Improving Google's Unsupervised Pixel-Level Domain Adaptation

Team Members:
Saad Waraich
Devin Cole
Maria Mosquera Chuquicusma
Gabriel Ribeiro

Mentor: Dr. Boqing Gong

Table of Contents

- Executive Summary 1
- Problem Objective and Motivation 2
- Broader Impacts 3
  - Current Applications 4
- Personal Motivations 5
  - Saad Waraich 5
  - Maria Mosquera Chuquicusma 5
  - Devin Cole 6
  - Gabriel Ribeiro 6
- Research 7
  - Background 7
    - Generative Adversarial Networks in Deep Learning 9
      - Generative Adversarial Networks (GANs) 9
      - Deep Convolutional GANs 11
      - Conditional-GAN and Info-GAN 13
    - Previous Frameworks for Domain Adaptation 17
  - Potential Applications 18
    - Adversarial Discriminative Domain Adaptation 18
    - Coulomb GANs: Provably Optimal Nash Equilibria via Potential Fields 20
    - Stacked Generative Adversarial Networks 23
    - Deep Hashing for Unsupervised Domain Adaptation 25
    - Energy Based GANs 28
    - From Source to Target and Back: Symmetric Bi-directional Adaptive GAN 31
    - Asymmetric Tri-training for Unsupervised Domain Adaptation 34
    - Generate to Adapt: Aligning Domains Using Generative Adversarial Networks 38
    - Unsupervised Domain Adaptation by Backpropagation 42
    - Coupled GANs 46
    - Unsupervised Transductive Domain Adaptation 51
    - Wasserstein GAN 55
    - Loss-Sensitive GAN 64
    - Strategies for Improving GANs Training 71
    - Improved Semi-Supervised Learning with GANs Using Manifold Invariances 74
    - Adversarial Feature Learning (Bidirectional Generative Adversarial Networks - BiGANs) 79
    - BEGAN: Boundary Equilibrium Generative Adversarial Networks 82
    - Deep Unsupervised Convolutional Domain Adaptation 87
- Objectives 92
  - Technical Objectives and Requirements 92
  - Communication 93
    - Mentor 93
    - Communication 93
- Design 94
  - Google's Unsupervised Pixel-Level Domain Adaptation Using GANs 94
  - General Model 108
  - Model Architectures 108
    - CoWGAN 109
    - CoIGAN 110

- Datasets 111
  - MNIST 111
  - MNIST-M 112
  - USPS 113
  - Synthetic Cropped LineMod and Cropped LineMod 114
- Languages and Libraries 115
  - Python 115
  - TensorFlow 116
  - Keras 117
  - NumPy 118

- Results 119
  - Replication of Google's Experiment 119
  - CoWGAN 120
  - CoIGAN 121
    - MNIST to MNIST Image Generation 121
    - MNIST to MNIST Graphs 123
    - MNIST to USPS Image Generation 125
    - MNIST to USPS Graphs 127
    - MNIST to MNIST-M Image Generation 129
    - MNIST to MNIST-M Graphs 131
  - CoPixGAN 133
- Administrative Content 134
  - Budget and Financing 134
  - Gantt Chart 135
  - Pert Chart 136
- Conclusion 137
- References 138

Executive Summary

Computer vision and machine learning have become the leading fields in computer science, both in academic and private sectors. There have been many advancements made in supervised and semi-supervised research; however, for the unsupervised side of research, there is a lot to learn. It is one of the more difficult tasks: learning a convolutional neural network framework that can take in unlabeled complex data and figuring out what the data is. A private sector company that has been working on this problem is Google. Their research, Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks [1], produced outstanding results. The Google project focused on the ability to identify objects in different domains. Domains could be a different location, photo setting (blur or brightness), or object orientation. Our project aims to replicate Google's results and modify Google's architecture to create better results. We will focus on creating a GAN that is better in feature identification (training of a model) and image replication (GAN's generator).

Problem Objective and Motivation

Domain adaptation allows objects in images to not to be affected by disturbances in the visual domain (i.e. blurs, lighting, etc.) they are in. Unsupervised domain adaptation focuses on transferring knowledge from a source domain with labels to a target domain with no labels [1]. There has been previous research in supervised domain adaptation; however, large datasets with labeled data are scarcely available. One motivation for this research, conducted by Google, is to be able to generalize between images in different settings and have ground-truth annotations readily available. Throughout their research, Google explored a very popular architecture for unsupervised learning called Generative Adversarial Networks (GANs). The aim for our project is to possibly improve upon Google's state-of-the-art results.

The program created will function in the following manner. A dataset will be passed to a Generative Adversarial Network (GAN), which generates fake images. Then, these fake images and the target domain images will be fed to the discriminator, which will discriminate between the two. The fake images will keep on changing, retaining the domain invariant features between fake images and the target domain. Ultimately, the GAN will start generating fake images that will be similar to the target domain images. These fake images, which contain domain invariant features and look like the target domain, will be fed to a classifier.

Our initial experiments will consist of finding an architecture that extracts rich domain-invariant discriminative features from images since unsupervised learning relies on representation learning. After a period of testing, data analysis will commence, comparing different techniques to find optimal results.

To be able to attain comparable results, we will use the three hand-written digit datasets used by Google: MNIST, MNIST-M, and USPS. Due to the time constraints of project delivery, algorithm training will be shortened. Additionally, there will be a hardware constraint for training the networks with large datasets. Final data quality may suffer.

Broader Impacts

The problem tackled in this project is very specific: is it possible to create better results than Google's "Unsupervised Pixel-level Domain Adaptation Using Generative Adversarial Networks" research? The process of finding this out can become too convoluted to where the why is missing. So why take on this challenge? This research comes from a branch of computer science called computer vision. It is the attempt to teach computers to process and recognize the world around them. By having a better understanding of their surroundings, computers can make major contributions to our daily lives. Humans will have a second pair of eyes that don't get tired and are always alert. Some examples of this technique being already applied include autonomous cars, face detection in cameras, selfie filters on Snapchat, and Google 3D maps. There are many potential applications of this technology. Doctors and nurses who work long hours on the hospital floor will be able to give better care for their patients. Computers will be able to indicate potential treatments based on patient symptoms. A large database to draw from and powerful cameras that can pick up details the human eye might not, will create more accurate diagnoses. Being able to process their environment, these machines will give sight to the blind, notifying them of every potential hazard and object just ahead. Eventually, they will have the ability to recognize and process written words and read them aloud such as words in restaurant menus. This creates more autonomy for the blind which will lead to better and healthier lives. As the field continues to progress, more viable applications will be available, increasing quality of life. Images in the next page show examples of how this computer vision research is being used.

Current Applications

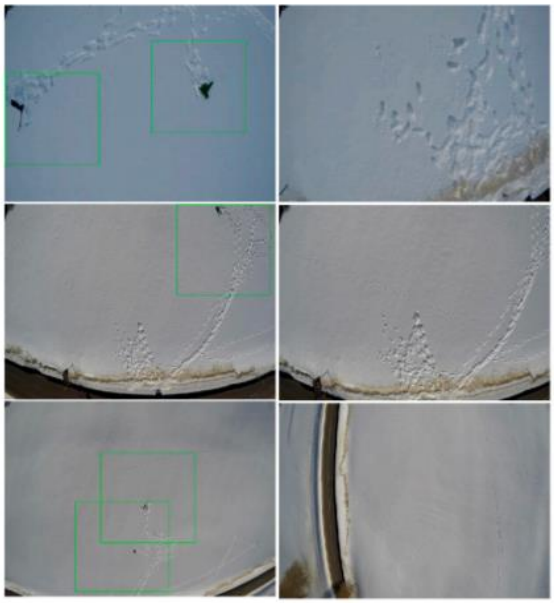


Figure 1. *"A convolutional Neural Network (CNN) Approach for Assisting Avalanche Search and Rescue Operations with Unmanned Aerial Vehicles (UAV) Imagery"* [2]. *The image above shows detection of potential avalanche victims detected by UAV. This is an attempt to speed up deployment of rescue teams.*

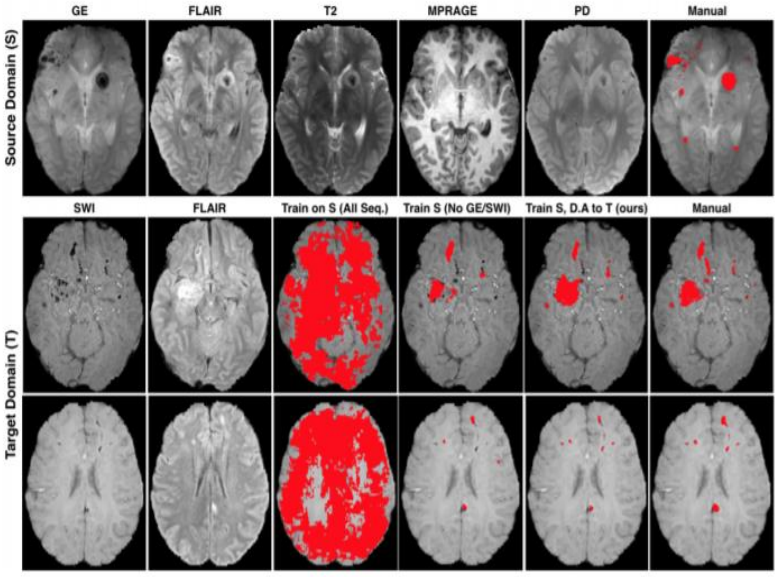


Figure 2. *"Unsupervised domain adaptation in brain lesion segmentation with adversarial networks"* [3]. *A system devised to learning segmentation of brain lesions. A system is created to classify the domain of the input data while another system attempts to counter that process, creating an algorithm that can increasingly detect brain lesions.*

Personal Motivations

Saad Waraich

During the summer, I interned at a software firm called Sighthound Inc., which mainly develops software in computer vision. When I started college, computer vision was my intended field to work in and getting an internship at a place which works in computer vision increased my interest in the field's research. During summer, I spent a lot of time reading different research papers related to computer vision and I enjoyed doing that because it's an emerging field and it will help improve technology in fields like medicine and robotics. In CVPR 2017, Google published a research paper called "Unsupervised Pixel-level Domain Adaptation Using Generative Adversarial Networks" which caught my interest. I liked what they were doing, and I wanted to do research in domain adaptation. I discussed my idea with my friends and they were also interested in domain adaptation, so as a team, we started exploring our potential approaches for domain adaptation to try to get better results than Google. Nonetheless, we are going to try to at least add something in the field of domain adaptation even if we cannot beat Google's results.

Maria Mosquera Chuquicusma

During the summer term of 2017, I spent my time doing research in the intersection of computer vision, machine learning, and medical imaging. One of the projects I worked on was involved with detecting the risk of pancreas cancer, which is very difficult to diagnose. Throughout my research, I learned that object settings usually propose many problems when they are distinct. For instance, a convolutional neural network could have problems generalizing a specific cancer lesion in datasets because of distinct settings, and that potentially could lead to an over or under diagnosis. My previous research made me become passionate about computer vision and machine learning, and had I not done it, perhaps I would have never chosen this research project. The problem we are trying to improve will be difficult, but then again, any problem in either of these fields will always pose a challenge. I believe that improving Google's results will not only allow us to help law enforcement agencies, hospitals, and a wide variety of other industries improve their software by allowing any type of objects to be detectable regardless of setting, but it will also help us grow as young researchers. In my opinion, having the opportunity to work on a research project will expand our creativity and encourage curiosity to approach other problems in the future.

Devin Cole

Before computer science, I spent my life dedicated to medicine and biochemistry. I am still interested in the medical field and believe that research into deep learning and neural networks can lead to better medical technology. There is already plenty of software available, such as mobile applications that can identify skin cancers, that utilize deep learning. If we can successfully improve the results of Google's experiment, we could increase the accuracy of these and future software. Doctors could be able to perform quicker and more accurate diagnoses and evaluate medical images more efficiently. Laboratory technicians could use deep learning machines to identify bacteria and viruses on slides or analyze sample data. This is a great opportunity to help doctors and patients which is why I wanted to take part in this project.

Gabriel Ribeiro

I've always been interested in technology, one of them being robots. One of the major components for a robot to function properly is to understand its environment. Computer vision plays a leading role in this process. The research being studied in our project can have broad influences in multiple industries, including robotics. Due to this fact, it's exciting to work on a project that can make object recognition more accurate. This can lead to better security in public areas and to better machine and human interaction in areas such as healthcare and retail. Assistant robots will be able to pick up and carry items for people who need help such as the elderly. They can better perform search and rescue missions by having a more accurate spatial awareness. They can also recognize victims in the middle of rubble. Anything that can better society will always be something I'm happy to work on.

Research

Background

The history of Computer Vision began in the summer of 1966. During that summer, MIT's artificial intelligence group decided to solve a computer vision problem. Little did they know how complex this would be and the many branches in computer science that would be created due to one research assignment. They experimented with using AI in different cases to interpret and extract information about vision data. That summer, the problem wasn't solved but the challenge was set.

At the start of AI, these systems relied heavily on hard-coded knowledge. As good as humans may be with interacting with their environment, they have a hard time describing it, missing minor details we take for granted on a daily basis. It became difficult for these systems to do their job, to make simple or complex decision due to the human error of their creators. This led to the idea that systems must have the ability to acquire their own knowledge by extracting patterns from raw data. This is known as machine learning. Machines/systems will eventually recognize patterns in their environment, through training and self-exploration. By recognizing these patterns, they better serve their purpose, being it an autonomous car able to sense potential hazards around it or a video processing program that can detect wrongdoers who commit a crime.

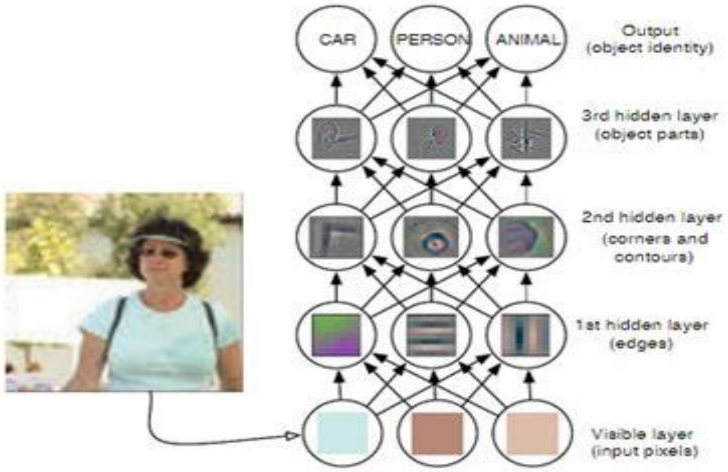


Figure 3. *Visible and hidden layer connected to create neural network.*

In the Figure 3, deep learning is used to break the complex attempt of mapping the image into smaller representations. As the data is filtered through each mapping, hidden layers extract abstract features from the image. They are considered hidden because their values are not given in the data, but are determined by the model.

There are a few key concepts we must first understand, to get a better view of what machine learning does. In machine learning, the algorithm attempts to create a representation of the data it has been given. It does this by breaking the data into smaller segments called features. Features are assigned based on what the algorithm believes is a significant piece of data to evaluate future given data. This is done during training. Once features are assigned, the algorithm will have a good mapping of what an effective representation looks like. The goal is to separate the factors of variation that explain the observed data [4]. An example of factors of variation for a car would be its color, position, and the brightness of the sun. Most times these factors affect all the pixels in the image. Because of their significant influence on the image, it becomes difficult to distinguish between what factors to keep and discard.

A solution to factor variation extraction is deep learning. Deep learning creates complex representation of images by building on smaller simpler representations.

Generative Adversarial Networks in Deep Learning

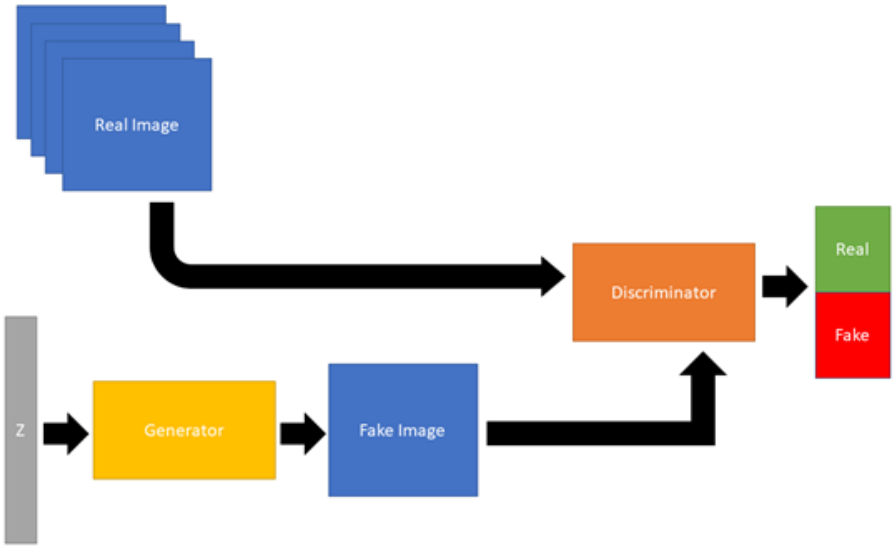


Figure 4. *Diagram of Generative Adversarial Network architecture.*

1. Generative Adversarial Networks (GANs)

Deep generative models that have been used in previous research that represent probability distributions consisted of Boltzman Machines, Deep Belief Networks, Variational Autoencoders, etc [4, 5]. However, Goodfellow et al. proposed a new framework called Generative Adversarial Networks (GANs), which have been proven to give remarkable results for image generation without any type of memorization in various fields of research studies, specifically in the areas of computer vision that focus on deep unsupervised learning. GANs can extract the most relevant discriminative features within respective images and focus on representation learning, which aids immensely in the image generation aspect. Tasks that have shown predominant results using GANs are single image super resolution, art generation, and image-to-image translations [6].

GANs are composed of a generator, G, and a discriminator, D, shown in Figure 4. The generator inputs random noise sampled from a random distribution p_z and learns to generate a fake image similar or identical to an original image, x. On the other hand, the discriminator attempts to learn from the distribution p_{data}. The discriminator uses a binary classifier for both the original images, x, and the generated image samples, $G(z)$, and tries to distinguish which one is real and which one is generated by outputting a probability score for the generated and real images. The main objective of the generator is to convince the discriminator into perceiving the generated image samples as the actual real images, so in other words $p_x = p_z$. Throughout training, the generator and discriminator will play a min-max game shown in the value function:

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

The idea is that theoretically both reach a Nash Equilibrium, meaning the losses converge to 0, however, this is not realistically possible, which makes GANs relatively difficult to train, even with the proposed optimal algorithm by Ian et al., shown in Figure 5. Some problems that arise during GAN training are overfitting, where the generator and discriminator are not necessarily learning which leads to production of low quality noisy generated images. To minimize this problem, researchers keep track of both generator and discriminator losses by plotting these losses for each desired range of iterations into graphs. Then researchers can see if these losses are converging to zero. Sometimes these metrics are not enough, and image generation has to then be inspected visually to check this issue and that there is variation in images [6, 7].

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Figure 5. *Algorithm proposed by Goodfellow et al. to train a standard GAN optimally. The training consists of training a dataset of images using minibatches, subsets of images from the dataset for k steps. The proposed training is to update the discriminator and generator loss function using stochastic gradient descent. The discriminator loss function is updated more frequently than the generator loss function throughout learning to prevent the zero-gradient learning problem.*

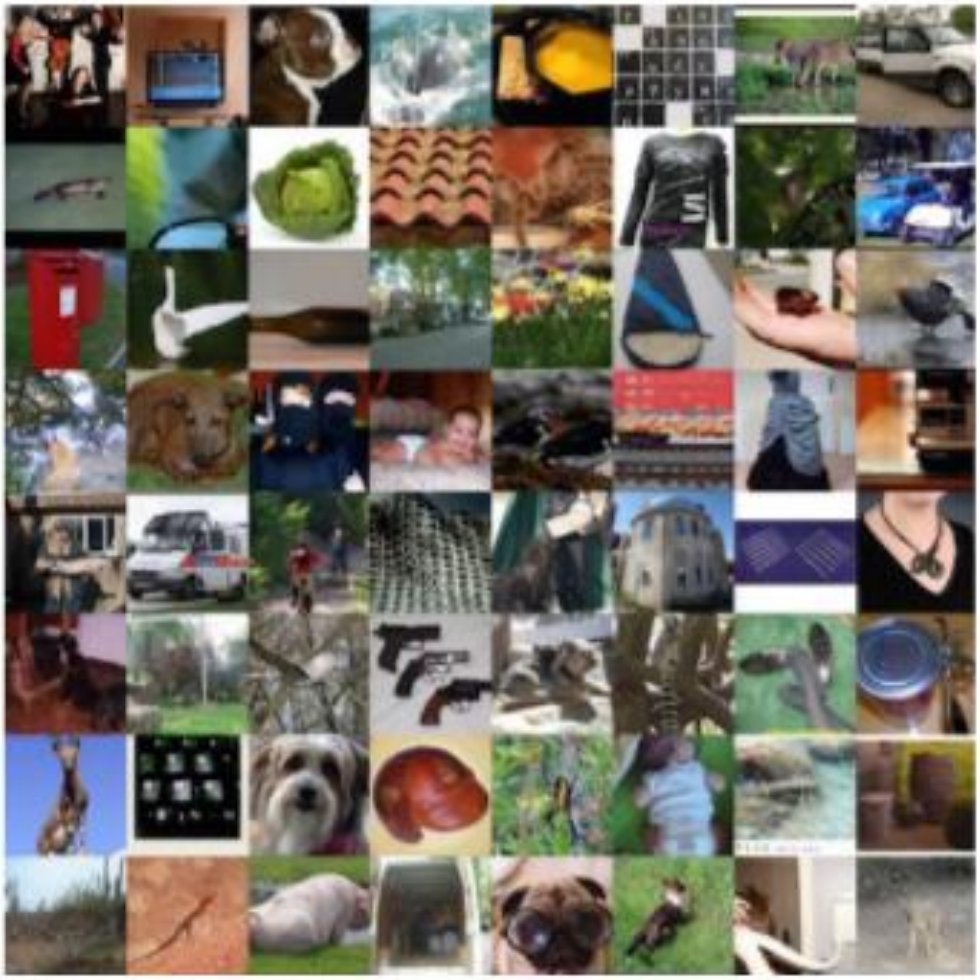


Figure 6. *Original images from the ImageNET dataset.*

2. Deep Convolutional GANs

Although GANs have shown remarkable results, the most used variant is called the Deep Convolutional GAN (DC-GAN). DC-GAN is an attempt to bridge the successes of CNN learning in both supervised and unsupervised conditions. DC-GANs use a limited architectural constraint. By creating and evaluating this set of constraints to make them stable to train in most settings. DC-GANs use trained discriminators for image classification tasks, showing competitive performance with other unsupervised algorithms. The generators have interesting vector arithmetic properties allowing for easy manipulation of many qualities of generated samples.

Figure 7. *Generated image samples using a standard GAN.*

The DC-GAN [8] architecture takes on a few core approaches to create stable training in various datasets.

1. All convolutional network, which uses strided convolutions. Giving the network the ability to learn its own downsampling. This concept is added to the generator (fractional-strided convolutions), for upsampling, and discriminator (strided convolutions).
2. The elimination of fully connected layers. The first layer of the GAN is fully connected. Convolutional features are directly connected to the input and the output of generator and discriminator. There are no fully connected hidden layers.
3. Addition of Batch Normalization allowed for stabilized learning. Batch Normalization is applied to all layers except for the generator output and discriminator input layer.
4. Use of ReLU activation in the generator except for the output layer. LeakyReLU for all layers in the discriminator.

Figure 8. *Generated image results of LSUN dataset using DCGANs.*

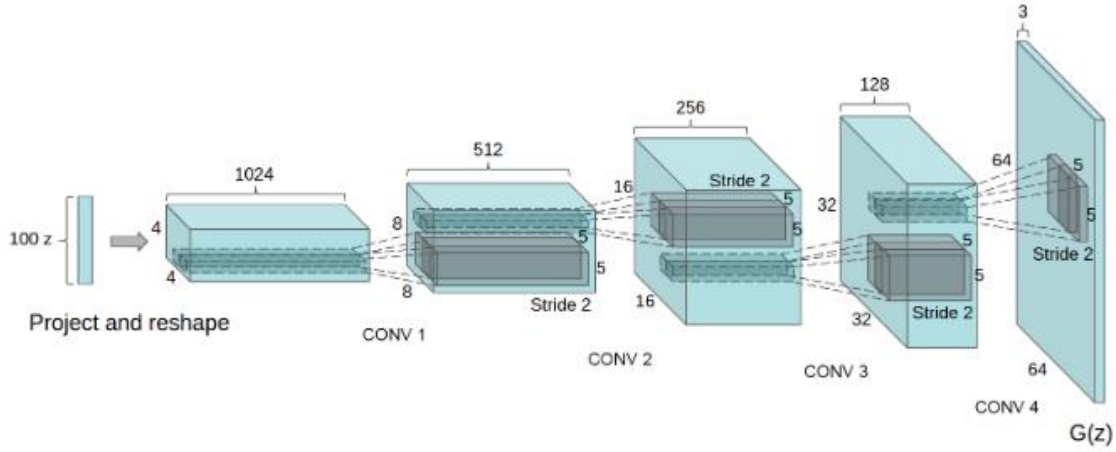


Figure 9. *DCGAN generator used for LSUN dataset scene modeling. A 100-dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four convolutions then convert this high-level representation into a 64 x 64 pixel image. No fully connected layer or pooling layers are used.*

3. Conditional-GAN and Info-GAN

Moreover, learning from a conditional architecture allows images to be classified into respective classes. GANs are not only used in non-conditional supervised, semi-supervised, and unsupervised learning. A variant GAN, called Conditional-GAN (C-GAN) [9], has been developed for supervised and semi-supervised

learning and another variant Info-GAN [10] for unsupervised, semi-supervised, and supervised learning. As opposed to the original GAN, C-GAN takes in labels y and random noise z as input to the generator to be able to generate samples from a specific class and it also input both y and x into the discriminator, so that the discriminator is able to separate features belonging to a specific class label. Info-GAN extends C-GAN to unsupervised learning and Google used the Info-GAN model to develop their classifier.

Info-GAN, an extension of the standard GAN, has proven to be a highly effective method in conditional unsupervised learning, since it is able to disentangle meaningful representations, which provide salient attributes, in images. To clarify, the disentanglement of representations happens for instance when facial attributes such as facial expression, eye color, hairstyle, etc. are found. This is something that supervised learning methods lack.

