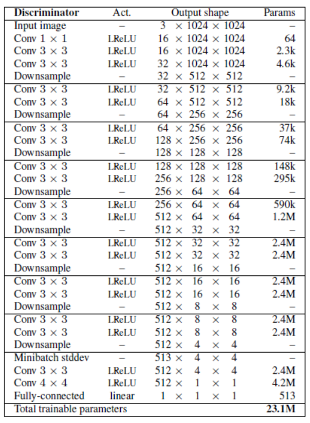


To make a smooth transition to a higher resolution from 16*16 to 32*32, residual blocks (toRGB & fromRGB) are used which allows the effect of 16*16 along with 32*32 pixels and reduces the shock of high fluctuation in pixel size to Neural network. The Residual block also increases the potential of reducing training error as compared to plain Networks, since it allows the weights(alpha) to get updated linearly and reduces the problem of vanishing gradient.

Below are the generator and discriminator neural network used for generating Fake cars.



Generator	Act.	Output shape	Params
Latent vector	–	512 × 1 × 1	–
Conv 4 × 4	LReLU	512 × 4 × 4	4.2M
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Upsample	–	512 × 8 × 8	–
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Upsample	–	512 × 16 × 16	–
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Upsample	–	512 × 32 × 32	–
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Upsample	–	512 × 64 × 64	–
Conv 3 × 3	LReLU	256 × 64 × 64	1.2M
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Upsample	–	256 × 128 × 128	–
Conv 3 × 3	LReLU	128 × 128 × 128	295k
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Upsample	–	128 × 256 × 256	–
Conv 3 × 3	LReLU	64 × 256 × 256	74k
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Upsample	–	64 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	18k
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Upsample	–	32 × 1024 × 1024	–
Conv 3 × 3	LReLU	16 × 1024 × 1024	4.6k
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 1 × 1	linear	3 × 1024 × 1024	51
Total trainable parameters			**23.1M**



Discriminator	Act.	Output shape	Params
Input image	–	3 × 1024 × 1024	–
Conv 1 × 1	LReLU	16 × 1024 × 1024	64
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 3 × 3	LReLU	32 × 1024 × 1024	4.6k
Downsample	–	32 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Conv 3 × 3	LReLU	64 × 512 × 512	18k
Downsample	–	64 × 256 × 256	–
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Conv 3 × 3	LReLU	128 × 256 × 256	74k
Downsample	–	128 × 128 × 128	–
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Conv 3 × 3	LReLU	256 × 128 × 128	295k
Downsample	–	256 × 64 × 64	–
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Conv 3 × 3	LReLU	512 × 64 × 64	1.2M
Downsample	–	512 × 32 × 32	–
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Downsample	–	512 × 16 × 16	–
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Downsample	–	512 × 8 × 8	–
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Downsample	–	512 × 4 × 4	–
Minibatch stddev	–	513 × 4 × 4	–
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Conv 4 × 4	LReLU	512 × 1 × 1	4.2M
Fully-connected	linear	1 × 1 × 1	513
Total trainable parameters			**23.1M**

In Generator: 1*1 convolution layer is used as a fully connected layer, where there are 3 filters of 1*1*16 used for image size of 1024* 1024*16.Since the previous layer channel size is 16, the 1*1 convolution has 16 channels, it does the element wise matrix multiplication of all the 16 channels of the previous layer with the 16 channels(weights) of the filter and applying linear Activation for 1024*1024 image. The layer becomes fully connected because all the 1024*1024*16 gets connected to 3 neurons(filters) of 1*1.

In Discriminator: The 1*1 convolution layer is used as a fromRGB layer and is used to create more channels (16) of 1024*1024 images, which increases the training of neural network.

The latent vectors correspond to random points on a 512-dimensional hypersphere and represent training and generated images in [-1,1]. Leaky ReLU is used with leakiness 0.2 in all layers of both networks, except for the last layer that uses linear activation. Pixelwise normalization is performed for the feature vectors after each Conv 3_3 layer in the generator.

