Project Report

Topic

GANs - Generative Adversarial Networks

CMPE 257: Group 6

1. John Wang:	015938845
2. Pragati Singh:	015971683
3. Madhuri Ranvirkar:	015273518
4. Alireza Mahinparvar:	014295437
5. Priyank Thakkar:	015944318

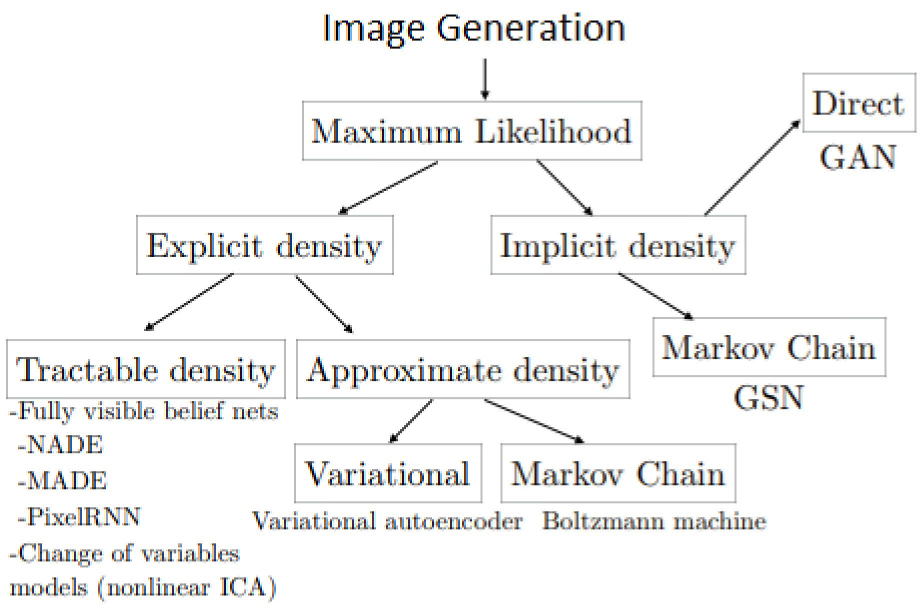
Guided By

CMPE 257-02 Professor: Chandrasekhar Mukherjee

Abstract

Automation of human vision related tasks such as classification and generation of images such as sorting through ripe vegetables or upscaling images has led to great improvements in the efficiency of human society. However, traditionally, machine learning algorithms for accomplishing these tasks of data classification, prediction and generation have been largely human supervision intensive tasks.

Convolutional neural networks revolutionized image processing in artificial intelligence by providing a technique that requires very little pre-processing of data. This in turn provided a general way of classifying and predicting images without much hand-engineered human intervention. This led to the automation of many previously human dominated tasks such as determining if a fruit was ripe or classifying a photo as explicit on Instagram. However, there is still a glaring hole in the AI automation paradigm. While CNNs are great at classifying and predicting images, they are unable to generate, translate, or mimic images.



Traditionally, image generation was handled by methods such as: variational autoencoders, Boltzmann machines, NADE, MADE, PixelRNN, and GSN [21]. Not only do these methods all require a lot of human supervision, but they are also all extremely computationally intensive. These methods are so computationally intensive that they are limited in the application for life purposes.

Just as CNNs revolutionized classification and prediction for AI vision, Generative Adversarial Networks (GANs) fulfilled this role for is the missing role for generation. This is the role that Generative Adversarial Networks (GANs) fills. While CNNs are broad tools great at the classification of existing data, GANs are the analog in generating data. GANs provides a way of automating data generation, translation, and mimicry in an automated and convolutedly intelligent manner in order to improve the speed and efficiency at which human society moves.

In our project, we are building 2 models: MNIST digital handwriting generator as well as the CelebA faces generator in order to better understand GANs systems in implementation, mathematics, strengths as well as limitations.

Introduction

“Deep learning has the possibility of discovering complex, hierarchical models that describe probability distributions over the types of data encountered in AI applications, such as natural pictures, audio waveforms containing speech, and symbols in natural language corpora” [1].

“Discriminative models, which transfer a high-dimensional, rich sensory input to a class label, have been the most spectacular triumphs in deep learning thus far [1, 2]”. There is the difficulty of approximating many intractable probabilistic computations that arise in maximum likelihood estimation. We cannot leverage the benefits of piecewise linear units in the generative context. To solve these difficulties, we propose a generative model. In the proposed study, an adversary is placed against the generative model. A discriminative model that learns to distinguish between samples from the model and data distributions. The generative model is analogous to the team of counterfeiters, the discriminative model is comparable to the police. In this game, the competition forces both sides to develop their procedures until the counterfeits are indistinguishable from genuine products. This framework can produce customized training algorithms for a wide range of models and optimization methods. In this study, we explore the case where a generative model passes random noise through a multilayer perceptron and generate samples. Descriptive model is also a multilayer perceptron. We call this process as adversarial nets. We used the extremely successful backpropagation and dropout methods to train both models. Only forward propagation is used to sample from the generative model.

The other solution in [1], “is demonstrating how it can be used to symbolically define mathematical functions, automatically derive gradient expressions, and compile these expressions into executable functions that outperform implementations using other existing tools” [1]. In [2], “those exploiting as building blocks unsupervised learning of single-layer models such as Restricted Boltzmann Machines, used to construct deeper models such as Deep Belief Networks” [2].

