Deep Convolutional Generative Adversarial Networks (DCGAN)

Report (710.085 Selected Topics for Computer Vision)

Peter LORENZ

Inst. of Computer Graphics and Vision
Graz University of Technology, Austria

Seminar/Project Computer Vision
Markus Oberweger
Graz, January 17, 2018

contact: Peter LORENZ peter.lorenz@student.tugraz.at

Abstract

This short report is based on the paper [6] of Radford published in 2016 and on a tensorflow implementation (ver1.3). Most time of this work is spend on training the network by different configurations and evaluating the result by finding the nearest neighbor in the training set.

Keywords: *Generative Adversarial Network, Tensorflow, Tensorboard, Deep Convolution, Image Synthesis, CelebA*

Acknowledgement

Big thanks to the admins. They provided me a good computer equipment for training all those variants of the deep convolution generative adversarial network.

1 Background

Supervised learning was paid more attention in past than unsupervised learning. Ian Goodfellow introduced an unsupervised approach, Generative Adversarial Nets [2], in year 2014. Basically, 2 networks are involved. The generator network produces from random values an image. The Discriminator is trained by real images and decides if the generated images from the network are real images, so that the generator retrieves feedback. This kind of learning counts to reinforcement learning (trial-and-error). This approach is refined by [6] and claim a rather stable unsupervised learning output. These modifications are summed up in the paper [6] as follows:

1. Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator). [7]
2. Use batch norm in both the generator and the discriminator. [8]
3. Remove fully connected hidden layers for deeper architectures. [4]
4. Use ReLU activation in generator for all layers except for the output, which uses the activation function Tanh. [5]
5. Use LeakyReLU activation in the discriminator for all layers. [3]

2 Is code available?

I would like to prefer to use this tensorflow version of DCGAN, which works on my personal computer out of the box: github.com/carpedm20/DCGAN-tensorflow They author of the repository stated that: "To avoid the fast convergence of D (discriminator) network, G (generator) network is updated twice for each D network update, which differs from original paper."
Some people have already published on GitHub frameworks, which includes different GAN implementations:
`https://github.com/sanghoon/tf-exercise-gan`
`https://github.com/khanrc/tf.gans-comparison`

Additional material are found:
Slides and Video
github.com/goodfeli/adversarial

3 What I have implemented

This work stays focused on the paper Unsupervised Representation Learning With Deep Convolution Generative Adversarial Networks. My personal computer for development purposes is set up of following components:

- **Intel(R) Core(TM) i7** CPU 960 @ 3.20GHz
- **GeForce GTX 980** 4GB (for calculation)
- **GeForce GT 730** 2GB (for monitoring)

30 epochs (standard configured) lasts approximately 1 day on my set up. After 6 epochs rather good results can be shown and after 10 epochs hardly changes can be distinguished. A drawback of GANs is that no inferences can be used, that means, only training examples can be used as output. Therefore, inferences can not be used, but the random input vector z of the generator network can be changed and send to the network to obtain different output images. The output of the GANs can also be difficult automated compared, and there are already some implementations available that makes setting up different GAN approaches (StackGAN, InfoGAN, DiscoGAN, and so on) redundant.
I set up a configuration file (e.g.: .yaml) where I can specify different modifications (activation functions, number of network layers, and so on) on DCGAN. The implementation should benchmark the stated modifications (1) by changing those parameters.
To clearing the results: The framework will be extended to do a nearest neighbor search by looking for the output the nearest neighbor of the training set. A simple pixel-wise comparison (SSIM, PSNR, RMSE) can not applied due to the high ambiguity of the output. Perhaps, a comparison of latent space (as in Auto-Encoders[1](AE)) would do this task.
However, using the latent space would be too high workload for this lecture and we want to find the nearest neighbor by using a l_2-Norm. Radford et. al. use a latent variable, a vector $\mathbf{z}$ [1], which we want to change and see what results do we get.