All bias is initialized to zero and all weights according to the normal distribution with unit variance. Minibatch standard deviation as an additional feature map at 4 _ 4 resolution toward the end of the discriminator to avoid mode collapse problems. The up sampling is done with nearest neighbors and down sampling operations is done by average pooling

More illustration on the working of progressive GANS can be found in Progressive GANS research paper [1]

METHOD/APPROACH:

Understanding the code of progressive GANS which generated CELEB fake images and replicating the same on CAR dataset.

Preparing the dataset to run and train the model.

Evaluating the results

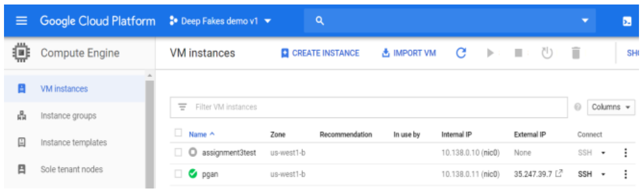
GANS take a large amount of time to converge the data into fake images, thus Google cloud platform was used to train the network.

Specifications of GPU:

1 x NVIDIA Tesla T4

n1-standard-4 (4 vCPUs, 15 GB memory)

framework - TensorFlow:1.13

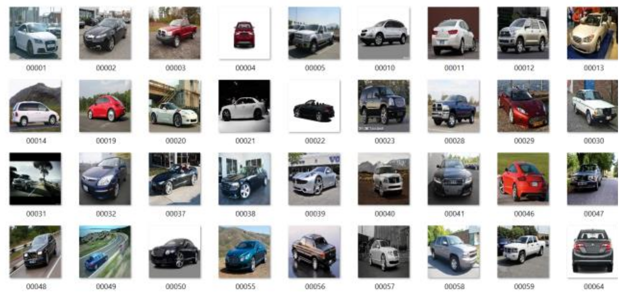


Name	Zone	Recommendation	In use by	Internal IP	External IP	Connect
assignment3test	us-west1-b			10.138.0.10 (nic0)	None	SSH
pgan	us-west1-b			10.138.0.11 (nic0)	35.247.39.7	SSH

Data Set Creation:

The dataset was taken form stanford.edu and had various car images with different sizes. The approach was to pre-process the images and resize it to 128*128 pixels to maintain the training data consistency.

Resized Images:

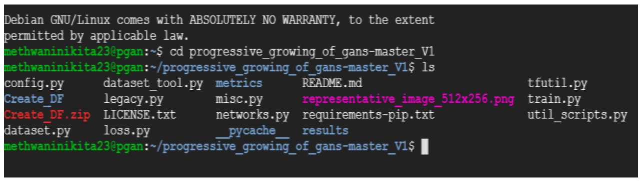


Tf.records file was generated by running

Python dataset_tool.py create_from_images datasets/car_results ~/downloads/cardataset which generated images varying from low to high resolution. This dataset was uploaded on the GCP to train the GANS.

Training the network:

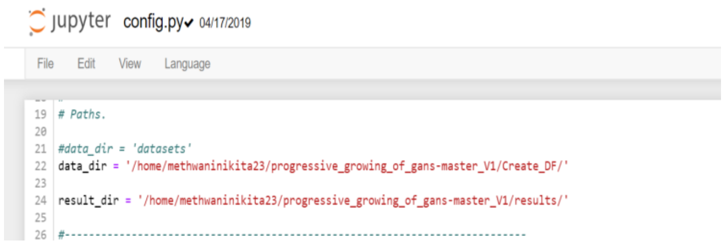
The code of progressive GANS[4](progreesive_growing_of_gans-master_V1) along with the tf.records (Create_DF) dataset was uploaded on GCP.



```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
methwaninikita23@pgan:~$ cd progressive_growing_of_gans-master_V1
methwaninikita23@pgan:~/progressive_growing_of_gans-master_V1$ ls
config.py      dataset_tool.py  metrics        README.md                      tfutil.py
Create_DF      legacy.py        misc.py        representative_image_512x256.png  train.py
Create_DF.zip  LICENSE.txt      networks.py    requirements-pip.txt           util_scripts.py
dataset.py     loss.py          __pycache__    results
methwaninikita23@pgan:~/progressive_growing_of_gans-master_V1$ 
```

The image is from GCP shell

The file config.py was modified to have car dataset and results directory. Modification of setting the GPUs used to train the model was also done. The image size of training was set to 128*128 pixels.