Materials and Methods

We first experimented with the MNIST dataset. We discovered how to develop a generative adversarial network with deep convolutional networks for generating handwritten digits.

The MNIST dataset is an acronym that stands for the Modified National Institute of Standards and Technology dataset. It's a collection of 70,000 little square grayscale photos of handwritten single numbers ranging from 0 to 9.

The mnist_load dataset () method in Keras gives you access to the MNIST dataset. It returns two tuples, one containing the input and output elements for the standard training dataset and the other containing the input and output elements for the standard test dataset.

The photos are grayscale with a black (0-pixel value) backdrop and white handwritten numerals (pixel values near 255). When plotted, the pictures would be predominantly black with a white number in the center. The photos in the training dataset served as the foundation for training a Generative Adversarial Network.

The generator model, in particular, learns how to produce new plausible handwritten digits between 0 and 9 using a discriminator that attempts to discriminate between genuine photos from the MNIST training dataset and new images generated by the generator model. Although the development of a grayscale output picture is required, this is a reasonably easy task that does not necessitate the use of a complicated generator or discriminator model.

Defining and training the discriminator model:

The model must be given a sample picture from our dataset and must predict whether the sample is real or false. This is a problem of binary categorization. The discriminator model comprises two convolutional layers, each with 64 filters, a tiny kernel size of 3, and a stride of 2 that is greater than typical. To predict whether the input sample is real or false, the model contains no pooling layers and only one node in the output layer with the sigmoid activation function. For binary classification, the model is trained to minimize the binary cross entropy loss function.

In creating the discriminator model, we followed various best practices, such as employing LeakyReLU instead of ReLU and Dropout.

```
Model: "sequential_1"
_________________________________________________________________
_
 Layer (type)                Output Shape              Param #
=================================================================
=
 conv2d (Conv2D)             (None, 14, 14, 64)        1664

 leaky_re_lu_3 (LeakyReLU)   (None, 14, 14, 64)        0

 dropout (Dropout)           (None, 14, 14, 64)        0

 conv2d_1 (Conv2D)           (None, 7, 7, 128)         204928

 leaky_re_lu_4 (LeakyReLU)   (None, 7, 7, 128)         0

 dropout_1 (Dropout)         (None, 7, 7, 128)         0

 flatten (Flatten)           (None, 6272)              0

 dense_1 (Dense)             (None, 1)                 6273

=================================================================
=
Total params: 212,865
Trainable params: 212,865
Non-trainable params: 0
_________________________________________________________________
_
```

We can see that the aggressive 2×2 stride acts to down-sample the input image, first from 28×28 to 14×14, then to 7×7, before the model makes an output prediction. This pattern is by design such that we do not use pooling layers instead use the large stride to achieve a similar down sampling effect.

