THE UNIVERSITY OF CHICAGO

MASTER'S THESIS ANALYSIS OF GENERATIVE ADVERSARIAL MODELS DRAFT

FOR THE DEGREE OF

MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY

STEVEN BASART

CHICAGO, ILLINOIS

MONTH 2017

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

We study Generative Adversarial Networks (GANs) and their applications to various tasks as well as the creation of a measure to analyze their performance. Regarding the study of GANs, we create a novel network that can be used for interpretability, image generation, and augmenting existing datasets. Last, we create a new measure to score sets of images as opposed to scoring individual images. The measure computes the relative divergence between the test set and generated set. This measure has several nice properties: tracks image quality, is unbiased, detects when the generator has missing modes, and captures the divergence of the generative model to a test distribution.

CHAPTER 1

INTRODUCTION

1.1 Background

Generative models have been used in such tasks as information retrieval [Wang et al., 2013], text classification [Antti, 2012], speech generation [van den Oord et al., 2016], and modeling income inequality [Pareto, 1897]. The problem of generating natural images was considered too difficult until recently. Since 2016 researchers have begun using generative models to generate plausible looking images using such methods as autoregressive models such as SPARN [Goessling and Amit, 2015] and pixelcnn [Salimans et al., 2017], probabilistic graphical models such as variational autoencoders [Kingma and Welling, 2013], and generative adversarial networks [Goodfellow et al., 2014a].

The generation of images is an important task for three main reasons: scientific, utilitarian and artistic value. The first scientific reason is that by having a good generator of natural images, one can use the generator as a prior over natural images. A good prior can be useful for Bayesian inference for image analyses. The second reason for the importance in generating images is for utilitarian purposes. There are many tasks that could benefit from generating natural images such as inpainting, colorization, sampling from rare categories, and generating artwork. The last reason is for artistic value. One can consider the task of "searching for similar images" which is currently approached as an information retrieval problem, but it could be reformulated as an generation task to become "generate similar images".

1.1.1 Objectives

The work presented in this paper covers a recent research direction of Generative Adversarial Networks or GANs. The main objective of our research involving Generative Adversarial

Networks is to explore a novel approach on utilizing the generator as a basis to query other models. Our first objective is to provide a solution to the problem that neural networks are considered uninterpretable black box models. In this way we can use a the generator of a generative adversarial network to query what examples would fool another model.

A secondary objective of GAN research is to explore how well generators can act to supplement or augment a dataset. By modeling the entire distribution it should be possible to generate samples that are harder and can help in generalization. We explore how diverse and distinct these images are.

The final objective of this work involves the creation of a novel evaluation measure for generative models. The measure aims to measure generalization error of the generator and ignores the discriminator of GAN. By formulating the measure in this way we can compare different image based generative models outside of GANs.

1.1.2 Contributions

Our contributions to GAN research are as follows:

- Creation of a novel GAN architecture. 2.5.1
- Novel approach for Neural Network interpretability. 2.5.3
- Demonstrating usefulness in augmentation of existing datasets. 2.5.4

The contributions relating to the measure are as follow:

- Measure approximates the relative divergence between generated and real distributions. 3.2
- The measure compares entire distributions rather than comparing on a per-image basis. 3.2

- The measure tracks image quality. 3.4.6
- The measure penalizes GANs that miss modes. 3.4.5

CHAPTER 2

GENERATIVE ADVERSARIAL NETWORKS

2.1 Overview

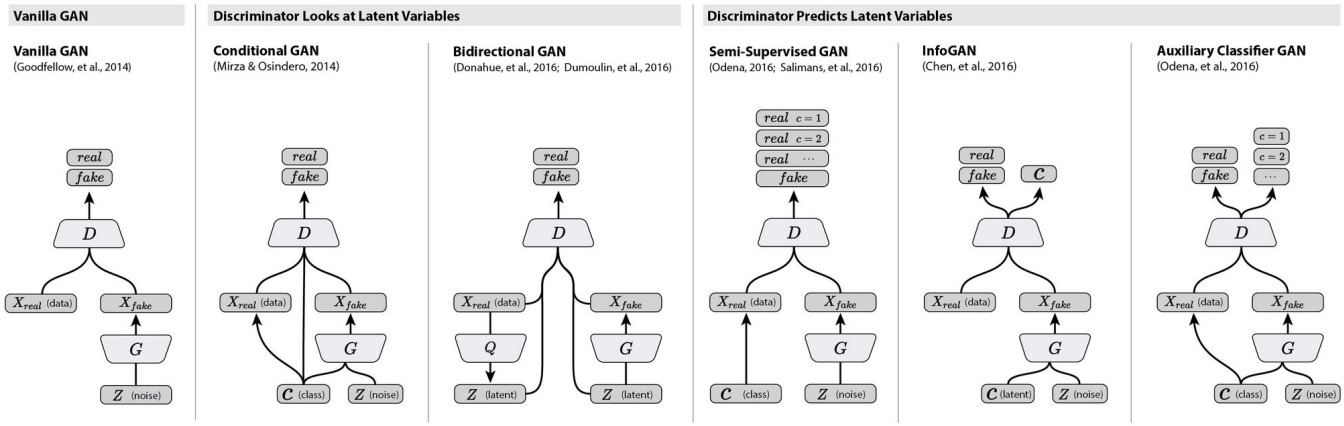


Figure 2.1: GAN variations

Generative Adversarial Networks (GANs) are a type of generative model that use neural networks and an adversarial training scheme. First proposed in Goodfellow et al. [2014b], the work displayed sharp images unlike those seen with other methods see Fig 2.10. At the core of Generative Adversarial Networks is the concept of two competing (adversarial) networks. One network the generator, termed G, takes in random noise as input and generates an image as an output. The other network, termed the discriminator D, has the task to predict the probability an image is real. Its task is therefore to discriminate between real and generated images by assigning a low score to generated or fake images while assigning a high score to real images. We can can call this a "Vanilla GAN" as it was the first proposed GAN model.

For other variations on the traditional "Vanilla GAN" see Fig 2.1. To highlight some of the differences between the models it can be roughly broken down into two other categories: whether the discriminator has access to the latent variables before making its prediction, or whether the discriminator itself predicts the latent variables. A brief discussion of each

model with the first being Conditional GAN. In this framework the GAN will condition its output on a latent variable of the target class. The discriminator will then take as input both an image and a label and then produce the same output [0, 1]. In bidirectional GAN, there is a separate network whose task is to generate a latent variable z for every input image. The discriminator will take in both z and an image and again produce an output in [0, 1]. The idea here is to learn a more informative mapping on the latent space as well. For the final three models there are a few subtle differences between them, but they all have the discriminator predict a class score along with the traditional real/fake score. The above classification of GANs focused on the discriminator; however, one can also examine the differences in the generator of GANs when doing a comparison.

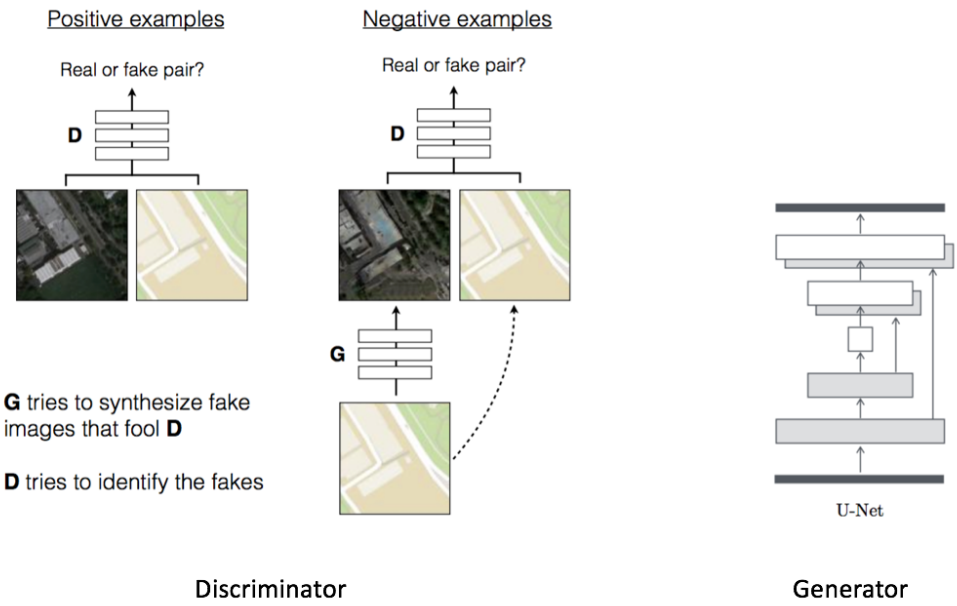


Figure 2.2: Conditional Adversarial Network

There is another somewhat confusingly named variant called a conditional generative adversarial network or CoGAN presented in [Isola et al., 2017] and shown in Fig 2.2. It is confusing because there are several works that use the title CoGAN and yet are quite

different such as those in Isola et al. [2017], Liu and Tuzel [2016], Mirza and Osindero [2014]. The main difference between a CoGAN and a "Vanilla GAN" is that the generator network G in a CoGAN will also take in as input an image to condition its output on. Unlike a "Vanilla GAN" the generator of a CoGAN does not take a noise variable z. The main focus for the remainder of this chapter will be on the CoGAN model. While we will refer to the model as GAN it will be understood that it is specifically a conditional generative adversarial model rather than a plain generative adversarial model.