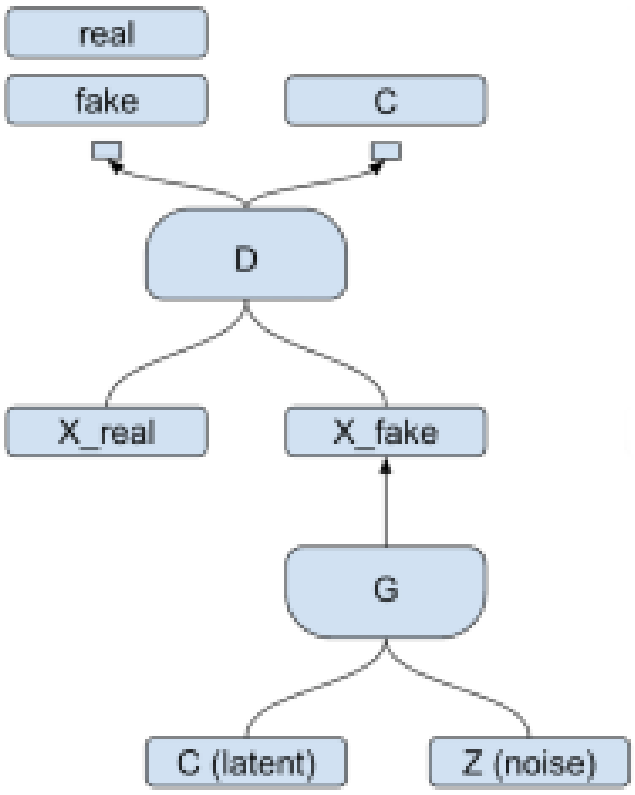


Figure 10. *Info-GAN architecture. Consists of a generator, G, that inputs a latent variable c and a random noise z. The discriminator takes in a generated sample and a real sample and classifies it as real or fake. It also is able to give real images a specified class.*

As opposed to the original GAN, which only focuses on one type of noise, z, which is treated as random noise for a generator, G. Info-GAN for unsupervised learning focuses on a noise, z, and a newly added noise, c, for a generator, G. The new noise, c, is considered the latent code and its main objective is to find salient

features in data. The variable c is seen as a label for each generated image. In C-GAN, c is known because the labels are available either fully or partially, however, in Info-GAN that is not the case. In Info-GAN, c is unknown, and it is hypothesized based on the given data representations. To avoid the problem of trivial codes posed by the noises z and c, information-theoretic regularization is proposed. This approach ensures that there is a high amount of mutual information between the latent codes c and $G(z,c)$, known as the generated samples. Since in Info-GAN we do not know c, we have to sample c from an auxiliary distribution network Q, which holds the conditional distribution for a given image. The network Q could be separate or merged with a discriminator, D. The minmax game value function for Info-GAN is shown below:

$$\min_{G,Q} \max_{D} V_{InfoGAN}(D,G,Q) = V(G,D) - \lambda L_1(C,Q)$$

With the latent variable, c, InfoGAN not only focuses on using discrete variables (specific classification), but also continuous variables. The latent variable, c, could be broken up into multiple latent codes, which means that variation could be controlled since there could be zero or a multitude of latent codes. Some examples of how InfoGAN uses this both categorical and continuous codes are shown in Figure 11 [10].

Figure 11. *Continuous variation is shown and there is a difference in lighting for each generated sample.*

Figure 12. *Discrete variation is shown, which just focuses on showing the correct digit instead of other variables.*

Previous Frameworks for Domain Adaptation

To improve the design of our architecture, we must also become informed, not only on the state-of-the-art frameworks, but also of previous approaches and the underlying challenges those approaches faced in the realm of domain adaptation. Aside from Google's approach to improve unsupervised domain adaptation, there have been numerous previous related works that have used other architectures, such as auto-encoders, rather than GANs.

Chen et. al proposed marginalized stacked denoising autoencoders (mSDAs) for unsupervised domain adaptation. A SDA is comprised of an encoder and decoder, just like the general autoencoder, but it has multiple hidden layers and noise added to corrupt the original input images in various forms to aid in the learning process. Several forms of corruption are additive isotropic Gaussian noise (GS), masking noise (MN), and salt-and-pepper noise (SPN). The training for a mSDA goes as follows: an input x is encoded into a latent variable z and then z is decoded to reconstruct an image and the process repeats until the loss converges. Their mSDA architecture competes in accuracies with the traditional SDAs and improves two ailments: high computational costs and the lack of scalability to high-dimensional features, an improvement from the standard SDA. Recent works by Clinchant et. al have made further contributions to the architecture proposed by Chen et. al. Some of their contributive work entails of adding a regularization aspect to mSDAs to prevent overfitting via unsupervised learning, however their work is focused on a text dataset rather than image dataset. Even though the generation of images for autoencoders is not as refined as GANs when it comes to image quality, it cannot be entirely assumed that GANs are better overall. The methods that we will be researching will mostly be GAN based or related to GANs and we will also be exploring minimal autoencoders, but very minimally [11, 12, 13].

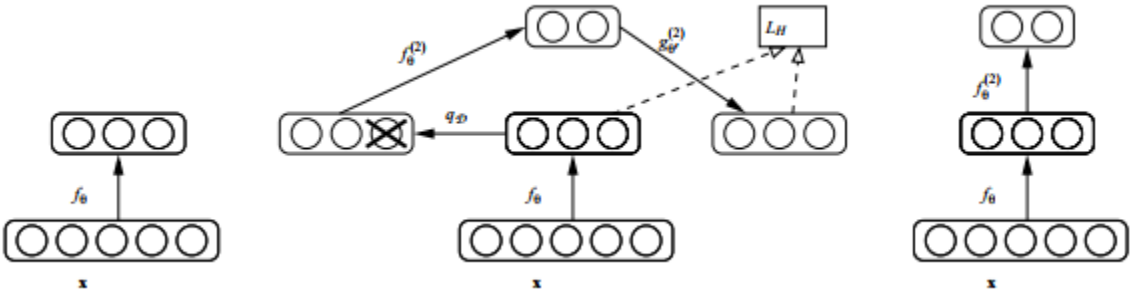


Figure 13. *Stacked denoising auto-encoder architecture.*

Potential Applications

Adversarial Discriminative Domain Adaptation

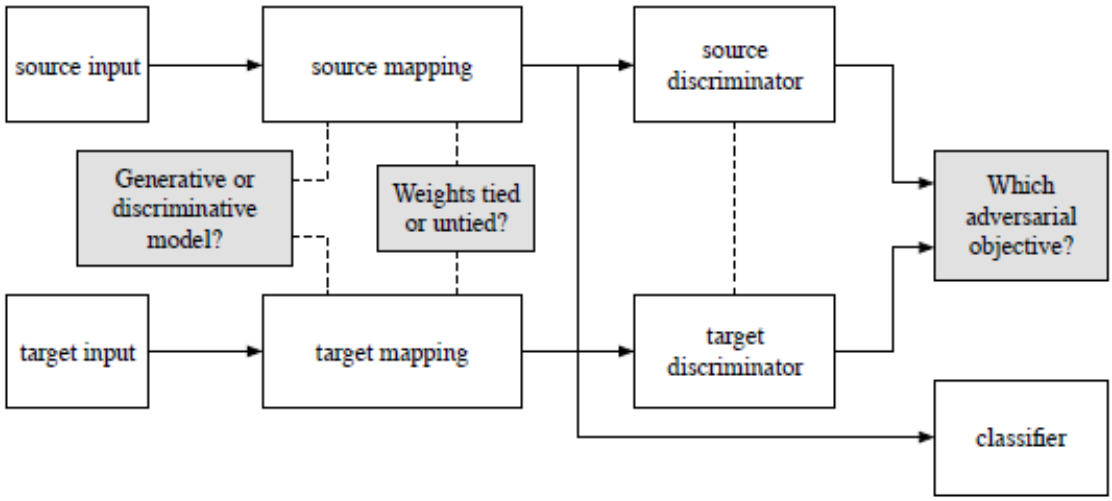


Figure 14. *Generic model for popular domain adaptation architectures that use independent networks to train source and target domains separately.*

Tzeng et al. gave a unified view of some of the most prevalent and innovative frameworks that have been explored in adversarial domain adaptation, which include gradient reversal, domain confusion, CoGAN, and their model, ADDA. The frameworks for these architectures focused on specific properties such as a base model (discriminative or generative), whether to weight share, and an adversarial loss function shown in Figure 14. The ADDA framework focused on using a discriminative base model, no sharing of weights, and the original GAN loss function.

In domain adaptation, the goal for an architecture is to train the source domain to appear as if it was from a target domain and for this the property of weight sharing is of high importance. With weight sharing there could be no, partial, or full sharing. With full sharing, the weights of both of the two networks are shared among them, however, no sharing and partially sharing have proven far more effective. This is because each network can focus on their particular domain-specific images during training and the network does not become overwhelmed by handling images from two different domains. During adversarial training, adversarial is of high importance, their work focused on using the original GAN loss function with parallel objectives. This means that there will be two independent objectives, thus two different loss functions, one for the source and one for the target domain, shown below:

$$\max_{D} \mathbb{E}_{x\sim p_S(x)}[\log D\,(M_S\,(x))] + \mathbb{E}_{x\sim p_T(x)}[\log(1 - D(M_T(x)))]$$
$$\max_{M_T} \mathbb{E}_{x\sim p_T(x)}\big[logD\big(M_T(x)\big)\big]$$

Our architecture will mainly focus on a discriminative base model, like the ADDA. The property of no-weight sharing is worth exploring and, also, the mentioned properties are of high importance as they have produced state-of-the-art results shown in Figure 15 [14].

Method	MNIST → USPS	USPS → MNIST	SVHN → MNIST
Source only	0.752 ± 0.016	0.571 ± 0.017	0.601 ± 0.011
Gradient reversal	0.771 ± 0.018	0.730 ± 0.020	0.739 [16]
Domain confusion	0.791 ± 0.005	0.665 ± 0.033	0.681 ± 0.003
CoGAN	0.912 ± 0.008	0.891 ± 0.008	did not converge
ADDA (Ours)	0.894 ± 0.002	0.901 ± 0.008	0.760 ± 0.018

Figure 15. *Classification results for the ADDA model compared to other state-of-the-art models.*

Coulomb GANs: Provably Optimal Nash Equilibria via Potential Fields

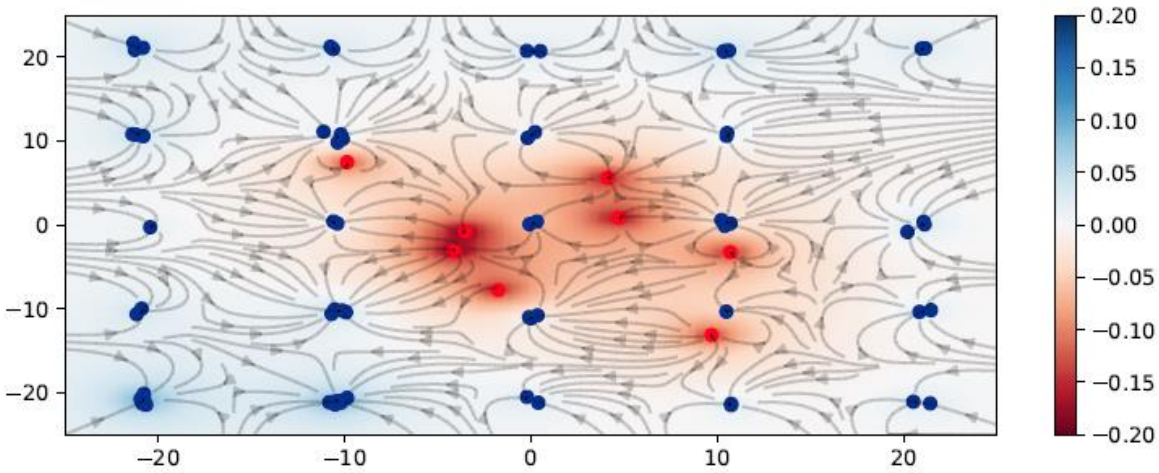


Figure 16. *Distribution of real and fake images in an electric field.*

Standard GANs have had the problem of achieving Nash Equilibrium, which is why they are so difficult to train, and they suffer from mode collapsing, which is lack of variation in images, especially if they don't use methods such as batch normalization. Another issue is the training samples may not be correctly modeled, since the discriminator cannot tell the generator whether an image is exactly real or exactly fake. Yet another critical issue is that the whole Vanilla GAN architecture can forget previous modeling errors in past epochs and in this case these errors are not discarded and can reoccur. This essentially leads to an oscillation behavior, which prevents the standard GAN from achieving Nash Equilibrium.

Figure 17. *Samples generated by Coulomb GAN using the CelebA, LSUN Bedrooms, and CIFAR-10 datasets.*

Coulomb GAN proposes to alleviate all the previous ailments found in the standard GAN and essentially learn the whole data distribution. The idea of the Coulomb GAN is based on Coulomb's Law. Basically, a set of real samples generate a potential field that attracts fake samples. Simply, as shown in Figure 16, the blue

particles represent the real data points and the red particles represent the fake data points. Opposite particles would attract; however, same colored particles will repel. Since the red colored particles are said to repel each other, this prevents mode collapse, since the red colored particles could never be like each other. The blue particles are held statically in place, they do not move, and they produce an electric field. Initially, the red particles would be randomly dispersed. The goal of the red particle is to find the nearest local blue particle and eventually become exactly like it. So, in other words, the red particle will be attracted and move towards a blue particle. Coulomb GANs also follow the law of conservation, which in this case, means a data point is neither created nor destroyed, it is merely edited.

Algorithm 1 Minibatch stochastic gradient descent training of Coulomb GANs for updating the the discriminator weights w and the generator weights θ.

while Stopping criterion not met **do**

- Sample minibatch of N_x training samples $\{x_1, \ldots, x_{N_x}\}$ from training set
- Sample minibatch of N_y generator samples $\{y_1, \ldots, y_{N_y}\}$ from the generator
- Calculate the gradient for the discriminator weights:

$$dw \leftarrow \nabla_{\boldsymbol{w}} \left[\frac{1}{2} \sum_{i=1}^{N_x} \left(D(\boldsymbol{x}_i) - \hat{\Phi}(\boldsymbol{x}_i) \right)^2 + \frac{1}{2} \sum_{i=1}^{N_y} \left(D(\boldsymbol{y}_i) - \hat{\Phi}(\boldsymbol{y}_i) \right)^2 \right]$$

- Calculate the gradient for the generator weights:

$$d\theta \leftarrow \nabla_{\boldsymbol{\theta}} \left[-\frac{1}{2} \frac{1}{N_x} \sum_{i=1}^{N_x} D\left(\boldsymbol{x}_i\right) \right]$$

- Update weights according to optimizer rule (e.g. Adam):

$$w_{n+1} = w_n + \texttt{ADAM}(dw, n)$$
$$\theta_{n+1} = \theta_n + \texttt{ADAM}(d\theta, n)$$

end while

Figure 18. *Training algorithm for Coulomb GANs.*

In order for generated samples to locally learn from real samples, sampling from both the training samples and generated samples is a must. Learning is done with gradients of the Coulomb potential field using a continuity equation, which tells whether samples move towards or away from a field or whether they are positive or negative. Moreover, another important aspect is the kernel, for Coulomb GANs. Generally, kernels are used to model the data patterns into clusters. A m-dimensional Plummer kernel, k, which is pushed to 0 is used, which has been proven to converge. Choosing the right m is crucial, since a large m could lead to unstable training [15].

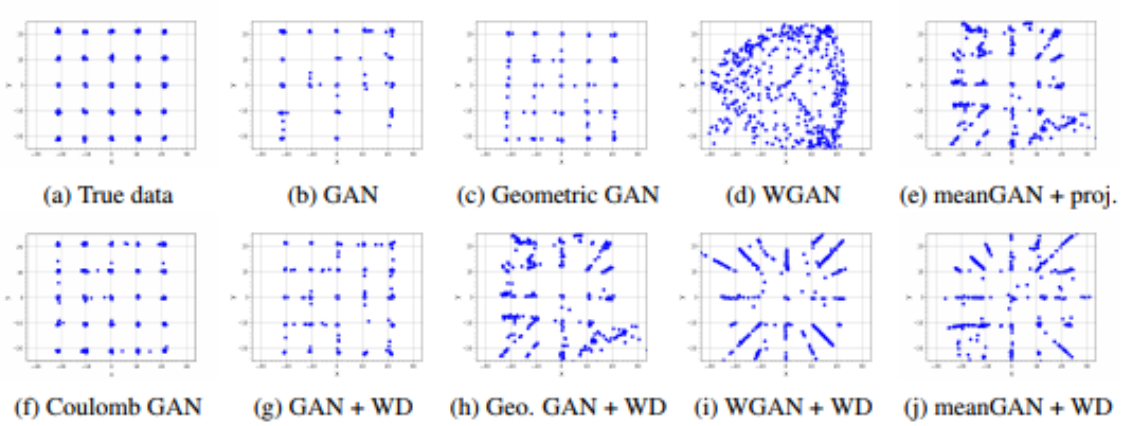


Figure 19. *Shows how the model is learned once training is finished. For Coulomb GANs, there are static well dispersed plots that show not only variation in data, but also that the generated samples can eventually become equivalent to the real samples. Coulomb GAN was able to outperform all the GANs mentioned above, except for the Wasserstein GAN (WGAN).*

Stacked Generative Adversarial Networks

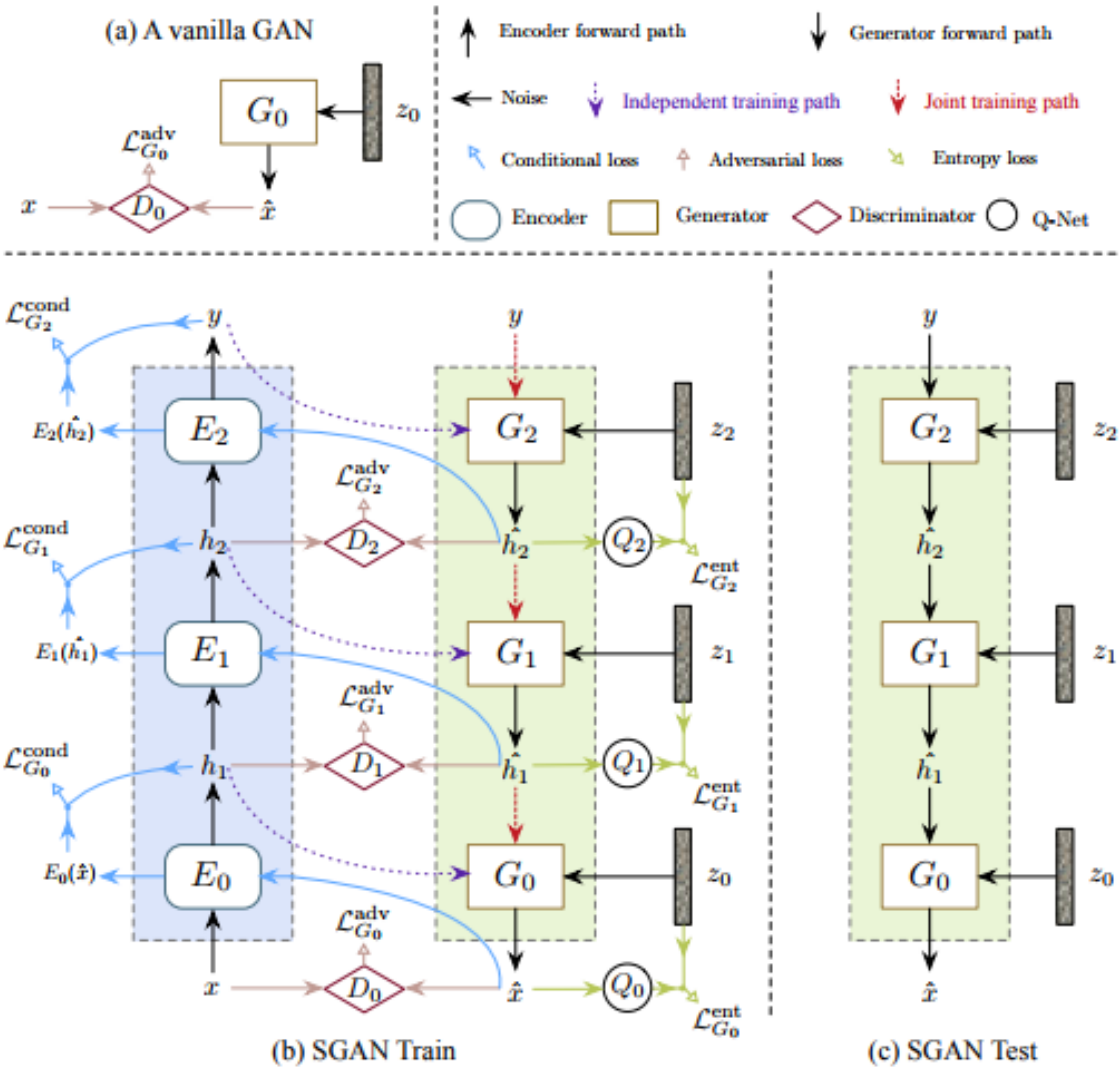


Figure 20. *Stacked GAN architecture used for supervised learning.*

Stacked architectures have shown high success in learning meaningful discriminative features in unsupervised learning settings and have outperformed standard GANs. Stacked GANs (S-GANs) propose a top-down generative model that could be used in both supervised and unsupervised settings, however, the implementation details mentioned are solely for supervised learning. The architecture setup consists of pre-training a deep neural network (DNN) of stacked encoders using a bottom-up approach. The first encoder takes an image, x, and encodes a non-linear mapping and passes it to the next encoder, and then eventually outputs a respective label y. Each encoder has a respective conditional loss.

The stacked generators are basically trained using a top-down approach to undo what each encoder has done. Each generator is trained independently and then it is trained end-to-end. A generator is given a real label, y, from an encoder, and it is given a random noise, z. Throughout the learning process, the generator captures lower level representations conditioned on higher level representations. High-level representations in images are denoted as attributes, objected categories, overall shape; they are not as detailed as low-level representations. Low level representations focus on specific details within an image. This concept allows generation of higher quality images. The encoders focus on grasping the high-level representations and the generators grasp the low-level representations.

Each generator learns by using 3 different losses: a conditional loss, an entropy loss, and an adversarial loss. The conditional loss is used to compute the higher-level representations using the low-level representations. An entropy loss is introduced to generate more diverse image samples. In order to create even more diverse samples, variational conditional entropy maximization is introduced, where the entropies between the lower-level representations and higher-level representations are maximized. An adversarial loss is introduced because of adversarial training [16].

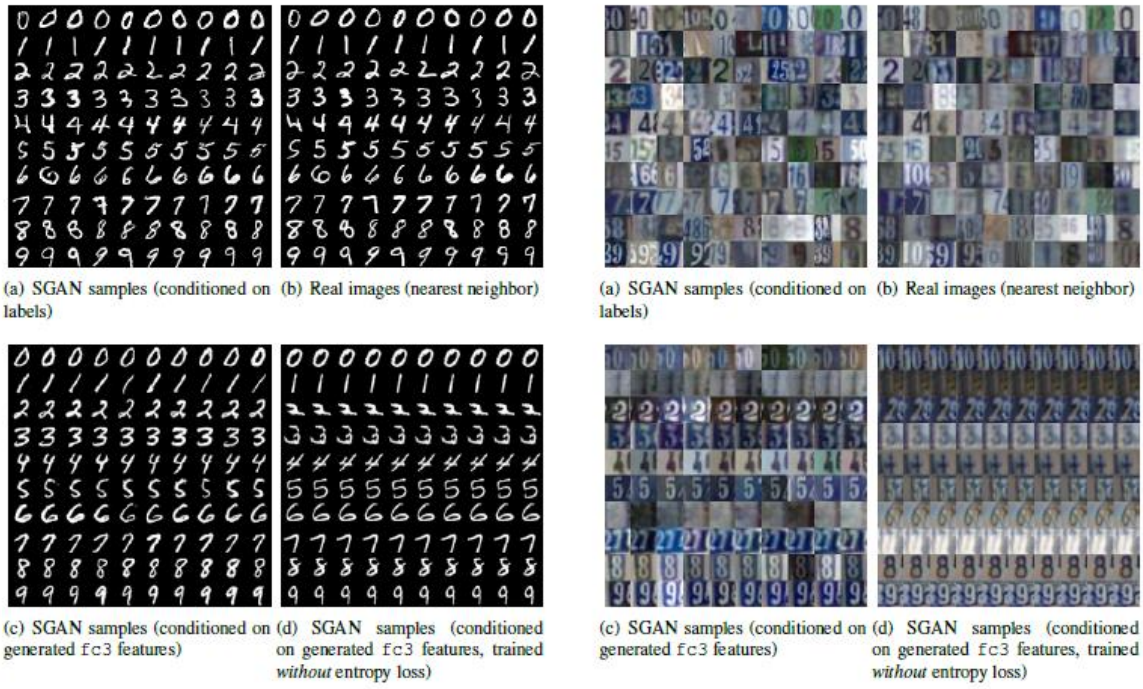


Figure 21. *S-GAN image generations results using the MNIST and SVHN datasets.*

Deep Hashing for Unsupervised Domain Adaptation

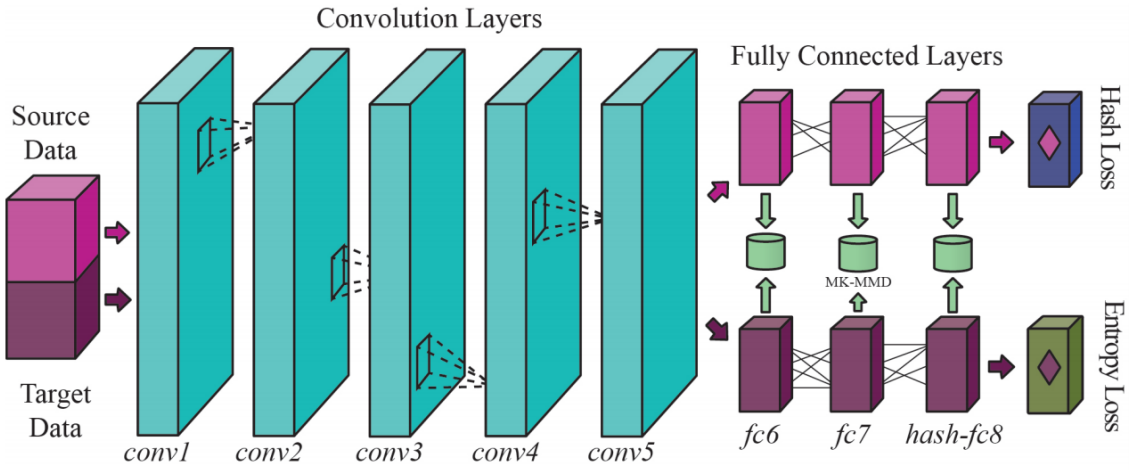


***Figure 22**. The Domain Adaptive Hash (DAH) architecture*.

In computer science, hashing is a very popular technique, since, in terms of run times, it has an O(1) access and requires small amounts of memory. Hashing techniques can take high dimensional data and transform it into compact binary codes, but not only that they can later assign these compact binary codes to other similar data. Although hashing techniques have deeply been explored in the computer science field, this paper proposes for the first time to use a hashing technique along with deep learning for domain adaptation.

Generally, a deep neural network is trained using a dataset and it outputs probability values for each specific class. However, in this paper, instead of outputting probability values, they output binary hash codes. The reason for doing that is that it provides a series of advantages. For instance, since in unsupervised domain adaptation, the target domain does not have any labels, hashing values allow to track a unique loss function for the unlabeled data and help achieve a highly robust category prediction.

The architecture, shown in Figure 22, is composed of 5 convolutional layers, which are conv1 through conv5, and 3 fully connected layers, which are fc6 through hash-fc8. The fc6 and fc7 are fined-tuned from the VGG-F network, meaning that they already have trained weights, which do not need much modification and are not randomly initialized. This helps since the network does not have to find the right weights, which could take some time.