4 Evaluation

The evaluations of the DCGANs capabilities are based on the dataset CelebA[2]. The current configuration is used as ground truth, where time and output are compared in regard to different configurations as listed on the table 1.

[1] https://medium.com/towards-data-science
[2] `http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html`

Dataset: CelebA								
#	**Generator**			**Discriminator**				**Batch N.**
	ReLU	L.ReLU	Tanh	ReLU	L.ReLU	Tanh	Pooling	
1	x				x			x
2		x			x			x
3			x		x			x
4	x			x				x
5	x					x		x
6	x							x
7	x				x		x	x
8	x				x			
9	x				x		x	

Table 1: Table of different possibilities to configure the .yaml file.

The table 1 describes different configurations, which can be configured per run. The configuration can only changed between "Generator", "Discriminator" and "Batch Normalization".
I expect different behavior on the network: No batch normalization means that the time to learn the network will increase [8]. Pooling may decrease time, but maybe also a down sampling of the results can be seen. The activation functions (ReLU, L.ReLU, Tanh) of the hidden layers may increase time until a minimal error is reached and may not be as good as the once (Generator: ReLU, Discriminator: L.ReLU) stated in the paper [6].

Dataset: CelebA			
#	Distribution		
	Uniform (-1,1)	Uniform (-100,100)	Normal($\mu = 0, \sigma = 0.2$)
10	x (see #1)		
11		x	
12			x

Table 2: For testing different distributions, the configuration number 1 of table 1 is used, which shows best results. Number 10 is the same as number

4.1 Nearest Neighbor Evaluation

After training I took 1st picture of each run and want to find the nearest neighbors

$$score = \left\| I^{(m)}_{trained}, I^{(n)}_{training} \right\|^2_2$$

The lower the score, the more similar is the output image to one image of the training set.

5 Results

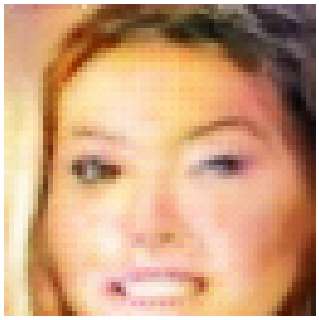
First of all, I analyzed the random vector z. The standard configuration of DCGAN is a uniform distribution and the vector has a size length of 64. The framework generates 100 samples, therefore, the sample_num is 100 and a matrix of 100 by 64 is created.

```
z = uniform(-1,1,size(sample_num, zdim))
z = uniform(-100,100,size(sample_num, zdim))
z = normal(mu=0,sigma=0.2,size(sample_num, zdim))
```

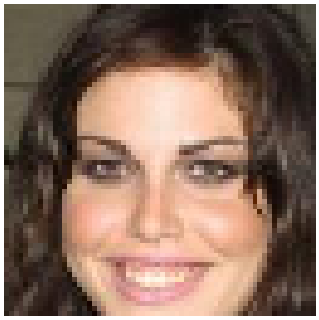
Some plots of the random vectors **z** can be found in in the appendix 6.1.
The normal distribution yields quite similar faces on all, while the uniform does not. This depends on the quite similar normal distribution, where only few values are very high.
Moreover, the time (see table 3) of a not proper chosen distribution increases dramatically. The output of second uniform distribution (-100,100) does also show the worst results.

5.1 Nearest Neighbor Output of different Parameters

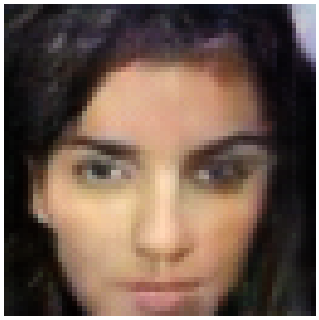
The nearest neighbor is calculated via the l_2 as described in the formulas 4.1. The left column represents the faces which are generated by the trained network. While the right column are the faces from the the dataset celebA for which they are trained for.