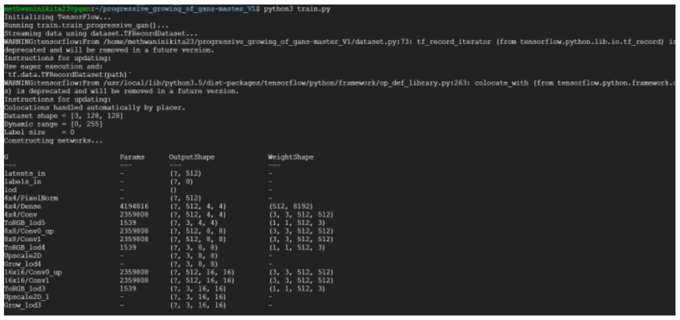
```
# Paths.

#data_dir = 'datasets'
data_dir = '/home/methwaninikita23/progressive_growing_of_gans-master_V1/Create_DF/'

result_dir = '/home/methwaninikita23/progressive_growing_of_gans-master_V1/results/'

#----------------------------------------------------------------------------
```

Train.py file was run to start the training of the GANS:



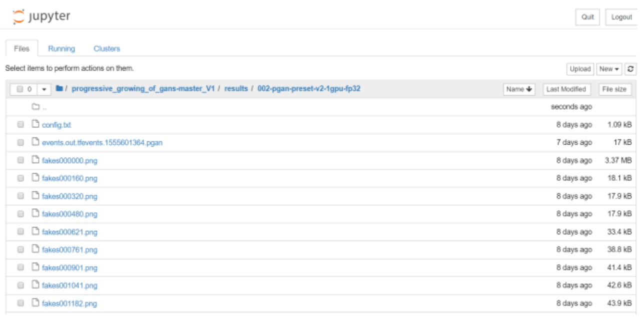
```
methwaninikita23@pgan:~/progressive_growing_of_gans-master_V1$ python3 train.py
Initializing TensorFlow...
Running train.train_progressive_gan()...
Streaming data using dataset.TFRecordDataset...
WARNING:tensorflow:From /home/methwaninikita23/progressive_growing_of_gans-master_V1/dataset.py:73: tf_record_iterator (from tensorflow.python.lib.io.tf_record) is
deprecated and will be removed in a future version.
Instructions for updating:
Use eager execution and:
`tf.data.TFRecordDataset(path)`
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.framework.o
s) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
Dataset shape = [3, 128, 128]
Dynamic range = [0, 255]
Label size    = 0
Constructing networks...

G                          Params      OutputShape            WeightShape
---                        ---         ---                    ---
latents_in                 -           (?, 512)               -
labels_in                  -           (?, 0)                 -
lod                        -           ()                     -
4x4/PixelNorm              -           (?, 512)               -
4x4/Dense                  4194816     (?, 512, 4, 4)         (512, 8192)
4x4/Conv                   2359808     (?, 512, 4, 4)         (3, 3, 512, 512)
ToRGB_lod5                 1539        (?, 3, 4, 4)           (1, 1, 512, 3)
8x8/Conv0_up               2359808     (?, 512, 8, 8)         (3, 3, 512, 512)
8x8/Conv1                  2359808     (?, 512, 8, 8)         (3, 3, 512, 512)
ToRGB_lod4                 1539        (?, 3, 8, 8)           (1, 1, 512, 3)
Upscale2D                  -           (?, 3, 8, 8)           -
Grow_lod4                  -           (?, 3, 8, 8)           -
16x16/Conv0_up             2359808     (?, 512, 16, 16)       (3, 3, 512, 512)
16x16/Conv1                2359808     (?, 512, 16, 16)       (3, 3, 512, 512)
ToRGB_lod3                 1539        (?, 3, 16, 16)         (1, 1, 512, 3)
Upscale2D_1                -           (?, 3, 16, 16)         -
Grow_lod3                  -           (?, 3, 16, 16)         -
```

The network formed a real image from a dataset by merging multiple cars and tried to create the fake cars.