There are a few reasons to use the formulation of a CoGAN instead of GAN. As discussed by Isola et al. [2017] there are many problems that can be formulated as an image to image mapping problem such as colorizing images, coloring sketches, and image segmentation. More generally the task can be seen as learning a map between two distributions which the general task can be suited to many problems outside of images.

2.2 Loss

Let us now define some notation used hereafter. Discriminator, D, corresponds to the neural networks where the input is an image and the output is in range [-1,1]. The generator, G, takes as input a noise vector and outputs an image. For the conditional generative adversarial network, the generator, G, can take as inputs an image and a target class variable y_t. Input image x is a real image drawn from p_{data}, and label y is the corresponding label of image x. Let y_t be a target label of the same list of classes as y but not necessarily equal to y. A noise vector z, is drawn from p_z which is typically standard multivariate Gaussian. Finally, let σ represent the logistic function.

The traditional GAN loss is

$$\min_G \max_D = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))] \tag{2.1}$$

At optimality the output of the generator would be equivalent to real data and the output of the discriminator would always be equal to 0.5 as it can no longer discriminate between real and generated (or fake) input. The optimality in practice is never reached.

We can summarize the loss we use to train our model as follows

$$L_{\text{GAN}} = L_G + L_D \tag{2.2}$$

$$L_G = \mathbb{E}_x[-\log(\sigma(D(G(y_t, x))))] \tag{2.3}$$

$$L_D = \mathbb{E}_x[-\log(\sigma(D(x))) - \log(1 - \sigma(D(G(y_t, x))))] \tag{2.4}$$

Our first loss is the typical GAN loss. Something of note here is that L_D sometimes has an alternative form $L_D = -\log(\sigma(D(x))) + \log(\sigma(D(G(y_t, x))))$. We found that the alternative form tended to be much less stable for our experiments. We present it as the sum of two distinct losses because the losses as originally presented in equation 2.1 may indicate that the loss can be optimized directly. In practice though the generator and discriminator objectives are optimized as two separate steps.

$$L_{label} = \mathbb{E}_x[-y \log(\text{softmax}(D(x)) - y_t \log(\text{softmax}(D(G(y_t, x))))] \tag{2.5}$$

The loss above that we employ is the standard cross entropy loss. Unlike many other GAN papers such as Salimans et al. [2016] we wish to encourage the discriminator to classify the real images x with true label y. At the same time we also encourage D to classify generated images $G(y_t, x)$ with the target label y_t. Next we define our modification penalty

$$L_1 = \mathbb{E}_x[\|x - G(y_t, x)\|_1] \tag{2.6}$$

This last loss is an ℓ_1 penalty between the generated image and the image it is conditioned on. In this way we try to force that the changes applied to an image remain minimal. Now, we have our final combined loss

$$L = L_{\text{GAN}} + L_{\text{label}} + \lambda L_1 \tag{2.7}$$

By combining all three losses, we can train our model to produce natural fooling examples. We add a hyper-parameter λ to the L_1 loss to control the strength of preservation over class adjustment.

2.2.1 f-Divergences

This section is a comment on the loss function that is typically used and about our own loss function that we used. The typical loss function used in GAN literature derived a relation of the Jenson Shannon Divergence (JSD) to the Adversarial Loss however the Adversarial loss is then modified from

$$-\log(1 - D(G(x))) \tag{2.8}$$

to

$$\log(D(G(x))) \tag{2.9}$$

The authors of paper Goodfellow et al. [2014b] lose any connection to JSD by performing this modification. This new loss function has no known theoretical implications.

Due to this new loss function having no known divergence counterpart another work Nowozin et al. [2016] has shown that the specific divergence chosen in Goodfellow et al. [2014b] is not necessary to make GANs work. In their work they test out a variety of divergences and term their GAN, f-GAN showing that the family of f-divergences can all work to train a GAN. Similarly there have been other loss functions applied such as the Wasserstein distance.

In GAN literature many of the comparisons are using "simple" datasets such as LSUN and MNIST. LSUN is the large scale scene understanding dataset, however when used in the GAN literature it is typically restricted to only include bedrooms. MNIST and other

related datasets are covered in section 2.4.3. Ultimately though there is no theoretical or even practical best loss function as each paper seems to use a slight modification.

This lack of a best function is highlighted in the literature. Papers either delay showing results or fail to show any comparative results and instead highlight their own cherry-picked images. In our own experiments we have found that loss function does affect the time to convergence and even the final qualitative outputs.

2.3 Architectural Variants

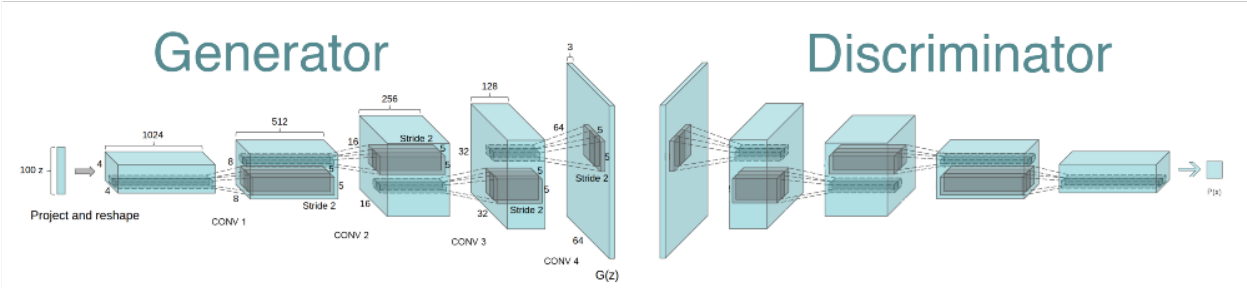


Figure 2.3: Typical GAN Architecture
This particular GAN architecture is from DCGAN, however, it is reflective of many other GAN architectures used. Note that while in this particular model the generator and discriminator are symmetric, this is not always the case.

The architectural space that has been explored by the research community relating to GAN research is extensive. The majority of the architectural variants that have won the ImageNet [Russakovsky et al., 2014] challenge have been explored and utilized in GANs to varying degrees of success. The first GAN architecture used only fully connected layers. The subsequent architectures utilized a fully convolutional architecture as shown in Fig 2.3. There have been several attempts to use more powerful architectures such as ResNet or others but most of the results thus far have been poor.

For our experiments we use a modification of the U-Net architecture (also presented in some literature as V-Net) shown in its original implementation in Fig 2.4. Our architectural modifications are described in more detail in section 2.5.1. The U-Net architecture was first

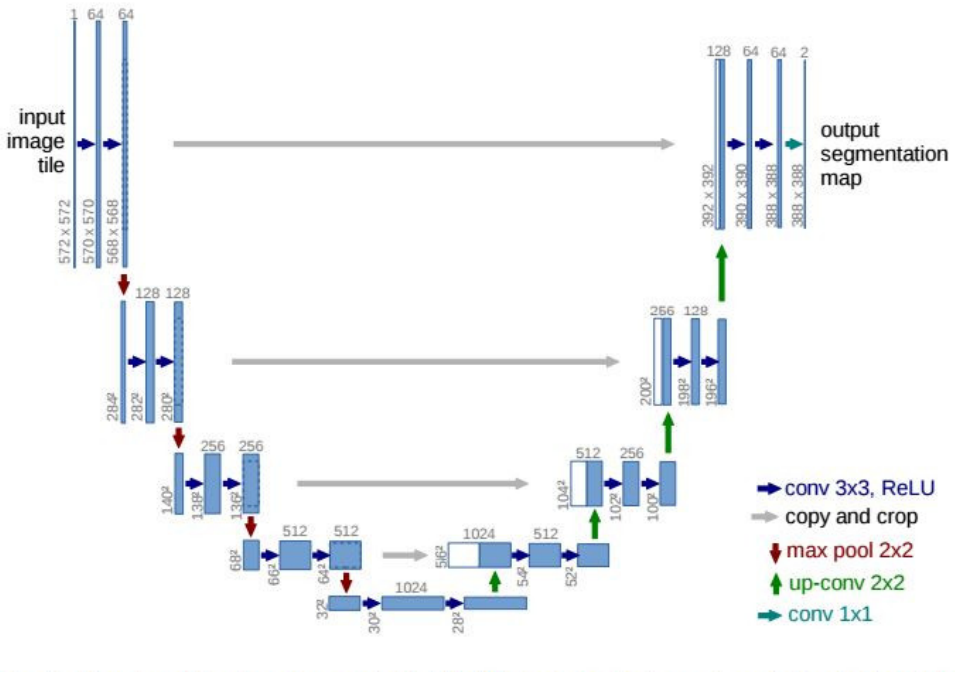


Figure 2.4: U-net architecture
This architecture was originally used for breast tumor detection. It is presented here in its original form. Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

used for GAN research in Isola et al. [2017]. The symmetric shape of the U-Net architecture is constructed in such a way that allows features from earlier layers to get concatenated onto a later layer in the network. Examples of using the U-Net as a generator in be seen in Fig 2.8.

2.4 Training

An important question that remains is how to train a "Vanilla GAN". The Discriminator is trained like a binary classifier where the positive labels correspond to real images and the negative labels correspond to the generated images. The Generator is trained as an adversary to the discriminator. It is trained to maximize the loss of the Discriminator and is

updated by back-propagating the loss through the discriminator network (while holding the discriminator fixed) to the generator network. The loss function is covered in more detail in 2.2.

There are a few differences in training a CoGAN as compared to a GAN. The main difference is that the loss function changes by the addition of an $L1$ norm penalty between the generated image and the target image. The additional loss term is sometimes weighted.