As seen, there is a set of fully connected layers for both the source and target domain. The source domain's hash-fc8 outputs a hash loss and the target domain's hash-fc8 outputs an entropy loss via respective loss functions. The source domain's hash loss (supervised) approximates a hash value for an object specific to a category and the target domain's entropy loss (unsupervised) ensures that the target hash values are aligned and/or connected to the source categories.

A supervised hashing of the source data is introduced to assign each data point in the training data a specific hash value, which is calculated by using the Hamming distance. If the Hamming distance is small, it means that two given data points x_i and x_j are similar, however, if the distance is large then they are different. With this setup, data points could be assigned to specific label categories. For unsupervised hashing of the target data, each data point is assigned a specific category based on how similar the hash codes for the target domain and source domain data are. Between the fully connected layers, a multi-kernel Maximum Mean Discrepancy (MK-MDD) is used to make sure that the features in each of the layers for the source and target domains have the least amount of differences in domains. Thus, the network's loss function for domain adaptation is split up into 3 parts, which consist of supervised hashing, domain disparity reduction, and unsupervised hashing and it is shown below:

$$\min_{\mathcal{U}} \mathcal{J} = \mathcal{L}(\mathcal{U}_s) + \lambda \mathcal{M}(\mathcal{U}_s, \mathcal{U}_t) + \eta \mathcal{H}(\mathcal{U}_s, \mathcal{U}_t)$$

In terms of classification performance using the Office dataset, DAH was successful, however, it was slightly surpassed by DANN, as shown in Figure 23. The Office dataset consists of 4,110 images total and each of those images belong to one of 31 categories. Aside from building an unsupervised domain adaptation network, Venkateswara et al. made further contributions by constructing a new dataset called the Office-Home dataset. The Office-Home dataset, shown in Figure 25, contains images from 4 distinct domains, were each domain containing 64 categories, so this dataset has a total of 15,500 images. Classification accuracies were also recorded using the Office-Home dataset and as shown in Figure 24, DAH was able to surpass the other architectures in terms of accuracy. So, it seen that DAH may perform better on larger datasets. To further show the performance, DAH, features are measured in terms of how much difference there is between the source and target domains (discrepancy) and by using t-SNE visualization, shown in Figure 26 [17].

Expt.	A→D	A→W	D→A	D→W	W→A	W→D	Avg.
GFK	48.59	52.08	41.83	89.18	49.04	93.17	62.32
TCA	51.00	49.43	48.12	93.08	48.83	96.79	64.54
CORAL	54.42	51.70	48.26	95.97	47.27	98.59	66.04
JDA	59.24	58.62	51.35	96.86	52.34	97.79	69.37
DAN	67.04	67.80	50.36	95.85	52.33	99.40	72.13
DANN	72.89	72.70	56.25	96.48	53.20	99.40	75.15
DAH-e	66.27	66.16	55.97	94.59	53.91	96.99	72.31
DAH	66.47	68.30	55.54	96.10	53.02	98.80	73.04

Figure 23. *Shows accuracies to compare against previous state-of-the-art models used in domain adaptation from source to target using the Office dataset. The dataset is composed of 3 domain categories which are: Amazon (A), Dslr (D), and Webcam (W).*

Expt.	Ar→Cl	Ar→Pr	Ar→Rw	Cl→Ar	Cl→Pr	Cl→Rw	Pr→Ar	Pr→Cl	Pr→Rw	Rw→Ar	Rw→Cl	Rw→Pr	Avg.
GFK	21.60	31.72	38.83	21.63	34.94	34.20	24.52	25.73	42.92	32.88	28.96	50.89	32.40
TCA	19.93	32.08	35.71	19.00	31.36	31.74	21.92	23.64	42.12	30.74	27.15	48.68	30.34
CORAL	27.10	36.16	44.32	26.08	40.03	40.33	27.77	30.54	50.61	38.48	36.36	57.11	37.91
JDA	25.34	35.98	42.94	24.52	40.19	40.90	25.96	32.72	49.25	35.10	35.35	55.35	36.97
DAN	30.66	42.17	54.13	32.83	47.59	49.78	29.07	34.05	56.70	43.58	38.25	62.73	43.46
DANN	33.33	42.96	54.42	32.26	49.13	49.76	30.49	38.14	56.76	44.71	42.66	64.65	44.94
DAH-e	29.23	35.71	48.29	33.79	48.23	47.49	29.87	38.76	55.63	41.16	44.99	59.07	42.69
DAH	31.64	40.75	51.73	34.69	51.93	52.79	29.91	39.63	60.71	44.99	45.13	62.54	45.54

Figure 24. *Shows accuracies for Office-Home dataset for different domain adaptation networks compared to DAH.*

Figure 25. *Office-Home dataset.*

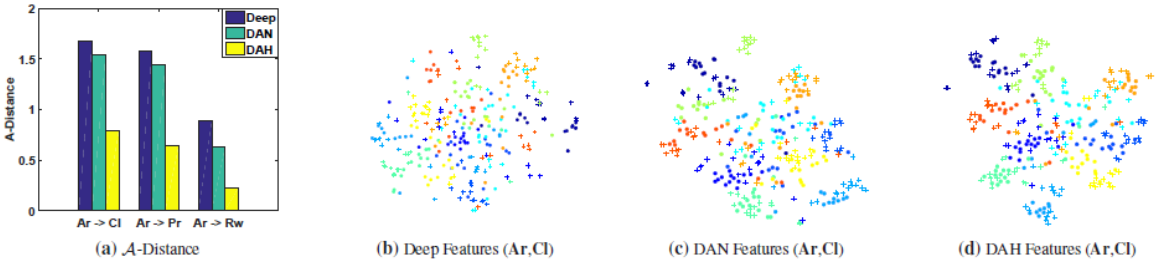


Figure 26. *It is seen that compared to other architectures DAH (part a) source and target domain images have minimal distinctions. In the t-SNE visualizations (b) through (d), it is also seen that there is more overlapping between domains for DAH compared to other architectures, which shows that DAH is a more effective domain adaptation method.*

Energy Based GANs

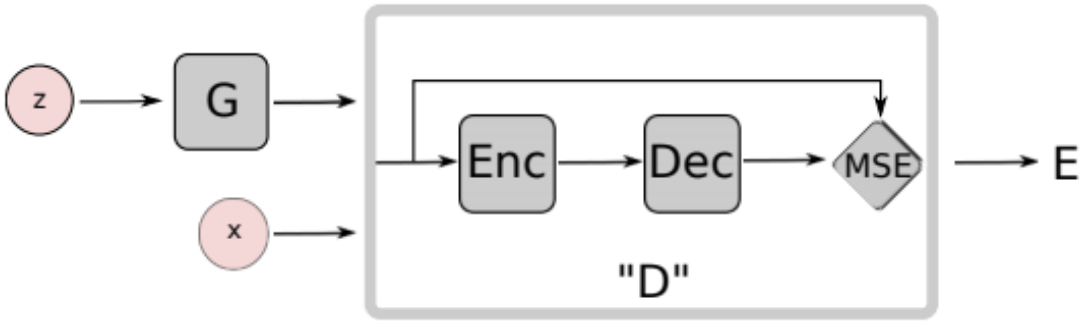


Figure 27. *EBGAN architecture.*

Energy Based GANs (EB-GANs) are purely based on an energy formulation, as denoted by its name, and are used in supervised and unsupervised learning. EB-GANs provide more stabilized training and higher-quality generated image samples when compared to standard GANs. Since EB-GANs are based on energies, there is no probabilistic interpretation whatsoever throughout learning. Instead original data samples, which are considered the correct configurations, are all given low energies, and the generated fake data samples, considered the incorrect configurations, are given high energies.

The architecture for an unsupervised EB-GAN is solely a single scale architecture and it has a generator and a discriminator that is made up of an auto-encoder. A random noise, z, is inputted into the generator, which tries to generate a fake sample with low energy, also referred to as a contrastive sample. The discriminator takes in $G(z)$ and x as inputs and encodes both images to latent vectors. These latent vectors are then decoded by the decoder and the reconstruction error calculated by the loss function, mean squared error (MSE) is updated. The discriminator's objective is to output high energies to generated samples.

The motivation for using autoencoders, even though they are not as efficient for image generation as opposed to standard GANs, is that there is no longer a binary logistic loss with 2 targets (real and fake) or [0,1], instead, the gradient could be denoted as a scalar that is not bounded. The reconstruction allows the gradient to go in different directions making generated images have more variability. The number of targets is denoted by a margin, m, meaning we bound our network [0, m], instead [0,1].

Choosing the right m is of high importance. Generally, some guidelines to choose the right m are looking at the discriminator's capacity and looking at how complex the dataset is. If m is too small, it can cause mode collapse because this tends to cause lower variation in images. However, if m is too large it can cause training to be more difficult. A suggested approach to choosing the right m, is to choose an

m that is large and gradually decrease the value of m until it reaches 0 throughout training.

The architecture also has the ability to use a “repelling regularizer” within the auto-encoder to avoid production of samples centered in few modes, making generated samples more variable. This repelling regularizer involves the Pulling-away Term (PT), which looks at the construction of a mini-batch and checks whether each sample representation is highly similar before putting it into a mini-batch.

Figure 28. *Shows images generated by DCGAN on the left and images generated EB-GAN-PT on the right. As it can be seen, EB-GAN-PT produces more variability in images.*

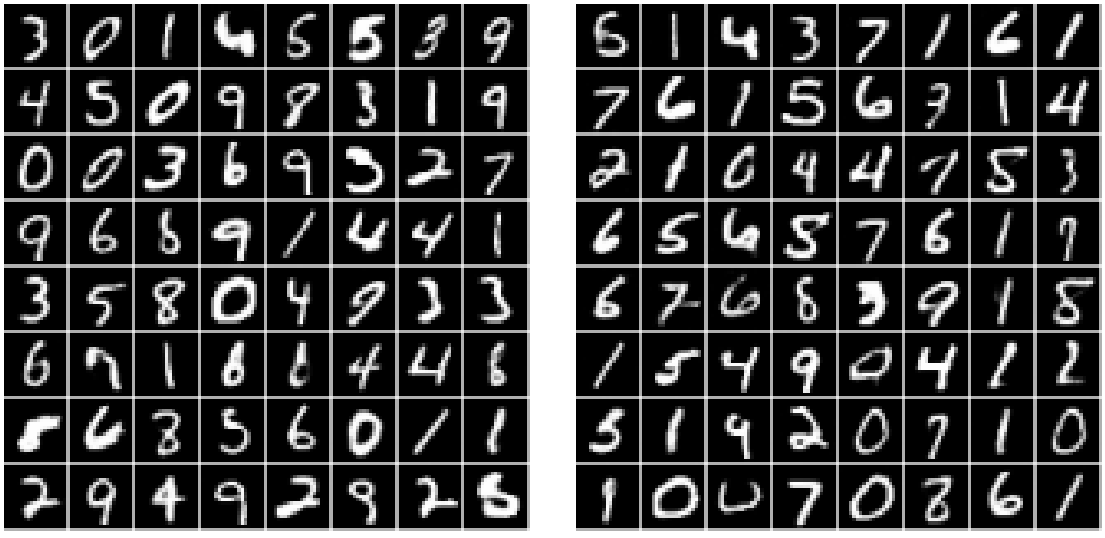


Figure 29. *EB-GAN-LN generated samples using the MNIST dataset.*

EB-GANs could not only be highly effective in supervised and unsupervised learning, Zhao et al. also explored EB-GAN-Ladder Network (EB-GAN-LN). This method makes use of generated contrastive samples, which helps semi-supervised learning overall since these samples are high quality samples with high variability that could be used throughout training [18].

From Source to Target and Back: Symmetric Bi-directional Adaptive GAN

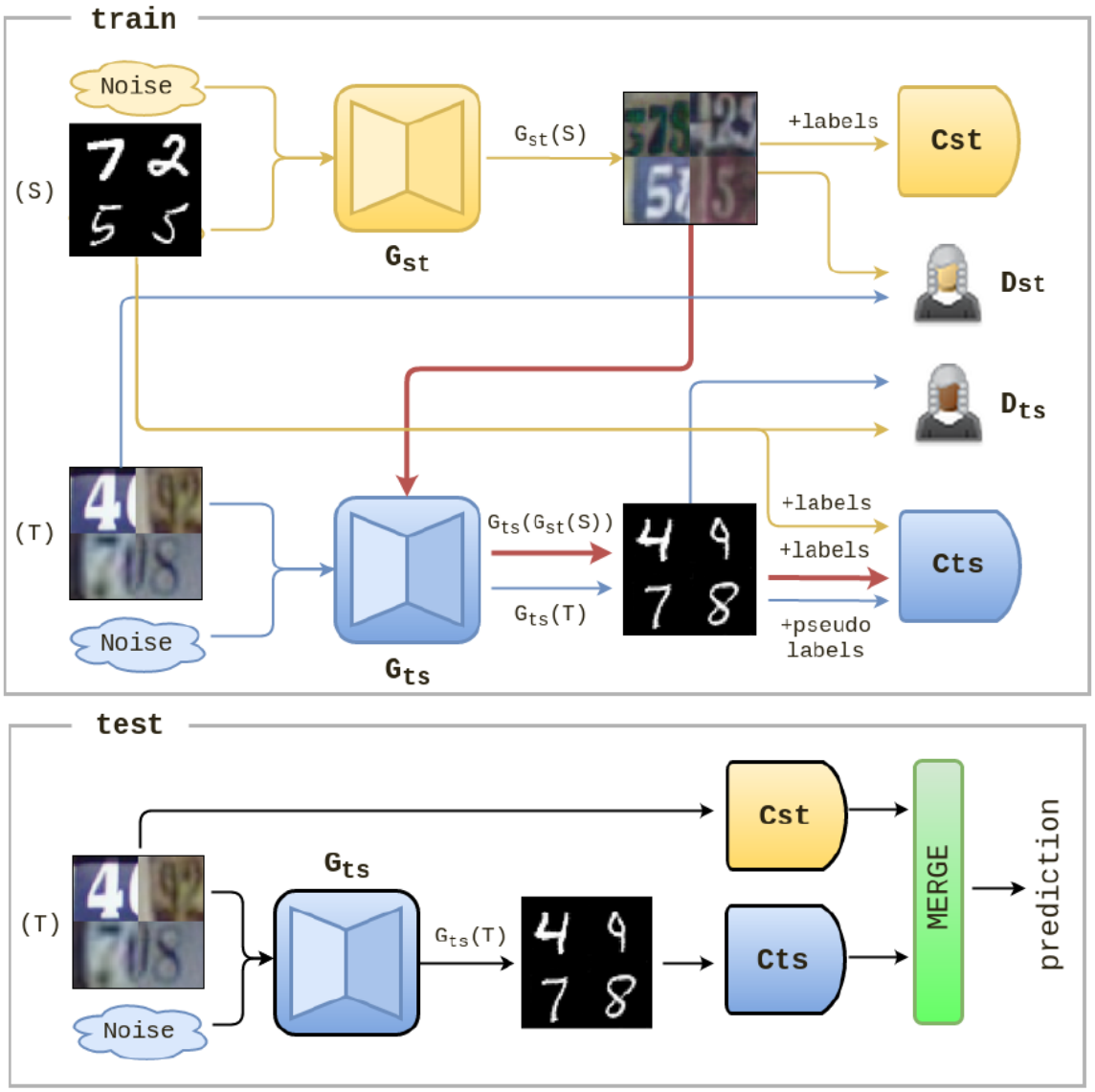


Figure 30. *SBADA-GAN architecture.*

The Symmetric Bi-Directional ADAptive Generative Adversarial Network SBADA-GAN proposes a new approach to domain adaptation. This architecture is composed of two generators, two discriminators, and two classifiers. Given a set of source domain images and their respective labels, $S = \{x_s, y_s\}$, and a set of target domain images without labels $T = \{x_T\}$, an image from the source domain is made to appear as it was from the target domain, just like Google's research paper. More systematically explained, a generator G_{ST} transforms images from the source domain to be like the ones from the target domain by inputting source domain images and a noise z, which is equivalent to $x_{ST} = G_{ST}(x_s)$. Once these

adapted images are generated they are inputted into a discriminator, D_{ST}, along with the original image target samples x_T. These image samples are then classified as being real domain adapted images or fake domain adapted images. Essentially, what happens is that throughout training samples, S, and samples from T will be correctly classified eventually. So, then the generated domain adapted images are inputted into a discriminator D_{ST}, which assigns the image a specific label referred to as a pseudo label.

In other domain adaptation, this would essentially be how overall training is done, however, with the following architecture they propose a symmetrical transformation system. Now, the generated target domain adapted images $G_{ST}(x_s)$ will be transformed to their original ground truth, which is X_s, by inputting them into a generator along with the respective pseudo labels $G_{TS}(G_{ST}(x_s))$. These newly source generated adapted images are then classified by the classifier C_{TS}. These generated images are evaluated by the discriminator D_{TS} to evaluate that the image generation actually goes back to the source domain. To aid in the learning process, instead of using a general image reconstruction loss, a class based reconstruction loss is used. The reason for this is that an image reconstruction loss tries to keep a lot of the input structure and this does not allow for changes in the images, however a class based reconstruction loss allows the various distinct features to be kept and has more liberty in image generation. The effectiveness of this is seen in Figure 31.c, where the class based reconstruction loss shows better image samples as opposed to the ones in Figure 31.a, with no reconstruction loss, and Figure 31.b, an image based reconstruction loss. Hence, we have the following objective function for SBADA-GAN:

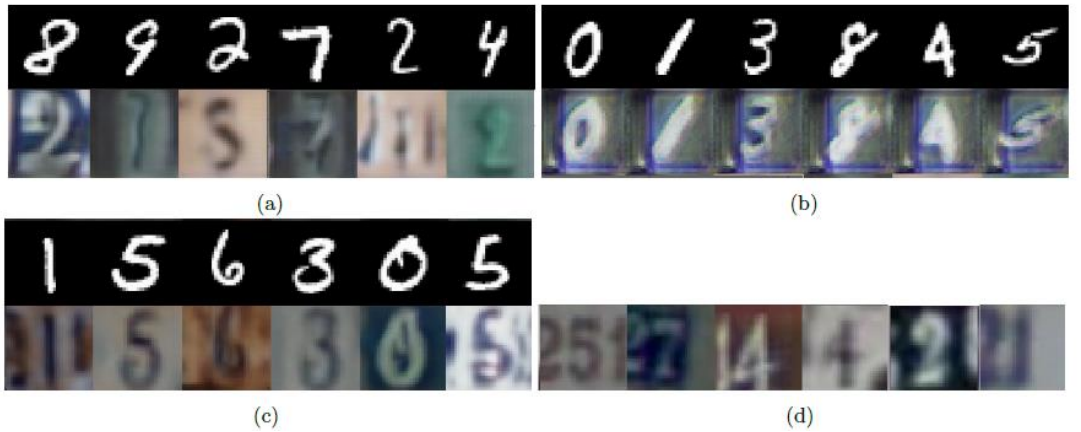


Figure 31. *Top row for sub figures show the original source domain images and bottom rows show the generated adapted target domain images. (a) Shows generated images with no reconstruction. (b) Shows generated images with an image-based cycle consistency loss. (c) Shows generated images using a class based reconstruction loss. (d) Shows original image samples from SVHN dataset.*

Furthermore, classification accuracies between domains are calculated to measure overall performance of domain adaptation using SBADA-GAN direct, which denotes domain adaptation from source to target domain, SBADA-GAN

inverse, which denotes domain adaption from the target to the source domain, and the general SBADA-GAN framework. It is seen that SBADA outperforms all of the previous state of the art models for domain adaptation, specifically Google's network, in all, but one dataset. The reason for this is that the other networks focused on using a non-RGB (not colored) SVHN dataset and instead used a grey-scale version, which is less difficult to train. While SBADA-GAN used the original RGB images from SVHN dataset. t-SNE visualizations are also viewed as a metric to evaluate domain adaptation shown in Figure 33 [19].

	MNIST to USPS	USPS to MNIST	MNIST to MNISTM	SVHN to MNIST	MNIST to SVHN
Source Only	78.9	57.1 ± 1.7	63.6(56.6)	60.1 ± 1.1	26.0 ± 1.2
CORAL[26]	81.7		57.7	63.1	
MMD [29]	81.1		76.9	71.1	
DANN [8]	85.1	73.0 ± 2.0	77.4	73.9	
DSN [3]	91.3		83.2	82.7	
CoGAN [17]	91.2	89.1 ± 0.8	62.0	no converge	
ADDA [28]	89.4 ± 0.2	90.1 ± 0.8		76.0 ± 1.8	
GenToAdapt [24]	92.5 ± 0.7	90.8 ± 1.3		84.7 ± 0.9	36.4 ± 1.2
DRCN [10]	91.8 ± 0.09	73.7 ± 0.04		82.0 ± 0.16	40.1 ± 0.07
GOOGLE [2]	95.9	–	98.2		
Target Only	96.5		96.4(95.9)	99.5	
SBADA-GAN direct	96.66	94.43	99.13	72.20	59.21
SBADA-GAN inverse	97.10	87.47	98.37	74.2	50.85
SBADA-GAN	**97.60**	**95.04**	**99.39**	76.14	**61.08**

Figure 32. *Classification accuracy comparison to other domain adaptation state-of-the-art models.*

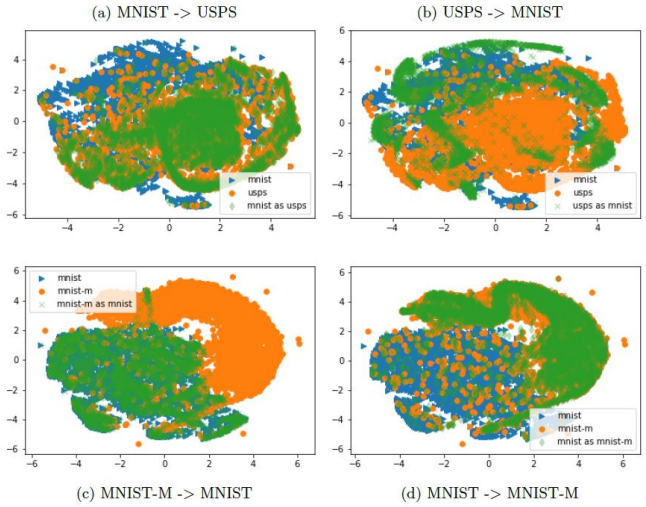


Figure 33. *The t-SNE visualization shows how the source, target, and mappings from source to target domain are overlapping. Hence showing the success of SBADA-GAN.*

Asymmetric Tri-training for Unsupervised Domain Adaptation

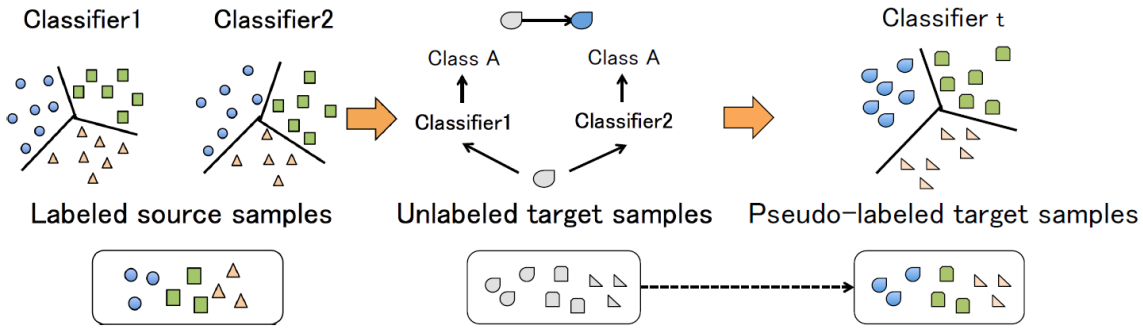


Figure 34. *The architecture involves 3 classifiers, which are shown above. Classifier 1 and 2 focus on learning the labels for the source domain images and Classifier t gives pseudo labels to the unlabeled target domain images.*

Previous methods have used co-training for domain adaptation, which involves the 2 different classifiers to assign pseudo labels to unlabeled target images. Saito et al. propose an asymmetric tri-training method to adapt images from a source domain to a target domain. Asymmetric means that there will be 3 classifiers used instead of 2 classifiers as used in co-training.

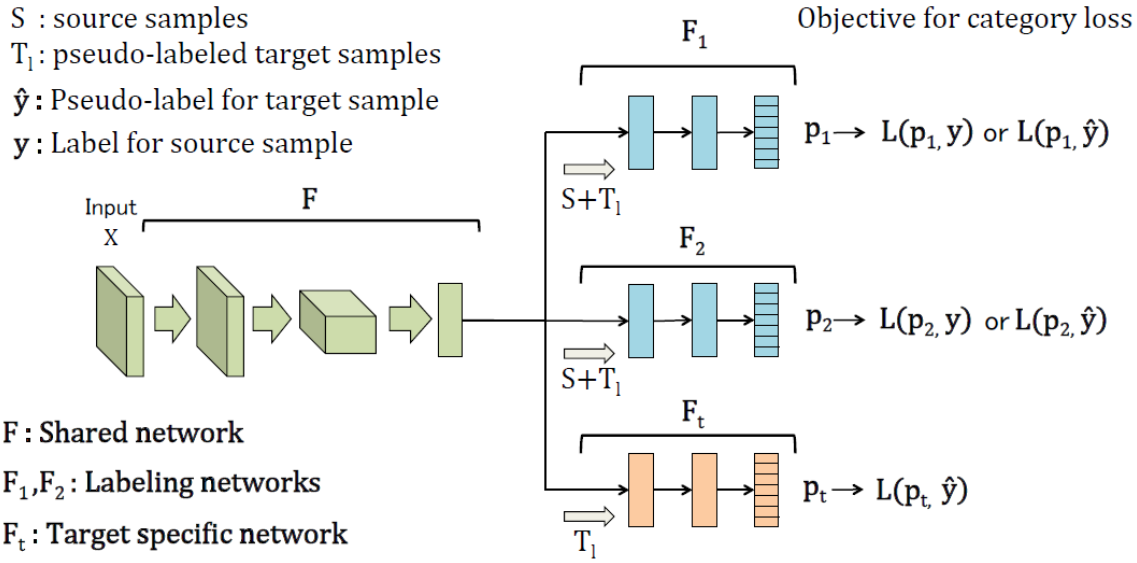


Figure 35. *The CNN network architecture used for unsupervised domain adaptation.*

The network architecture consists of 1 CNN network, F, which then splits into 3 sub-networks $F_1, F_2,$ and F_3.The F network takes in a set of images from both the source (labeled) and target domain (unlabeled). This network acts as a feature extractor and outputs a set of features with applied batch normalization in the last

layer. Batch normalization is used because it increases the overall performance of the network and it also increases the accuracy of the network. These features are then used in throughout sub-networks, which contain classifiers C_1, C_2, and C_t. Other methods usually partition features into different views, however, this method does not do that. Instead it initializes each classifier differently, so that each classifier obtains different sets of features from images.

