# Generating Paintings With InfoGAN

Waseem Khan
Rutgers University
Professor Ahmed Elgammal

## Abstract

InfoGAN, a generational network with feature control, is used to produce unique paintings with varying details controlled by latent codes that the user can specify. Since optimization of the loss function requires a correlation between the codes and the image, the network can control simple and creative qualities of a generated image and produce nice results with little training (few minutes on GPU).

## Introduction

Generational Adversarial Networks or GAN is a new and popular field in contemporary deep learning. Deep models have gotten very accurate at image classification and identification, but the newer and more interesting challenge is can a computer create a unique image? By using deconv networks, an array of random inputs along with a class can be used to generate an image. But how can this model be trained?

A GAN is essentially two networks, a generational network and an adversarial network, hence the name. The generational network (G) generates images that try to replicate the original, while the adversarial (A) network tries to distinguish between the generated and the real images from the training set, without knowing which is which ahead of receiving them. The analogy is that of a counterfeiter (G) making counterfeit money and the police officer (A) trying to figure out which money is real and which is not. As the police becomes better at distinguishing which money is fake, the counterfeiter must try different attempts to deceive the police. By connecting the two networks together and having correlated loss functions, one can achieve good results.

Attempts like Variational Autoencoder generate fuzzy images because they restrict the encoded latent vector to a unit gaussian, typically. GAN's currently provide the sharpest results. InfoGAN was a variation on the GAN that tries to have the generated image share as much mutual information with several latent codes that are initially used to generate the image. By adding this mutual information optimization in the loss function, the latent codes can control aspects or qualities of the image.

With this feature, the question was raised: can a network generate art and can varying these latent codes affect the resulting image in creative ways? This was investigated starting from a simple handwritten numbers all the way to art.

## Methods and Materials

The datasets used were MNIST, SVHN, and WikiArt - holding handwritten numbers, street house numbers, and an assortment of 80k paintings spanning 26 categories. The language used was Lua on the Torch framework, and the network was trained on a GTX Titan.

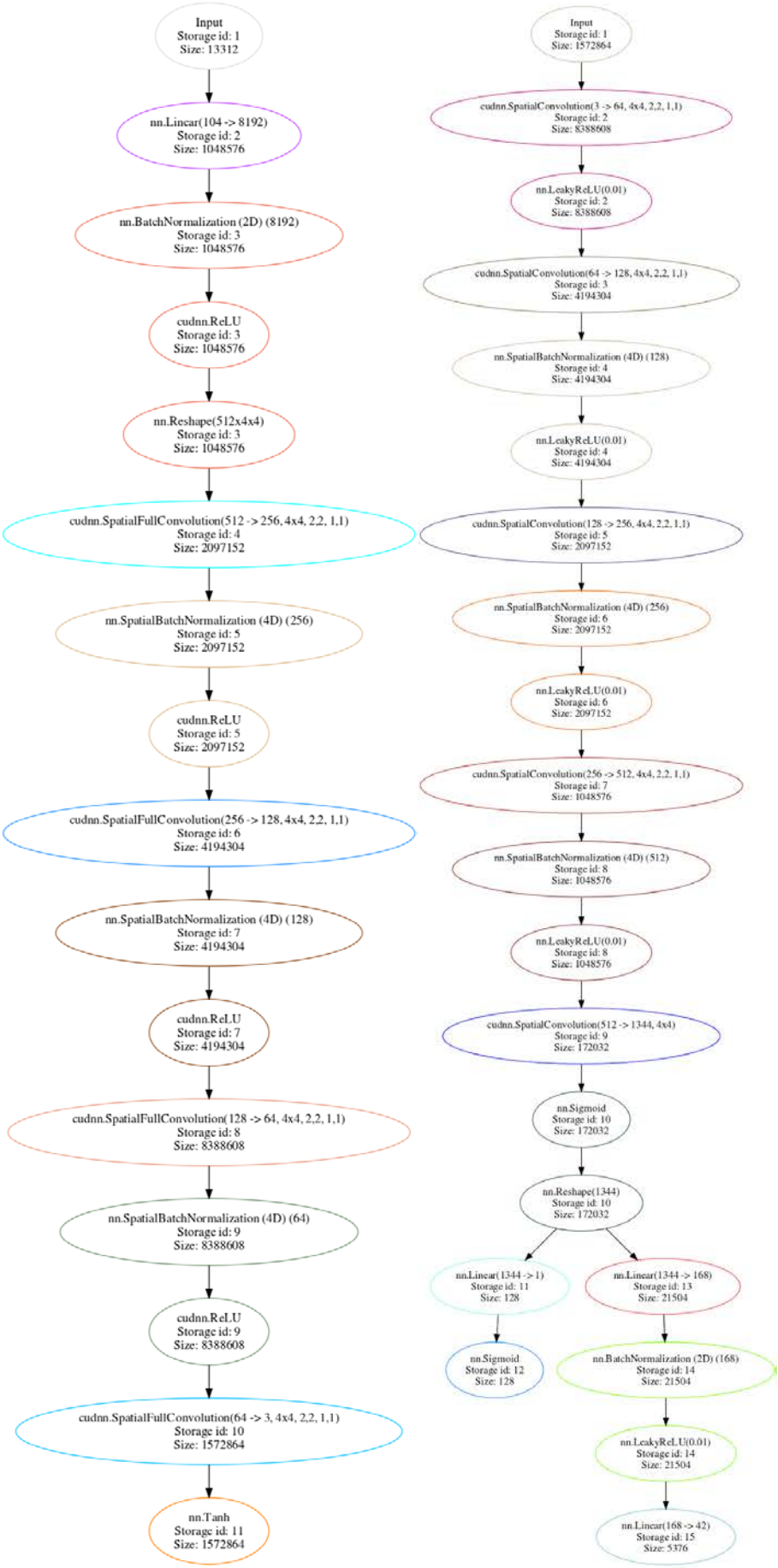
The torch implementation of OpenAI's InfoGAN is found here: https://github.com/anibali/infogan, and only implements a model for a simple mnist example - which is a 28x28x1 tensor. The data is stored in a large tensor that contains all the training images and labels, so it can be passed in as one input - based on the batch size.