2.4.1 Training Heuristics

In addition to what was mentioned in section 2.4, the first GAN paper [Goodfellow et al., 2014a] employed a differential or varied update schedule between the discriminator and that of the generator. The generator is updated twice as often as the discriminator. For every update step to the discriminator, it will be frozen for the next two while only the generator is updated. There are a few commonly held intuitions behind this heuristic. One is that the discriminator has an easier time learning to discriminate. Another is that the generator has a harder time learning to generate a new distribution that can fool the discriminator. The above training scheme more often than not ends up with a collapsed model that produces output akin to white noise. Several heuristics employed to deal with model collapse described below.

Other heuristics employed also appear to target the discriminator by making it weaker. A heuristic termed label flipping adds noise to the positive target labels. This asymmetric targeting tries to smooth the two distributions that the discriminator learns. Similarly another technique adds noise directly to the real and generated inputs fed into the discriminator. The level of noise is dropped on an exponentially decaying schedule [Salimans et al., 2016].

Inspired by these techniques we tested freezing a layer or several layers of the discriminator. This acts as a stronger corruption by transforming the intermediate representation via a nonlinear function as opposed to an linear function presented earlier. We found this tech-

nique to outperform adding noise or label flipping and can be augmented with label flipping. This leads to hypothesis that weak discriminators are optimal at least in the beginning of training to make GANs work. It still does not elucidate how to improve GAN training.

The success or failure of a particular heuristic is still hard to determine apriori. However a common theme among the heuristics is that they all aim to weaken the discriminator.

2.4.2 GAN Issues

Section 2.4.1 highlighted some of the issues dealing with model collapse for GANs. When the generator stops learning or only produces one output. However there are other issues that are encountered when training GANs. One of the issues with the training of GANs that a lot work in the literature has focused on is how to stabilize the training.

Finally there is the issue of having a meaningful loss function. Currently the loss function can at best be a reflection as to how well the generator is "fooling" or the discriminator is successfully discriminating. Performing at an optimum for this loss function does not necessarily correspond to realistic generations. The optimum itself is also typically not reached when training GANs.

2.4.3 Datasets

The datasets used for the experiments presented in this paper include: MNIST, SVHN, CIFAR, Char74k.

MNIST. The MNIST dataset consists of black and white handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples.

SVHN. The Street View House Numbers (SVHN) dataset is of cropped 32 by 32 digits found through Google Street View. There are 73,257 images in the training set, 26,032 images in the test set, and 531,131 images for additional training.

CIFAR. The two CIFAR datasets consist of colored natural scene images, with 32x32

pixels each. CIFAR-10 consists of images drawn from 10 and CIFAR-100 consists of images drawn from 100 classes. For both datasets there are 50,000 training images and 10,000 test images.

Char74k. We used the subset of English characters that consists of 12,503 images. We randomly shuffled the dataset and split it into 10503 for training and 2000 for testing. Even on a dataset with this few examples per class ModNet is able to generate compelling examples.

2.5 Contributions

While in some previous sections we have highlighted a few of the contributions, in this section we will directly speak to some of the more prominent contributions of our recent work. We will discuss the new architecture, new method for interpretability, results in semi-supervision and memorization.

2.5.1 Architecture

We call our network the Modification Network or ModNet for short. Our generator is a modified U-Net architecture. Previously it was used as a generator in Isola et al. [2017] to produce an image but in our work we modified the output to be that of an image that would be added point-wise to the input image. We call this image a "delta image" as it acts to highlight or mask out features present in the original image. After the point-wise addition we either clip or renormalize and shift to bound the values into range [-1,1]. We direct the label of the generated image by conditioning the generator on a one-hot vector y_t that is fed in along with the original image. This allows us to determine which class we want the input image to change to. An example is shown in Fig 2.5.

ModNet differs from its sibling network presented in the work of Isola et al. [2017] in three key ways. In utilizing the UNet architecture to serve as their generator the network has to

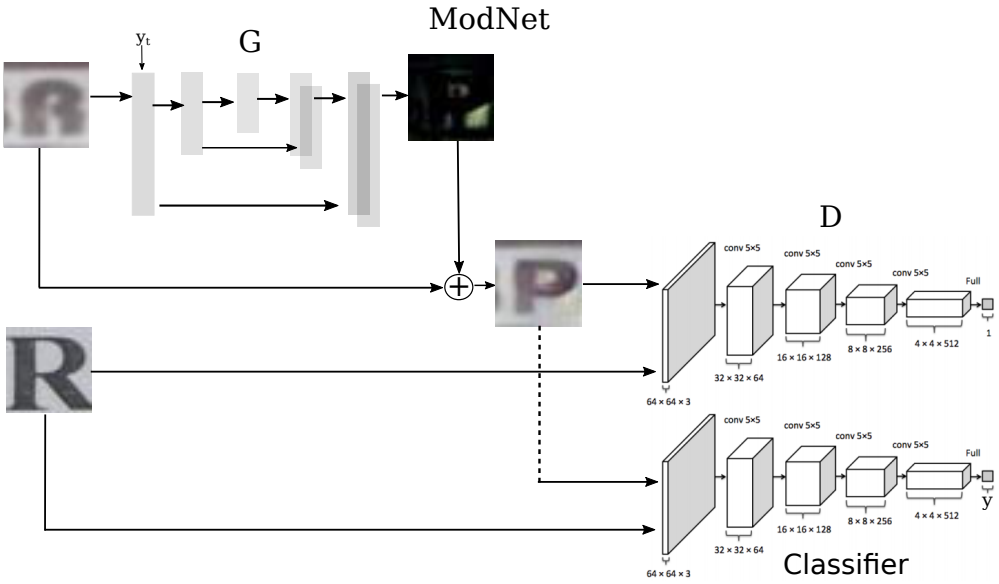


Figure 2.5: ModNet
One variant of the proposed architecture where the classifier and discriminator are two separate networks. The Generator in the top left of the figure takes in both an input image and a target label y_t to then produce a delta image to be added point-wise to to the input image. The final output after the addition is passed through a tanh. The inputs to the discriminator are normalized to be in the range [-1,1]. The classifier is tasked to predict the true label for real image and the target label for a generated image.

learn the complex task of generating images. However, because the network is conditioned on natural images its task is easier than a "Vanilla GAN" which map from the space of a standard Gaussian to natural images. The generator in ModNet could be argued to have an even easier task. The task has changed from generating an entire image to generating the deltas required to be applied to the original image. This now changes the focus to learning to change relevant portions of an image and it can learn to ignore generating backgrounds.

Another modification from Isola et al. [2017] is that now the final nonlinear activation in the generator has to be changed from a sigmoid unit to a tanh. The reasoning behind this change is that if the deltas are bounded in [0,1] instead of [-1,1] the network is constrained to only learn to add to the image. When testing under this setting we observed poor results mostly just adding a bright filter or white mask over areas instead of learning a proper transformation.

Lastly by adding the additional classification loss, see equation 2.5, we now can create two variants of our architecture. One in which the discriminator acts as both discriminator and classifier as seen in Odena et al. [2016]. The other new variant is where we can split our classifier from our discriminator this allows for two different gradient signals to be received by the generator see Fig 2.5. There exists the option of having the classifier to be pretrained, and optionally frozen.

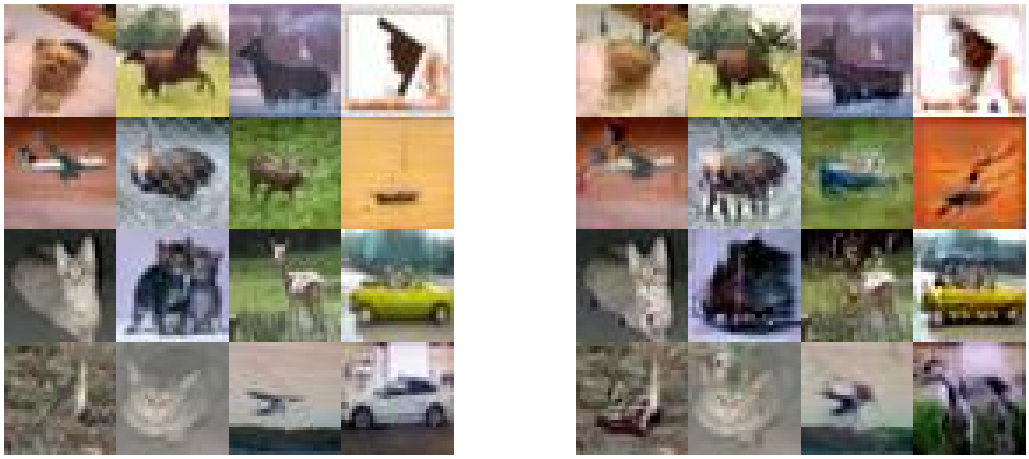
2.5.2 Training Procedure

Based on the use case of our model there two distinct training procedures that could be employed. The two use cases are for interpreting a pretrained model or for augmentation. The training for the latter follows the training procedure described in section 2.4. The training for interpretability requires that the discriminator be separate from the classifier. The classifier is never updated during training but its error signal is used to update the the generator. See training algorithm for more details.

2.5.3 Interpretability

In this section we highlight how with this new approach we can better understand what neural networks are learning. This can be used to attempt to debug, or make them better. To run the interpretability experiments we needed to use the variant of our architecture where the classifier and discriminator are separate from one another. In this way our goal is to generate images with a high classification score that fool a classifier.

For Fig 2.6 we show what happens when you take a shallow classifier and use our method to fool it. We are able to show some small and also large image changes that will lead to a high classification score.