Run 1

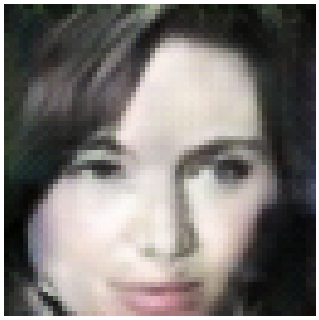


1 Neighbor

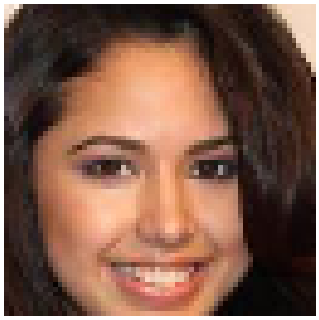


Run 2

2 Neighbor

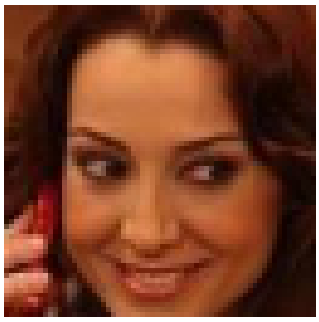


Run 3



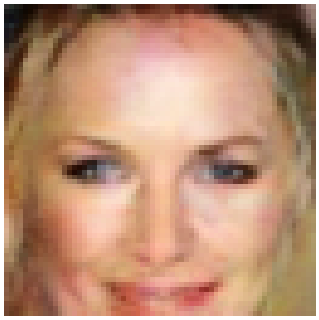
3 Neighbor

Run 4



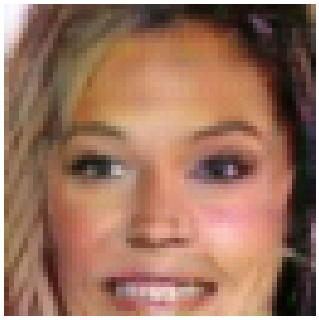
4 Neighbor

Figure 1: Nearest Neighbor 1 of 3.

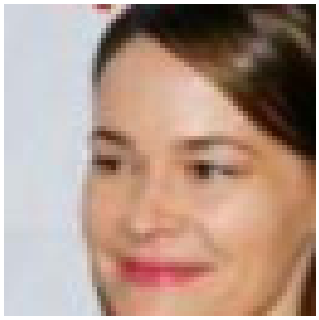


Run 5

5 Neighbor

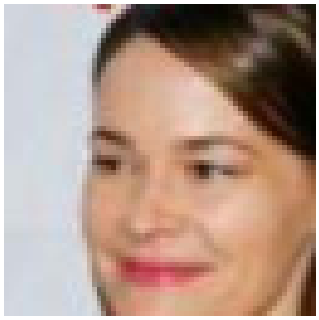


Run 6

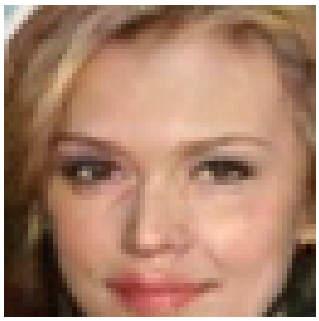


6 Neighbor

Run 7



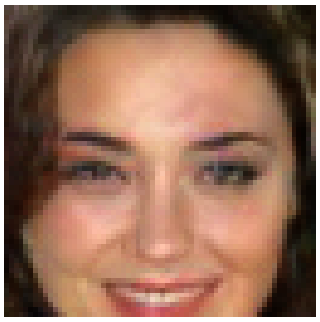
7 Neighbor



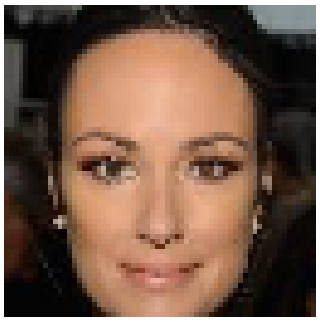
Run 8

8 Neighbor

Figure 2: Nearest Neighbor 2 of 3.