To expand this to color as done in the original paper, I downloaded SVHN where each individual image is 32x32x3, having three channels corresponding to the RGB spectrum. By trial and error, I was able to build a more complex model that could handle the larger amount of input and also produce original-looking results. This was achieved through adding more weights into the MNIST model and moved on once the change in latent code across the generations could be noticed.

Next, I moved onto the wikiart data. I wrote a script to resize the paintings to 32x32x3 and stored the images and labels in a tensor file for torch. I used the naive SVHN model as test run on wikiart and the generated images were unrecognizable. By adding more weights in the right areas, I was able to have decent generation (shown in results). Next, I added a layer each to the generator and discriminator networks to build up to 64x64x3 images. By adding another Conv and DeConv layer to the two networks, I was able to let the network learn more features and generate a higher quality image.

The model used for the last 64x64x3 wikiart images can be found below. The left image being the generator which takes in inputs of the latent codes and random variables and outputs a generated image tensor. The right image is the discriminator which has both a regular discriminator head from a regular GAN and also an Info head which allows for latent codes to be factored into it.

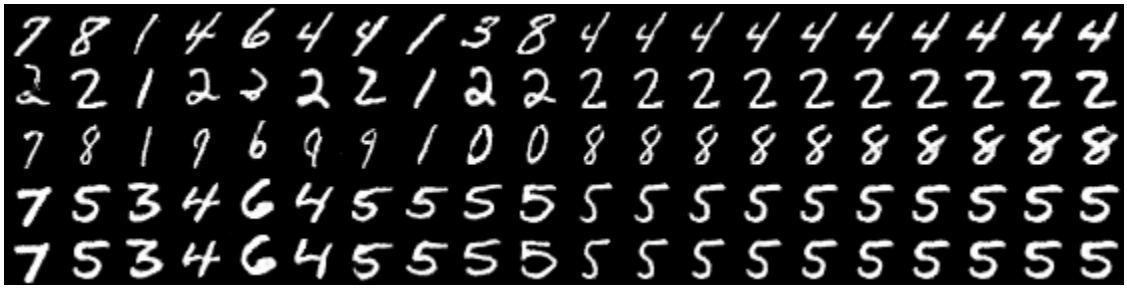
I stopped the training at 100 epochs usually because the loss function of the generated images tends to increase after that and the generated images do not improve in quality. General Sharpening is added only to the generated wikiart64 images to see the picture better.