(a) Original CIFAR-10 images. (b) Generated images with corrected label

Figure 2.6: Interpretability samples
A sample of CIFAR-10 images that fool a weak classifier. We can see that the model learns to associate wheels with trucks by putting them on a cat and switching the label. Target labels of generated images are as follows. 1st row: Deer, Deer, Truck, Cat. 2nd row: Cat, Horse, Ship, Plane. 3rd row: Truck, Truck, Deer, Truck. 4th row: Car, Dog, Bird, Horse.

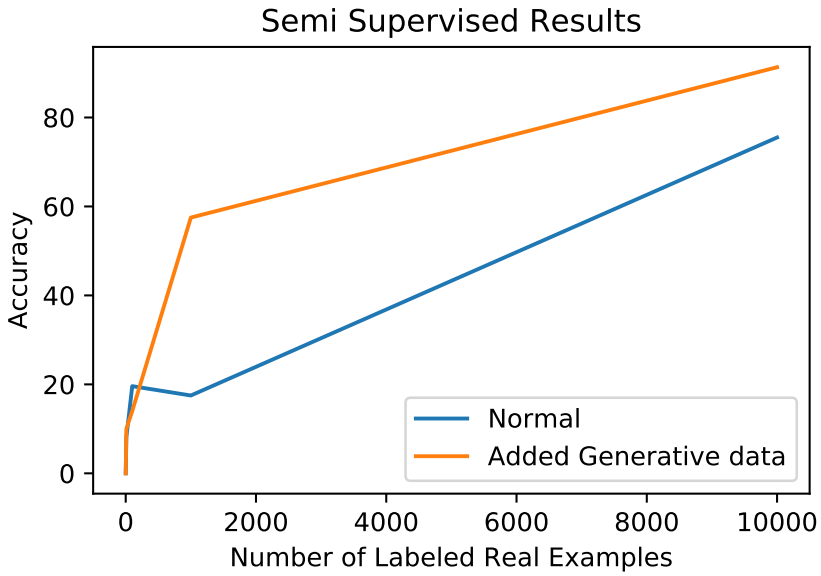


Figure 2.7: Semi-supervised
Semi-supervised results on SVHN. The figure is generated from four query points.

2.5.4 Semi-Supervision

Previous work from Salimans et al. [2016] used GANs as a tool to expand a small dataset. They created several thousand images from a GAN and used a fraction of the training data

to train a new classifier. They reported the total error as in the total amount of mistakes made on a dataset as opposed to percent error as we suspect it did not perform that well compared to other data augmentation methods.

We were inspired by their idea to use a generator to create extra images that can be used for semi supervised learning. We ran two different semi supervised experiments. The first experiment involves creating an entirely generated conditional dataset. From this we train a new classifier and measure its performance. Using only generated images under the condition that the target label not equal the source label we can achieve 88%. This result is rather low accuracy compared to state of the art accuracy achievable on the regular dataset. This implies that using the generator causes some of the features in the training data to be removed and not allow for as great generalization as the original dataset.

The second experiment involves taking a subset of the original training data and using only those images as the conditional images to generate the other labels in the dataset. We experimented with [10, 100, 1,000, 10,000] real examples and for the added generated data we generated the other nine labels possible for each example. The added generated data then has [100, 1000, 10,000, 100,000] images in each respective data point (it corresponds to the sum of the real and generated). Then a new classifier is trained on both the generated and real subset of the images and is compared to a classifier trained on only real data. The results for this experiment are shown in Fig 2.7.

2.5.5 Memorization

Lastly we hypothesize that the traditional GANs might be memorizing the training data, we wanted to test if this was also the case for the CoGAN. This hypothesis was prompted by the fact that some work utilize a per pixel l2 loss during GAN training. This loss encourages the generator to memorize the training images as opposed to learning the entire distribution.

To test our model we computed the nearest neighbors of all of the training images using

ℓ_2 distance computed on pixels and looked up the nearest neighbor of all the generated images against training images. We observed that all of the generated images were always closest to image they were conditioned on rather than any other image. This led us to the conclusion that our training scheme was not retrieving another image within the training data that was of the target label. We have yet to run this experiment on other GANs to determine if they are memorizing the training data.

2.6 Related Work

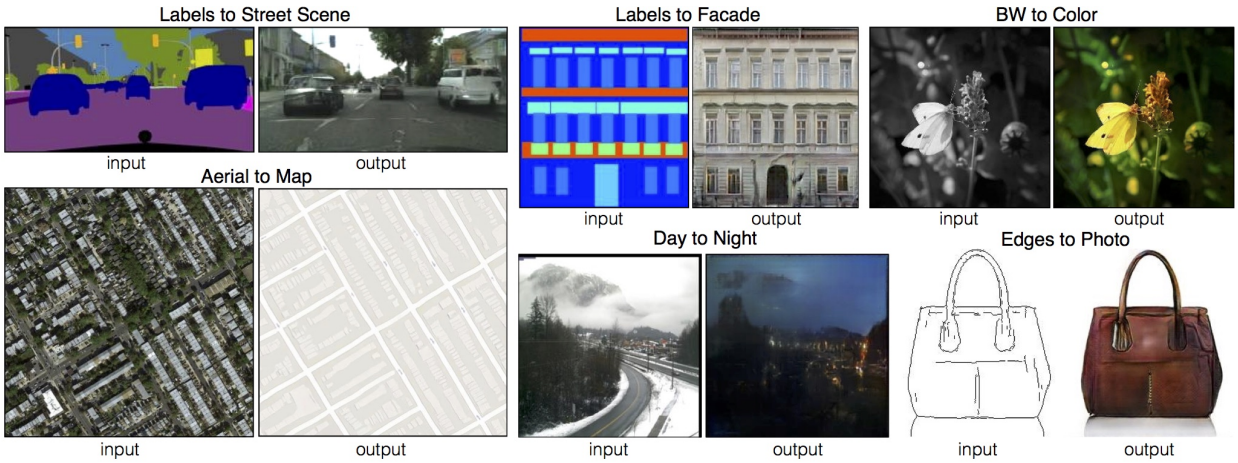


Figure 2.8: Pix2Pix Example

The work most similar to ours is that of Isola et al. [2017]. The task is given paired image datasets, to generate an image conditioned on the input image. The problem in image-to-image mapping is a supervised problem where for each image there is a corresponding image that is its ground truth. The task of the generator is to generate the corresponding image pair conditioned on the input image. The task of the discriminator is to evaluate an image pair as being real or generated. The loss function they employed was the traditional GAN loss plus the ℓ_1 penalty between the generated image and its corresponding paired image. Examples of the various tasks they employed is shown in Figure 2.8.

While GANs have become a popular generative model there are other generative models

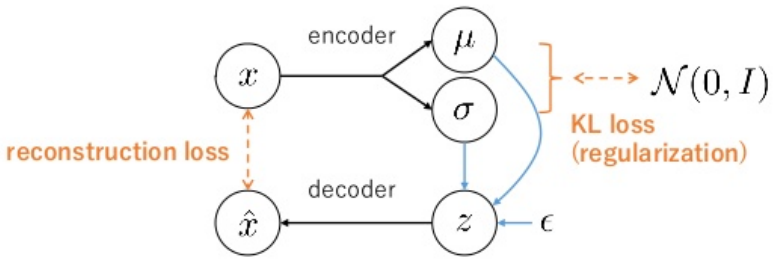


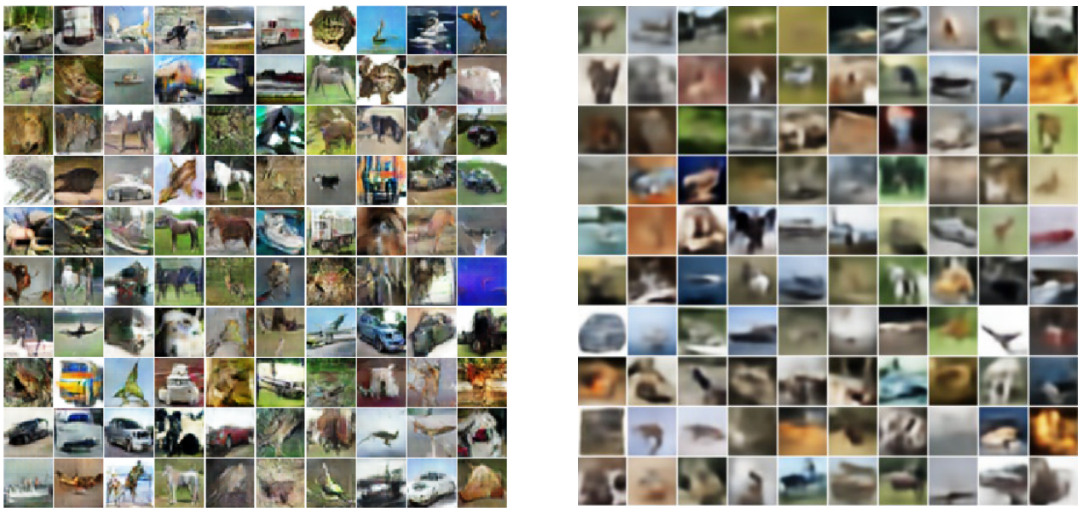
Figure 2.9: VAE Diagram
Diagram exhibiting the typical structure of a Variational Autoencoder.

that exist. Variation Autoencoders (VAEs) are neural network models that approximate some automorphism. The main difference between an autoencoder and a variational autoencoder is that the bottleneck is of the form of two vectors where the first vector models the mean and the second vector models the covariance of a multidimensional Gaussian. The reasoning behind generating these two vectors is we want to encode our input as a point on some multivariate Gaussian. To enforce this encoding resembles a multivariate Gaussian, there is the additional loss of the form $-KL((\mu, \Sigma), \mathcal{N}(0, 1))$. The additional loss term is a penalty for how far the means and variances differ from the standard multidimensional Gaussian and in practice it is typically scaled by some lambda factor.