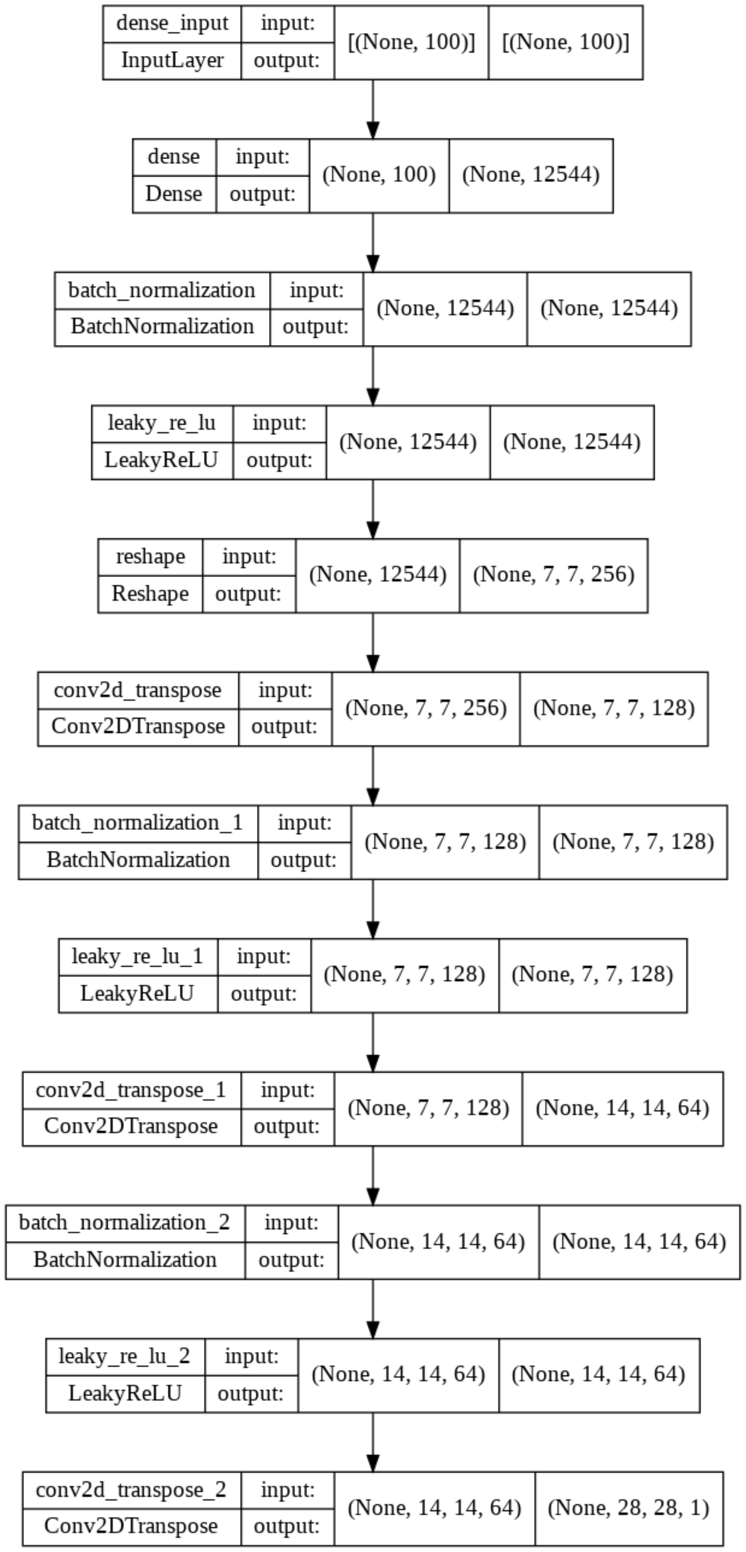


Figure 1: Discriminator Model

Defining the Generator Model:

```
Model: "sequential"
_________________________________________________________________
_
 Layer (type)                Output Shape              Param #   
=================================================================
=
 dense (Dense)               (None, 12544)             1254400   
                                                                 
 batch_normalization (BatchN  (None, 12544)            50176     
 ormalization)                                                   
                                                                 
 leaky_re_lu (LeakyReLU)     (None, 12544)             0         
                                                                 
 reshape (Reshape)           (None, 7, 7, 256)         0         
                                                                 
 conv2d_transpose (Conv2DTra  (None, 7, 7, 128)        819200    
 nspose)                                                         
                                                                 
 batch_normalization_1 (Batc  (None, 7, 7, 128)        512       
 hNormalization)                                                 
                                                                 
 leaky_re_lu_1 (LeakyReLU)   (None, 7, 7, 128)         0         
                                                                 
 conv2d_transpose_1 (Conv2DT  (None, 14, 14, 64)       204800    
 ranspose)                                                       
                                                                 
 batch_normalization_2 (Batc  (None, 14, 14, 64)       256       
 hNormalization)                                                 
                                                                 
 leaky_re_lu_2 (LeakyReLU)   (None, 14, 14, 64)        0         
                                                                 
 conv2d_transpose_2 (Conv2DT  (None, 28, 28, 1)        1600      
 ranspose)                                                       
                                                                 
=================================================================
=
Total params: 2,330,944
Trainable params: 2,305,472
Non-trainable params: 25,472
```