Input
Storage id: 1
Size: 13312
nn.Linear(104 -> 8192)
Storage id: 2
Size: 1048576
nn.BatchNormalization (2D) (8192)
Storage id: 3
Size: 1048576
cudnn.ReLU
Storage id: 3
Size: 1048576
nn.Reshape(512x4x4)
Storage id: 3
Size: 1048576
cudnn.SpatialFullConvolution(512 -> 256, 4x4, 2,2, 1,1)
Storage id: 4
Size: 2097152
nn.SpatialBatchNormalization (4D) (256)
Storage id: 5
Size: 2097152
cudnn.ReLU
Storage id: 5
Size: 2097152
cudnn.SpatialFullConvolution(256 -> 128, 4x4, 2,2, 1,1)
Storage id: 6
Size: 4194304
nn.SpatialBatchNormalization (4D) (128)
Storage id: 7
Size: 4194304
cudnn.ReLU
Storage id: 7
Size: 4194304
cudnn.SpatialFullConvolution(128 -> 64, 4x4, 2,2, 1,1)
Storage id: 8
Size: 8388608
nn.SpatialBatchNormalization (4D) (64)
Storage id: 9
Size: 8388608
cudnn.ReLU
Storage id: 9
Size: 8388608
cudnn.SpatialFullConvolution(64 -> 3, 4x4, 2,2, 1,1)
Storage id: 10
Size: 1572864
nn.Tanh
Storage id: 11
Size: 1572864
Input
Storage id: 1
Size: 1572864
cudnn.SpatialConvolution(3 -> 64, 4x4, 2,2, 1,1)
Storage id: 2
Size: 8388608
nn.LeakyReLU(0.01)
Storage id: 2
Size: 8388608
cudnn.SpatialConvolution(64 -> 128, 4x4, 2,2, 1,1)
Storage id: 3
Size: 4194304
nn.SpatialBatchNormalization (4D) (128)
Storage id: 4
Size: 4194304
nn.LeakyReLU(0.01)
Storage id: 4
Size: 4194304
cudnn.SpatialConvolution(128 -> 256, 4x4, 2,2, 1,1)
Storage id: 5
Size: 2097152
nn.SpatialBatchNormalization (4D) (256)
Storage id: 6
Size: 2097152
nn.LeakyReLU(0.01)
Storage id: 6
Size: 2097152
cudnn.SpatialConvolution(256 -> 512, 4x4, 2,2, 1,1)
Storage id: 7
Size: 1048576
nn.SpatialBatchNormalization (4D) (512)
Storage id: 8
Size: 1048576
nn.LeakyReLU(0.01)
Storage id: 8
Size: 1048576
cudnn.SpatialConvolution(512 -> 1344, 4x4)
Storage id: 9
Size: 172032
nn.Sigmoid
Storage id: 10
Size: 172032
nn.Reshape(1344)
Storage id: 10
Size: 172032
nn.Linear(1344 -> 1)
Storage id: 11
Size: 128
nn.Linear(1344 -> 168)
Storage id: 13
Size: 21504
nn.Sigmoid
Storage id: 12
Size: 128
nn.BatchNormalization (2D) (168)
Storage id: 14
Size: 21504
nn.LeakyReLU(0.01)
Storage id: 14
Size: 21504
nn.Linear(168 -> 42)
Storage id: 15
Size: 5376

# Results

The outputs are formatted so that the latent code is varied across the columns and each row has a slightly different latent code distribution.

MNIST



Generated images for MNIST: first image on left shows varying c1, which is a discrete latent code that controls the number generated. The image on the right shows a continuous latent code being varied which correlates to how much each digit is stretched, with the left side having the narrowest numbers and the right side having the widest digits.

SVHN

These are the generated images for SVHN: the image on the left shows varying a discrete c1, which manipulates the number generated but only to a number close to it in latent space in color. We can see the transition between a ‘2’ to an ‘8’ in the middle row. The continuous latent code c5 on the right being altered corresponds to width, similar to MNIST, but also to the brightness of the digit.

WIKIART-32

Here are two generated images of wikiart 32 pixel size. The left image shows a discrete latent code being changed, which corresponds to general shapes of objects in the painting. The right image shows a continuous latent code being altered and relates to a transition between two paintings.

WIKIART-64

These images have been sharpened to see the images more clearly.

This first image has a discrete latent code which manipulates color and shapes of objects.

This next image has a continuous latent code that controls the simplicity and colors of an image. We can see that the second row is a generated painting of a person and the simplest version of it is on the right (black and white almost) and the rightmost picture has the deepest colors and range.

An animated version of the paintings while being generated as the model trains can be found along with the source code at my github page: [https://github.com/waseemkhan96/wikiart-infogan](https://github.com/waseemkhan96/wikiart-infogan)

# Conclusion

While working on this project, I have learned the many steps in making a deep learning model. I can now preprocess and store images into a dataset file, create a neural network corresponding to the dataset, and train the network on GPU using Torch. InfoGAN optimizes the mutual information between latent codes and the generated images to control the features of a generated image. By adding weights and layers, the training time is increased but the network is capable of learning deeper features from the real images that it can reproduce.

From training infoGAN on MNIST and SVHN, it can be seen that the latent codes control the number and width/brightness of the output. By applying it to something as abstract as art, we see features such as shape, content, simplicity/complexity and color being affected as seen in the results. Next steps would be to potentially build a better model to handle even larger image sizes and create more latent codes to see what other features could be controlled.

# Literature Cited

X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. ArXiv e-prints, June 2016.

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. ArXiv e-prints, June 2014.

D. P Kingma and M. Welling. Auto-Encoding Variational Bayes.ArXiv e-prints, December 2013.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng Reading Digits in Natural Images with Unsupervised Feature Learning *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278{2324,November 1998. MNIST database available at http://yann.lecun.com/exdb/mnist/.