To train the variational autoencoder, the re-parameterization trick is utilized where you take ϵ which is sampled from a standard Gaussian and multiply it by the generated σ and add it to generated μ from the encoder. To convert the output z into a real image typically deconvolution is employed similar in style to the generator in GANs.

The main benefit of this model is that once trained it becomes easy and efficient to sample from the model. The generation of the image happens by initializing the bottleneck with parameters sampled from a multivariate Gaussian and running the model beginning from the bottleneck layer through the rest of the network.

Another recent generative model is known as Pixel CNN [Salimans et al., 2017]. There have been a few variants of this model such as Pixel RNN and Pixel CNN++. The model



(a) Samples from a GAN (b) Samples from a VAE

Figure 2.10: GAN vs VAE samples

predicts the next pixel given the previous context which are all of the previous pixels in raster order. As a generative model one just needs to initialize the first pixel the model sees and then it can be used to generate all of the other pixels in raster order. It has had its greatest success as a means for compression.

Other autoregressive models that predict in pixel order are MADE from Germain et al. [2015] and NADE from Uria et al. [2016]. The former model modifies an autoencoder architecture to apply binary masks to the weights (effectively a binary dropout) to ensure that each pixel prediction does not depend on another it is conditioned on. This ensures valid log-likelihoods can be computed from the model. The latter model NADE computes the log-likelihoods directly from the inputs by conditioning each neuron to only examine all future pixels. This also works to ensure that the likelihood can be broken into its conditional probabilities. Both works were evaluated on the MNIST dataset.

Other work from Goessling [2016] utilize Gaussian mixtures to learn a distribution over images. The work shows some improvements over the other autoregressive models namely MADE and NADE in some datasets and inferior performance in other datasets such as MNIST. Overall the work appears to be comparable in terms of performance.

Training Algorithm: We typically set the batch size, m, to be the maximum that could fit on a GPU which is on the order of hundreds for small images and ~ 20 for large images. For the values of j, and k we used either (1,2) or (1,1). Both achieved similar results. Finally λ is in the range of $[10^{-3}, 10^{-5}]$. The value varied depending on the dataset but we tried to keep the ℓ_1 penalty to be on the same order as the other two losses. To remove the hand tuning of λ we also used the average ℓ_1 penalty which achieved similar results.

for *number of training iterations* **do**

for *j steps* **do**

- Let V be the set of m examples $\{(x^1, y^1),(x^2, y^2) \ldots (x^m, y^m)\}$ sampled from the data distribution.
- Let U be the union of the set of m examples $\{x_t^1, x_t^2, \ldots x_t^m\}$ from the data distribution and the set m examples $\{y_t^1, y_t^2 \ldots y_t^m\}$ from the data distribution. Note that we use t in x_t and y_t to denote different set of examples.
- Update the discriminator by ascending its stochastic gradient.

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [-\log(\sigma(D(x^i))) - \log(1 - \sigma(D(G(y_t^i, x_t^i))))$$
$$- y^i \log(\text{softmax}(D(x^i)) - y_t^i \log(\text{softmax}(D(G(y_t^i, x_t^i))))]$$

end

for *k steps* **do**

- Let U be the union of the set of m examples $\{x_t^1, x_t^2, \ldots x_t^m\}$ from the data distribution and the set m examples $\{y_t^1, y_t^2 \ldots y_t^m\}$ from the data distribution.
- Update the generator by descending its stochastic gradient.

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} [-\log(1 - \sigma(D(G(y_t^i, x_t^i))))$$
$$- y_t^i \log(\text{softmax}(D(G(y_t^i, x_t^i))))$$
$$- \|x_t^i - G(y_t^i, x_t^i)\|_1]$$

end

The gradient-based updates can use any standard gradient-based learning rule. We used ADAM[Kingma and Ba, 2014] in our experiments.

end

CHAPTER 3

EVALUATION OF GENERATIVE MODELS

In this section we address the concerns about memorization and a lack of any evaluation metrics or measures. We define the purpose of a GAN as the creation of a generative model. This statement is important because in the literature there are various different goals that papers propose for GANs including as a pre-training scheme or for creating a good prior. Given this purpose we set out to create an evaluation measure that captures how well the generator approximates the true distribution.

3.1 Background

3.1.1 Generative Adversarial Networks

The specific GANs we consider in this chapter are that of DCGAN Radford et al. [2016], Improved GAN Salimans et al. [2016], and Improved Wasserstein GAN Gulrajani et al. [2017]. The following descriptions are oversimplification of the changes from the initial GAN architecture as developed in Goodfellow et al. [2014a]. The DCGAN is an architectural modification in that they replaced the multilayer perceptron with a convolutional neural network (CNN). The Improved GAN builds from DCGAN by using a CNN and modifies the loss by penalizing the difference between layers of generator and discriminator. Finally the Improved Wasserstein GAN changes the loss to $\mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}[D(G(z))]$ and applies a penalty to the gradients so that the CNN approximates a L_1 Lipschitz function.

3.1.2 Previous Measures

Parzen Window Estimation. Parzen [1962] uses a kernel centered around each data point to compute the density function of the data. It belongs to a family of kernel methods that estimate the log likelihood of the data. The problem with this family of methods is that

they fail to properly track image quality and can be manipulated to achieve a desired result. Moreover Theis et al. [2015] demonstrate how image quality provides no information about likelihood and vice-versa. In addition the estimates from the methods can be quite far from the true likelihoods. We also demonstrate how Parzen Windows fails to track image quality in section 3.4.1.

Annealed Importance Sampling. Wu et al. [2016] follows in a similar vein by trying to estimate the log-likelihood of the data. To estimate the log-likelihood in AIS, they consider the geometric mean of many intermediate distributions between the prior, assumed to be Gaussian with respect to the decoder model, and the target, the output of the decoder. However, in GANs there is no restriction or implicit constraint to its target distribution to be that of a Gaussian. The Gaussian prior assumption is met within variational autoencoders (VAE) by explicitly penalizing the model from deviating from a Gaussian code. This loss in VAEs explains why they score best using AIS as compared to GANs.

Still the main drawback to AIS was explained by Theis et al. [2015] who show that log-likelihood based estimates are a poor measure. We further demonstrate how even within a fixed model the estimates can vary by many nats by changing the variance of the Gaussian prior in Figure 3.5.

The Inception Score. Salimans et al. [2016] uses the inception model which is trained on Russakovsky et al. [2015] and takes the form: $\exp(\mathbb{E}_x \left[\mathrm{KL}(p(y|x)\|p(y))\right])$. This measure might arguably be the most popular measure currently used to measure quality of a GAN. This measure was shown to correlate well with image quality assessment by Mechanical Turkers and also promotes diversity of samples.

There are several faults with the Inception Score namely: its internal bias, its discouragement of intraclass diversity, and its failure to capture repeated modes. The first point attests

to how the model used in the Inception Score is biased to images most similar to those found in ImageNet. This is due to its training procedure and therefore makes the model unreliable for images that differ significantly from those found in ImageNet. There exist many natural datasets which differ significantly from ImageNet, such as SVHN Goodfellow et al. [2013], satellite imagery Mnih and Hinton [2010], and NORB LeCun et al. [2004]. This fact alone concerning the model's bias would not represent an issue, but the score does not take into account the target distribution, so the score is only ever relative to the generated distribution. The second point can be observed by examining what increases the score most. The score is maximized given images that activate a particular ImageNet class. Therefore the measure promotes activation of many ImageNet classes as opposed to activating a single or small number of classes, which can be regarded as promoting interclass variation. Having a large intraclass variation i.e. many distinct samples of one Imagenet class would achieve a low score. The last point we demonstrate experimentally in section 3.4.5, but can also be observed as well in the formulation of the score. Where the score penalizes for only containing a small number of modes by way of the $p(y)$ being close to $p(y|x)$ if there are only a handful of modes. However once the number of modes exceeds some threshold, $p(y)$ will approach the maximum entropy for a given dataset and then the measure itself becomes saturated.

A measure that we do not consider in this paper is classifier two sample test Lopez-Paz and Oquab [2017]. The method involves training a neural network to classify images as being real or fake, and then computing the average amount that it gets fooled to serve as the statistic. This method is akin to training a discriminator during the training of GANs. By relying on a neural network to classify in versus out of distribution there are no guarantees it will produce any meaningful ordering for two models which are both in or both out of distribution. As well the model would fail to detect the difference between an autoencoder and a generative model.

Figure 3.1: CIFAR-10 samples
CIFAR-10 samples from Improved GAN without Batch Normalization and the normal Improved GAN. Images are randomly sampled during the end of training.

3.2 Measure

In this paper we aim to create a measure that is optimal if the distribution learned by the generator matches the true distribution. Since the true distribution is unknown to us, we approximate the true distribution with a set of finite samples S drawn from the true distribution. Given S we can split the samples into a held out test set T and a training set R. Similar to other computer vision tasks such as classification we would like to measure how well the generator performs by evaluating the distance of the generated distribution from the test distribution.

One approach to approximating images as distributions to consider each image as a delta function around which some mass exists. This creates a point-wise mass distribution that can be used in common for each distribution. Even with this representation though computing the actual probability distribution of space is still intractable. From here we further relax the distributions to be modeled as a Gaussian mixture. Computing the diverge of two Gaussian mixtures is costly in their general form but we can approximate them with a variational approximation which will provide us with a lower bound on the true divergence Goldberger et al. [2003].