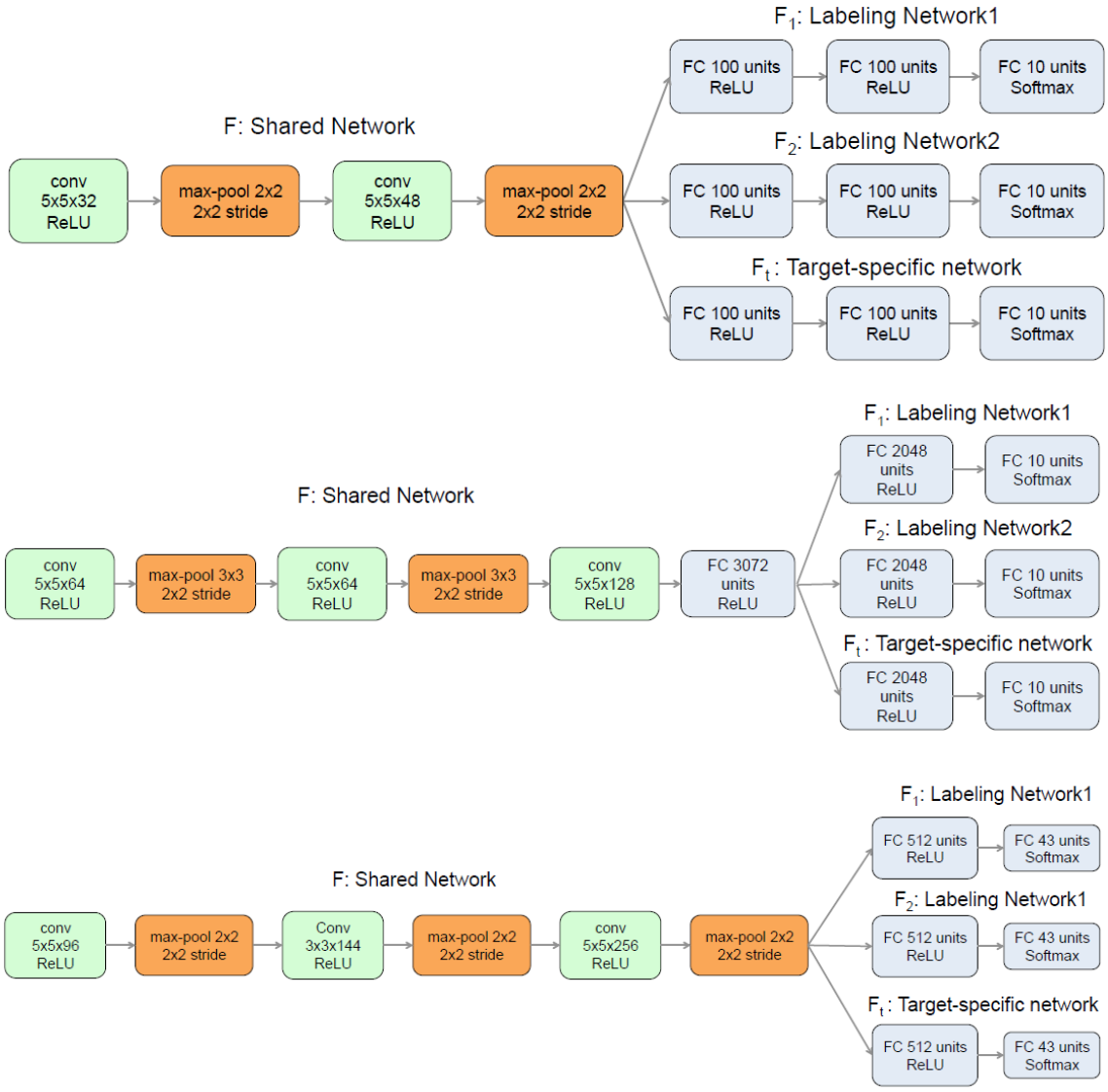


Figure 36. *Shows the detailed network architectures used for training and testing and testing domain adaptation in the MNIST, SVHN, and Synthetic Signs datasets.*

F_1 and F_2 classify features from F by outputting predictions. Their predictions help in giving pseudo labels to the unlabeled target samples. In order to give pseudo labels, both classifiers have to either agree on a specific label or at least one of the classifiers has to have a high confidence in assigning a specific pseudo label. In the paper, for different experiments the threshold was set to probabilities 0.9 or 0.95 and varied throughout their experiments, where 0 means low confidence and

1 means high confidence. The main objective of the networks F_1 and F_2 are to learn target-discriminative representations, which will help in the target images classification process. The F_T network assigns a pseudo label to a given target image. The idea is that both the real labels and pseudo labels will be merged together and be regarded as labels overall to increase the overall classification of target images.

METHOD / SOURCE	MNIST	SVHN	MNIST	SYN DIGITS	SYN SIGNS
TARGET	MNIST-M	MNIST	SVHN	SVHN	GTSRB
Source Only w/o BN	59.1(56.6)	68.1(59.2)	37.2(30.5)	84.1(86.7)	79.2(79.0)
Source Only with BN	57.1	70.1	34.9	85.5	75.7
MMD (Long et al., 2015b)	76.9	71.1	-	88.0	91.1
DANN (Ganin & Lempitsky, 2014)	81.5	71.1	35.7	90.3	88.7
DRCN (Ghifary et al., 2016)	-	82.0	40.1	-	-
DSN (Bousmalis et al., 2016)	83.2	82.7	-	91.2	93.1
kNN-Ad (Sener et al., 2016)	86.7	78.8	40.3	-	-
Ours w/o BN	85.3	79.8	39.8	**93.1**	**96.2**
Ours w/o weight constraint ($\lambda = 0$)	**94.2**	**86.2**	49.7	92.4	94.0
Ours	94.0	85.0	**52.8**	92.9	**96.2**

Figure 37. *Shows classification results of the asymmetric tri-training network used compared to other similar state-of-the-art models.*

As shown in Figure 37, this network was able to surpass other similar methods by a good margin. The only domain adaptation set up they had difficulty obtaining a high accuracy was for MNIST → SVHN. The reason being is that the SVHN dataset images contain more than 1 number in each image, and the MNIST dataset only has 1 number per image. More results in Figure 38 below, show that domain adaptation between MNIST→ MNIST-M is relatively easier to achieve since the clusters overlap.

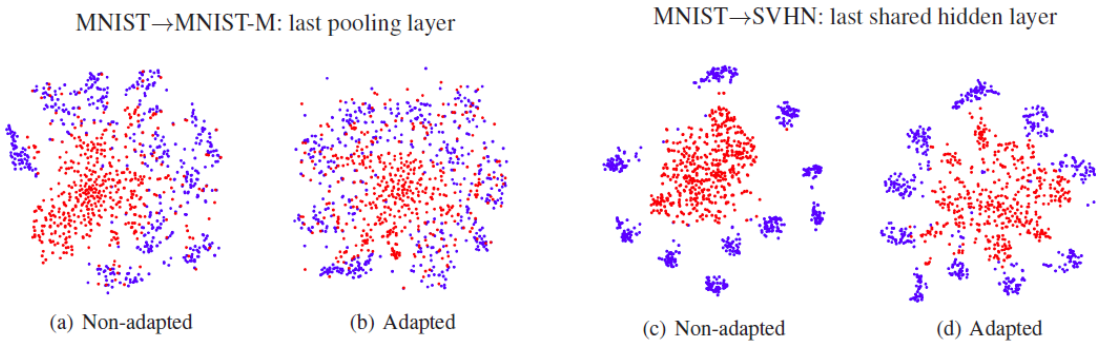
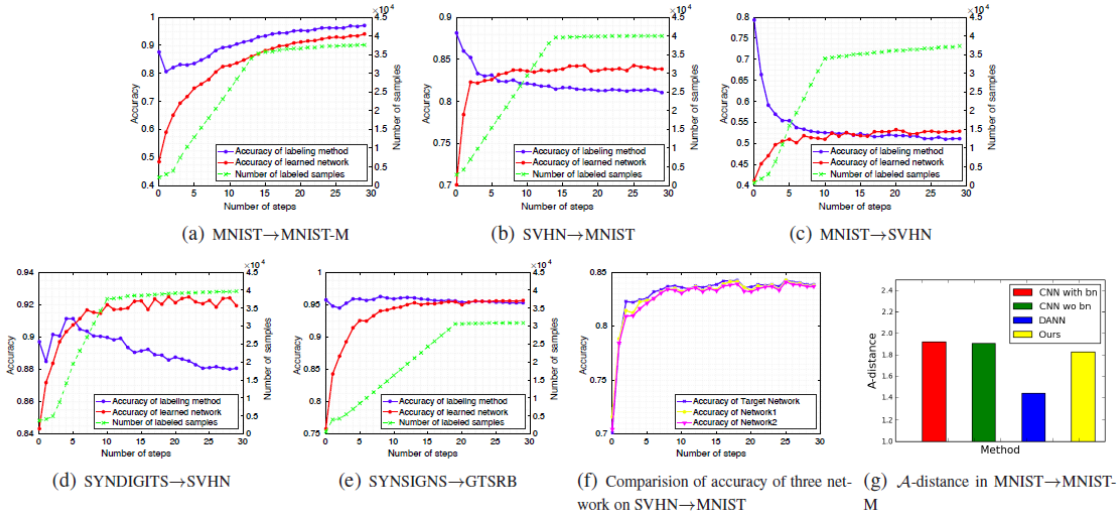


Figure 38. *Show t-SNE visualizations of domain adaptation set-ups.*



(a) MNIST→MNIST-M (b) SVHN→MNIST (c) MNIST→SVHN

(d) SYNDIGITS→SVHN (e) SYNSIGNS→GTSRB (f) Comparision of accuracy of three network on SVHN→MNIST (g) $\mathcal{A}$-distance in MNIST→MNIST-M

Figure 39. *Shows accuracy graphs and measure A-distance to show effectiveness of the method proposed in this paper.*

Figure 39, shows various quantitative metrics that show the performance of the networks used. For subfigures (a) through (e), accuracies are measured for each domain adaptation set up. The green line shows target labelled samples for each iteration. The blue line shows pseudo label accuracy during the training phase. Finally, the red line shows the accuracy learned by the overall network. One thing to note is that the overall accuracy seems to constantly grow and that is due to a very powerful factor in the network architecture. When pseudo labels are assigned, only pseudo labels that are deemed accurate are saved in the overall quantity of labels, the rest are discarded. This allows the network to not be tricked and augment the performance rate overtime. In subfigure (f), the three subnetworks were compared, and it is seen that all three achieve similar high-performance accuracy. This shows that all three networks are learning target-specific discriminative features. The final subfigure (g) shows the evaluation of the A-distance for the domain adapted samples versus the real target samples, however, here this method does not surpass other methods. However, this metric is not a valid measure of how well the network performs [20].

Generate to Adapt: Aligning Domains Using Generative Adversarial Networks

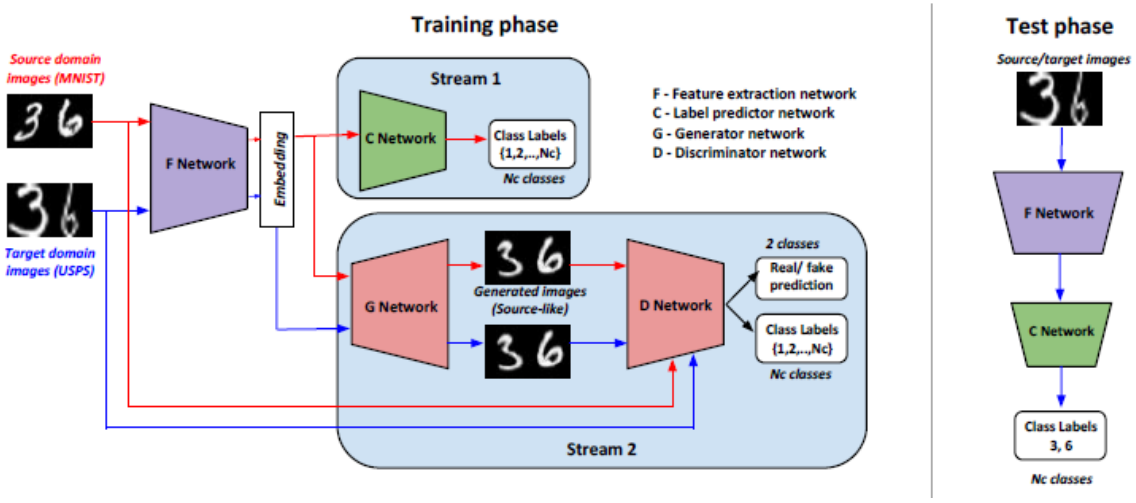


Figure 40. *GAN network architecture used for both training and testing phases.*

Sankaranarayanan et al. propose a new framework using GANs and streams for domain adaptation. The basic idea of this network is that since there are no labels for the target domain, an alignment between the source domain and target domain distribution can be learned using an embedding. Throughout the training and testing phases, two different network architectures are used, as shown in Figure 40.

The training phase network takes in a source image into a network F and forms an embedding. This embedding consists of the embedded image into a feature vector, a random noise z, and a class label. This embedding vector is inputted into 2 different streams. Stream 1 consists of a classifier network C, which takes this embedding vector and outputs a respective label in a supervised manner. In Stream 2, the embedding vector is inputted into a network G, which generates a fake source samples. The discriminator network D, then takes the generated fake source samples and the original images as input. The network D then outputs a probability of whether a given image is fake or real, D_{data} and it also outputs the respective class for the image, D_{cls}. Both factors are used to backpropagate the gradients, so the network can learn from the source domain images.

The training phase for the target domain images is a little bit different. The target domain images get inputted into a network F and this network outputs an embedding vector comprised of an embedding, random noise, and since the labels for the classes are not known, instead a binary label is used, to point whether that image is fake or real. Then this embedding vector is only inputted into one stream, Stream 2. So, the embedding vector gets inputted into a network G, which then generates a fake sample. A fake sample and real sample from the target domain are inputted into the network D. However, at this point network D can only output

whether an image is fake or real because the target images have no labels. So, only gradients that are backpropagated are only for D_{data}.
For the testing phase, the concepts of the streams is no longer used. A set of images are inputted into the network F. The network F then outputs a specific label for each respective image inputted using the classifier network C.

Algorithm 1 Iterative training procedure of our approach

1: training iterations = N
2: **for** t in 1:N **do**
3: Sample k images with labels from source domain $\mathcal{S}$: $\{s_i, y_i\}_{i=1}^{k}$
4: Let $f_i = F(s_i)$ be the embeddings computed for the source images.
5: Sample k images from target domain $\mathcal{T}$: $\{t_i\}_{i=1}^{k}$
6: Let $h_i = F(t_i)$ be the embeddings computed for the target images.
7: Sample k random noise samples $\{z_i\}_{i=1}^{k} \sim \mathcal{N}(0,1)$.
8: Let f_{g_t} and h_{g_t} be the concatenated inputs to the generator.
9: Update discriminator using the following objectives:

$$L_D = L_{data,src} + L_{cls,src} + L_{adv,tgt} \tag{9}$$

- $L_{data,src} = \max_D \frac{1}{k}\sum_{i=1}^{k} \log(D_{data}(s_i)) + log(1 - D_{data}(G(f_{g_t})))$
- $L_{cls,src} = \max_D \frac{1}{k}\sum_{i=1}^{k} \log(D_{cls}(s_i)_{y_t})$
- $L_{adv,tgt} = \max_D \frac{1}{k}\sum_{i=1}^{k} \log(1 - D_{data}(G(h_{g_t})))$

10: Update the generator, only for source data, through the discriminator gradients computed using real labels.

$$L_G = \min_G \frac{1}{k}\sum_{i=1}^{k} -\log(D_{cls}(G(f_{g_t}))_{y_t}) + \log(1 - D_{data}(G(f_{g_t}))) \tag{10}$$

11: Update the embedding F using a linear combination of the adversarial loss and classification loss. Update the classifier C for the source data using a cross entropy loss function.

$$L_F = L_C + \alpha L_{cls,src} + \beta L_{F_{adv}} \tag{11}$$

- $L_C = \min_C \min_F \frac{1}{k}\sum_{i=1}^{k} -\log(C(f_i)_{y_t})$
- $L_{cls,src} = \min_F \frac{1}{k}\sum_{i=1}^{k} -\log(D_{cls}(G(f_{g_t}))_{y_t})$
- $L_{F_{adv}} = \min_F \frac{1}{k}\sum_{i=1}^{k} \log(1 - D_{data}(G(h_{g_t})))$

12: **end for**

Figure 41. *Shows the algorithm setup for the training of the architecture.*

Further explained, to be able to optimize the networks $C, D, G,$ and F, such that they can learn an embedding, there are a series updates that need to be done throughout the training session. As previously discussed, when network D is given a set of source images it outputs both D_{data} and D_{cls}, which are optimized by a binary cross entropy and a cross entropy loss. Then the network G is updated using the gradients of D with both an adversarial loss and a classification loss, so that it can produce class-specific high-quality samples. The networks F and C are updated in a supervised manner by using the source images and source labels. The adversarial gradients from network D also serve to update F. This allows for cutting-edge learning of features and generation of images. To be able to reach the goal of domain adaptation using the embedding learned, the gradients from D_{data} for the target images serve to update F.

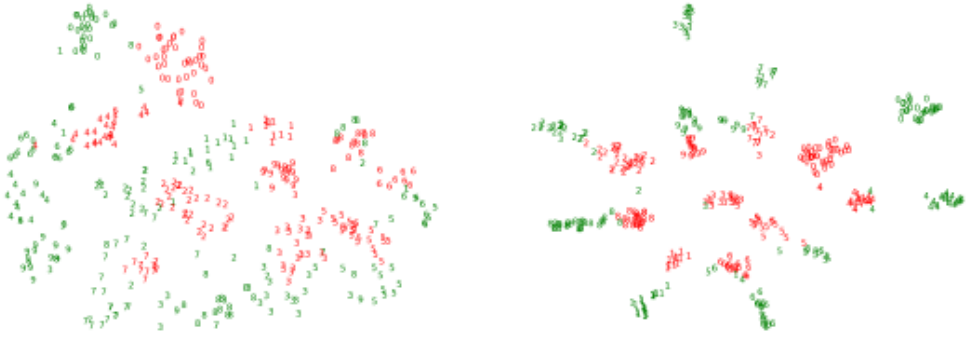
Method	MN → US (p)	MN → US (f)	US → MN	SV → MN
Source only	75.2 ± 1.6	79.1 ± 0.9	57.1 ± 1.7	60.3 ± 1.5
RevGrad [4]	77.1 ± 1.8	-	73.0 ± 2.0	73.9
DRCN [5]	91.8 ± 0.09	-	73.7 ± 0.04	82.0 ± 0.16
CoGAN [14]	91.2 ± 0.8	-	89.1 ± 0.8	-
ADDA [29]	89.4 ± 0.2	-	90.1 ± 0.8	76.0 ± 1.8
PixelDA [1]	-	**95.9**	-	-
Ours	**92.8 ± 0.9**	95.3 ± 0.7	**90.8** ± 1.3	**92.4** ± 0.9

Figure 42. *Classification accuracies for the domain adaptation using the MNIST, USPS, and SVHN datasets. (MN: MNIST, US: USPS, SVHN: SV).*

Method	A → W	D → W	W → D	A → D	D → A	W → A	Average
ResNet - Source only [9]	68.4 ± 0.2	96.7 ± 0.1	99.3 ± 0.1	68.9 ± 0.2	62.5 ± 0.3	60.7 ± 0.3	76.1
TCA [22]	72.7 ± 0.0	96.7 ± 0.0	99.6 ± 0.0	74.1 ± 0.0	61.7 ± 0.0	60.9 ± 0.0	77.6
GFK [6]	72.8 ± 0.0	95.0 ± 0.0	98.2 ± 0.0	74.5 ± 0.0	63.4 ± 0.0	61.0 ± 0.0	77.5
DDC [30]	75.6 ± 0.2	76.0± 0.2	98.2 ± 0.1	76.5 ± 0.3	62.2 ± 0.4	61.5 ± 0.5	78.3
DAN [15]	80.5 ± 0.4	97.1 ± 0.2	99.6 ± 0.1	78.6 ± 0.2	63.6 ± 0.3	62.8 ± 0.2	80.4
RTN [17]	84.5 ± 0.2	96.8 ± 0.1	99.4 ± 0.1	77.5 ± 0.3	66.2 ± 0.2	64.8 ± 0.3	81.6
RevGrad [4]	82.0 ± 0.4	96.9 ± 0.2	99.1 ± 0.1	79.4 ± 0.4	68.2 ± 0.4	67.4 ± 0.5	82.2
JAN [18]	85.4 ± 0.3	97.4 ± 0.2	**99.8** ± 0.2	84.7 ± 0.3	68.6 ± 0.3	70.0 ± 0.4	84.3
Ours	**89.5** ± 0.5	**97.9** ± 0.3	**99.8** ± 0.4	**87.7** ± 0.5	**72.8** ± 0.3	**71.4** ± 0.4	**86.5**

Figure 43. *Classification accuracies using the Office dataset with images in different domains (A: Amazon, W: Webcam, D: DSLR).*

Moreover, this network has achieved state-of-the-art accuracy results compared to other networks for not only domain adaptation within the MNIST, USPS, and SVHN datasets, but also for the Office dataset. This is quite an achievement as GANs typically require a vast amount of data to obtain high accuracies, which shows the network has successfully learned to adapt images.



(a) Non adapted (b) Adapted

Figure 44. *t-SNE visualization of domain adaptation from SVHN to MNIST.*

Further metrics such as t-SNE visualization are used to visualize both the source and target domain distributions. In Figure 44 (a), shows a poor domain adapted t-SNE visualization, while (b) shows a very well adapted visualization, where there are clusters that represent different classes. Within these clusters, it is seen that the data points for both domains have overlapping.

Setting	Accuracy(in %)
Stream 1 - Source only	68.4
Stream 1 + Stream 2 (C_1 only)	80.5
Stream 1 + Stream 2 ($C_1 + C_2$)	89.5

Figure 45. *Classification accuracies using different modifications to the proposed architecture with the Office dataset.*

More experiments were conducted to see the effectiveness of the overall network. By using only Stream 1 and using the source domain images to train, the network achieves an accuracy of 68.4%, which is relatively poor. However, using Stream 1 and Stream 2 with only the real and fake classifier the accuracy increases drastically to 80.5%, Lastly, by using the whole architecture proposed, it achieves an even more outstanding accuracy, 89.5%. With this experiment it is seen that all streams and attributes within the network are needed to achieve optimal results [21].

Unsupervised Domain Adaptation by Backpropagation

A frequent problem for deep feed-forward architectures is the scarcity of large amounts of labeled data. It may be possible to obtain training sets that are big enough to train large-scale deep models, even if problems lack labeled data. A drawback to this process is the possibility of creating a shift in the data distribution compared to actual data encountered at test time. An example of this is the training of a network on totally synthetic data, where this training of the network on synthetic data will inevitably have a different distribution from real data. Learning a classifier when there is a shift between training data and testing data is known as domain adaptation. This research paper focuses on combining domain adaptation and feature learning in one training process called deep domain adaptation [22]. The goal is to embed domain adaptation with learning representations, as this will make the classification much more efficient since it will be based on both discriminativeness and the invariance of changing domains. Using this approach will help the feed forward network be applicable to the target domain, ignoring the shift between the source and target domain.

This basically will focus on two things:

1) Discriminativeness
2) Domain-invariance

For this approach to work, two classifiers will be needed:

1) Classifier 1 with a label predictor used for both training and testing of images which will predict the class labels
2) Classifier 2 with a domain discriminator which will discriminate between source and target domain for training.

The proposed architecture takes as an input, sample images $x \epsilon X$, and will output labels $y \epsilon Y$. There will be two distributions $S(x, y)$ and $T(x, y)$ on $X \otimes Y$ where S is the source distribution and T is the target distribution. The proposed architecture for deep domain adaptation looks like in Figure 46 below.

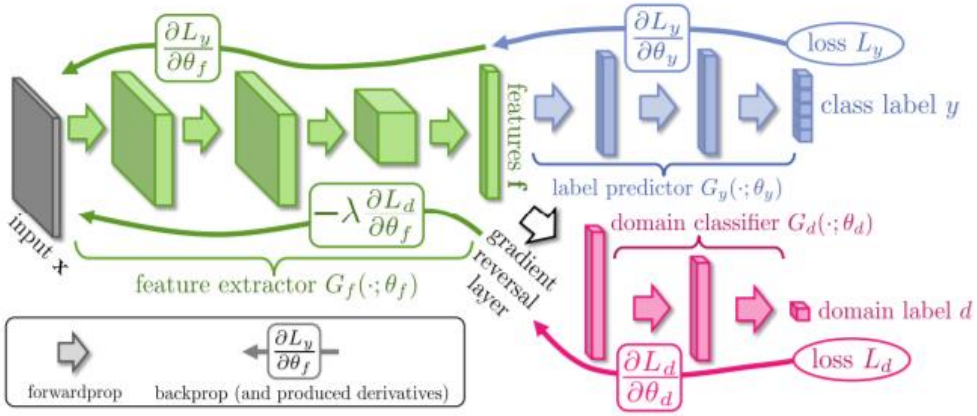


Figure 46. *Proposed deep domain adaptation architecture.*

The goal is to predict the labels y when an input x from the target domain is provided. Labeled source images are used to train this network. During training there are no labels for the target domain, but at test time, labels for the target domain will be predicted.

The deep feed-forward architecture not only does the labelling $y \epsilon Y$, but also gives the domain label $d \epsilon \{0,1\}$.

The input x is mapped to G_f (a feature extractor) to a D-dimensional feature vector$f \epsilon R^D$. Several feed-forward layers are present for feature mapping, and the vector of parameters in this layer is θ_f, i.e. $f = G_f(x; \theta_f)$. After the input is mapped, the feature vector f is mapped by G_y, and the parameters of this mapping are denoted by θ_y. Lastly, the feature vector f is mapped to the domain label d by mapping G_d with parameters θ_d.

During the learning stage, the goal is to minimize the label prediction loss on source images of the training set, meaning the parameters of the feature extractor and label predictor are optimized to reduce empirical loss for source images. The feature vector f needs to be domain-invariant, so the distributions look like:

$$S(f) = \{G_f(x; \theta_f) | x \sim S(x)\} \; and \; T(f) = \{G_f(x; \theta_f) | x \sim T(x)\}$$

While training, to get domain-invariant features, θ_f is seeked to make two feature distributions as similar as possible to maximize the loss of the domain classifier. Simultaneously, θ_d is seeked to minimize loss of the domain classifier and loss of the label predictor. The final function looks like:

$$E(\theta_f, \theta_y, \theta_d) = \sum_{\substack{i=1..N \\ d_i=0}} L_y\big(G_y\big(G_f(x_i; \theta_f); \theta_y\big), y_i\big) - \lambda \sum_{i=1..N} L_d\big(G_d\big(G_f(x_i; \theta_f); \theta_d\big), y_i\big)$$

$$= \sum_{\substack{i=1..N \\ d_i=0}} L_y^i(\theta_f, \theta_y) - \lambda \sum_{i=1..N} L_d^i(\theta_f, \theta_d)$$

where L_y is the label prediction loss and L_d is the domain classification loss. The saddle point is also seeked of the function above from parameters $\theta_f, \theta_y, \theta_d$.

$$(\hat{\theta}_f, \hat{\theta}_y) = arg \min_{\theta_f, \theta_y} E(\theta_f, \theta_y, \hat{\theta}_d)$$

$$\hat{\theta}_d = arg \max_{\theta_d} E(\hat{\theta}_f, \hat{\theta}_y, \theta_d)$$

A saddle will be a stationary point of the following parametric updates:

$$\theta_f \leftarrow \theta_f - \mu\left(\frac{\delta L_y^i}{\delta \theta_f} - \lambda \frac{\delta L_d^i}{\delta \theta_f}\right)$$

$$\theta_y \leftarrow \theta_y - \mu\left(\frac{\delta L_y^i}{\delta \theta_y}\right)$$

$$\theta_d \leftarrow \theta_d - \mu\left(\frac{\delta L_d^i}{\delta \theta_d}\right)$$

where μ acts as a learning rate.