Real image

The converged fake car images were downloaded in the mentioned directory



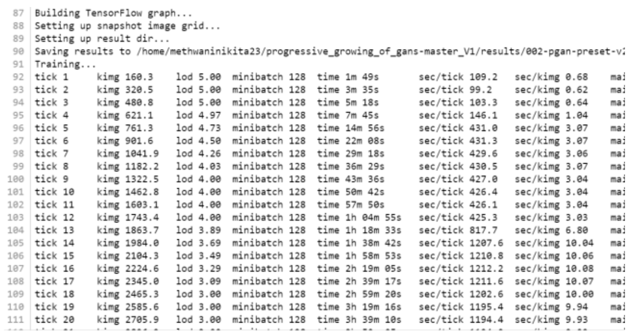
Select items to perform actions on them.

/ progressive_growing_of_gans-master_V1 / results / 002-pgan-preset-v2-1gpu-fp32

Name	Last Modified	File size
..	seconds ago	
config.txt	8 days ago	1.09 kB
events.out.tfevents.1555601364.pgan	7 days ago	17 kB
fakes000000.png	8 days ago	3.37 MB
fakes000160.png	8 days ago	18.1 kB
fakes000320.png	8 days ago	17.9 kB
fakes000480.png	8 days ago	17.9 kB
fakes000621.png	8 days ago	33.4 kB
fakes000761.png	8 days ago	38.8 kB
fakes000901.png	8 days ago	41.4 kB
fakes001041.png	8 days ago	42.6 kB
fakes001182.png	8 days ago	43.9 kB

RESULTS

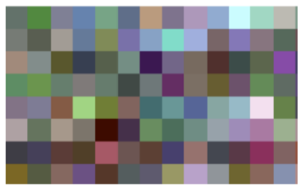
Log file generated at the end of the training network illustrates the minutes it took for each image to be generated



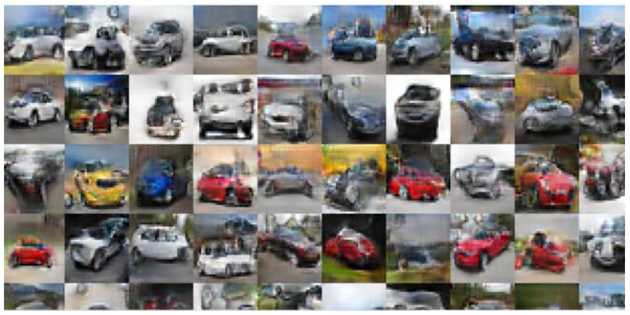
```
Building TensorFlow graph...
Setting up snapshot image grid...
Setting up result dir...
Saving results to /home/methwaninikita23/progressive_growing_of_gans-master_V1/results/002-pgan-preset-v
Training...
tick 1     kimg 160.3    lod 5.00  minibatch 128  time 1m 49s        sec/tick 109.2   sec/kimg 0.68    mai
tick 2     kimg 320.5    lod 5.00  minibatch 128  time 3m 35s        sec/tick 99.2    sec/kimg 0.62    mai
tick 3     kimg 480.8    lod 5.00  minibatch 128  time 5m 18s        sec/tick 103.3   sec/kimg 0.64    mai
tick 4     kimg 621.1    lod 4.97  minibatch 128  time 7m 45s        sec/tick 146.1   sec/kimg 1.04    mai
tick 5     kimg 761.3    lod 4.73  minibatch 128  time 14m 56s       sec/tick 431.0   sec/kimg 3.07    mai
tick 6     kimg 901.6    lod 4.50  minibatch 128  time 22m 08s       sec/tick 431.3   sec/kimg 3.07    mai
tick 7     kimg 1041.9   lod 4.26  minibatch 128  time 29m 18s       sec/tick 429.6   sec/kimg 3.06    mai
tick 8     kimg 1182.2   lod 4.03  minibatch 128  time 36m 29s       sec/tick 430.5   sec/kimg 3.07    mai
tick 9     kimg 1322.5   lod 4.00  minibatch 128  time 43m 36s       sec/tick 427.0   sec/kimg 3.04    mai
tick 10    kimg 1462.8   lod 4.00  minibatch 128  time 50m 42s       sec/tick 426.4   sec/kimg 3.04    mai
tick 11    kimg 1603.1   lod 4.00  minibatch 128  time 57m 50s       sec/tick 426.1   sec/kimg 3.04    mai
tick 12    kimg 1743.4   lod 4.00  minibatch 128  time 1h 04m 55s    sec/tick 425.3   sec/kimg 3.03    mai
tick 13    kimg 1863.7   lod 3.89  minibatch 128  time 1h 18m 33s    sec/tick 817.7   sec/kimg 6.80    mai
tick 14    kimg 1984.0   lod 3.69  minibatch 128  time 1h 38m 42s    sec/tick 1207.6  sec/kimg 10.04   mai
tick 15    kimg 2104.3   lod 3.49  minibatch 128  time 1h 58m 53s    sec/tick 1210.8  sec/kimg 10.06   mai
tick 16    kimg 2224.6   lod 3.29  minibatch 128  time 2h 19m 05s    sec/tick 1212.2  sec/kimg 10.08   mai
tick 17    kimg 2345.0   lod 3.09  minibatch 128  time 2h 39m 17s    sec/tick 1211.6  sec/kimg 10.07   mai
tick 18    kimg 2465.3   lod 3.00  minibatch 128  time 2h 59m 20s    sec/tick 1202.6  sec/kimg 10.00   mai
tick 19    kimg 2585.6   lod 3.00  minibatch 128  time 3h 19m 16s    sec/tick 1195.4  sec/kimg 9.94    mai
tick 20    kimg 2705.9   lod 3.00  minibatch 128  time 3h 39m 10s    sec/tick 1194.4  sec/kimg 9.93    mai
```