We arrive at our variational approximation by simplifying Goldberger et al. [2003] work in approximating Gaussian mixtures. The following derivation begins from said work. We will use π and ω to correspond to the mixing weights of the Gaussian mixture. We use the

subscripts a and b to refer to the index of the mixing weight for each Gaussian mixture. We will use a to refer to the GMM f and b to refer to GMM g.

The likelihood $L_f(g) = \mathbb{E}_{f(x)}[\log g(x)]$ relates to the KL divergence by $\mathrm{KL}[f\|g] = L_f(f) - L_f(g)$. Therefore any estimate of likelihood can be related back to the KL. In the following we will introduce a variational parameter $\phi_{b|a}$ which will have the following constraints: $\phi_{b|a} > 0$ and $\sum_b \phi_{b|a} = 1$.

By Jensen's inequality we have

$$\begin{aligned} L_f(g) &= \mathbb{E}_{f(x)} \log\, g(x) \\ &= \mathbb{E}_{f(x)} \log \sum\, \omega_b g_b(x) \\ &= \mathbb{E}_{f(x)} \log \sum\, \phi_{b|a} \frac{\omega_b g_b(x)}{\phi_{b|a}} \\ &\geq \mathbb{E}_{f(x)} \sum\, \phi_{b|a} \log \frac{\omega_b g_b(x)}{\phi_{b|a}} \\ &= \mathcal{L}_f(g, \phi) \end{aligned} \tag{3.1}$$

This being a lower bound on $L_f(g)$, we get the best bound by maximizing $\mathcal{L}_f(g, \phi)$ with respect to ϕ. We leave the details of the maximization to Goldberger et al. [2003].

$$\hat{\phi}_{b|a} = \frac{\omega_b e^{-\mathrm{KL}(f_a||g_b)}}{\sum_{b'} \omega_{b'} e^{-\mathrm{KL}(f_a||g_{b'})}} \tag{3.2}$$

A similar bound can be achieved for $L_f(f)$. Finally we define the variational approximation by substituting ϕ into $L_f(g)$ and the corresponding ψ into $\mathcal{L}_f(f)$.

$$\mathrm{KL}(f\|g) = \sum_a \pi_a \log \frac{\sum_{a'} \pi_{a'} e^{-\mathrm{KL}(f_a\|f_{a'})}}{\sum_{b'} \omega_{b'} e^{-\mathrm{KL}(f_a\|g_b)}} \tag{3.3}$$

In our specific case we can simplify the variational approximation to what's below due to some assumptions. Namely we assume an equal weighting of each Gaussian in the mixture. Thus π and ω become a normalization parameter equal to the number of examples in each

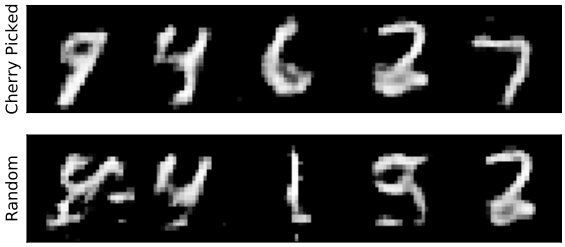


Figure 3.2: Cherry-picked results
An example GAN on trained on MNIST showing that cherry-picked examples accrued from a GAN can be deceptive.

respective mixture. We can also ignore the numerator as in our case it is a constant, only adding a linear shift to our results. As a reminder for the following we use G as the generated set $|G|$ as the number of samples in the generated set, and g as a sample from G. Similarly T is the test set, $|T|$ the number of examples, and t a sample from T.

$$\sum_{t \in T} -\frac{1}{|T|} \log \left[\frac{1}{|G|} \sum_{g \in G} e^{-\mathrm{D}(t,g)} \right] \tag{3.4}$$

Lastly there were a few popular choices for the approximations for the divergence, D. The first is to approximate each GMM as a single multivariate Gaussian and then compute the divergence of two multivariate Gaussians which has a closed form solution. The second choice and the one we opted for is to consider D to be the pairwise distances between each Gaussian mixture center. We found this approach to work better and led to less of a difference in conceptualization from our original assumptions to the final approximation.

Given this final formulation we can observe a few nice properties of the measure. It can achieve its optimal performance when the generator matches the test distribution.

An important note about the variational approximation is that it provides a lower bound for KL, however, the restriction of positivity is lost. Therefore the results presented in the paper are negative despite being presented as the KL.

3.2.1 Image Embedding

In the previous section we derived the Distribution Divergence Measure as a lower bound of the KL Divergence between two Gaussian Mixtures. It still leaves open the question of which embedding to use when computing the measure, and if an embedding is even necessary.

The main reason to have the measure computed on an embedding rather than on the pixels themselves is twofold. The first reason is that by considering an embedding space that is much smaller than the original space we can avoid the curse of dimensionality. This also works to ensure long term use of the measure as images are increasingly growing in resolution so any measure that works directly in pixel space will increase its computation time. The second reason to utilize an embedding is that it can compress images into the core components of what is desired. There are many factors such as slight changes in blur, brightness, or contrast that all allow an image to remain natural so the embedding should be immune to slight variations of these modifications.

These requirements lead us to a consider embeddings that are invariant to slight changes in brightness, contrast, or blur, of which a natural choice are neural networks. Neural Network models are currently insensitive to slight changes in all of the aforementioned criterion and can produce at above human level performance in some tasks such as that of classification. Given this we chose our embedding to be the logits of a classifier trained on the dataset of interest. The requirement of a pretrained classifier is currently a non-issue due to the wide availability of pretrained models on most if not all of the standard datasets of which GANs are trained on.

Logits. are the final output of a neural network before taking the softmax. The logits represent a natural choice as they are the most compact dimensional representation while still containing semantic information. The main reason to choose logits over the final output probabilities is that Hinton et al. [2015] has shown that the logits contain a lot of information

that is lost after the softmax. The logits themselves can be used to train other networks Romero et al. [2014].

3.2.2 Alternative Forms

Besides the form of the measure we have presented in equation 3.4 we also considered two alternative forms of the measure. For the first alteration we considered the relative relative divergence two pairs of distributions (train R and test T), and (train R and generated G). The formulation is presented below in equation 3.5.

$$\sum_{r \in R} \frac{1}{|R|} \log \frac{\sum\limits_{t \in T} \exp[-D(r,t)]/|T|}{\sum\limits_{g \in G} \exp[-D(r,g)]/|G|} \tag{3.5}$$

This formulation captured most of the desirable properties that we sought but it ended up not capturing when the generator missed several modes. A second formulation we considered was the summation of equations 3.4 and 3.5. This had all of the desirable properties that the original one had but failed to add anything obvious that the first measure missed. We therefore kept the formulation presented in equation 3.4 due to it capturing all of the desirable properties and its simpler formulation.

3.3 Methods

3.3.1 Datasets

We used the following four datasets and visualize the results as GANs train.

MNIST. The MNIST database Lecun et al. [1998] of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples.

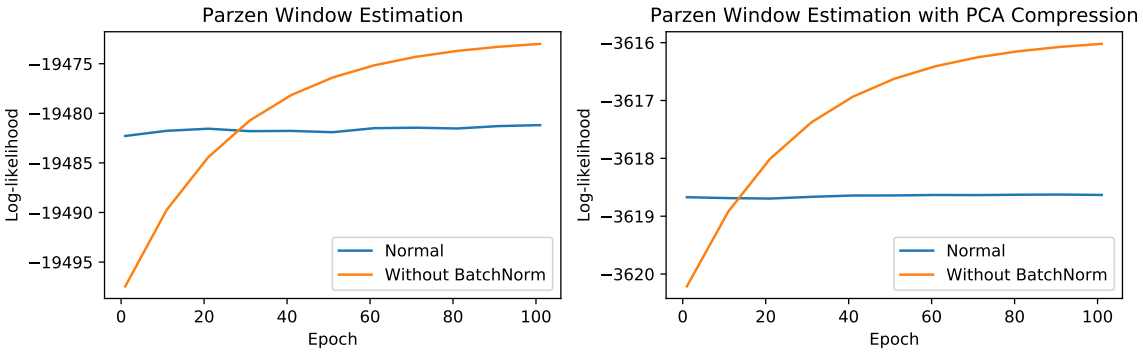


Figure 3.3: Parzen Window Estimation.
Here we demonstrate how the log-likelihood improves over time on the model without Batch Normalization even though the model never achieves better images than the typical model. We further demonstrate that even under PCA compression Parzen Windows fails to capture the desired properties.

SVHN. The Street View House Numbers (SVHN) dataset Goodfellow et al. [2013] is of cropped 32 by 32 digits found through Google Street View. There are 73,257 images in the training set, 26,032 images in the test set, and 531,131 images for additional training.

CIFAR. The two CIFAR datasets Krizhevsky and Hinton [2009] consist of colored natural scene images, with 32-by-32 pixels each. CIFAR-10 (C10) consists of images drawn from 10 classes and CIFAR-100 consists of images drawn from 100 classes. For both datasets there are 50,000 training images and 10,000 test images.

ImageNet. The ILSVRC 2012 classification dataset Russakovsky et al. [2015] consists of 1000 classes, in total 1.2 million for training, 50,000 for validation, and 100,000 for testing.