A gradient reversal layer is inserted between the feature extractor and domain classifier. Backpropagation then processes the features through the gradient reversal layer. The partial derivative of the loss is the downstream of the gradient reversal layer and the parameters are the upstream of the gradient reversal layer, so running stochastic gradient descent converges to a saddle point. The gradient reversal formulation looks like:

$$R_\lambda(x) = x$$
$$\frac{dR_\lambda}{dx} = -\lambda I$$

where I is the identity matrix. Therefore, the function E, after being optimized, looks like:

$$\bar{E}(\theta_f, \theta_y, \theta_d) = \sum_{\substack{i=1..N \\ d_i=0}} L_y\big(G_y\big(G_f(x_i;\theta_f)\theta_y\big), y_i\big) + \sum_{i=1..N} L_d\left(G_d\left(R_\lambda\left(G_f(x_i;\theta_f)\right);\theta_d\right), y_i\right)$$

The feature extractor is composed of two to three convolutional layers and chosen configurations from pervious works. The domain Adaptor contains three connected layers, but for the MNIST dataset, a two-layered architecture is used. The loss functions L_y and L_d are set to logistic regression loss and binomial cross-entropy.

Training is done on 128 sized batches. Half of the images in the batch are labeled from the source domain and half are un-labeled from the target domain. The adaptation factor λ is gradually changed between 0 to 1 for training to suppress noise signal.

$$\lambda_p = \frac{2}{1 + \exp(-\gamma \cdot p)} - 1$$

t-SNE projection is used to visualize feature distribution with color coding.
Figure 47 shows the effect of adaptation on the distribution of extracted features.

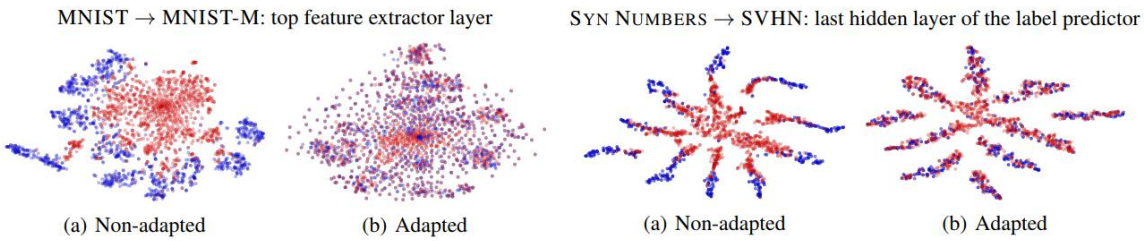


Figure 47. *Blue represents the source domain and red represents the target domain.*

Every case was trained on a certain source dataset and then tested on a certain target dataset, looking for the considerable shifts between the domains. The results are summarized in Figure 48 and 49 given below.

METHOD / SOURCE / TARGET	MNIST / MNIST-M	SYN NUMBERS / SVHN	SVHN / MNIST	SYN SIGNS / GTSRB
SOURCE ONLY	.5749	.8665	.5919	.7400
SA (FERNANDO ET AL., 2013)	.6078 (7.9%)	.8672 (1.3%)	.6157 (5.9%)	.7635 (9.1%)
PROPOSED APPROACH	**.8149** (57.9%)	**.9048** (66.1%)	**.7107** (29.3%)	**.8866** (56.7%)
TRAIN ON TARGET	.9891	.9244	.9951	.9987

Figure 48. *Classification accuracies for digit image classification for different source and target domains.*

METHOD / SOURCE / TARGET	AMAZON / WEBCAM	DSLR / WEBCAM	WEBCAM / DSLR
GFK(PLS, PCA) (GONG ET AL., 2012)	.464 ± .005	.613 ± .004	.663 ± .004
SA (FERNANDO ET AL., 2013)	.450	.648	.699
DA-NBNN (TOMMASI & CAPUTO, 2013)	.528 ± .037	.766 ± .017	.762 ± .025
DLID (S. CHOPRA & GOPALAN, 2013)	.519	.782	.899
DECAF$_6$ SOURCE ONLY (DONAHUE ET AL., 2014)	.522 ± .017	.915 ± .015	–
DANN (GHIFARY ET AL., 2014)	.536 ± .002	.712 ± .000	.835 ± .000
DDC (TZENG ET AL., 2014)	.594 ± .008	.925 ± .003	.917 ± .008
PROPOSED APPROACH	**.673 ± .017**	**.940 ± .008**	**.937 ± .010**

Figure 49. *Accuracy evaluation of different DA approaches on the OFFICE dataset.*

Coupled GANs

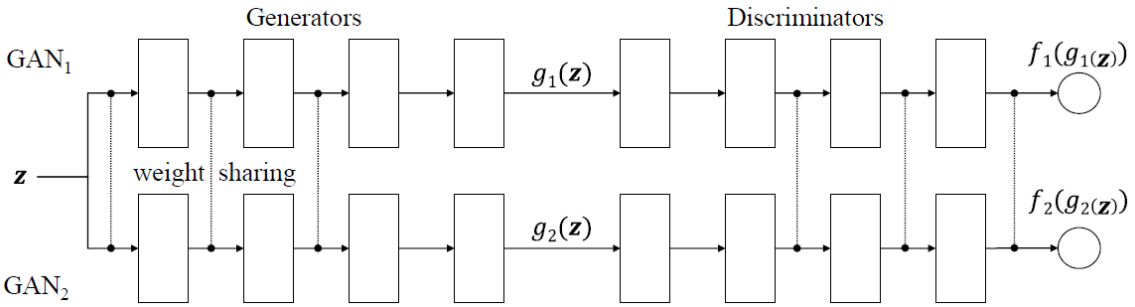


Figure 50. *CoGAN architecture.*

Previous frameworks have concentrated on learning a joint distribution by using datasets with tuples of images, meaning that images of the same class with distinct domains have the same respective label. However, these types of datasets are scarcely available and constructing them is highly challenging. Coupled GANs (CoGANs) [23] are used to bypass this challenge and learn the joint distribution of multi-domain images effectively.

The CoGAN architecture could be adapted to multiple domains; however, the architecture mentioned in the paper only focuses on domain adaptation in two domains. The CoGAN architecture presented in this paper is comprised of two GANs, one for each domain. Each of these GANs is responsible for synthesizing images in one domain. Throughout training, the CoGAN learns a marginal distribution as opposed to a joint distribution. Weight-sharing is introduced in this architecture to learn the joint distribution indirectly. Although this technique causes confusion between both discriminative models, it leads to the models learning from each other's domains.

The CoGAN architecture as discussed previously is comprised of GAN_1 and GAN_2, which are represented mathematically as g_1 and g_2. The generative models of the architecture try to independently learn from images x_1 and x_2 drawn from marginal distributions in each respective domain. An input noise, z, is input into each generative model and is mapped to each domain-specific image. Each GAN could have different number of layers, which are denoted m_1 and m_2. Below are the generator functions:

$$g_1(z) = g_1^{(m_1)}\left(g_1^{(m_1-1)}\left(\dots g_1^{(2)}\left(g_1^{(1)}\,(z)\right)\right)\right)$$

$$g_2(z) = g_2^{(m_2)}\left(g_2^{(m_2-1)}\left(\dots g_2^{(2)}\left(g_2^{(1)}\,(z)\right)\right)\right)$$

The generative model from the first to last layer decode information from abstract concepts to more details. The first layer decodes high-level semantics and the last layer decodes low-level details.

The first layers of both GAN_1 and GAN_2 are forced to have an identical layer structure denoted by k and share the same weights denoted by θ, as shown below:

$$\theta_{g_1^{(i)}} = \theta_{g_2^{(i)}}, for\ 1, 2, \dots, k$$

With this set up, high-level semantics will be decoded the same way in both generative models, except for the last layers. The last layers take the shared weights and observe them in a different manner, which eventually allows the generator to deceive the discriminator.

The CoGAN is also comprised of two discriminative models, f_1 and f_2, denoted by:

$$f_1(x_1) = f_1^{(n_1)}\left(f_1^{(n_1-1)}\left(\dots f_1^{(2)}\left(f_1^{(1)}\ (x_1)\right)\right)\right)$$
$$f_2(x_2) = f_2^{(n_2)}\left(f_2^{(n_2-1)}\left(\dots f_2^{(2)}\left(f_2^{(1)}\ (x_2)\right)\right)\right)$$

For the discriminative models, the number of layers are denoted by n_1 and n_2. The role of the discriminative models is to give a probability score to an input image, and this probability score tells whether that respective image is from the true data distribution. The first layer of the discriminative model extracts low-level features and the last layer extracts high-level features. Since input images from different domains have the same high-level semantics, the discriminative model for both GAN_1 and GAN_2 needs to have the same last layer, hence the weights of the last layer are shared. The sharing weights between layers, where l is the number of shared layers, is denoted by:

$$\theta_{f_1^{(n_1-j)}} = \theta_{f_2^{(n_2-j)}}, for\ j = 0,1, \dots, l-1$$

Similar to the standard GAN model, CoGANs also play a minimax game; however, for CoGANs, the models are split up into two different teams. The generative models make up the first team and their goal is to generate images from two different domains to deceive the discriminative models. The discriminative models, the second teams, have a different objective, and it is to differentiate between images created by the respective generative models and the images from the respective domains. Weight sharing is used in both generative and discriminative model as discussed earlier, which leads to better training of the whole system. The function below denotes the minimax game:

$$\max_{g_1,g_2}\min_{f_1,f_2} V(f_1, f_2, g_1, g_2)\ , subject\ to:$$

$$\theta_{g_1^{(i)}} = \theta_{g_2^{(i)}}, \quad for\ i = 1,2 \ldots,$$
$$\theta_{f_1^{(n_1-j)}} = \theta_{f_2^{(n_2-j)}}, \qquad for\ j = 0,1, \ldots, l-1$$

$$where\ V\ is\ denoted\ by:$$

$$V(f_1, f_2, g_1, g_2) = \mathbb{E}_{x_1 \sim p_{x_1}}[-log f_1(x_1)] + \mathbb{E}_{z \sim p_z}\left[-\log\left(1 - f_1\left(g_1(z)\right)\right)\right]$$
$$+ \mathbb{E}_{x_2 \sim p_{x_2}}[-log f_2(x_2)] + \mathbb{E}_{z \sim p_z}\left[-\log\left(1 - f_2\left(g_2(z)\right)\right)\right]$$

Several experiments were conducted with CoGANs using the MNIST training set for two respective tasks. The first task ($\mathbb{A}$) is to learn a joint distribution of a digit and its edge image. Thus, for this task, the source domain is the original digit images and the target domain is the edge images. The second task ($\mathbb{B}$) is to learn a joint distribution of a digit and its negative image, and for this task, the source domain is digit images and the target domain is the negative images. Due to weight sharing, the CoGAN network was able to render well-defined corresponding images for both tasks, respectively, even though it was not trained without these corresponding images.

The generators used for this experiment were composed of 5 convolutional layers, which used fractional strides, batch normalization, and ReLU. On the other hand, the discriminators used a variant of LeNet. The results of the trained tasks mentioned are shown in Figure 51 below.

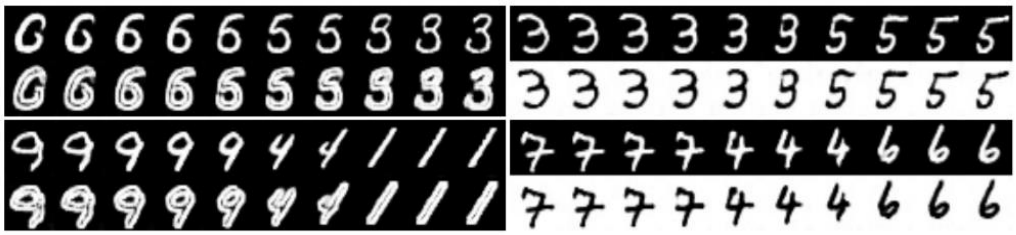


Figure 51. *Left: generation of digits and corresponding edge images. Right: generation of digits and corresponding negative images.*

To test the efficiency of CoGANs, more experiments were conducted with a different number of weight sharing layers used. The experimental set up consisted of using GAN_1 to adapt an image from the 1st domain to the 2nd domain. Then, the adapted image was compared to the generated image from the 2nd domain to visualize the quality of the adapted image. A joint distribution learning is perfect if it can render two identical images. To evaluate the adapted image quantitatively, a ratio of agreed pixels between 10K pairs of images generated by each network is used. Each network is trained 5 times with different weights and the average pixel agreement ratios are reported for 5 trials. The results are shown in Figure 52 below.

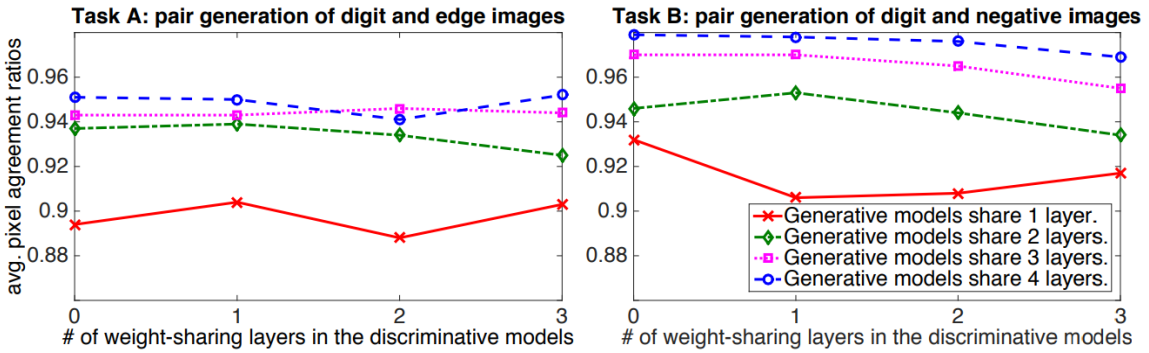


Figure 52. *The generative models' performance is positively correlated with the number of weight-sharing layers.*

Furthermore, another experiment was done using the CelebFaces Attributes dataset to learn the joint distribution of faces. This dataset consisted of faces with distinct pose variations and backgrounds. Multiple CoGANs were trained to generate a face with a specific attribute and a face without an attribute. The first domain was the face images with attributes and the second domain was the face images without attributes. Both generative and discriminative models had a deep convolutional neural network with 7 layers. CoGAN does not require corresponding images to be inputted into the network to learn another domain. This experiment was able to yield generated pairs of corresponding faces of the same person with and without a specific attribute. The results of this experiment are shown in Figure 53.

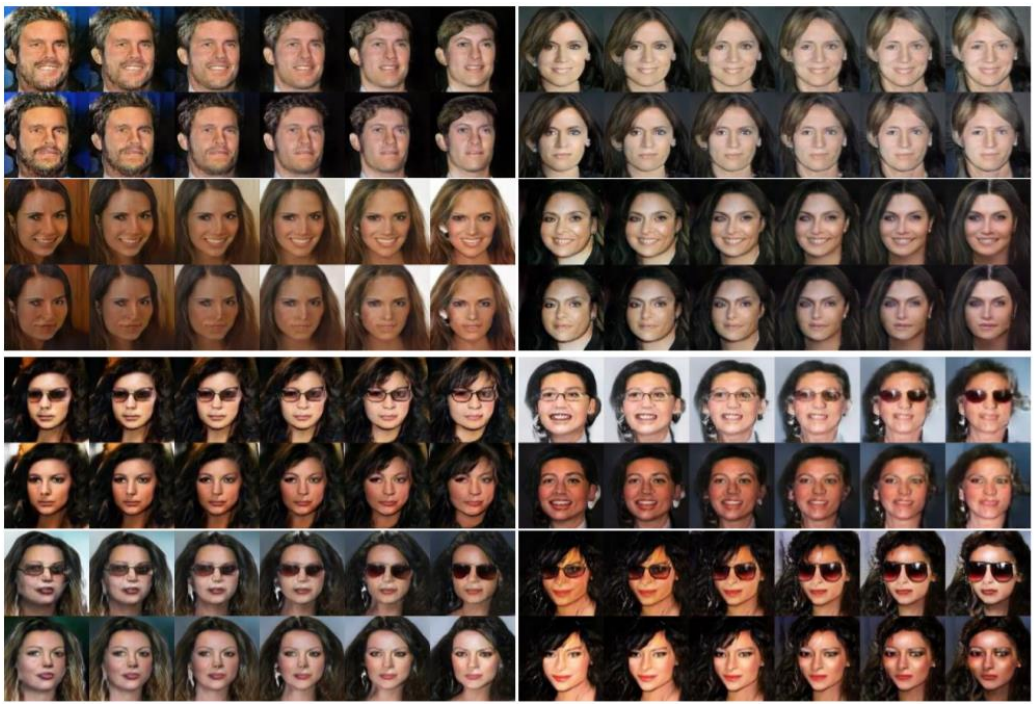


Figure 53. *Face images with different attributes generated using CoGAN. A distinct attribute is shown per row and each row shows distinct variations of these attributes.*

Unsupervised domain adaptation experiments were conducted by training a classifier on one domain (with labels) and testing it on another (without labels). The classifier consisted of a softmax layer, which was attached to the last layer of the discriminative model.

2000 images and labels from the MNIST dataset were randomly selected which were referred to as being of domain D_1, and 1800 images were randomly selected from the USPS dataset which were referred to as being of domain D_2. USPS images were resized so that they had the same dimensions as the MNIST images. A CoGAN was trained by jointly solving digit classification with labels for D_1 and only images for D_2. This joint learning produced two classifiers for the MNIST and USPS datasets shown below:

$$c_1(x_1) = c\left(f_1^{(3)}\left(f_1^{(2)}\left(f_1^{(1)}(x_1)\right)\right)\right)$$

$$c_2(x_2) = c\left(f_2^{(3)}\left(f_2^{(2)}\left(f_2^{(1)}(x_2)\right)\right)\right)$$

In the equation above, c denotes the softmax layers in both classifiers due to weight sharing. Using the same classifiers, adaptation from USPS to MNIST can be achieved. The results for unsupervised domain adaptation are shown in Figure 54 below.

Method	[17]	[18]	[19]	[20]	CoGAN
From MNIST to USPS	0.408	0.467	0.478	0.607	**0.912** ±0.008
From USPS to MNIST	0.274	0.355	0.631	0.673	**0.891** ±0.008
Average	0.341	0.411	0.554	0.640	**0.902**

Figure 54. *Unsupervised domain adaptation performance for CoGAN compared to other state-of-the-art models.*

Unsupervised Transductive Domain Adaptation

In unsupervised domain adaptation, other approaches have generally looked at learning a problem where the task is to discover a transformation matrix which aligns the source domain data distribution, which has labels, to the target domain data distribution, which has no labels. However, one thing the previous approaches do not consider is using the target data during training. This paper proposes a novel approach where the target labels will already be included into the training phase by using a transductive target inference technique [24].

The framework proposed in this paper is composed of two stages. The first stage is the transduction stage in which all the target labels are automatically inferred by solving a discrete multi-label energy minimization problem given a fixed domain transform parameter. The second stage is the adaptation stage in which the optimal asymmetric metric is calculated and found between the source and target domains given a fixed target label assignment. This gives the framework an advantage: a domain transformation parameter can be learned, and it is conscious of the actual transductive inference stage.

The unsupervised transductive domain adaptation method proposed is approached by solving label assignments of an unsupervised target domain (transductive stage) and domain shift jointly (adaption stage). The source domain is fully supervised and is denoted as:

$$\{\hat{x}_i, \hat{y}_i\}_i \in [N^s]$$

The target domain is fully unsupervised and is denoted as:

$$\{x_i\}_i \in [N^u]$$

It is also assumed that the source domain and target domain have different distributions on the same space as shown below:

$$\hat{x}_i, x_i \in X, where\ \hat{x}_i \sim p_s\ and\ x_i \sim p_t$$

Another component their model used is a feature function which is applicable on both domains and has a parameter θ:

$$\Phi_\theta : X \to R^d$$

The domain shift for the adaptation stage is explicitly modeled as an asymmetric similarity so that if two data points are high from both the source and target domains, they are from the same class. The function that proposes this is shown below:

$$s_w(\hat{x}_i, x_j) = \Phi(\hat{x}_i)^T W \Phi(x_j)$$

The transduction stage is modeled after the nearest neighbor algorithm. The k-nearest neighbor (k-NN) algorithm, which finds given an image, finds the closest class to that image via pixel measurements. For each source data point, a margin is enforced between the similarity of the source data point and the nearest neighbor from the target domain. Using the target labels and the similarity metric, W, transduction and adaptation are solved. This is shown by the following optimization function:

$$\min_{W, y_1, \ldots y_{N^u}} \sum_{i \,\epsilon\, [N^s]} [s_w(\hat{x}_i, x_{i^-}) - s_w(\hat{x}_i, x_{i^+}) + \alpha] + \lambda \sum_{i \,\epsilon\, N^u} \sum_{j \,\epsilon\, \mathcal{N}(x_i)} sim(x_i, x_j) \mathbb{I}(y_i \neq y_j)$$

$$s.t. \;\; i^+ = argmax_{j | y_j = \hat{y}} s_w(\hat{x}_i, x_j)$$

$$i^- = argmax_{j | y_j \neq \hat{y}_i} s_w(\hat{x}_i, x_j)$$

Here, $\mathbb{I}(a)$ serves as an indicator function, where if it is 1, then a is true and, if is 0, then a is false. On the other hand, $[a_+]$ serves as a rectifier function which is equal to $\max(0, a)$ and sim could be any similarity function.

To enforce consistency of pairwise target points, a k-nearest neighbor (k-NN) graph of unsupervised target points is created in terms of:

$$\psi_{ij} = sim(x_i, x_j) \mathbb{I}(y_i \neq y_j)$$

As for the closest supervised source point to each class, use:

$$\psi_{i\hat{k}} = s_w(\hat{x}_k, x_i)$$

After the k-NN graph is created, as mentioned, and since this algorithm requires arduous amounts of computations, the optimization problem is solved using α-β swapping. Also, stochastic gradient descent is used in the adaptation stage with a specific batch size, which will solve transduction only over a respective batch. The visualization of the above created model with the k-NN concept, where k = 4, and 4-class classification is shown in Figure 55.

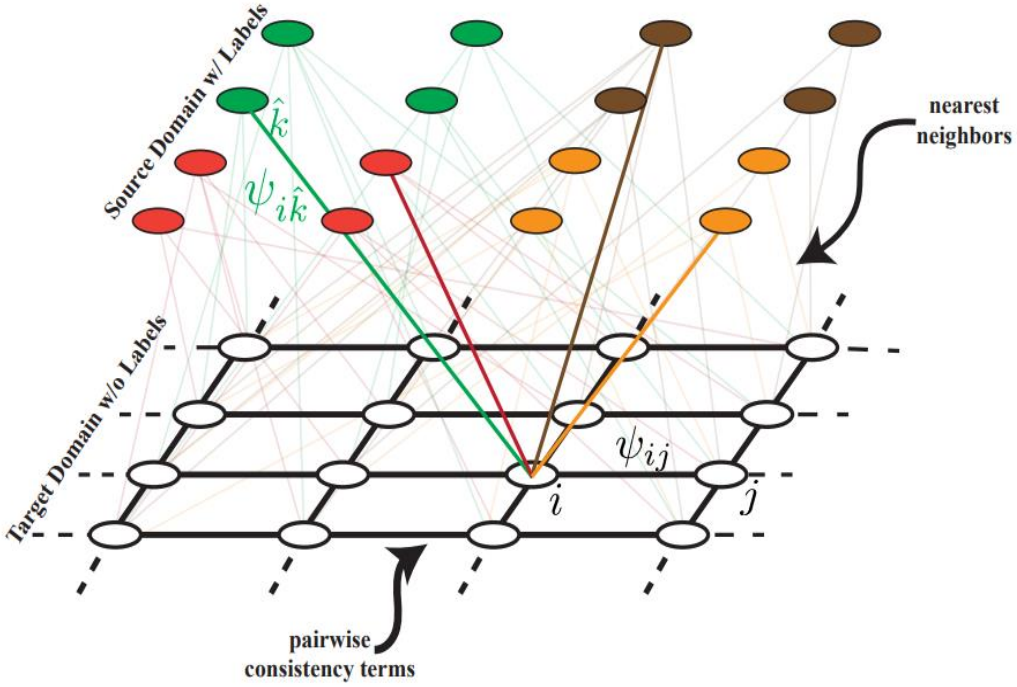


Figure 55. *Visualization of label propagation.*

Algorithm 1 Transduction with Domain Shift

Input: source $\mathbf{x}_i$, target $\hat{\mathbf{x}}_i$, y_i, batch size B
repeat
  Sample $\{\mathbf{x^b}_{1\cdots B}\}$, $\{\hat{\mathbf{x}}^b_{1\cdots B}, \hat{y}^b_{1\cdots B}\}$
  Solve (4) for $\{y_{1\cdots B}\}$
  for $i = 1$ **to** B **do**
    if $\hat{y}_i$ **in** $y_{1\cdots y_B}$ **then**
      Compute (i^+, i^-) using $\{y_{1\cdots B}\}$ in (5)
      Update $\frac{\partial loss}{\partial \theta}$ and $\frac{\partial loss}{\partial \mathbf{W}}$ using (7,8)
    end if
  end for
  $\mathbf{W} \leftarrow \mathbf{W} + \alpha \frac{\partial loss(y_i, \mathbf{W})}{\partial \mathbf{W}}$
  $\theta \leftarrow \theta + \alpha \frac{\partial loss(y_i, \mathbf{W})}{\partial \theta}$
until CONVERGENCE or MAX_ITER

Figure 56. *Algorithm for transduction with domain shift.*

Source Target	Amazon Webcam	D-SLR Webcam	Webcam D-SLR	Webcam Amazon	Amazon D-SLR	D-SLR Amazon
GFK (Gong et al., 2013)	.398	.791	.746	.371	.379	.379
SA* (Fernando et al., 2013)	.450	.648	.699	.393	.388	.420
DLID (Chopra et al., 2013)	.519	.782	.899	-	-	-
DDC (Tzeng et al., 2014)	.618	.950	.985	.522	.644	.521
DAN (Long & et al., 2015)	.685	.960	.990	.531	.670	.540
Backprop (Ganin & Lempitsky, 2015)	.730	**.964**	**.992**	.536	.728	.544
Source Only	.642	.961	.978	.452	.668	.476
Our Method	**.804**	.962	.989	**.625**	**.839**	**.567**

Figure 57. *Shows state-of-the-art results compared to the unsupervised transductive domain adaptation used and it is seen that this model achieves remarkable results*.