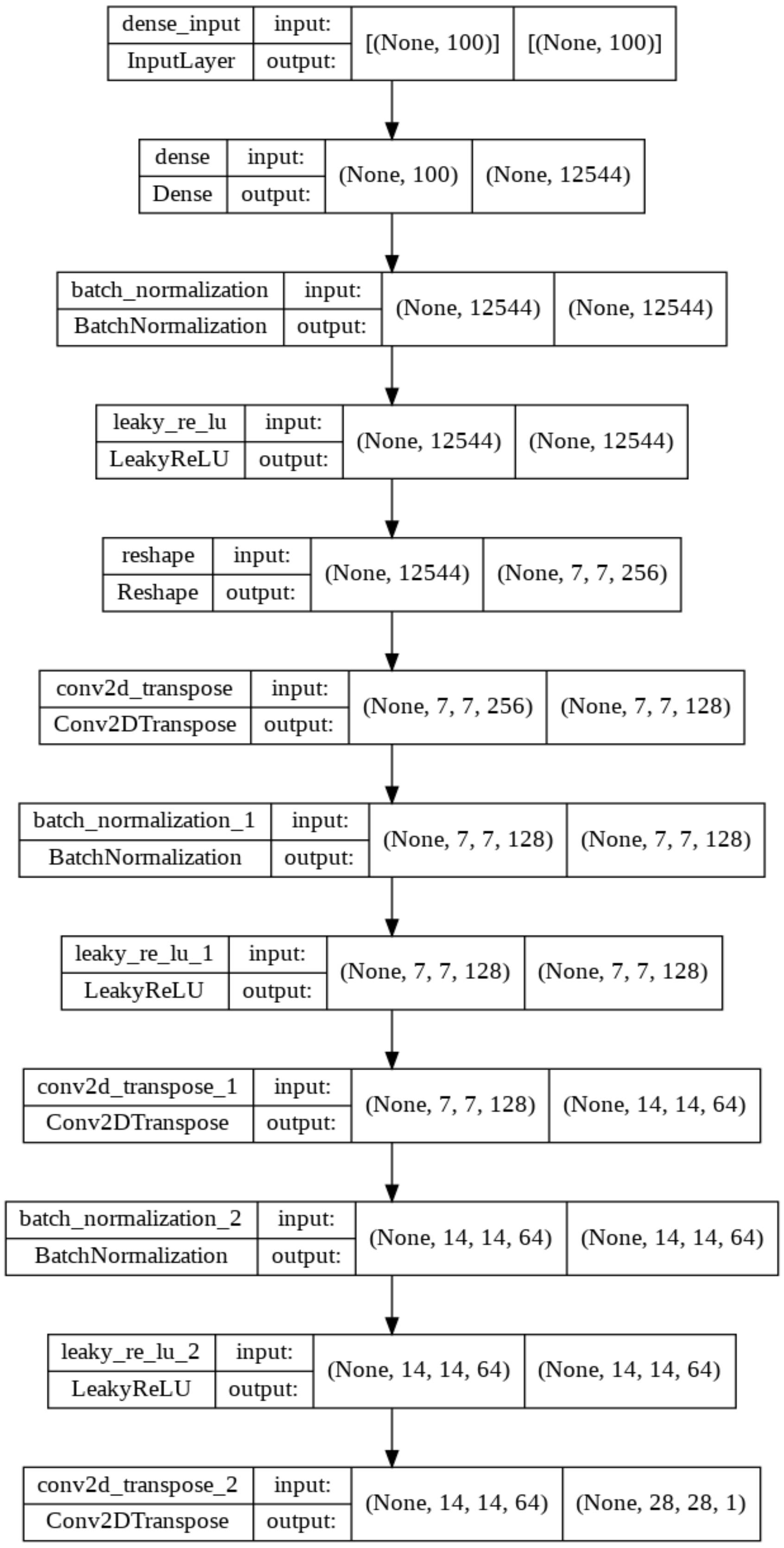

dense_input | InputLayer | input: | output: | [(None, 100)] | [(None, 100)]
dense | Dense | input: | output: | (None, 100) | (None, 12544)
batch_normalization | BatchNormalization | input: | output: | (None, 12544) | (None, 12544)
leaky_re_lu | LeakyReLU | input: | output: | (None, 12544) | (None, 12544)
reshape | Reshape | input: | output: | (None, 12544) | (None, 7, 7, 256)
conv2d_transpose | Conv2DTranspose | input: | output: | (None, 7, 7, 256) | (None, 7, 7, 128)
batch_normalization_1 | BatchNormalization | input: | output: | (None, 7, 7, 128) | (None, 7, 7, 128)
leaky_re_lu_1 | LeakyReLU | input: | output: | (None, 7, 7, 128) | (None, 7, 7, 128)
conv2d_transpose_1 | Conv2DTranspose | input: | output: | (None, 7, 7, 128) | (None, 14, 14, 64)
batch_normalization_2 | BatchNormalization | input: | output: | (None, 14, 14, 64) | (None, 14, 14, 64)
leaky_re_lu_2 | LeakyReLU | input: | output: | (None, 14, 14, 64) | (None, 14, 14, 64)
conv2d_transpose_2 | Conv2DTranspose | input: | output: | (None, 14, 14, 64) | (None, 28, 28, 1)

Figure 2: Generator Model

The generator model is in charge of developing fresh, fictitious yet believable handwritten digit pictures. It accomplishes this by taking a latent space point as input and producing a square grayscale picture.

The latent space is a vector space of Gaussian-distributed values that can be freely defined. The generator model will give meaning to the latent points and, as a result, the latent space, until the latent vector space reflects a compressed representation of the output space, MNIST images, which only the generator understands how to transform into credible MNIST images at the conclusion of training.

To create a generator model, we must convert a vector from a 100-dimensional latent space to a 2D array with 2828 or 784 values. There are several ways to do it.

The first hidden layer is a Dense layer, which has enough nodes to depict a low-resolution version of the output image. A picture half the size (one quarter the area) of the output image would have 1414 or 196 nodes, whereas an image one eighth the size (one eighth the area) would have 707 or 49 nodes. The discriminator model comprises two convolutional layers, each with 64 filters, a tiny kernel size of 3, and a stride of 2 that is greater than typical. To predict whether the input sample is real or false, the model contains no pooling layers and only one node in the output layer with the sigmoid activation function. The model is trained to minimize the binary cross entropy loss function, appropriate for binary classification.

We don’t just want one low-resolution version of the image; we want many parallel versions or interpretations of the input. This is a pattern in convolutional neural networks where we have many parallel filters resulting in multiple parallel activation maps, called feature maps, with different interpretations of the input. We want the same thing in reverse: many parallel versions of our output with different learned features that can be collapsed in the output layer into a final image. The model needs space to invent, create, or generate.

We have used some best practices in defining the discriminator model, such as the use of LeakyReLU instead of ReLU, using Dropout, and using the Adam version of stochastic gradient descent with a learning rate of 0.0002 and a momentum of 0.5.

Algorithms and Theory

The adversarial network introduced as implicit generative model which cost function can be interpreted as minmax game between generator and discriminator and the advantage that this model has rather than generative model is that this model can bring pictures and results with good quality and that is main difference between GAN and CNN networks that we know.