3.3.2 Architectures

For this work we consider two primary GAN architectures for our evaluation, DCGAN Radford et al. [2016], and Improved GAN Salimans et al. [2016]. We modified both architectures to create a weaker generator by removing all of the batch-normalization layers. To create our cherry picked examples for MNIST we removed convolutional layer 4 and modified the

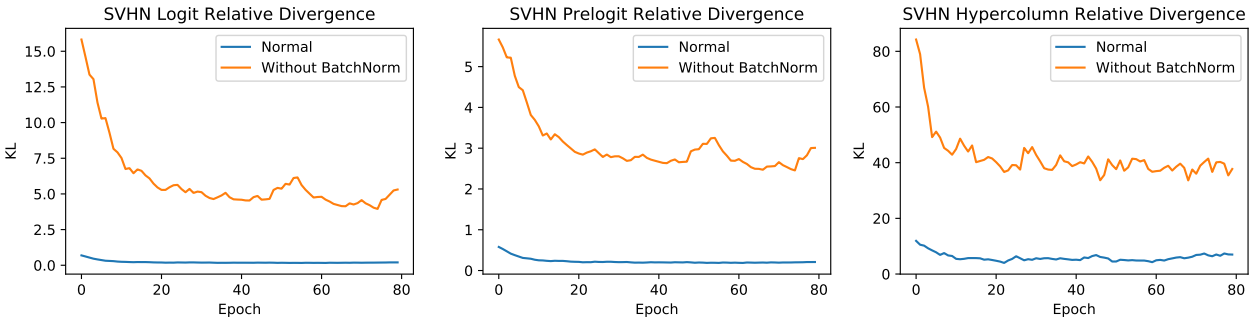


Figure 3.4: Alternative semantic embeddings.
We show that other semantic representations namely logits, prelogits, and hypercolumns track each other with DDM. This highlights that DDM is insensitive to any particular embedding so long as it contains enough semantic information to meaningfully discriminate among the samples.

third convolutional layer to match the output of convolutional layer 4 from DCGAN. We call this modified architecture small DCGAN. To run our final comparison over models we compared the regular DCGAN, Improved GAN, and Improved Wasserstein GAN Gulrajani et al. [2017].

For the purposes of creating the embedding we used Wide Residual Networks Zagoruyko and Komodakis [2016] for both SVHN and CIFAR-10. To embed MNIST we trained a 4 layer CNN architecture, followed by a fully connected layer. We used Xavier initialization Glorot and Bengio [2010] for the neural network used to embed MNIST and we used He initialization for the wide residual networks.

3.4 Experiments

3.4.1 Comparisons to Other Measures

The first measure we examine is Parzen Windows as shown in 3.3. The measure shows how the weaker model (the model without Batch Normalization) improves with respect to log-likelihood, yet the samples never achieve anything resembling natural images. This is in contrast to the regular Improved GAN architecture that appears more "natural" over time.

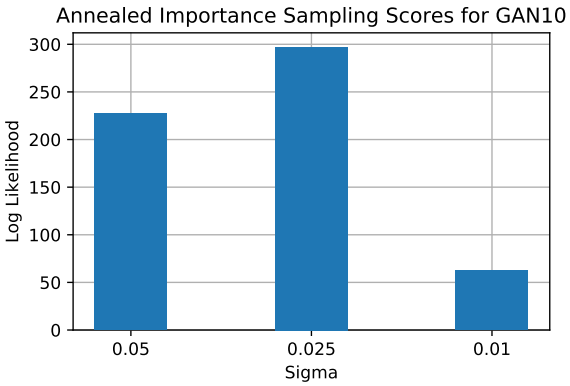


Figure 3.5: AIS
Even for a fixed GAN the Annealed Importance Sampling Score can vary by an order of magnitude depending on the sigma used for the Gaussian prior.

Thereby highlighting a result that agrees with Theis et al. [2015] that log-likelihood models do not correlate with perceptual image quality.

Additionally Parzen Windows suffers from the curse of dimensionality. Fitting a kernel to the space of natural images results in a very poor estimation as slight changes in the pixel space do not correspond to any natural changes. Leaving much of the space empty resulting in much difficulty estimating the likelihood. Another experiment that we ran was compressing natural images by taking the top principle components that capture 95% of the variation of the data and then running Parzen Windows. In this setting it would be expected that Parzen Windows would perform better but we still observed the same behavior as highlighted in 3.3 where the weak model outperformed the standard Improved GAN model. The code to run Parzen Windows was obtained from *github.com/goodfeli/adversarial/blob/master/parzen_ll.py*.

With regards to Annealed Importance Sampling in figure 3.5 we run AIS several times while varying the variance parameter on a fixed GAN architecture, the "GAN10" architecture from the AIS paper. We can observe that the final log-likelihood is heavily dependent on the choice of sigma. This shows how the measure can be manipulated by varying sigma, a hyperparameter, until the desired score is achieved.

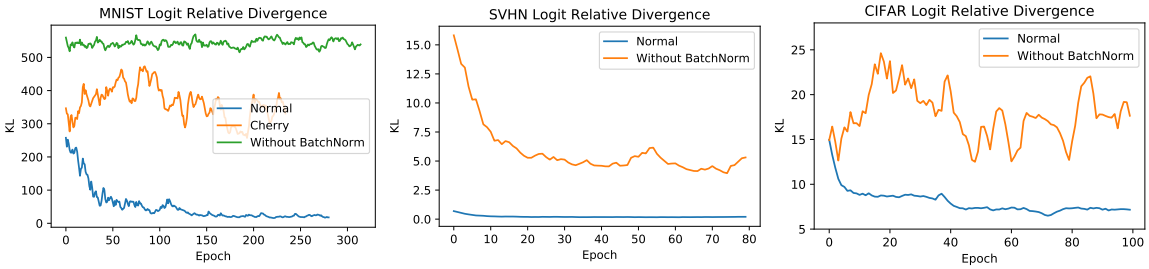
It also highlights how the measure is inappropriate for implicit density models as their claims of convergence only hold true for explicit density based generative models of which variational autoencoders are. This is discussed in Wu et al. [2016] whereby to estimate the exact posterior one requires simulated data. Therefore the final estimates that they derive can be quite far from the true likelihood except when certain conditions are met of which GANs do not meet.

The Inception Score has a few issues worth mentioning. The first issue occurs when the generated set has a small number of images that resemble ImageNet images. This will generate a high Inception Score, even though the majority of samples are of poor quality. If a generator is able to produce a few samples that activate only one class, then it becomes equivalent to taking the KL divergence between a one hot vector and uniform over 1000 classes. Afterwards one takes the exponential of this quantity so it becomes $e^{\log(N)}$ where $N = 1000$. Implying the maximum inception score is 1000 which is quite far from any of the current datasets even computed on actual images and making the score ultimately difficult to interpret.

Another issue of the Inception Score is that it fails to detect missing modes of the distribution. We created a dataset where we take natural images and replicate them until we match N the number of samples in the training set. By increasing the number of samples we thereby decrease the replication factor of the dataset. The Inception Score reaches its optimal performance once there are 1000 distinct images. Whereas DDM does not reach the optimum with any replication amount.

3.4.2 Distribution Divergence Measure

In this section we compute the Distribution Divergence Measure, DDM, on three different datasets MNIST, SVHN, and CIFAR-10 and compare the results of the architectures de-



(a) MNIST results. (b) SVHN results. (c) CIFAR-10 results.

Figure 3.6: Distribution Divergence Measure plotted over time
Distribution Divergence Measure plotted over time for the datasets. The baseline for all of the experiments is plotted as the black line. For some datasets we can observe that the model does not learn to generalize much more after the first epoch namely that of Improved GAN on SVHN. The main reason it is likely performing worse under DDM is that it is likely memorizing the training set and therefore becoming less like the test distribution.

scribed in section 3.3.2. We demonstrate that DDM achieves the correct ordinal rankings for each dataset. Due to the architectures having a great dissimilarity in their outputs . We want to start with a baseline task to ensure the model works under supervision before proceeding to more complicated comparisons that may look equivalent to human observers and vary subtlety. We would like to note that removing batchnormalization from the Improved GAN architecture produced the worst results both perceptibly and often several orders of magnitude higher (worse) than all of the other architectures, thus they are not plotted.

For our MNIST experiment we tested DCGAN, weakened DCGAN, and small DCGAN. We can observe that the correct ordinal ranking is achieved by the measure, highlighting that the measure detects the missing modes of the distribution. By ranking the small DCGAN better than weakened GAN it highlights that it is not fooled by noise and by ranking DCGAN better than small GAN it further highlights the importance of the full distribution for a better score.

For the SVHN experiment we tested DCGAN, weakened DCGAN and Improved GAN. We left out small DCGAN as we had difficulty determining when small DCGAN was performing better or worse (i.e. producing more visibly natural images) than weakened DCGAN. In our CIFAR-10 experiment we ran all of the same architectures as with SVHN.

3.4.3 Insensitivity to Embedding

To show that the DDM is not explicitly tied to the logits of a neural network we tested out other representations that contain semantic information encoded from neural networks shown in Fig 3.4. In this way we demonstrate that the measure still produces the correct ordering among the compared architectures. We test three different embedding: logits, prelogit, and hypercolumns.

Logits are the values of the final layer of the neural network before applying the softmax function. Prelogits are the inputs to the layer before the logits. Hypercolumns Hariharan et al. [2014] are a sampled representation of each layer in the neural network which contains information from many different receptive fields.

3.4.4 Insensitivity to Architecture

We also tested whether the measure varies based upon the particular architecture used to create the embedding. We experimented with determining if the measure varied based on architecture used by varying the initialization of each architecture and experimenting with different architectures. Ultimately while the final values from each architecture vary slightly the overall ordering remains unchanged. We would like to note that all of the models have sufficient model complexity to achieve at or near state of the art in the task of classification.