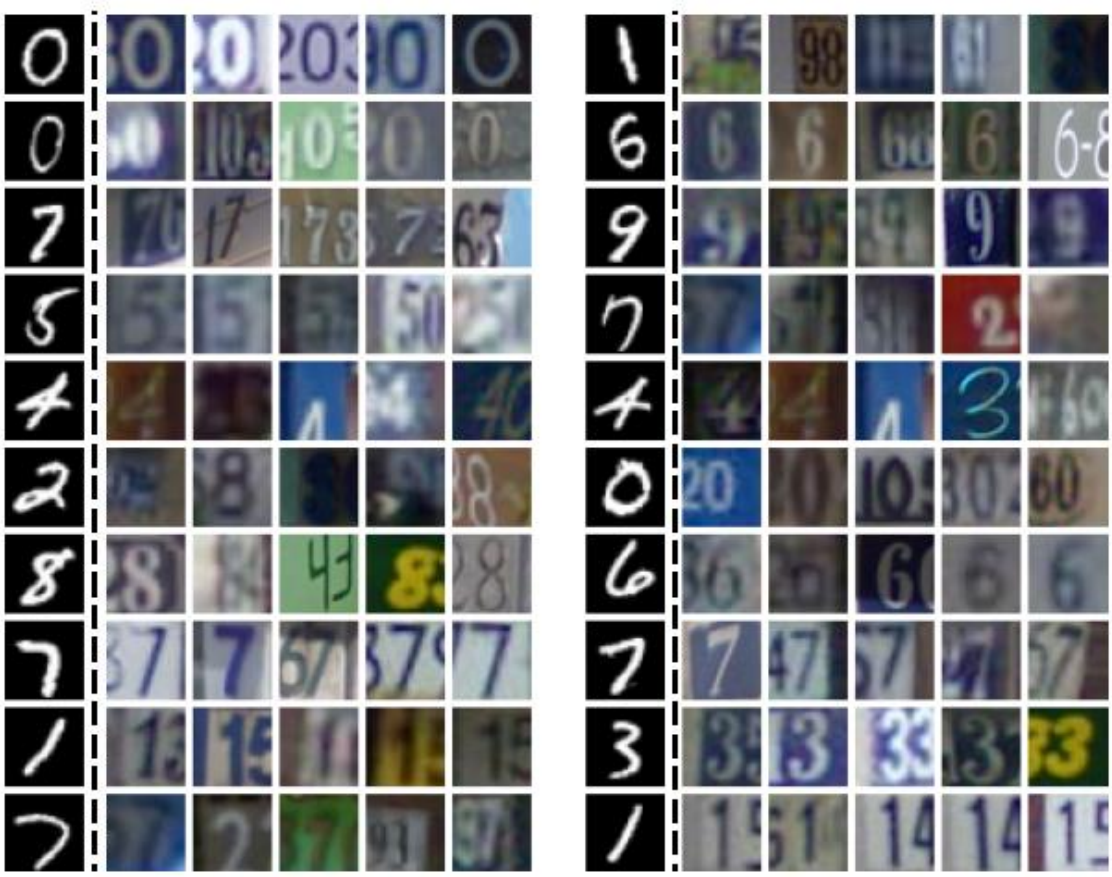


Figure 58. *Illustrates the nearest neighbors for the SVHN (domain) and MNIST datasets (target).*

As seen in Figure 57 and Figure 58, using unsupervised transductive domain adaptation method proposed in this paper has achieved state-of-the art classification results and it has also mapped different domain images effectively.

Wasserstein GAN

A constant problem with GANs has been convergence and knowing when to stop training. The Wasserstein GAN (WGAN) [25, 26], proposed by researchers at the Courant Institute of Mathematical Sciences, is a model that attempts to deal with this.

Before delving into the WGAN, the authors of the WGAN paper explain the process of image generation and its relation to various distances and divergences. The main objective of a generative model is to learn a probability distribution. In learning a probability distribution, we assume that the real data comes from a distribution, $\mathbb{P}_r$, and we want to learn a distribution, $\mathbb{P}_\theta$, that approximates $\mathbb{P}_r$. There are two ways to do this: have the model density P_θ exist and learn it directly through estimation, or learn a parametric function g_θ that takes a random variable Z with a fixed distribution $p(z)$ and creates samples that follow $\mathbb{P}_\theta$.

The first approach is problematic because P_θ does not exist when working with distributions supported by low dimensional manifolds [27] (a surface or shape that consists of various points and has 4 or fewer dimensions). Furthermore, the solution to this, adding a noise component to P_θ, introduces error when training generative models. Noise has been known for degradation and blurriness of generated images.

The second approach, on the other hand, can represent distributions limited to a low dimensional manifold and can easily generate images. Therefore, we choose this approach and train g_θ using a defined distance/divergence, $\rho(\mathbb{P}_\theta, \mathbb{P}_r)$, between two distributions. This is the approach used in GANs.

In order to pick the best distance/divergence for training g_θ, we have to look at each one's impact on the convergence of sequences of probability distributions. We want to define the distribution $\mathbb{P}_\theta$ in a way where the mapping of the parameter θ ($\theta \mapsto \mathbb{P}_\theta$) is continuous. If the mapping is continuous, then minimizing $\rho(\mathbb{P}_\theta, \mathbb{P}_r)$ with respect to θ will cause $\mathbb{P}_\theta$ to approach $\mathbb{P}_r$. Four distances were examined: the Total Variation (TV) distance, Kullback-Leibler (KL) divergence, Jensen-Shannon (JS) divergence, and Earth-Mover (EM) or Wasserstein-1 distance.

- The *Total Variation* (TV) distance

$$\delta(\mathbb{P}_r, \mathbb{P}_g) = \sup_{A \in \Sigma} |\mathbb{P}_r(A) - \mathbb{P}_g(A)| \ .$$

- The *Kullback-Leibler* (KL) divergence

$$KL(\mathbb{P}_r \| \mathbb{P}_g) = \int \log\left(\frac{P_r(x)}{P_g(x)}\right) P_r(x) d\mu(x) \ ,$$

where both $\mathbb{P}_r$ and $\mathbb{P}_g$ are assumed to be absolutely continuous, and therefore admit densities, with respect to a same measure μ defined on $\mathcal{X}$.[2] The KL divergence is famously assymetric and possibly infinite when there are points such that $P_g(x) = 0$ and $P_r(x) > 0$.

- The *Jensen-Shannon* (JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \| \mathbb{P}_m) + KL(\mathbb{P}_g \| \mathbb{P}_m) \ ,$$

where $\mathbb{P}_m$ is the mixture $(\mathbb{P}_r + \mathbb{P}_g)/2$. This divergence is symmetrical and always defined because we can choose $\mu = \mathbb{P}_m$.

- The *Earth-Mover* (EM) distance or Wasserstein-1

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\left[\|x - y\|\right] \ ,$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$. Intuitively, $\gamma(x, y)$ indicates how much "mass" must be transported from x to y in order to transform the distributions $\mathbb{P}_r$ into the distribution $\mathbb{P}_g$. The EM distance then is the "cost" of the optimal transport plan.

Figure 59. *List of the different distances examined in the WGAN paper. The EM distance is the minimal effort needed to move the mass in a probability distribution so that the distribution changes into a different distribution.*

It was found that simple sequences of distributions converge under only the EM distance, and every distribution that converges under the other distances, also converges under the EM distance. It was also shown that the EM distance induces the weakest topology and is the most reasonable loss function when learning distributions supported by low dimensional manifolds. For these reasons, the EM distance is a good loss metric for training GANs.

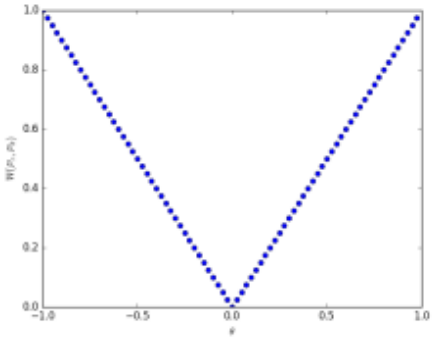


Figure 60. *Plot of the EM distance $W(\mathbb{P}_\theta, \mathbb{P}_r)$ as a function of θ. The plot shows that the EM distance is continuous and has no cases where the gradient is 0.*

Calculating the exact EM distance, $W(\mathbb{P}_\theta, \mathbb{P}_r)$, however, is intractable. Thus, the supremum (least upper bound) over 1-Lipschitz functions was replaced with the supremum over K-Lipschitz functions for some K to get $K \cdot W(\mathbb{P}_\theta, \mathbb{P}_r)$. Moreover, if we suppose there is a parameterized family of functions $\{f_w\}_{w \in \mathcal{W}}$ that are all K-Lipschitz (w is the weights of a function and $\mathcal{W}$ is the set of all possible weights), we can find the optimal critic f_w to obtain the Wasserstein distance up to an unknown constant, K. A method to do this is to train a neural network with weights to get an optimal f_w and then back-propagate through $W\big(\mathbb{P}_r, g_\theta(Z)\big)$. The WGAN paper also mentions using weight clipping to enforce the Lipschitz constraint for this approach, but states there is a better way to do it.

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size.
n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, \ldots, n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Figure 61. *The proposed WGAN algorithm. First, the EM distance is estimated for a fixed θ by training the critic to convergence. Then, the θ is calculated by sampling a batch of prior samples. θ is subsequently updated and the same process is run until θ has converged.*

Since the EM distance is continuous and differentiable, we train the critic to optimality. By doing this, we get a more reliable and cleaner gradient descent and prevent mode collapse. The authors ran an experiment to show the difference between a standard GAN algorithm and the WGAN algorithm by training the GAN's discriminator and WGAN's critic to optimality. The standard GAN discriminator results in vanishing gradients, while the WGAN critic converges to a linear function with clean gradients.

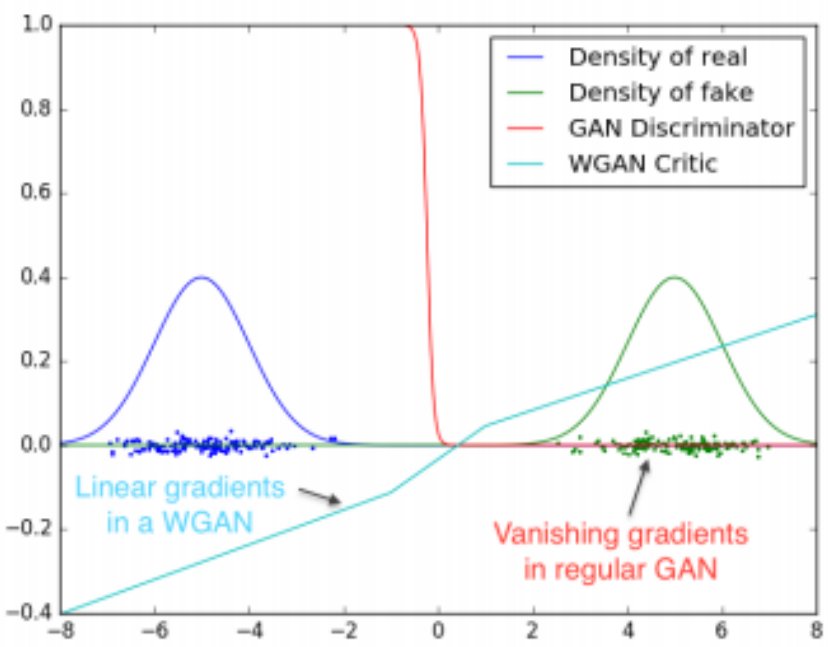


Figure 62. *The output curves for the GAN discriminator and WGAN critic over the space containing real and fake density distributions. The GAN discriminator has vanishing gradients while the WGAN critic has clean gradients everywhere.*

The authors ran sets of experiments and stated two advantages of the WGAN: a loss function that correlates with image quality and the generator's convergence, and better optimization stability (not as reliant on architecture). They tested WGAN image generation against a DCGAN with a standard GAN algorithm. They used the LSUN-Bedrooms dataset containing 64 x 64-pixel images of indoor bedrooms and the same hyper-parameters as the WGAN algorithm proposed above. Additionally, they ran experiments where they replaced the generator or the generator and critic with a 4-layer ReLU-MLP (rectified linear unit multilayer perceptron) with 512 hidden units.

The first set of experiments was comparing the evolution of the WGAN estimate of the EM distance to the evolution of the GAN estimate of the JS divergence during training. It was observed that the EM distance correlates with image quality while the JS divergence does not. Also, it was found that using momentum-based optimizers or higher learning rates can sometimes result in unstable WGAN training.

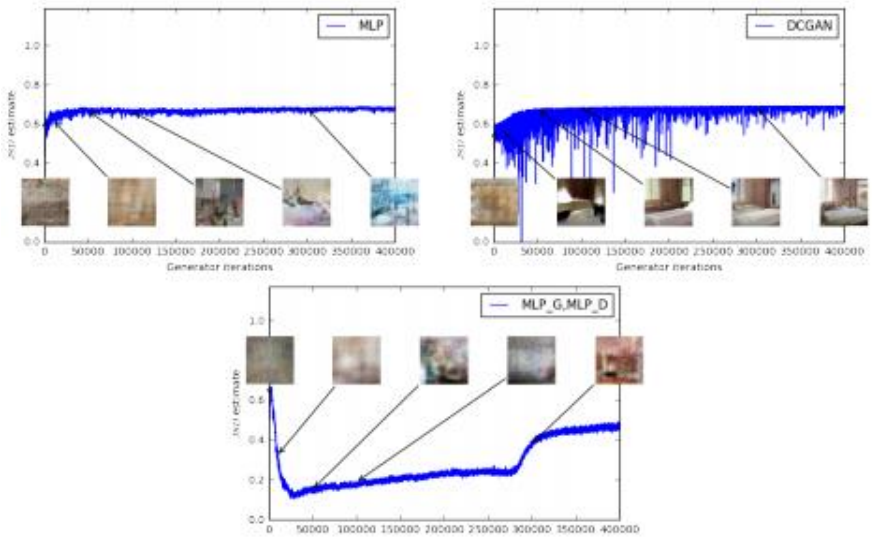


Figure 63. *Plots of the training curves at various training stages using the JS divergence as the loss function. Samples are included to show the progression of image quality. In the top left, top right, and bottom, the generator was the MLP, the generator was the standard DCGAN, and the generator and discriminator were the MLP, respectively. The curves show no correlation between the loss and image quality.*

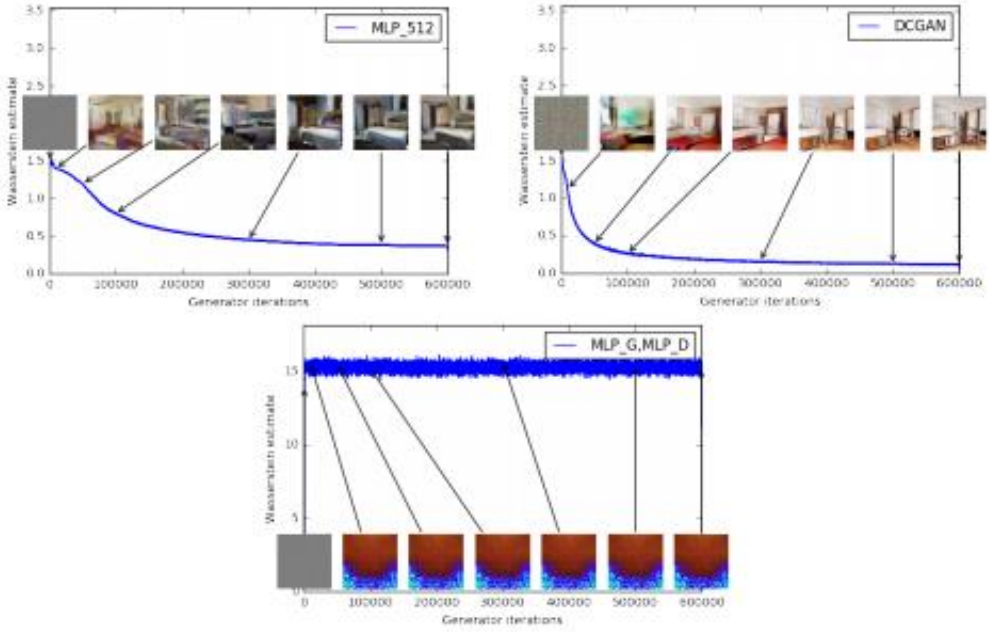


Figure 64. *Plots of the training curves at various training stages using the EM distance as the loss function. Samples are included to show the progression of image quality. In the top left, top right, and bottom, the generator was the MLP, the generator was the standard DCGAN, and the generator and critic were the MLP, respectively. As the loss decreases, image quality improves, showing a correlation between the two.*

The second set of experiments tested image generation using a convolutional DCGAN generator, convolutional DCGAN generator without batch normalization and with a constant number of filters, and the same ReLU-MLP used in the first set of experiments. For the standard GAN discriminator and WGAN critic, the convolutional DCGAN architecture was used. Images showed that the WGAN produced images just as well as the baseline DCGAN and, in the generator architecture without batch normalization, the WGAN generated samples while the standard GAN did not learn. An important note is that mode collapse was never experienced for the WGAN algorithm in any of the experiments.

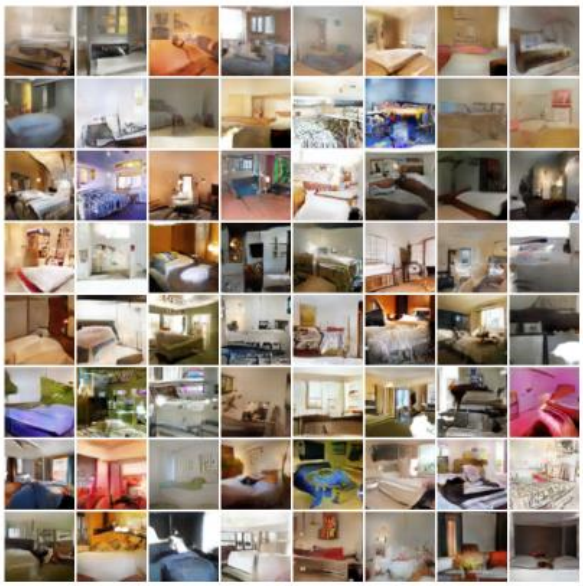


Figure 65. *WGAN algorithm trained with a DCGAN generator.*

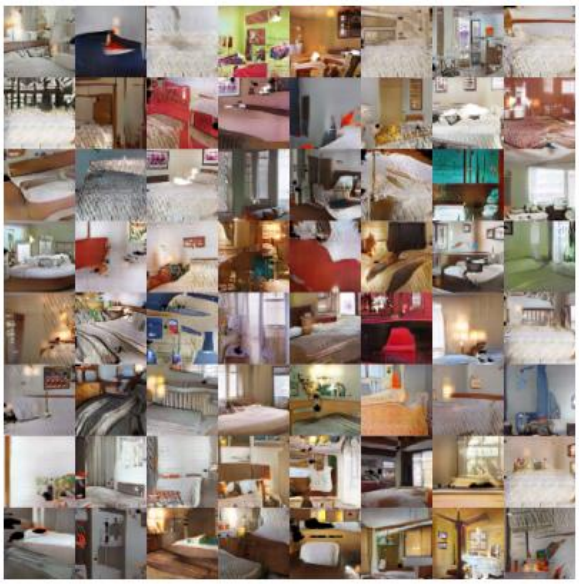


Figure 66. *Standard GAN algorithm trained with a DCGAN generator.*

Figure 67. *WGAN algorithm trained with a DCGAN generator without batch normalization and with a constant number of filters. The WGAN still produced images, although they had lower quality.*

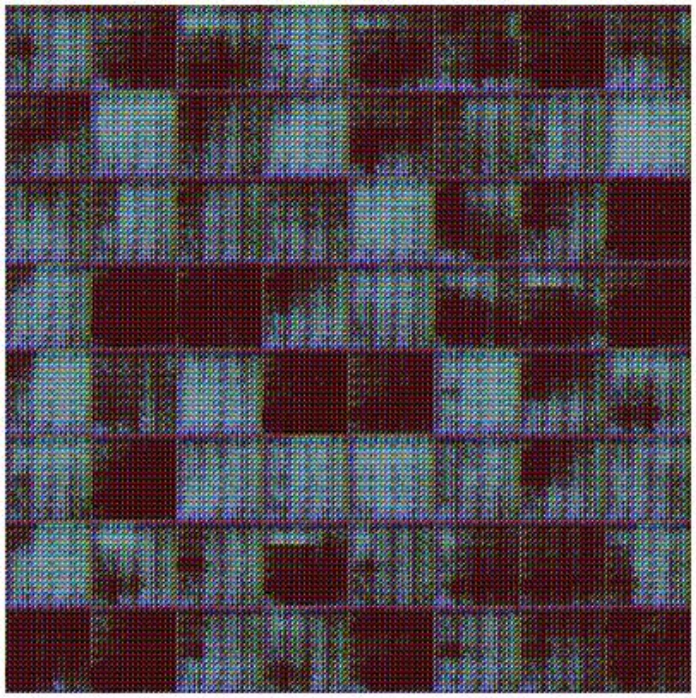


Figure 68. *Standard GAN algorithm trained with a DCGAN generator without batch normalization and with a constant number of filters. The standard GAN could not produce images and failed to learn.*

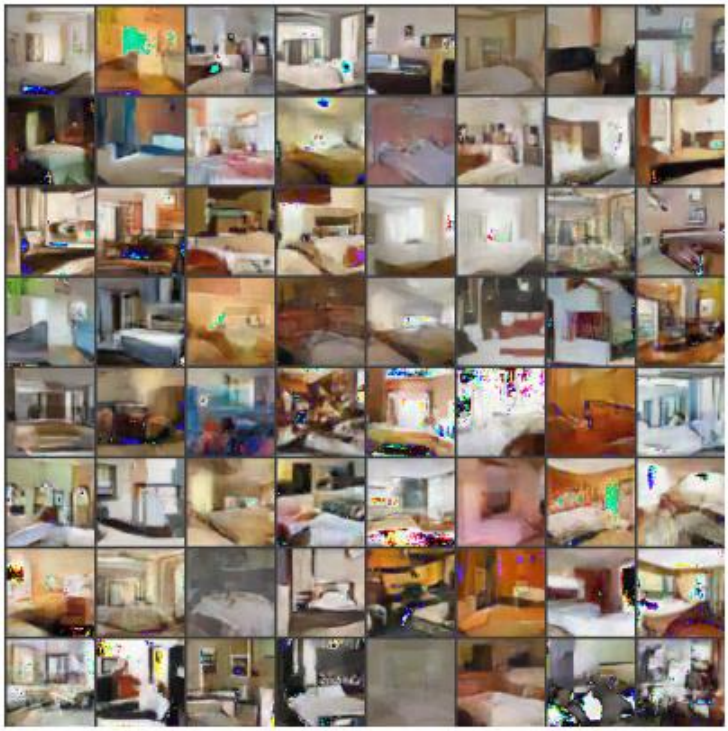


Figure 69. *WGAN algorithm trained with an MLP generator. The image quality was lower than that of the standard GAN trained with a DCGAN generator, but was higher than the image quality of the GAN MLP.*

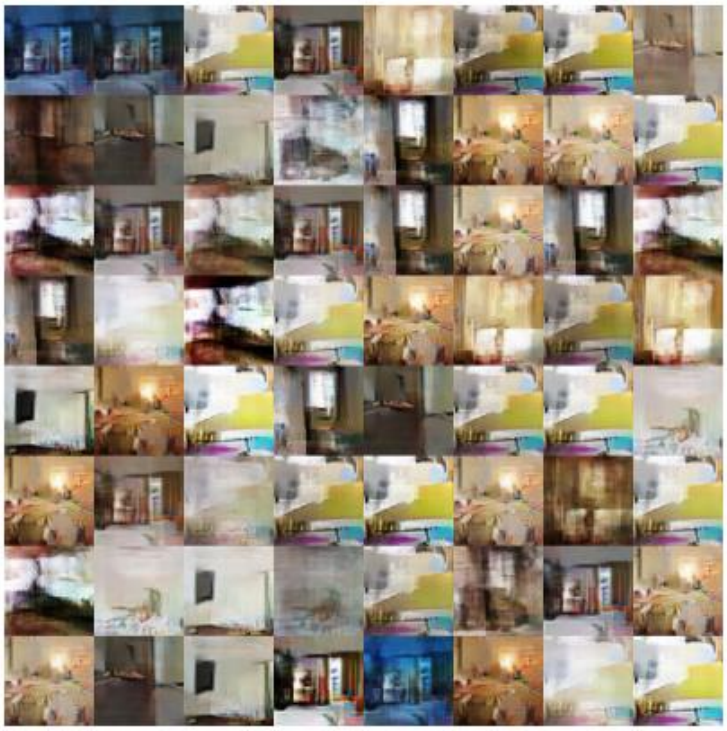


Figure 70. *Standard GAN algorithm trained with an MLP generator. The GAN MLP experienced significant mode collapse.*

Overall, the WGAN is an interesting GAN model that can be implemented for high training stability, prevention of mode collapse, and a meaningful loss function. It is a potential option for trying to beat Google's PDA experiment.

Loss-Sensitive GAN

In the training of a GAN model, there is a problem with treating real and generated data as positive and negative examples. Even though, by doing this, fake images would become closer to the manifold of real images, we run into trouble with the discriminator. By treating the generated samples as negative examples, we can have a discriminator that overly leans towards identifying an image as fake. This, in turn, would limit the extent to which the generator can be trained. Additionally, with classic GANs, there is an assumption that GANs have infinite modeling capacity. GAN's with infinite modeling capacity can experience vanishing gradients as well as mode collapse where the generator produces the same data point and there is no diversity among generated images.

The creators of the Loss-Sensitive GAN (LS-GAN) [28] looked at creating models without infinite modeling capacity, resulting in the new GAN model. They first proposed a loss function to measure the quality of generated images and imposed a constraint so that the loss of a real image is smaller than that of a generated one by an unfixed margin (dependent on the distance between the images in a metric space). They then analyzed the LS-GAN and model data on a specific class of Lipschitz densities. They proved that the generator can produce quality images by enforcing a Lipschitz constraint on the LS-GAN.

Additionally, the creators proposed a non-parametric solution to the LS-GAN that points out the optimal loss function for all Lipschitz functions. With the optimal loss function, we can update the LS-GAN generator with gradient without having to worry about the gradient vanishing. Finally, they applied the LS-GAN algorithm to a Conditional GAN (CLS-GAN) to produce images and evaluated it by image classification.

Going more in depth, the creators introduced a loss function that can tell the difference between real and fake images instead of training a discriminator that uses a probability to determine the classification of real images. They trained this loss function with a constraint that requires real and fake images to be separated by a data-dependent margin of their losses. By choosing a viable margin, as the generator creates better images, the margin can vanish, allowing the GAN to focus on improving the generation of images that are farther away from real samples. This hard constraint was then relaxed with a slack variable so that they could use a fixed generator G_ϕ to train the loss function L_θ parameterized with θ. In the case that the loss function is fixed (L_θ**.**), we can find an optimal generator by solving the minimization problem for the fixed loss function and generator. In general, L_θ and G_ϕ can be optimized by solving an equilibrium where the loss function's parameter is minimized.