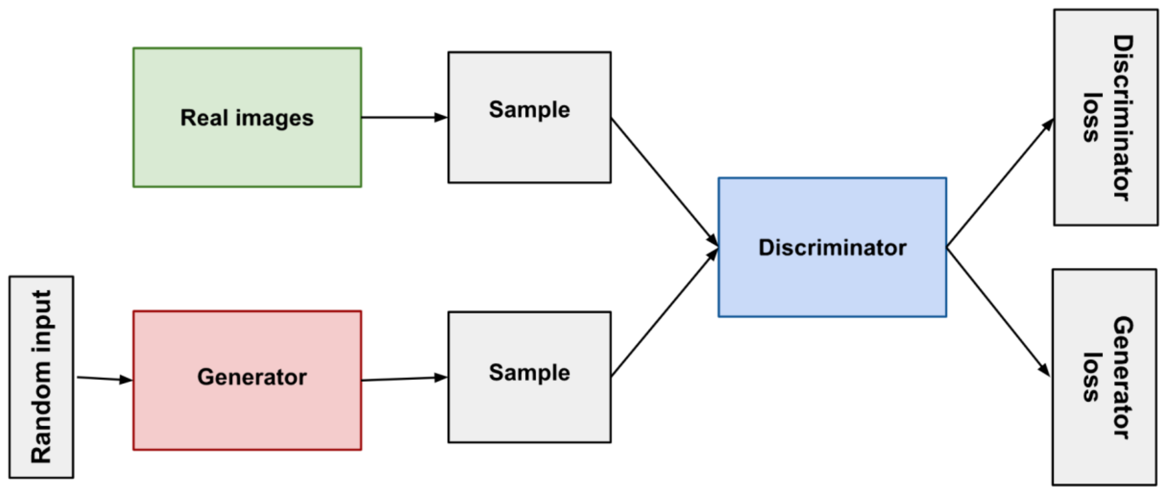


Figure 3: Algorithm Structure

A generator is a neural network that generates fake data in order to train the discriminator. A discriminator is a neural network that distinguishes between true and fraudulent data. Data for discriminator training originates from two different sources. One of the actual data examples is a real human, bird, or other animal image. Generator creates fictitious data instances. We must ensure that our data cannot be readily rejected or accepted in this game between discriminator and generator because if, for example, data is easily detectable, the discriminator will pass all test cases and if data is too difficult to recognize, the discriminator will reject all data test instances. As a result, the data we use for training and testing must fall somewhere in the middle. In other words, the information presented should neither be too excellent or excessively negative.

Learning with GANs is driven by a novel cost function:

$$min_G max_D V(D,G) = E_{x\sim Pdata(x)}[log\ D(x)] + E_{z\sim P(z)}[log(1 - D(G(z)))]$$

The cost function is composed of 2 components, the entropy of the discriminator D(x) and the entropy of the generator G(z). We are trying to maximize G(z) while minimizing D(x). We then use the gradient of this cost function to drive changes through backpropagation to update to the next step.

But how do we know when the GAN has reached completion? Consider a proposition for an optimal discriminator:

$$D_G(x) = \frac{P_{data}(x)}{P_{data}(x) + P_g(x)}$$

Based on the formulas we have above the minmax game equation can be written like below:

$$C(G) = min_G max_D V(D, G)$$

$$= E_{x\sim Pdata(x)}[log\ D(x)] + E_{z\sim P(z)}[log(1 - D(G(z)))]$$

$$= E_{x\sim Pdata(x)}[log\ D(x)] + E_{z\sim P(z)}[log(1 - D(z))]$$

$$= E_{x\sim Pdata(x)}[log\ \frac{P_{data}(x)}{P_{data}(x)+P_g(x)}] + E_{z\sim P(z)}[log \frac{P_{data}(x)}{P_{data}(x)+P_g(x)}]$$

The minimum C(G) can be achievable if and only if $p_g = p_{data}$ and at this moment C(G) achieves the value -log 4. And to prove that we can say,

for $p_g = p_{data}$ we have $D_g(x) = \frac{1}{2}$

So $E_{x\sim Pdata(x)}[-log\ 2] + E_{z\sim P(z)}[-log\ 2] = -\ log\ 4$

So, by subtracting above expression from $V(D, G)$ we will obtain:

$$= -log(4)\ + KL(p_{data}|\ \ |\frac{P_{data} + P_g}{2}) + KL(P_g|\ \ |\frac{P_{data} + P_g}{2})$$
$$= -log\ (4)\ + 2.JSD(P_{data}|\ \ |P_g)$$

In second proposition we have the case that if Discriminator (D) and generator (G) have enough capacity then discriminator is able to reach optimum given G and P_g is updated and P_g converge to P_{data}

$$E_{x\sim Pdata(x)}[log\ D(x)] + E_{x\sim P(g)}[log(1 - D(G(x)))]$$

Results

As we see in Figure 3 after the data was passed to the discriminator we see some discriminant loss and generator (adversary loss) and the results we got from our model on figure 4 and figure 5 shows that the discriminator loss is between 0.5 to 1. Also, we see adversary loss between 0.75 to 1.25. Based on diagrams below we see consistency between the ranges of discriminator and adversary loss. For both cases in low number of epoch cases, the loss is high but after we increase epochs the consistency shows up in loss ranges.