3.4.5 Missing Modes

In the work of Che et al. [2016] they highlight how a traditional GAN under toy settings will not capture several modes of a Gaussian mixture. To simulate how a GAN may miss modes of a distribution we create a dataset of real images that we repeat several times. In this way we can claim to have a generator that only learned a few modes of the distribution namely n examples where $n \in \{10, 20, 100, 1,000, 10,000\}$. At 10,000 examples we no longer say that the simulated data is missing several modes as it is probably covering a large portion of the

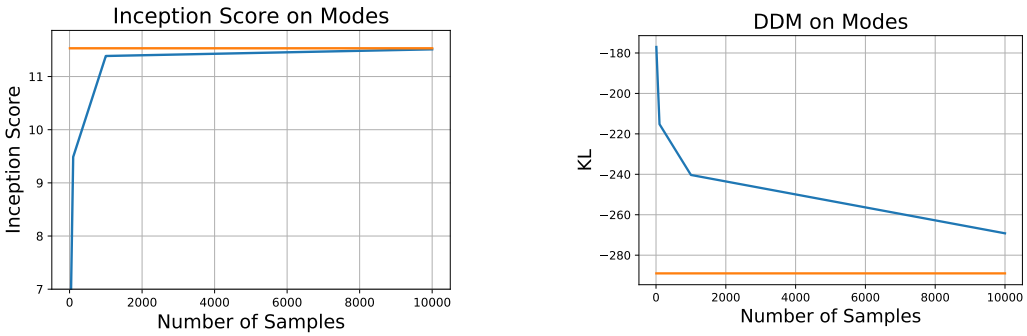


Figure 3.7: Comparison of Measures on Modes.
For this experiment we sampled 10,20,100,1000,10000 images from the training set and replicated the samples up to the size of the original dataset of 50000. The inception score saturates around 1000 images or a a replication amount of 5. For the DDM since we also compare to a test set it does not saturate but instead approaches the relative divergence. For both measures the orange line represents the best performance on each respective measure.

dataset.

We test how well DDM and Inception Score can handle the above simulated data. We take n random images from the training set and duplicate the images in n until we reach the same size as the training set N. Then we run this duplicated subset of the training set through the measure and the Inception Score. We take samples of CIFAR-10 training images. To get our final score we repeat this experiment 5 times and take the average of the scores. We show in Fig 3.7 that the Inception Score approaches its optimum at 1,000 images while DDM does not achieve its optimum as we penalize the model lack of generalization to the test set.

3.4.6 ImageNet Results

To ensure that the results work for arbitrary sized images we tested DDM on 46,000 images from the ImageNet dataset. We randomly set aside 45,000 images from the validation set that we partition into 40,000 training and 5,000 testing images. We took the remaining one thousand images to serve as our samples. We ran a series of corruptions to the 1,000 image samples, those of additive Gaussian noise, uniform random noise and Gaussian blur. For

CIFAR-10 Samples from	Measure Score
DCGAN	−163.14
Improved Wasserstein GAN	−227.66
Improved GAN	−232.74
Training Set	−289.23

Table 3.1: GAN Scores for various architectures on the CIFAR-10 dataset.

Gaussian blur with a fixed mean of zero, each additive point to the variance, sigma, roughly corresponds to a loss of 10 bits of information according to DDM. The results for additive noise tended to be on the order of 12 bits.

3.4.7 GAN Generator Comparisons

We wanted to demonstrate that DDM could be used to measure the quality of various GANs, so we chose 3 popular GAN architectures namely DCGAN, Improved GAN, and Improved Wasserstein GAN. We trained each GAN architecture until convergence as deemed their respective implementation on the CIFAR-10 dataset. Then we sample 5000 images uniformly at random from each generator to use as our generated set. See Table 3.1 for the list of results.

Here we demonstrate that the Improved GAN best approximates the true distribution, while Improved Wasserstein GAN performs slightly worse. The worst performing model is the DCGAN, as expected.

CHAPTER 4

CONCLUSIONS

4.1 Conclusion and Future Work

In this work we created both a novel GAN architecture, and approach that we demonstrate can be utilized for both interpretability and augmentation. However this method does not solve the issue of instability in GAN training. This approach still fails to produce interpretable results on image datasets that GANs still fail on such as Imagenet or COCO.

With regards to our measure, we believe it to be quite robust as to the settings we have tested against. It is also a relatively fast measure to compute. The current main downside of the method is the space complexity grows as the square of the dataset. We are currently exploring ways to mitigate this impact perhaps through clustering of the representations before computing the pairwise distances. However assuming the dataset size is not very large our approach outperforms all other methods.

For future work we plan to examine curriculum learning or training schedules. This can be approached in two ways: first start with easy examples then make it harder. The other approach can be to progressively make the discriminator stronger over time and allow the generator to consistently "win". Neither of these have been studied partially because coming up with a good curriculum is rather difficult. The idea of a discriminator curriculum has never been proposed.

REFERENCES

Puurula Antti. *Scalable Text Classification with Sparse Generative Modeling*, pages 458–469. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-32695-0.

Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode regularized generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, volume abs/1612.02136, 2016. URL `http://arxiv.org/abs/1612.02136`.

M. Germain, K. Gregor, I. Murray, and H. Larochelle. MADE: Masked Autoencoder for Distribution Estimation. *ArXiv e-prints*, February 2015.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL `http://proceedings.mlr.press/v9/glorot10a.html`.

M. Goessling and Y. Amit. Mixtures of Sparse Autoregressive Networks. *ArXiv e-prints*, November 2015.

Marc Goessling. *High-Dimensional Generative Models: Shrinkage, Composition and Autoregression.* PhD thesis, 2016. URL `https://knowledge.uchicago.edu/handle/11417/242?show=full`.

Jacob Goldberger, Shiri Gordon, and Hayit Greenspan. Approximating the kullback leibler divergence between gaussian mixture models. In *International Conference on Computer Vision (ICCV)*, 2003.

I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. *ArXiv e-prints*, December 2013.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. pages 1–9, 2014a. ISSN 10495258. doi: 10.1001/jamainternmed.2016.8245. URL `http://arxiv.org/abs/1406.2661`.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2014b.

I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved Training of Wasserstein GANs. *ArXiv e-prints*, March 2017.

B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Hypercolumns for Object Segmentation and Fine-grained Localization. *ArXiv e-prints*, November 2014.

G. Hinton, O. Vinyals, and J. Dean. Distilling the Knowledge in a Neural Network. *ArXiv e-prints*, March 2015.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *ArXiv e-prints*, December 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013. URL `https://arxiv.org/abs/1312.6114`.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Tech Report*, 2009.

Yann Lecun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

Yann LeCun, Fu Jie Huang, and Léon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'04, pages 97–104, Washington, DC, USA, 2004. IEEE Computer Society. URL `http://dl.acm.org/citation.cfm?id=1896300.1896315`.

M.-Y. Liu and O. Tuzel. Coupled Generative Adversarial Networks. *ArXiv e-prints*, June 2016.

David Lopez-Paz and Maxime Oquab. Revisiting classifier two-sample tests. In *International Conference on Learning Representations (ICLR)*, 2017.

M. Mirza and S. Osindero. Conditional Generative Adversarial Nets. *ArXiv e-prints*, November 2014.

Volodymyr Mnih and Geoffrey E. Hinton. Learning to detect roads in high-resolution aerial images. In *Proceedings of the 11th European Conference on Computer Vision: Part VI*, ECCV'10, pages 210–223, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-15566-9, 978-3-642-15566-6. URL `http://dl.acm.org/citation.cfm?id=1888212.1888230`.

S. Nowozin, B. Cseke, and R. Tomioka. f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization. *ArXiv e-prints*, June 2016.

A. Odena, C. Olah, and J. Shlens. Conditional Image Synthesis With Auxiliary Classifier GANs. *ArXiv e-prints*, October 2016.

Vilfredo Pareto. The new theories of economics. *History of Economic Thought Articles*, 5, 1897. URL https://EconPapers.repec.org/RePEc:hay:hetart:pareto1897.

Emanuel Parzen. On estimation of a probability density function and mode. *Ann. Math. Statist.*, 33(3):1065–1076, 09 1962. doi: 10.1214/aoms/1177704472. URL https://doi.org/10.1214/aoms/1177704472.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2016.

A. Romero, N. Ballas, S. Ebrahimi Kahou, A. Chassang, C. Gatta, and Y. Bengio. FitNets: Hints for Thin Deep Nets. *ArXiv e-prints*, December 2014.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014. URL http://arxiv.org/abs/1409.0575.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Neural Information Processing Systems (NIPS)*, 2016.

Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *CoRR*, abs/1701.05517, 2017. URL http://arxiv.org/abs/1701.05517.

Lucas Theis, Aaron van den Oord, and Matthis Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations (ICLR)*, 2015.

B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle. Neural Autoregressive Distribution Estimation. *ArXiv e-prints*, May 2016.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016. URL `http://arxiv.org/abs/1609.03499`.

Yanshan Wang, In-Chan Choi, and Jae-Sung Lee. Indexing by latent dirichlet allocation. *CoRR*, abs/1309.3421, 2013. URL `http://arxiv.org/abs/1309.3421`.

Y. Wu, Y. Burda, R. Salakhutdinov, and R. Grosse. On the Quantitative Analysis of Decoder-Based Generative Models. *ArXiv e-prints*, November 2016.

S. Zagoruyko and N. Komodakis. Wide Residual Networks. *ArXiv e-prints*, May 2016.