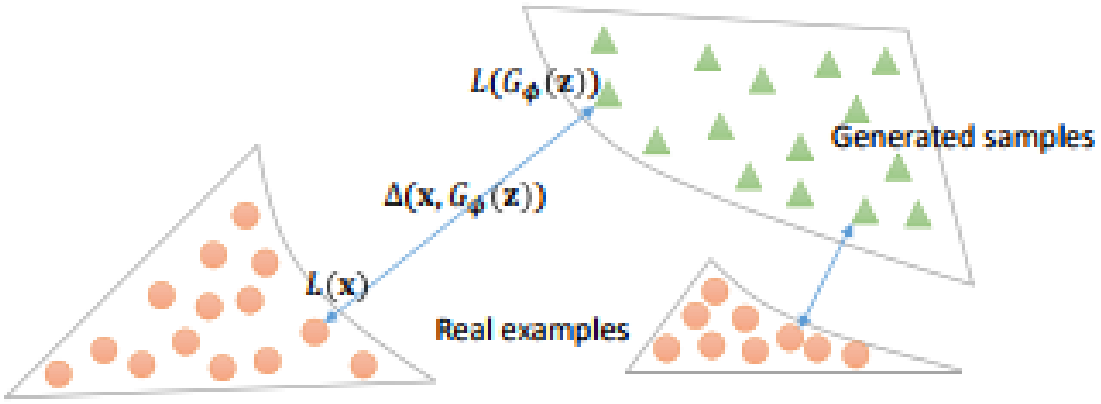


Figure 71. *Illustration showing the LS-GAN concept of the loss margin between the fake and real images.*

To prove that this approach could work, they showed that the density distribution of generated images P_G**.** will converge to the density distribution of real images P_{data}. If we use the Lipschitz assumption for L_θ and generator density P_G, we can relax the assumption that the discriminator and generator have infinite modeling ability to show P_G**.** and P_{data} are the same. Nash equilibrium can then be shown to exist when L_θ**.** and P_G**.** are Lipschitz, proving the above assertion.

Although they proved the density of generated images and real data are consistent, they were concerned about the generalizability of the LS-GAN if more training images were introduced. To test the LS-GAN's generalizability, they examined two objectives of the model: how well the loss function can distinguish between real and fake images, and the limit to which the generator can be trained to minimize the loss function. It was shown that the model does have generalization ability, but we should manage the sample complexity when training the generator and loss functions which can be done by limiting their Lipschitz constants.

They then looked at the class of optimal loss functions based on an objective that deals with all the Lipschitz loss functions. They first stated the optimal loss functions that are the upper and lower bounds for the class of these functions. Afterwards, they ensured the loss functions had a bounded Lipschitz constant by using weight decay. Additionally, it was shown that the loss function can give enough gradient to constantly update the generator regardless of the speed at which it is trained to optimality.

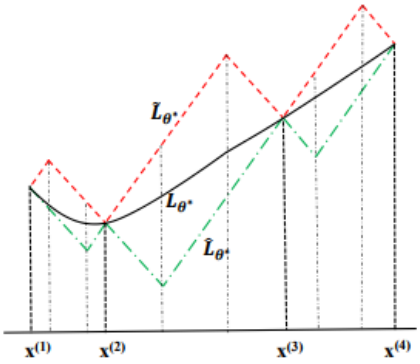


Figure 72. *Comparison of the optimal loss functions that are the upper and lower bounds for the class of optimal functions. The functions are cone-shaped and have almost no vanishing gradients.*

Furthermore, the LS-GAN was extended to create images based on a condition y, resulting in the CLS-GAN. The CLS-GAN generator takes in a condition and noise vector, y and z, respectively, and produces an image. The creators defined a loss function $L_\theta(\mathrm{x},\mathrm{y})$ (x is a data sample) to train the model and applied a constraint to it, similar to the LS-GAN. The loss function and generator get trained, resulting in a generator that makes images that are close to the data samples for a given condition y. This is proven by stating there exists a Nash equilibrium where the trained loss function is Lipschitz continuous in x for each y, and the conditional density is Lipschitz continuous and becomes the real data density as the model is trained.

The trained loss function of the CLS-GAN can be used to predict the label of an image, making the model a useful classifier. The benefits of having a CLS-GAN classifier is it has the potential for improved training and gives a method for assessing the model based on its classification accuracy. The CLS-GAN can also be used for both supervised and semi-supervised learning.

Algorithm 1 Learning algorithm for LS-GAN.

Input: m data examples $\mathcal{X}_m$, and λ.
for a number of iterations **do**
  for a number of steps **do**
    Update the loss function over minibatches;
    Sample a minibatch from $\mathcal{X}_m$;
    Sample a minibatch from $\mathcal{Z}_m$;
    Update the loss function L_θ by descending the gradient of (8) with weight decay over the minibatches;
  end for
  Sample a set of $\mathcal{Z}'_k$ of k random noises;
  Update the generator G_ϕ by descending the gradient of (9) with weight decay;
  Update the generated samples $\mathbf{x}^{(m+j)} = G_{\phi^*}(\mathbf{z}_j)$ for $j = 1, \cdots, m$ on $\mathcal{Z}_m$;
end for

Figure 73. *The proposed LS-GAN algorithm. The loss function is updated over multiple minibatches and then the generator is trained with a set of random noises.*
To evaluate the LS-GAN, three experiments were run: the comparison of image generation of the regular LS-GAN with that of other models, the evaluation of the

LS-GAN by using the CLS-GAN classifier, and the qualitative assessment of the CLS-GAN's generations of images for various classes. The architecture used for the generator and loss function networks in the experiments were based on the DCGAN architecture. Batch normalization layers were added to the two networks. Furthermore, the LS-GAN loss function network did not use a sigmoid layer as the output and the CLS-GAN loss function network had a global mean-pooling layer. For training, a mini-batch of 64 images and the Adam optimizer were used.

For the first experiment, they compared the DCGAN and LS-GAN image generation on the CelebA dataset. There were no significant differences between the two, but when the batch normalization layers were removed from the generator of the DCGAN, the DCGAN experienced mode collapse and failed to produce face images while the LS-GAN still performed well. In addition, there was no evidence of mode collapse of the generated images and the LS-GAN was shown to provide enough gradient to constantly update the generator.

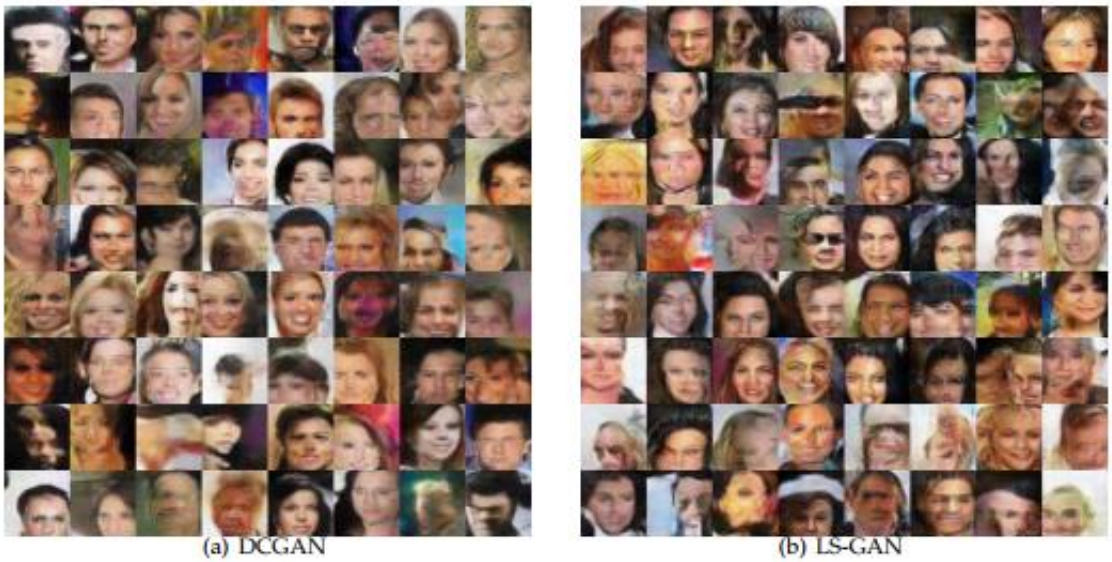


Figure 74. *DCGAN and LS-GAN generated images on the CelebA dataset. There are no notable differences between the image quality for both models.*

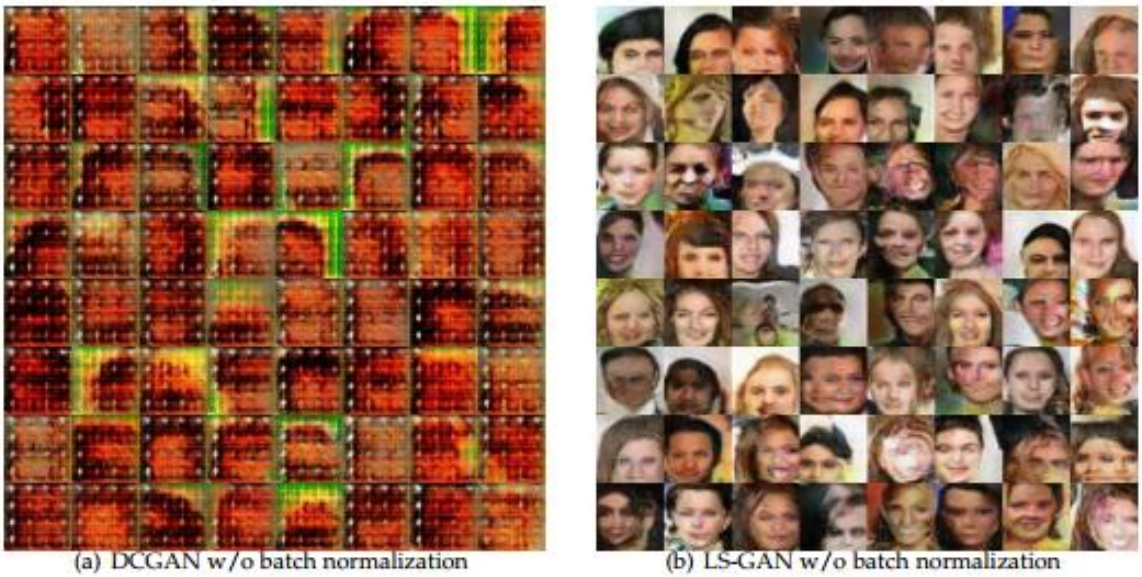


Figure 75. *DCGAN (left) and LS-GAN (right) generated images on the CelebA dataset without using batch normalization. The LS-GAN was still able to produce images, but the DCGAN failed to generate any face images. The DCGAN generator also experienced mode collapse.*

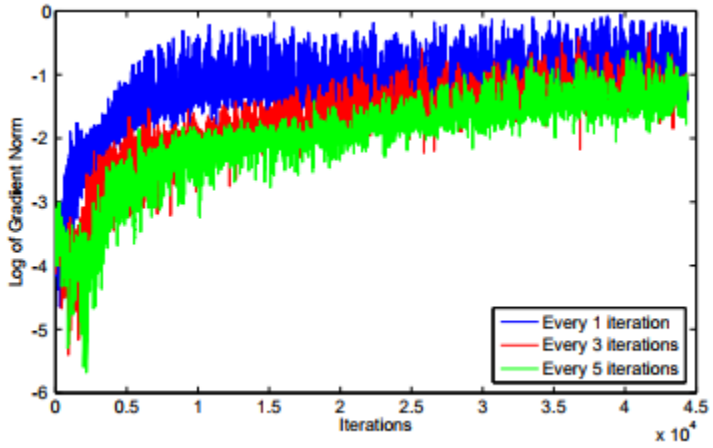


Figure 76. *Plot of the log of the gradient norm vs. the number of iterations. The colors represent how often the generator is updated. The plot shows non-vanishing gradients for the generator.*

Next, they compared the classification accuracy of the LS-GAN to other models on the CIFAR-10 and SVHN datasets. The CIFAR-10 dataset consists of 32 x 32 colored images of 10 different classes and the SVHN contains 32 x 32 colored images of house numbers. First, the classification accuracies of the CLS-GAN and other models on CIFAR-10 were calculated in two different situations. The first had all the dataset images labeled while the second had only 400 images labeled per class. The results of the experiment showed the CLS-GAN performed better than the other models in supervised and semi-supervised learning.

Methods	*Accuracy (All)*	*Accuracy (400 per class)*
1 Layer K-means [7]	80.6%	63.7% (± 0.7%)
3 Layer K-means Learned RF [29]	82.0%	70.7%(± 0.7%)
View Invariant K-means [30]	81.9%	72.6%(± 0.7%)
Examplar CNN [31]	84.3%	77.4%(± 0.2%)
Conditional GAN [32]	83.6%	75.5%(± 0.4%)
DCGAN [7]	82.8%	73.8%(± 0.4%)
Ladder Network [20]	–	79.6%(± 0.5%)
CatGAN [22]	–	80.4%(± 0.4%)
ALI [23]	–	81.7%
Improved GAN [2]	–	81.4%(± 2.3%)
CLS-GAN	**91.7%**	**82.7%(± 0.5%)**

Figure 77. *Classification accuracies for supervised and semi-supervised learning of the CLS-GAN and other models on the CIFAR-10 dataset. The CLS-GAN outperformed the other models in both settings.*

SVHN was then used in place of CIFAR-10, but training the model was done in only a semi-supervised setting. In this case, the CLS-GAN outperformed the other models again.

Methods	*Error rate*
KNN [7]	77.93%
TSVM [7]	66.55%
M1+KNN [18]	65.63%
M1+TSVM [18]	54.33%
M1+M2 [18]	36.02%
SWWAE w/o dropout [35]	27.83%
SWWAE with dropout [35]	23.56%
DCGAN [7]	22.48%
Conditional GAN [32]	21.85%±0.38%
Supervised CNN [7]	28.87%
DGN [18]	36.02%±0.10%
Virtual Adversarial [36]	24.63%
Auxiliary DGN [37]	22.86%
Skip DGN [37]	16.61%±0.24%
ALI [23]	7.3%
Improved GAN [2]	8.11%±1.3%
CLS-GAN	**5.98%± 0.27%**

Figure 78. *The classifier error rates for the CLS-GAN and other models on the SVHN dataset. The LS-GAN had the lowest error rate, outperforming all the other models.*

Last, they analyzed generated images by the CLS-GAN for the MNIST, CIFAR-10, and SVHN datasets. The CIFAR-10 images were distinguishable, but the resolution was low, and some visual details were missing. Moreover, mode collapse was observed when the hyperparameter λ was set to a small value.

Figure 79. *Images generated by the CLS-GAN on the MNIST, CIFAR-10, and SVHN datasets. The image quality is high, but the CIFAR-10 images are missing some details.*

In summary, while the LS-GAN can produce high quality images, has mainly non-vanishing gradients, and prevents mode collapse, there appears to be better models for unsupervised learning such as the Wasserstein GAN. We could, however, look into using the CLS-GAN as our classifier or use the LS-GAN if we start working with supervised and semi-supervised learning.

Strategies for Improving GANs Training

Instead of focusing on solely creating variations of GANs, we can examine strategies for optimizing each model. Although, since GANs are considered multilevel optimization problems, we cannot simply apply gradient descent or other ordinary methods to increase stability as these can result in poor performance of the models. In order to actually improve GANs training, we have to focus on the optimization of both the discriminator and generator. Professionals in machine learning have already looked into strategies [29, 30] that can help with this and some of them are listed below.

Method	GANs	AC
Freezing learning	yes	yes
Label smoothing	yes	no
Historical averaging	yes	no
Minibatch discrimination	yes	no
Batch normalization	yes	yes
Target networks	n/a	yes
Replay buffers	no	yes
Entropy regularization	no	yes
Compatibility	no	yes

Figure 80. *Strategies for stabilizing and improving the training of GANs and AC (actor-critic methods) and whether they have been shown to improve performance. This project will focus on the strategies for GANs.*

1. Freezing learning

Initially with GANs, the discriminator needs to be trained first with a sample set of images and some noise. Afterwards, the generator is trained with the output of the discriminator. The problem with training both is that one or the other will progress faster than the other which can lead to the GAN having degenerate behavior. A solution to this is to freeze the generator or the discriminator (training is set to false and no back propagation occurs) while the other is being trained.

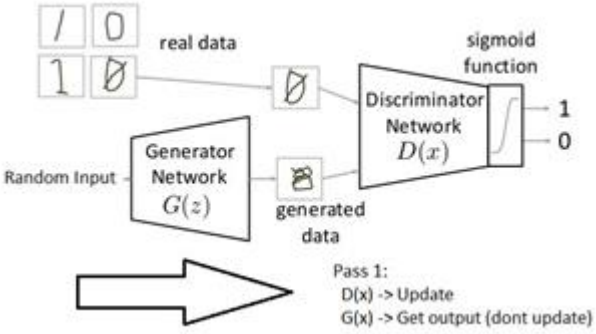


Figure 81. *The discriminator is trained with a dataset of real images while the generator is frozen.*

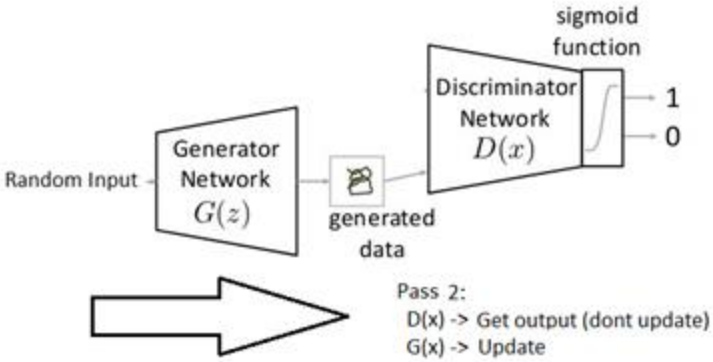


Figure 82. *The generator is trained using output from the discriminator, but the discriminator is frozen. Freezing the discriminator/generator during training prevents the GAN from producing a degenerate solution.*

2. Label smoothing

One way to stop gradients from disappearing when the discriminator is confident in detecting that an image is fake is to change the target labels. Instead of having single values of 0 and 1 for fake and real images, respectively, make the targets a value from a range. For example, choose a value between 0 and 0.3 if the image is fake and a value between 0.8 and 1.2, if the image is real.

3. Historical averaging

In historical averaging, a penalty term is added to gradient descent. This term punishes weights that deviate too much from previous average values. The benefit of historical averaging is that it helps the GAN find equilibria which the gradient descent will not do.

4. Minibatch discrimination

Rather than having the discriminator look at a single image, multiples images from the same minibatch are fed into it. Doing this helps prevent the generator from collapsing by increasing the diversity or entropy of images produced. Without minibatch discrimination, the outputs of the generator could converge to a single point and cause collapse, resulting in failure of the training algorithm.

5. Batch normalization

During training, the distribution of each input's layer can change. This is known as internal covariate shift and causes training to be slowed down. Batch normalization stops internal covariate shift by transforming the inputs so that they have 0 mean and unit variance. Implementing batch normalization in training has the advantages of increasing the learning rate and improving the accuracy of GANs

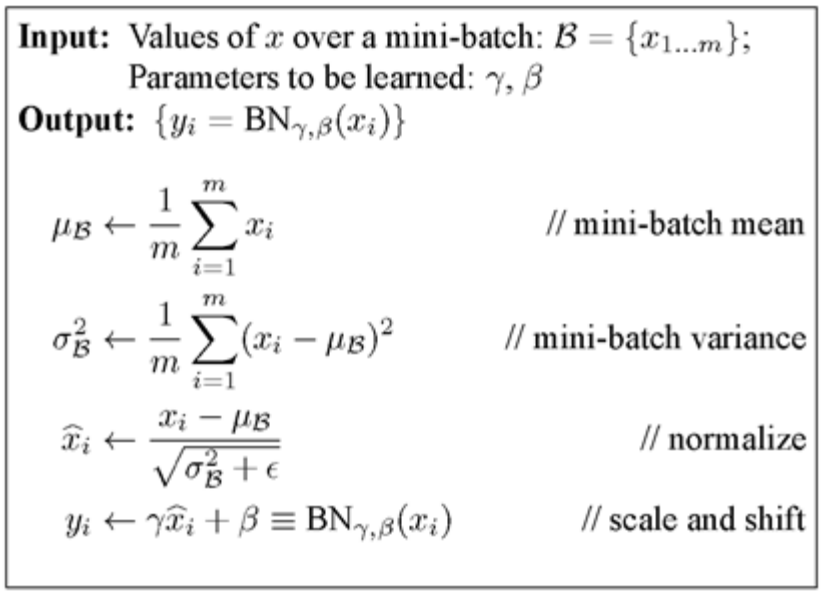


Figure 83. *Batch normalization algorithm. Here, the algorithm is used on x over a mini-batch.*

We could reasonably use some of these strategies in our GAN models to improve the classification accuracy of Google's DCGAN. If we wanted to, we could also look at variations of a few of these such as virtual batch normalization and one-sided label smoothing. At the very least, we should aim to include batch normalization when designing our GANs.

Improved Semi-Supervised Learning with GANs using Manifold Invariances

Most methods of semi-supervised learning with GANs [33] use a method where the discriminator and classifier are shared. The neural network will not only discriminate between real images samples and generated (fake) images, but it will also predict the class label. GANs can capture the data manifold. The data manifold being all the data points which encompass all the features, the entire mapping of features in the neural network. This research proposes the idea of capturing the manifold and estimating the tangent space of said manifold. Using it to inject invariances into the classifiers.

In most semi-supervised learning methods that use GANs, the regular GAN's discriminator is modified to have k outputs, each output corresponds to K real classes. There are other times where the output is $k + 1$, which corresponds to the probability that a sample is fake. In this research, the generator is used to obtain the tangents to the image manifold. Once these tangents are found they are used to inject invariances into the classifier and discriminator.

The paper argues that although there has been work done using autoencoders to estimate the local tangent space, GANs have been established to generate better quality images. Therefore it (GANs) can be argued to learn a more accurate parameterization of the image manifold.

First, we begin with the training of the generator. The generator will serve as a mapping from low dimensional space (raw pixel data) to a manifold embedded in the higher dimensional data, which is when hidden layers have extracted unique and specific features from the image such as edges, contours and lighting.

To be able to get tangents to the manifold at x, the corresponding latent representation z must be found first. There must exist an encoder that can do this. This encoder will be used to get the tangent directions to the manifold at a point $x = g(z)$ from M.

There are three approaches to train the generator to learn a close approximation to the true data manifold. Which is an attempt to find the corresponding latent representation with hopefully a small error.

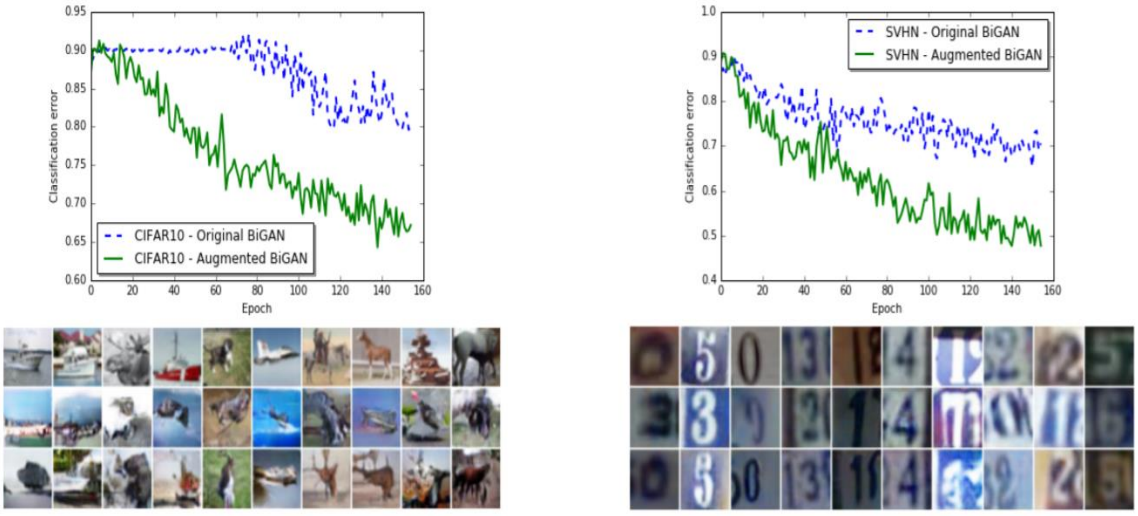


Figure 84. *Image demonstrating classification error difference between BiGAN and Augmented BiGAN.*

- Decoupled Training: A process where the generator is trained first and fixed after. The encoder is trained by optimizing a suitable reconstruction loss in the Z (latent) space. This does not yield satisfactory results.
- BiGAN: In BiGAN's encoder and generator are trained jointly. With the use of feature matching loss. There are better semantic results compared to decoupled training, but when the generator receives an input of the latent space of real images, the results that are acquired are results in a different class. This is called class switching.
- Augmented-BiGAN: An attempt to better augment the issue of class switching, and third pair is constructed. $\big(h(x), g\big(h(x)\big)\big)$, where h finds the latent space of x (real image) and the generator in turn uses that latent space to construct image x.

Depending on the kind of fake image that is being inputted into the GAN's discriminator, the results are affected leaning to good or bad results.

- Weak fake examples: When examples are coming from the generator and are easy for the current discriminator to distinguish between, it becomes a terrible approach for semi-supervised learning. This leads to rendering of unlabeled data, which is not worth much.
- Strong fake examples: When the current discriminator can't know which image is fake or real.
- Moderate fake examples: When images are at a happy medium to the discriminator. This gives the classifier/discriminator more opportunities to learn a supervised loss.