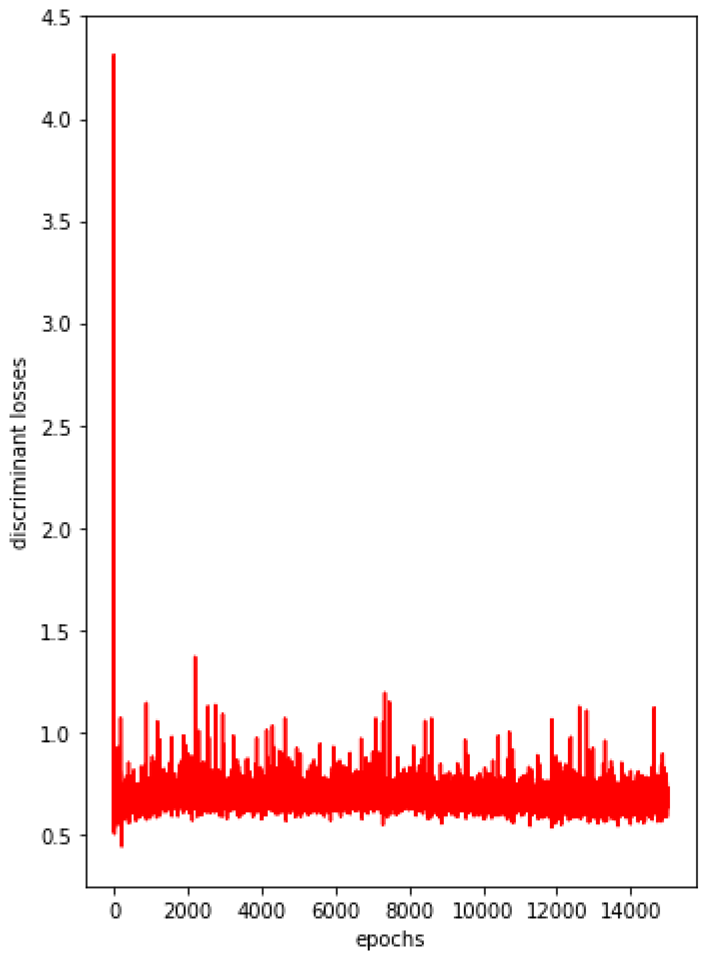


Figure 4: Discriminative Losses

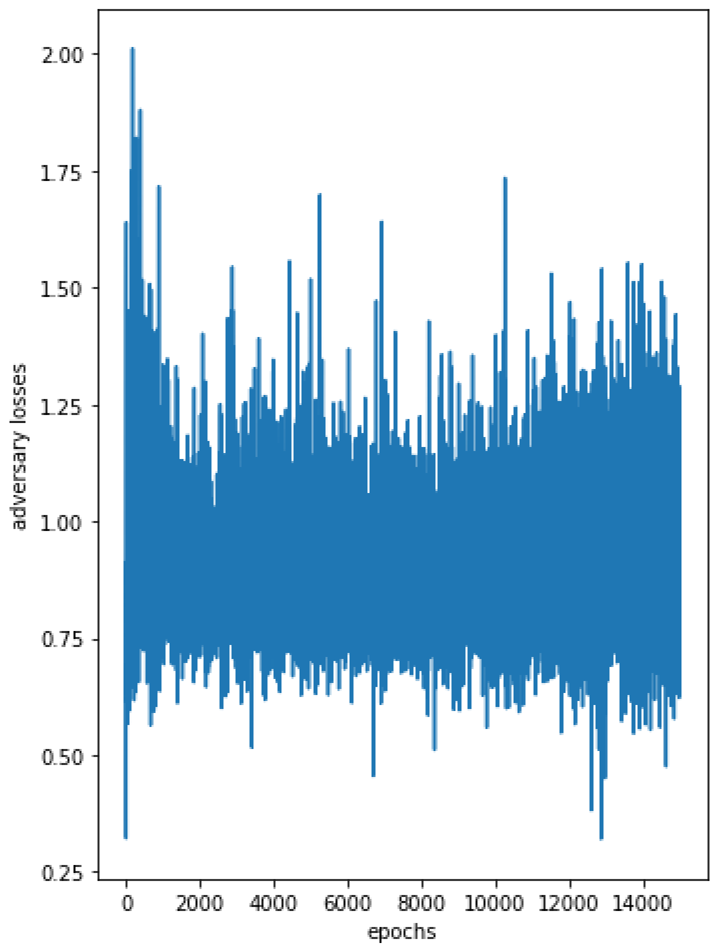
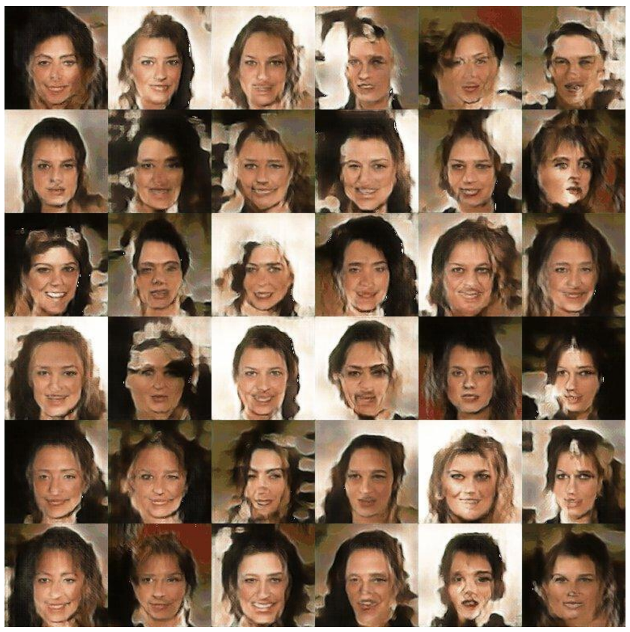