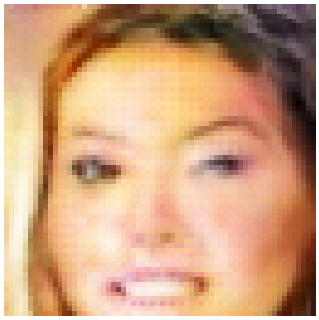
Run 9



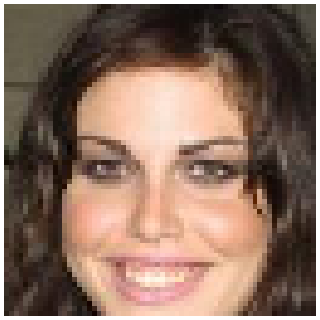
9 Neighbor

Figure 3: Nearest Neighbor 3 of 3.

5.2 Nearest Neighbor Output of different Random Vectors z

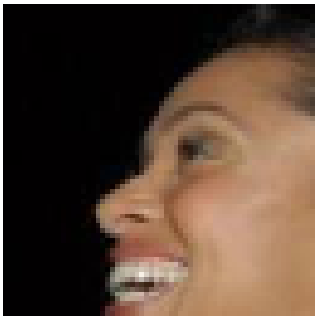


Run 10

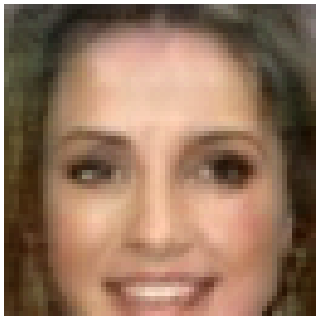


10 Neighbor

Run 11



11 Neighbor



Run 12

12 Neighbor

Figure 4: Nearest Neighbor based on the table 2.

6 Appendix

6.1 Random Vectors z

The following plots show the random vectors **z**. One column corresponds to one random vector which has the size of 64x1. For 100 samples there are 100 hundred columns. The uniform distribution in figure 5 is the standard configuration. The uniform distribution is modified by a bigger range between -100 to 100 in figure 6. Finally, a normal distribution with a $\mu = 0$ and $\sigma = 0.2$ is tried, which shows up in figure 7.

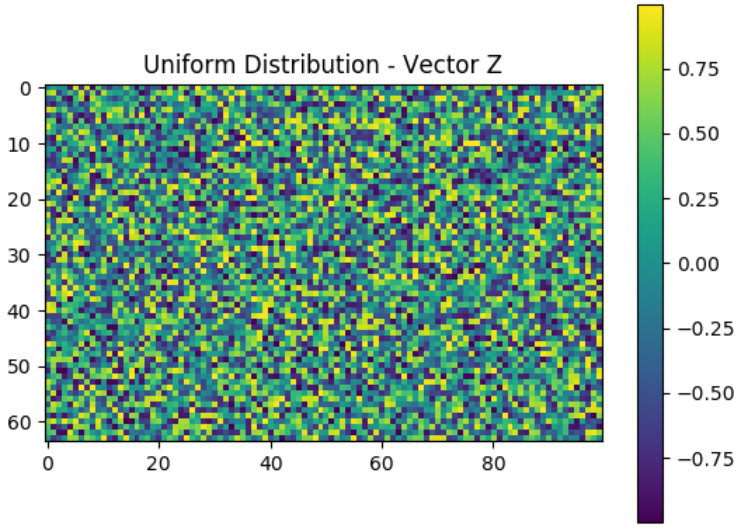


Figure 5: Uniform Distribution (-1,1)