The fake car images were generated after continuous training for 12 hours. The below shows the convergence of noisy images in the initial stages to the final images

Generation of Images in the Initial Stages:



Final Images of Fake cars



DISCUSSION/CONCLUSION

Though the results generated are of very low resolution, this project implemented the understanding of Progressive GANS using custom dataset. Analyzing the Network functioning, structure and layers of neural network and implementation of GANS on GCP helped to produce fake like images. The low resolution of the imaged could be due to the less training time for converging. The training time can be increased further along with the use of Style based generator network which uses various factors to introduce styles in the generated image.

REFERENCES

[1] Progressive Growing of GANs for Improved Quality, Stability, and Variation -- Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen

(Submitted on 27 Oct 2017 (v1), last revised 26 Feb 2018 (this version, v3))

[2] A Style-Based Generator Architecture for Generative Adversarial Networks

Tero Karras, Samuli Laine, Timo Aila

(Submitted on 12 Dec 2018 (v1), last revised 29 Mar 2019 (this version, v3))

[3] Mathematical working of GANS: https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

[4]Progressive GANS implementation of CELEB dataset:https://github.com/tkarras/progressive_growing_of_gans

[5]GANS: https://hackernoon.com/how-do-gans-intuitively-work-2dda07f247a1

[6] https://medium.com/deep-math-machine-learning-ai/ch-14-general-adversarial-networks-gans-with-math-1318faf46b43

[7]https://www.youtube.com/watch?v=5WoItGTWV54

[8]https://www.floydhub.com/diegoalejogm/projects/gans/10/output/1.%20Vanilla%20GAN%20TensorFlow.ipynb

[9] Running Jupyter Notebook on Google Cloud Platform (Medium) (Anaconda) https://towardsdatascience.com/running-jupyter-notebook-in-google-cloud-platform-in-15-min-61e16da34d52

SCOPE

The scope of the project was to read and understand various research papers having GANS and progressive GANS implemented. The aim was to reimplement the actual existing code on the car dataset to produce fake looking images.

CONTEXT

The code was referred from the research paper on Progressive GANS [1] which implemented the GANS for generating fake CELEB images. network structure of the code was understood and the config.py file was changed as per the car dataset such as the input image size to 128*128 and the number of GPU on which the dataset will be trained on.