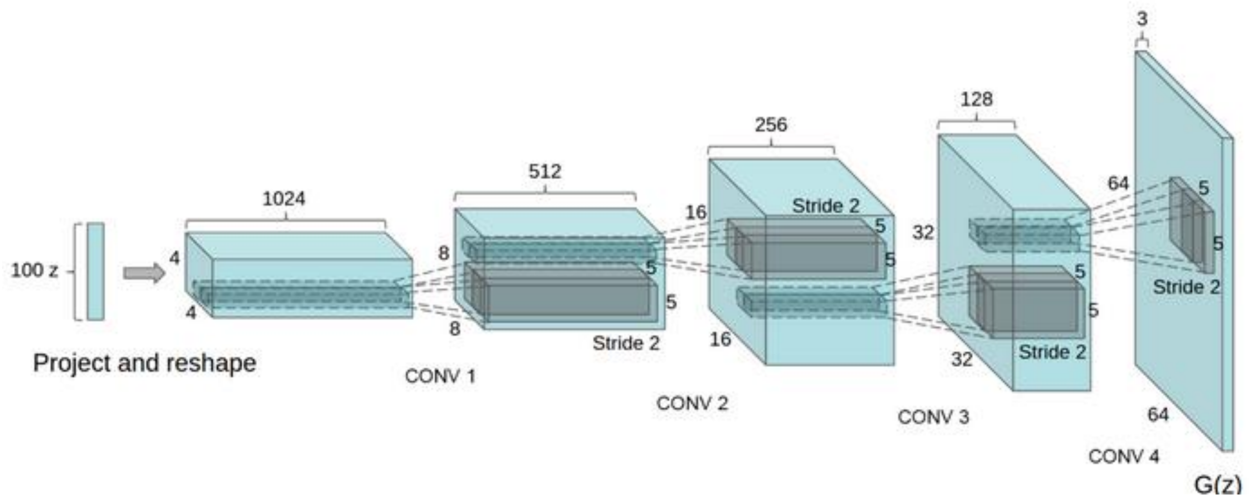
Figure 5: Adversary Losses



Conclusion and Discussion

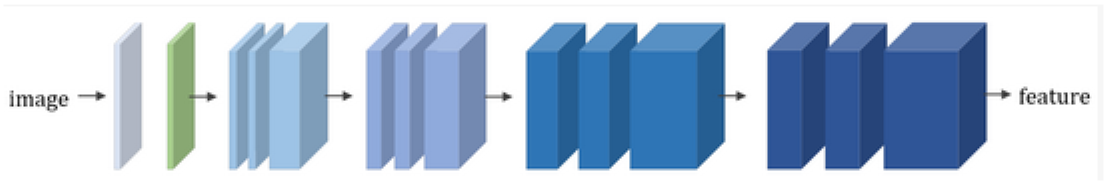
Summary

Now, in this section we will discuss our learning and the journey to find different algorithms that we can implement for the Generative Adversarial Networks. The faces dataset that is been shown here was our second implementation of our own for the GANs. We also did implement MNIST digit dataset for the learning purposes. In which we explored one of the current algorithms called DCGAN (Deep convolutional generative adversarial networks). It mainly composes of convolution layers. Max pooling and fully connected layers are not the part of it.



As you can see above we do not have max pooling layers but we have convolution stride, which you can see and confirm it from our open source code. It certainly uses the up sampling and eliminate fully connected layers. Batch normalization excepts the output layer for the generator. And finally, we use LeakyReLU or ReLU as our activation function for our generator or discriminator.

However, our second implementation of the project which is of the faces, we do not use the DCGAN. While in our learning process we learnt that we still can generate our own models and we can create our own version of it. We did explore and connected layers that stays the same sized but do not decreases. In the generator of the for this code we did start with 128 layers then we generated (256 x 3) layers and again for the last time we did (512 x 2) layers at the end. Which does follow the previous structure but there is no implementation of batch normalization. Moreover, we did implement new activation function as ‘tanh’. In the discriminator we generated the convolutional network that do not decreases.



We implemented it with (256 x 5) from start to end without any changes which exactly follows the structure above.

The idea of this project was to learn more about Generative adversarial networks or GAN. the way that this class of machine learning works and how the two neural networks we see as generative and discriminator contest with each other in a game. This technique learns to generate new data with the same statistics as the training data set. By using two different models where one of them is MNIST digital handwriting generator and the other one CelebA faces generator the main concept of GANs systems in implementation, mathematics, and strengths was analyzed.

The idea of this project was to learn more about Generative adversarial networks or GAN. the way that this class of machine learning works and how the two neural networks we see as generative and discriminator contest with each other in a game. This technique learns to generate new data with the same statistics as the training data set. By using two different models where one of them is MNIST digital handwriting generator and the other one CelebA faces generator the main concept of GANs systems in implementation, mathematics, and strengths was analyzed.

The assumption of the CelebA Faces Gans were:

1. Assume system convergence is possible
2. Assume celebA faces data set is representative of all faces
3. Assume no background artifacts
4. Assume faces should all be in same angle
5. Assume gender and race is negligible
6. Assume hair style, makeup and other accessories are negligible

From the results and discussion, it is not clear that the system will converge, so assumption 1 does not appear to be true. This is probably due to assumptions 5 and 6 in which the dataset mixes gender and jewelry. The target samples are binary for each of these factors, as in a picture will either be male or be female. There is no such thing as being half a male or half a female. The same can be said of having half an earring. However, the generator is mixing each of the male and female features as well as images with and without jewelry. In this sense, assumptions 5 and 6 are complicating the generator in such a way that it appears to be stuck in a local minimum spitting out androgenous images that are neither male nor female, and half formed jewelry. Unable to wiggle out of the local minimum, the GANs seemed to suffer the collapse problem and started to change the background color to no improvement (the gradient asks the generator to make a change but the change is just to flip colors). Interestingly enough, unlike assumption 5 and 6, 4 should be unnecessary as neural networks are great with dealing with translated or rotated shapes. The faces probably need not all be facing in the same angle. 2 and 3 are more subjective to the use case and not relevant to running the algorithm. Assumption 2 and 3 reminds the users of the GANs that the numbers the GANs will generate will be devoid of artifacts (stray marks) and will be similar to those digits inside the celebA faces library.