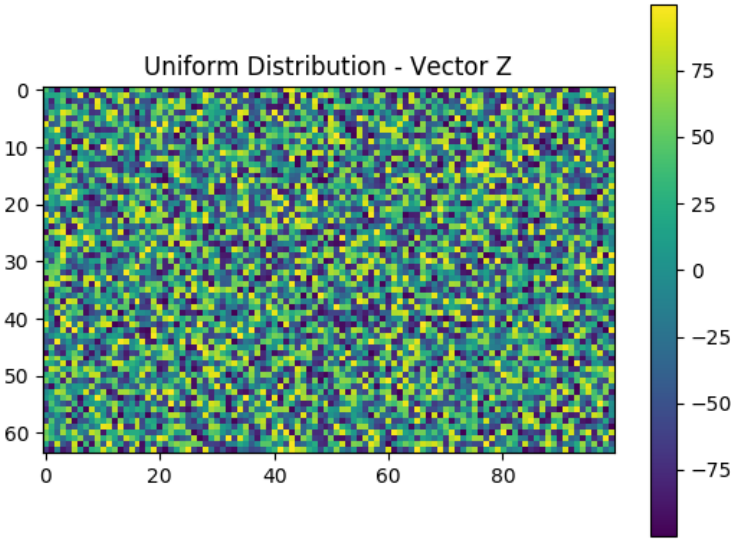


Figure 6: Uniform Distribution (-100,100)

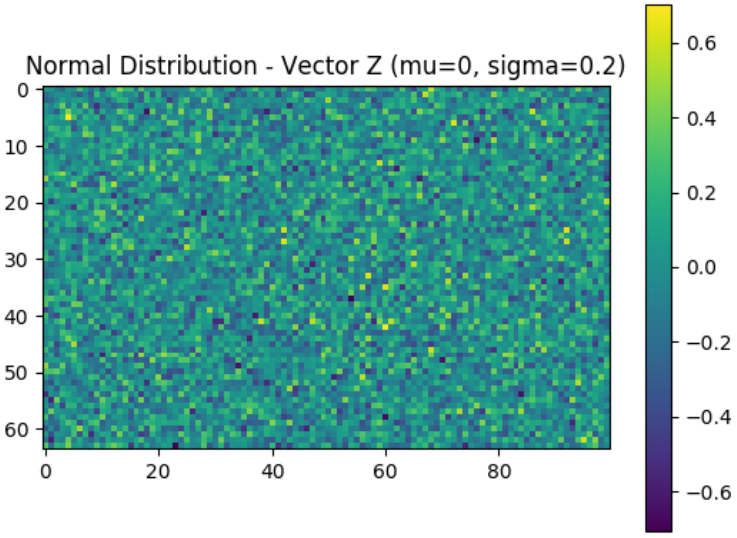
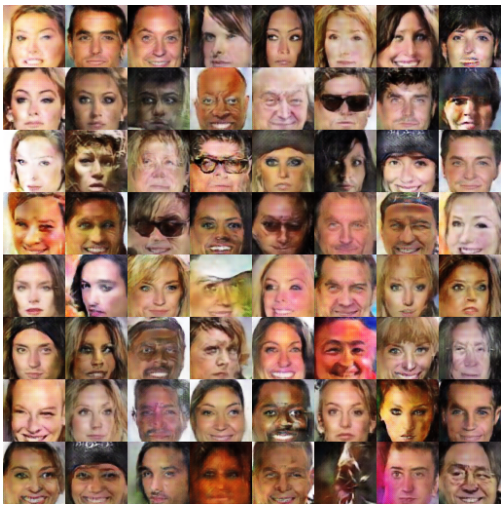


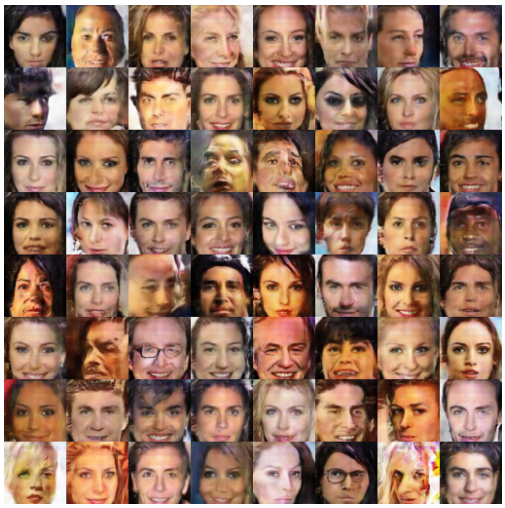
Figure 7: Normal Distribution ($\mu = 0$,$\sigma = 0.2$)

6.2 Trained Faces

Numbers in the caption refer to the table 1. 64 samples are are shown on one of the figures, where each face corresponds to one random vector $\mathbf{z}$, shown in 5.

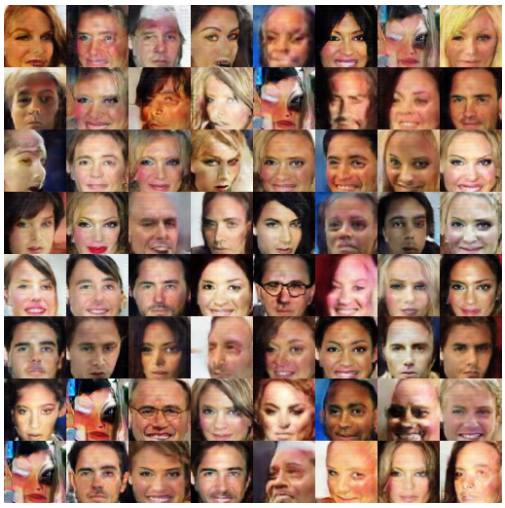


Run 1

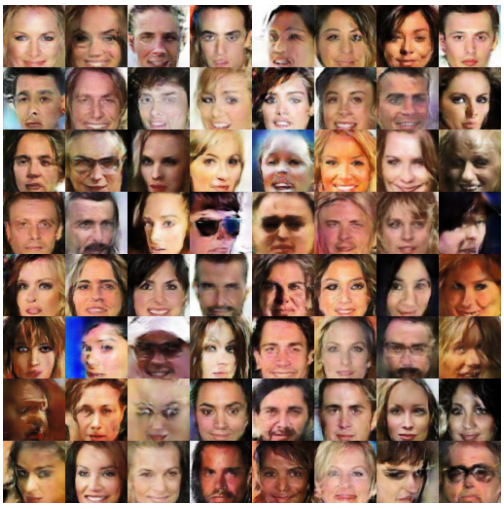


Run 2

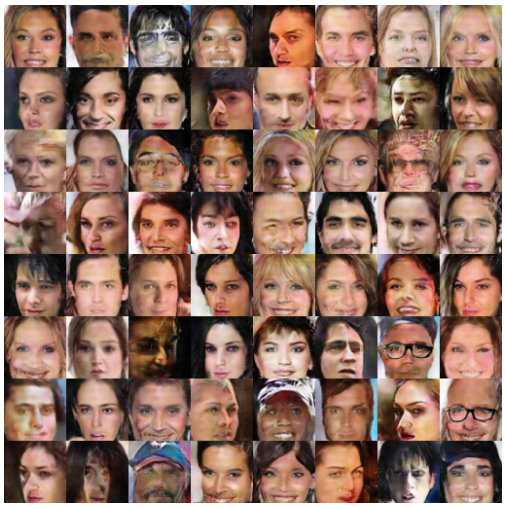
Run 3



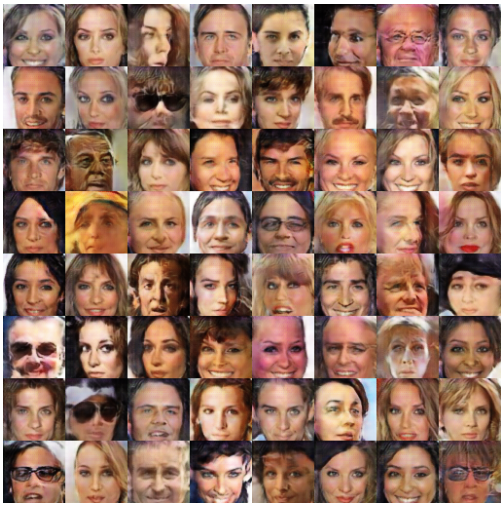
Run 4



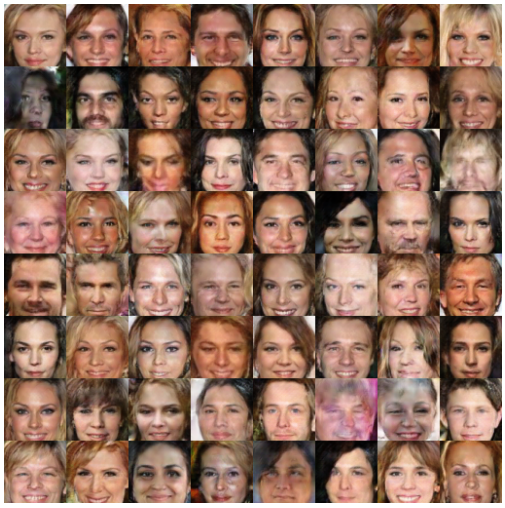
Run 5



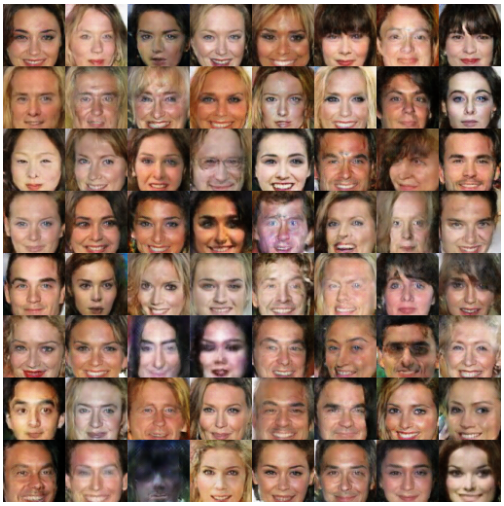
Run 6



Run 7

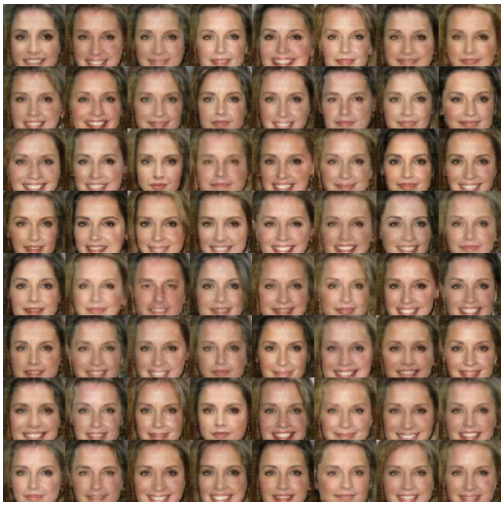


Run 8



Run 9

Run 11



Run 12

6.3 Time Measurements

#	Time	
	seconds	hours
1	76336.74	21.2
2	78964.65	21.9
3	76549.25	21.2
4	74689.87	20.7
5	74546.76	20.7
6	75443.76	20.9
7	76476.46	21.2
8	74635.61	20.7
9	72713.31	20.2
10	see run #1	
11	90718.89	25.2
12	126220.64	35.1

Table 3: Time table of different runs.

References

[1] Ian Goodfellow. NIPS 2016 Tutorial: Generative Adversarial Networks. 2016. 3

[2] Ian Goodfellow, Jean Pouget-Abadie, and Mehdi Mirza. Generative Adversarial Networks. pages 1–9, 2014. 2

[3] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. 2

[4] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Research Blog: Inceptionism: Going Deeper into Neural Networks. 2

[5] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*, (3):807–814, 2010. 2

[6] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. pages 1–16, 2015. ii, 2, 4

[7] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for Simplicity: The All Convolutional Net. 2014. 2

[8] Christian Szegedy. Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift. 2015. 2, 4