All in all, while the celebA faces GANs did not converge, the MNIST digital handwriting did converge. Additionally, even though the celebA face did not converge, the output was nonetheless impressive. While the generated faces were not perfect, they did resemble that of a human being. In the future, we would like to run the same simulation in 3 separate batches:

1. Male
2. Female with jewelry
3. Female without jewelry

We believe this will get the system out of the local minimum because it will avoid the scenario of having half an earring, an impossibility in real life. Gender, and jewelry are boolean values. The image is either male or female and either has jewelry or not. There are neither half females nor half earrings. Makeup, hairstyle/length and race are not boolean values but continuous and can be kept mixed.

Our project uses aspects of FCC (fully connected) GANs and DC (deep convolutional) GANs, but does not entirely meet the criteria of either. This is because our data set is finely curated which allowed us to skip some fully connected layers and normalization that would have been standard to either of those two models.

The topic of GANs is incredibly rich and the future possibilities are endless. Beyond this small project, people have created GANs projects such as: deep fakes (augmenting a celebrity image to perform the actions of a target dummy), DLSS (taking a low resolution image and automatically upscaling it), generating drug targets for cancer, jukebox.ai (using GANs to generate music,) lyrebird (using GANs to mimic voices,) and many more. This project has greatly enriched our understanding of neural network data generation and we hope to explore more GANs projects in the future.

References

[1] Bastien, F. et al., (2012), “Theano: new features and speed improvements”, in Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

[2] Yoshua Bengio (2009), "Learning Deep Architectures for AI", Foundations and Trends in Machine Learning: Vol. 2: No. 1, pp 1-127.
[Available] http://dx.doi.org/10.1561/2200000006

[3] Bengio, Yoshua and Mesnil et al., “Better Mixing via Deep Representations” in Machine Learning (cs.LG), FOS: Computer and information sciences.
[Available] https://arxiv.org/abs/1207.4404

[4] Hinton, G. E., Osindero, S., and Teh, Y. (2006), “A fast learning algorithm for deep belief nets”, Neural Computation, 18, pp. 1527–1554.

[5] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998), “Gradient-based learning applied to document Recognition”, Proceedings of the IEEE, 86(11), pp. 2278–2324.

[6] Rifai, S., Bengio, Y., Dauphin, Y., and Vincent, P. (2012), “A generative process for sampling contractive auto-encoders”, In ICML’12.

[7] Smolensky, P. (1986), “Information processing in dynamical systems: Foundations of harmony theory”, In D. E. Rumelhart and J. L. McClelland, editors, Parallel Distributed Processing, volume 1, chapter 6, pp. 194–281. MIT Press, Cambridge.

[8] Tieleman, T. (2008), “Training restricted Boltzmann machines using approximations to the likelihood gradient”, In W. W. Cohen, A. McCallum, and S. T. Roweis, editors, ICML 2008, pp. 1064–1071, ACM.

[9] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, “Extracting and composing robust features with denoising autoencoders” In ICML 2008.

[10] ounes, L. (1999), “On the convergence of Markovian stochastic algorithms with rapidly decreasing ergodicity rates”, Stochastics and Stochastic Reports, 65(3), pp. 177–228.

[11] Krizhevsky, A., Sutskever, I., and Hinton, G. (2012), “ImageNet classification with deep convolutional neural networks”, In NIPS’2012.

[12] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998), “Gradient-based learning applied to document recognition, Proceedings of the IEEE”, 86(11), pp. 2278–2324.

[13] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014), “Stochastic backpropagation and approximate inference in deep generative models”, Technical report, arXiv:1401.4082.

[14] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009), “What is the best multi-stage architecture for object recognition?”, In Proc. International Conference on Computer Vision (ICCV’09), pp. 2146–2153.

[15] Kingma, D. P. and Welling, M. (2014), “Auto-encoding variational bayes”, In Proceedings of the International Conference on Learning Representations.

[16] Krizhevsky, A. and Hinton, G. (2009), “Learning multiple layers of features from tiny images”, Technical report, University of Toronto.

[17] Glorot, X., Bordes, A., and Bengio, Y. (2011), “Deep sparse rectifier neural networks”, In
AISTATS’2011.

[18] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y., “Maxout networks”, In ICML’2013

[19] Goodfellow, I. J., Mirza, M., Courville, A., and Bengio, Y., “Multi-prediction deep Boltzmann
Machines”, In NIPS’2013.

[20] Gutmann, M. and Hyvarinen, A. (2010), “Noise-contrastive estimation: A new estimation principle for unnormalized statistical models” In AISTATS’2010.

[21] Goodfellow, I. J., “NIPS 2016 Tutorial: Generative Adversarial Networks” Technical Report, Cornell University