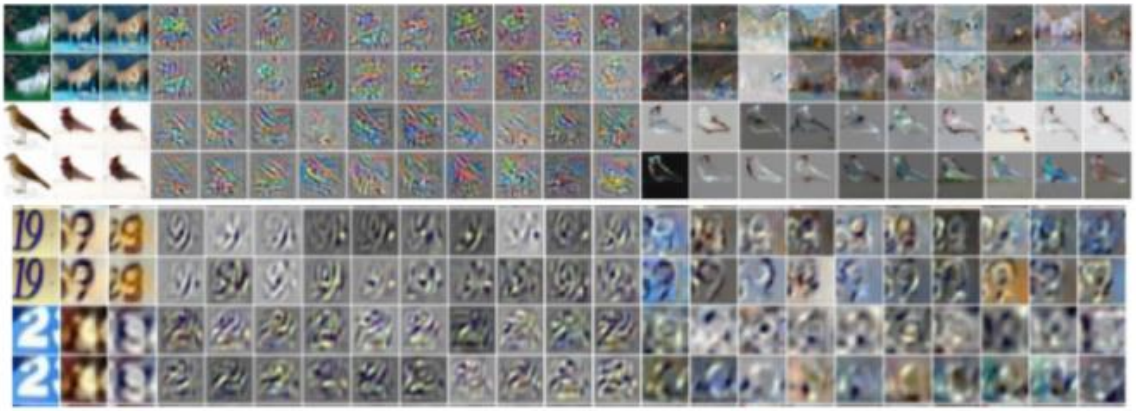


Figure 85. *The image above demonstrates the visualization of the tangents found in the images. Top row (1-4) used CIFAR10 dataset. Bottom row (5-8) uses the SVHN dataset. Columns from left to right. Original Image, reconstructed image using gh (generator and encoder), reconstructed image using g-pph, tangents using encoder (4-13) and tangents using generator (14-23). More examples of generated results can be found in the next consecutive pages.*

Tangents Plot Results

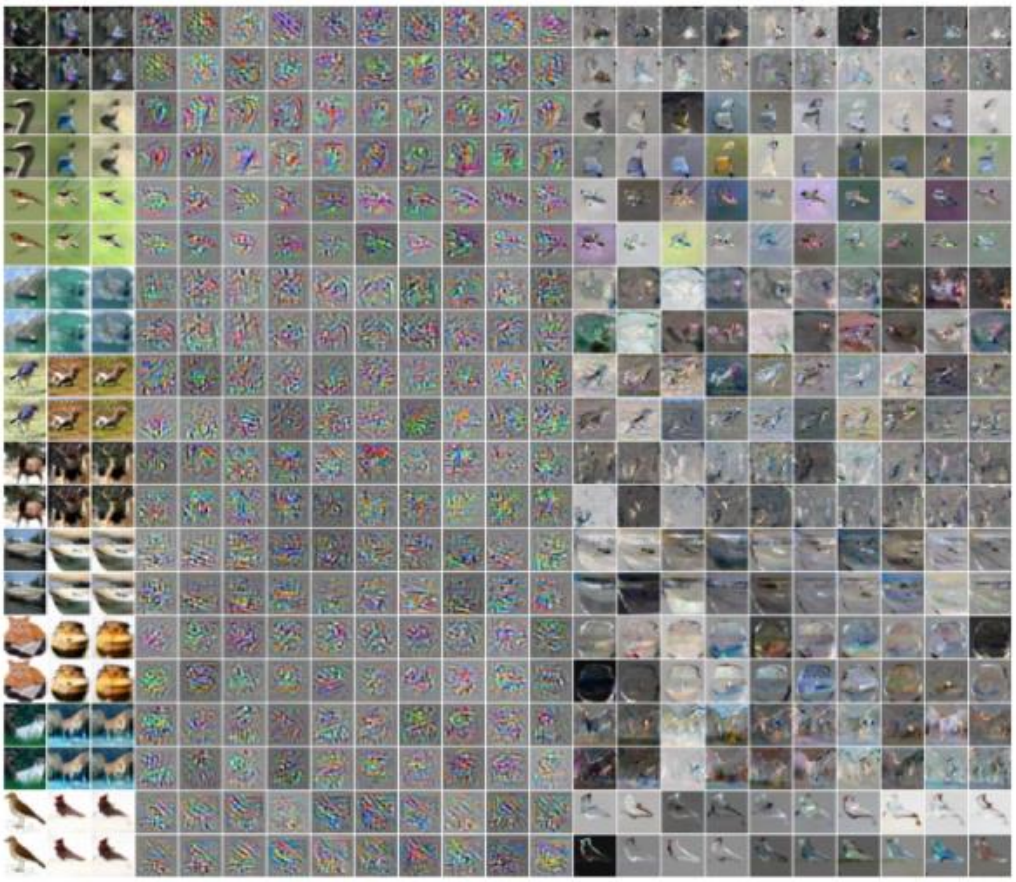


Figure 86. *Tangent plots results on CIFAR10 dataset.*

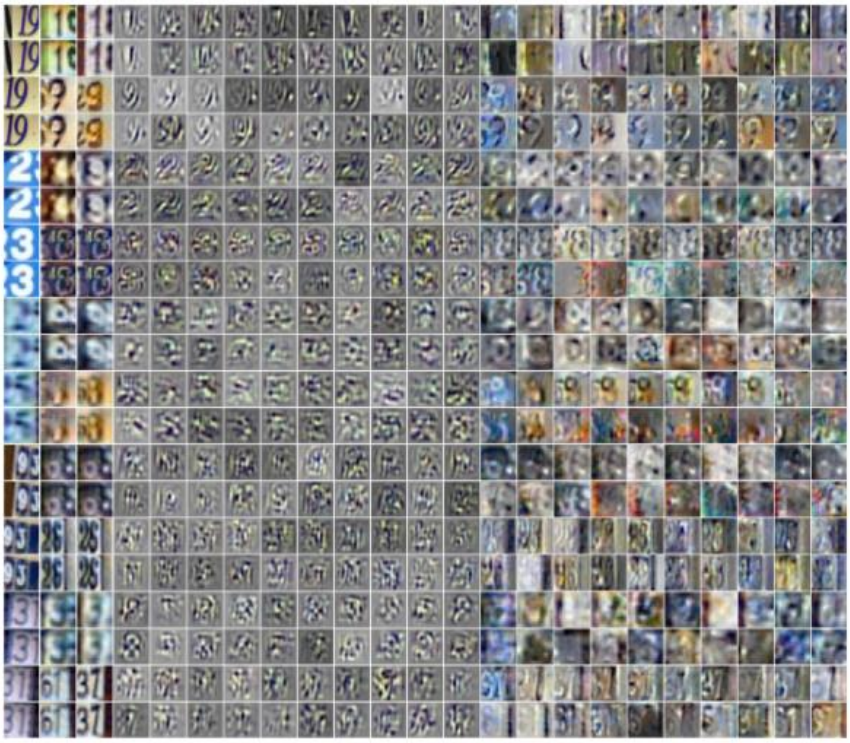


Figure 87. *Tangent plots results on SVHN dataset.*

Reconstruction Plot Results

Figure 88. *CIFAR10 (dataset) reconstructions, on the left are standard BiGAN reconstruction images and on the right are Augmented-BiGAN reconstruction images.*

Figure 89. *SVHN (dataset) reconstructions, on the left are standard BiGAN reconstruction images, and on the right are Augmented-BiGAN reconstruction images.*

Adversarial Feature Learning (Bidirectional Generative Adversarial Networks - BiGANs)

GANs have demonstrated impressive results in capturing semantic variation in data distributions. The ability of distinguishing between groups of data based on identifying that group's label (semantic value). Creating a great mapping from simple latent space distribution to complex data distributions (generator), where a generator is given a certain set of parameters to create a "fake" image. This algorithm can be useful in the applications where semantics are important. But there doesn't exist the learning of the inverse of that mapping.

The proposed framework to include an inverse mapping from data to latent representation is the Bidirectional Generative Adversarial Network (BiGAN) [34]. Framework Diagram seen below. Like regular GANs this framework includes Generator (G) and Discriminator (D) that adversarially play a game against each other. Included in the framework is an encoder (*E*). This encoder maps data x (real data/ images from the dataset) to latent representations (z or $E(x)$). The discriminator will discriminate between tuples of data and its latent space. $(G(z), z)$ and $(x, E(x))$. The encoder and the generator are not connected; therefore, they do not directly learn from each other. It is argued that the encoder and the generator are learning to invert each other to fool the discriminator.

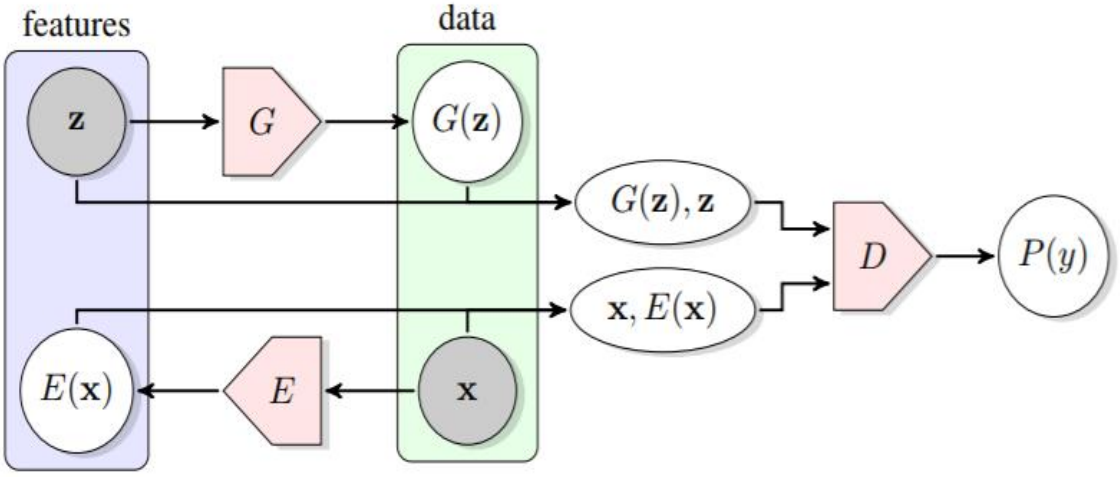


Figure 90. *Structure of Bidirectional Generative Adversarial Networks (BiGANs).*

BiGANs capabilities are evaluated by first training in an unsupervised manner. After, transferring the encoder's learned feature representation for use in auxiliary supervised learning tasks. Tests are run on MNIST and ImageNet datasets. In all experiments the discriminator, generator, and encoder are a multi-layer network (parametric deep network). The discriminator takes initial input data and at each new layer that it encounters the latent space representation is modified.

The first experiment presented is the permutation-invariant MNIST. Each image of the dataset in the dataset is 28 x 28 pixels and must be treated like an unstructured 784D vector. To accomplish this each module is a multi-layer perceptron, unconnected to the spatial structure in the data. The latent space distribution is set to a 50D continuous distribution.

BiGAN	D	LR	JLR	AE (ℓ_2)	AE (ℓ_1)
97.39	97.30	97.44	97.13	97.58	97.63

Figure 91. *One Nearest Neighbors classification accuracy on the permutation-invariant MNIST. Comparing BiGAN trained encoders with other existing experiments. All methods seem to perform at the same level, possibly due to MNIST being a simpler dataset.*

Figure 92. *Qualitative results for permutation-invariant MNIST BiGAN training, including generator samples $G(z)$, real data x, and corresponding reconstructions $G(E(x))$.*

ImageNet results differ from the MNIST dataset. Generally, GANs trained on ImageNet cannot reconstruct images, but are able to capture interesting aspects. In these experiments the encoder architecture is based on AlexNet through the fifth and last convolutional layer. An attempt on AlexNet's discriminator was also tested. Latent distribution is set to 200D continuous uniform distribution.

	conv1	conv2	conv3	conv4	conv5
Random (Noroozi & Favaro, 2016)	48.5	41.0	34.8	27.1	12.0
Wang & Gupta (2015)	51.8	46.9	42.8	38.8	29.8
Doersch et al. (2015)	53.1	47.6	48.7	**45.6**	30.4
Noroozi & Favaro (2016)*	57.1	56.0	52.4	48.3	38.1
BiGAN (ours)	**56.2**	**54.4**	**49.4**	43.9	33.3
BiGAN, 112 × 112 E (ours)	55.3	53.2	49.3	44.4	**34.8**

Figure 93. *Classification accuracy with the ImageNet dataset compared to other standard experiments.*

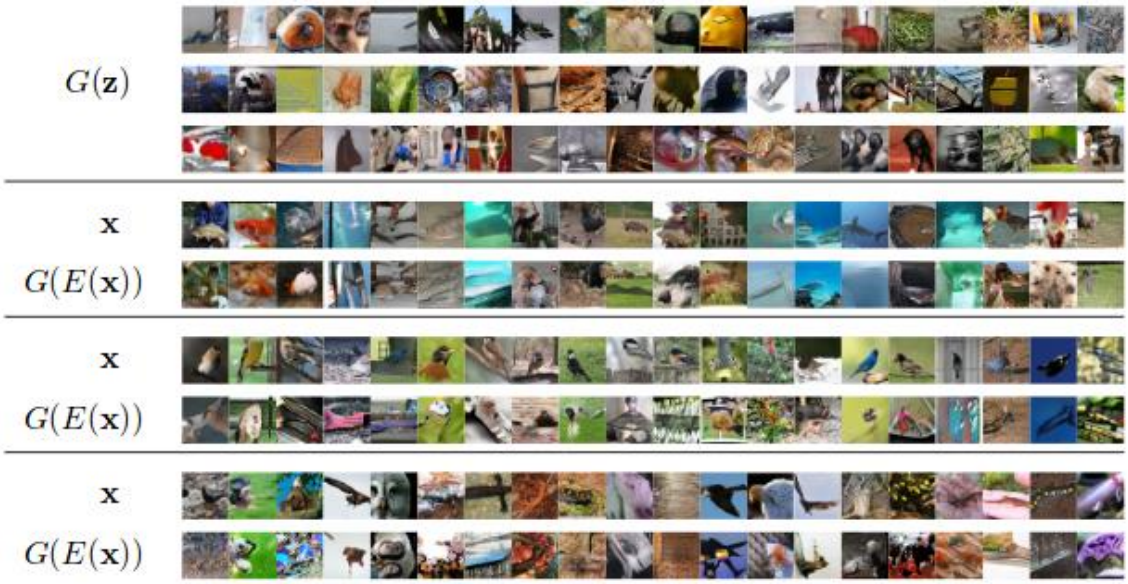


Figure 94. *Qualitative results for ImageNet BiGAN training, including generator sample* $G(z)$*, real data* x*, and corresponding reconstructions* $G(E(x))$*.*

BEGAN: Boundary Equilibrium Generative Adversarial Networks

The Boundary Equilibrium Generative Adversarial Network (BEGAN) [35] is a network that enforces equilibrium. This equilibrium enforcement is paired with a loss derived from the Wasserstein distance. Auto-encoders are trained based on Generative Adversarial Networks (GANs). Helping with the balancing of generators and discriminators during training. This method derives an approximate convergence method. This leads to fast and stable training, which results in high visual quality of images during generation. Image generation is the main focus of this research paper.

During the process of training and testing, the auto-encoder is used as a discriminator. In standard applications of GANs, an attempt to make a data distribution is made. BEGANs match the auto-encoder loss distribution using loss derived from the Wasserstein distance. Using the typical GAN objective with equilibrium term to balance discriminator and generator. This creates an architecture that is easier and simpler to train compared to traditional GANs.

To study the effects of matching the error distributions, auto-encoder loss in introduced. Once that is found, the lower bound of the Wasserstein distance between the autoencoder loss distributions of real and generated samples is found.

1. Introduction of loss for training pixel-wise autoencoder:

$$\mathcal{L}(v) = |v - D(v)|^{\eta} \text{ where } \begin{cases} D : \mathbb{R}^{N_x} \mapsto \mathbb{R}^{N_x} & \text{is the autoencoder function.} \\ \eta \in \{1, 2\} & \text{is the target norm.} \\ v \in \mathbb{R}^{N_x} & \text{is a sample of dimension } N_x. \end{cases}$$

2. Using two distributions of the auto-encoder loss, the Wasserstein distribution is found:

$$W_1(\mu_1, \mu_2) = \inf_{\gamma \in \Gamma(\mu_1, \mu_2)} \mathbb{E}_{(x_1, x_2) \sim \gamma}[|x_1 - x_2|]$$

3. Derivation of Wasserstein lower bound:

$$\inf \mathbb{E}[|x_1 - x_2|] \geqslant \inf |\mathbb{E}[x_1 - x_2]| = |m_1 - m_2|$$

The discriminator is designed to maximize the above equation (derivation of the Wasserstein lower bound). The discriminator function is similar to that of the Wasserstein Generative Adversarial Networks (WGAN), except it matches distributions between losses. It does not require the discriminator to be K-Lipschitz because Kantorovich and Rubinstein duality theorem isn't used.

In most cases, generator and discriminators are not well balanced. The discriminator usually has an easier time winning. To mitigate this situation the equilibrium method is introduced. The generator and discriminator losses must remain balanced. They are balanced when loss of the real images and loss of the generated images have reached a point where they are close to each other. See equation below:

$$\mathbb{E}\left[\mathcal{L}(x)\right] = \mathbb{E}\left[\mathcal{L}(G(z))\right]$$

If the discriminator is unable to identify the difference between generated samples and real ones, the error distributions between both inputs must be similar. A new hyperparameter is introduced to relax the equilibrium. In this model the discriminator will have two competing goals: to auto-encode real images and discriminate real images from generated images. [S] A new equation is created (γ) to balance the discriminator goals. When γ is low, the autoencoder will focus on auto-encoding real images, resulting in lower image diversity.

$$\gamma = \frac{\mathbb{E}\left[\mathcal{L}(G(z))\right]}{\mathbb{E}\left[\mathcal{L}(x)\right]}$$

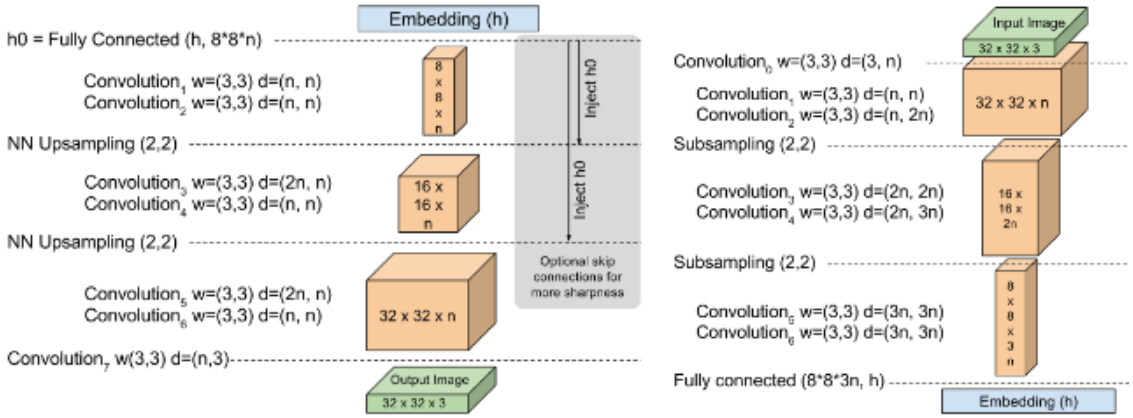


Figure 95. *Demonstrating the network architecture for BEGANs, where both the generator and the discriminator are deep convolutional neural network architecture as an auto-encoder with an encoder and decoder. One distinct difference between generator and discriminator is how the decoder is weighted.*

To maintain equilibrium, Proportional Control Theory is used. Proportional control is a linear feedback system where a controlled variable is corrected directly proportional to the difference between a desired value and measured value. The controlled variable here, regulates the emphasis put on the loss function from the generator during gradient descent. In the beginning of training, the generated images are easy to construct for the auto-encoder. This yields a bigger loss of the distribution of real data. This is maintained during the training process with the equilibrium constraint. The BEGAN method does not require alternating training

between the discriminator and generator or pretraining the discriminator. During training a batch size of 6 is used.

The discriminator is a convolutional deep neural network with an auto-encoder architecture. Auto-encoders have a deep encoder and decoder. The generator has the same architecture as the discriminator decoder. The generator has different weights than the discriminator. The structure, shown in Figure 95**,** has convolutions with a 3 x 3 kernel, which has exponential linear units applied to the outputs. Each layer is repeated an average of 2 times, with more repetitions resulting in better visual results. At each down-sampling the convolutional filters increase linearly. Down-sampling is also done along with sub-sampling with a stride of 2. Up-sampling is implemented using the nearest neighbor algorithm. Between the encoder and decoder, processed data is mapped through full connected layers.

The model was trained using the adam optimizer with an initial learning rate of 0.0001. There was a decaying by a factor of 2 at convergence measurement stalls. High initial learning rates created modal collapses. Models were trained in for various resolutions (32 x 32 pixels to 256 x 256 pixels), with a final down-sampled image of size 8 x 8 pixels. For a more varied data set, 360K celebrity face images was used for training.

Figure 96. *EBGAN results (left) vs. BEGAN results (right).*

The image in Figure 96 demonstrates results of BEGAN at a resolution 128 x 128 pixels. To the left of the group of generated images is the result of EBGANs' results at 64 x 64 pixels resolution. Higher resolution images resulted in loss of sharpness. There is a variation of characteristics, but there are some missing or not existent. No glasses seemed to be generated, there are fewer older people, and there are fewer male generations.

Figure 97. *Demonstration of variation in diversity ratio (γ). The model appears well behaved with a degree of image diversity across the range of values.*

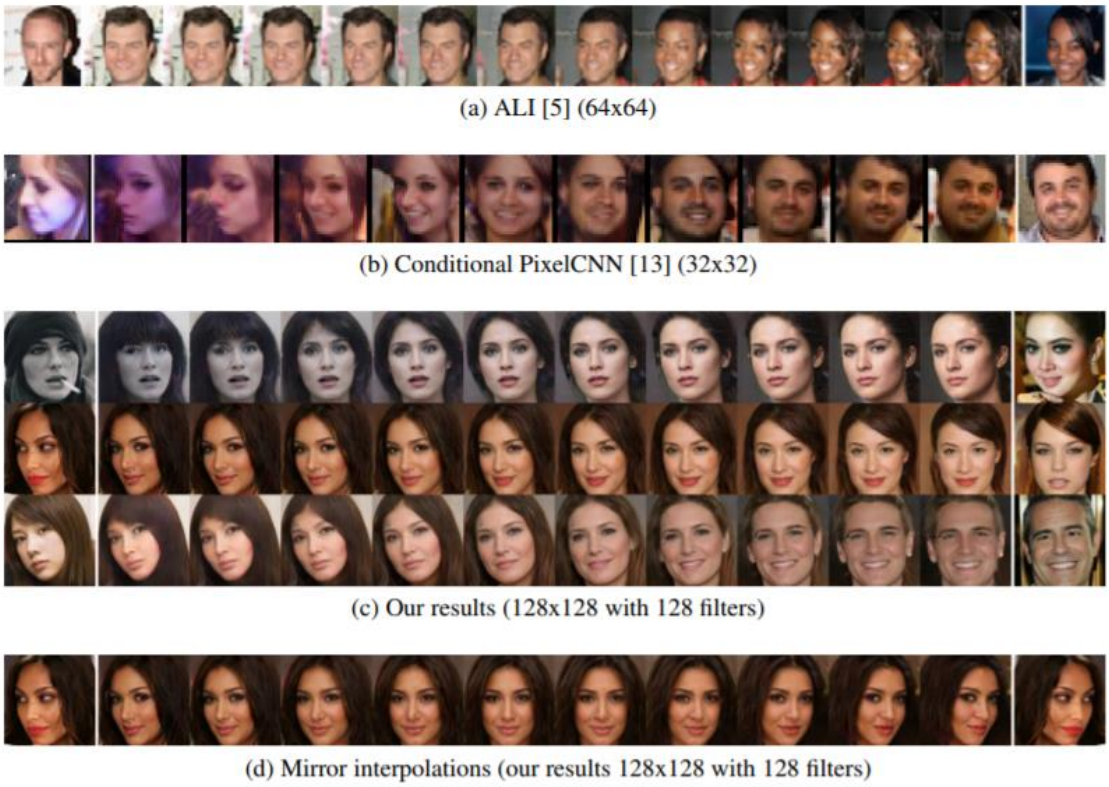


Figure 98. *Interpolations of real images in latent space.*

By interpolating the latent space between two real images, the team can verify that the model can generalize images, and did not just memorize it. In Figure 98.c, the images are not part of the training data. The first and last image are real images that are to be represented and interpolated. Images next to the outer images are their corresponding approximations. Images in-between are results of linear interpolation. To compare ALI and Conditional PixelCNN visual results are included.

During the results evaluation, we can see that there is a diversity in the image samples. Generated images look like the real ones. Interpolations seem to show good continuity. There are some misses, some features went without representation such as the cigarette on the left image.

To check the effect of the equilibrium balancing techniques, experiments are performed where the discriminator has more advantage over the generator. Other experiments where conducted where the generator has more advantage over the discriminator. The model was continuously stable and converged to meaningful results when the equilibrium was maintained. During the reduction of capacity, the discriminator image quality suffered. While for generator reduction there seemed to be small effects.

(a) Starved generator ($z = 16$ and $h = 128$) (b) Starved discriminator ($z = 128$ and $h = 16$)

Figure 99. *Advantaging one network over the other.*

The inception score is computed to measure quality and diversity. The inception score is used in GANs to measure single sample quality and diversity on the inception model. An unconditional version of the model is trained and compared to previous unsupervised results. Hopefully, generating a distribution like the original data. Visual results of BEGAN show exciting potential as shown in Figure 100.

Method (unsupervised)	Score
Real data	11.24
DFM [19]	7.72
BEGAN (ours)	**5.62**
ALI [5]	5.34
Improved GANs [16]	4.36
MIX + WGAN [2]	4.04

Figure 100. *Comparing inception scores compared to other figures*.

Deep Unsupervised Convolutional Domain Adaptation

We know that Domain Adaptation means to adapt the feature representation learned in the source domain (with rich label information) to the target domain (less or no label information). There already has been work done in this field such as the attempt to align feature distribution between source and target domain in fully connected layers (unsupervised DNN-based models), but this creates a constraint near the output level of DNN models. Since there exists this constraint, to output level of models, the knowledge transfer is degraded.

A proposed solution to this problem would be to develop an attention transfer process. Leading to a minimized domain discrepancy on the second-order correlation statistics of attention maps for both source and target domains. Here domain discrepancy is measured in correlation alignment loss. To establish this solution a new method must be created: Deep Unsupervised Convolutional Domain Adaptation (DUCDA) [36]. DUCDA will minimize supervised classification loss of labeled source data. Also minimize unsupervised correlation alignment loss (measured in both convolutional layers and fully connected layers). A multilayer domain adaptation process, that reinforces individual domain adaptation components and enhances the generalization of the CNN models. Multilayer domain adaptation has shown to outperform most state of the art approaches with potential for large scale real world applications.

Generally, to construct a visual recognition system, there is a need to for a large amount of labeled data. This creates a challenging task of manually annotating images for each specific target domain. It's difficult to obtain manually labeled training data for every real-world scenario. Deep CNNs is an attempt to solve this problem in the unsupervised research field. With the great promise of deep CNNs, there has been some bumps on the road. As discrepancy between the distributions of training data and test examples increases there is a proportional increase in the test error.

To help reduce this performance degradation, domain-invariant models were introduced. Domain-invariant models encourage knowledge transfer. They help bridge source and target domains in an isomorphic latent feature space. To decrease the size of that bridge, an attempt to minimize the distanced metric of domain discrepancy (measure by maximum mean discrepancy or correlation) is made. This new CNN-based domain adaptation, minimizes domain discrepancy measure on the feature responses of the convolutional layer. Since spatial information and semantic context are neglected when domain adaptation is only performed in the fully connected (FC) layers, domain adaptation is extended to convolutional layers. This extension better captures spatial context information. Also ensure a better FC layer representation.

The challenges comes when high dimensionality of the convolutional feature responses are in the picture. Requiring substantial number of training data.

Leading to ill-posed solutions. An attention transfer process for the convolutional domain adaptation must be developed. At each convolutional layer the activation maps are calculated by L_p-norm pooling at all convolutional response channels (for both source and target domains). Next, domain discrepancy minimization is performed on second-order correlation statistics of attention maps. The correlation between discriminative part of an image or a visual scene. Unlike existing convolutions informative spatial context in the convolutional attention maps can be transferred from source to target domain. More discriminative object parts can be discovered in the target domain. Leads to enhanced discriminative representation power. Obtaining more effective feature response for both source and target domain. Even without labeling, more perception-level knowledge can be transferred (Source to Target) in lower Convolutional layers of a CNN with some guidance.

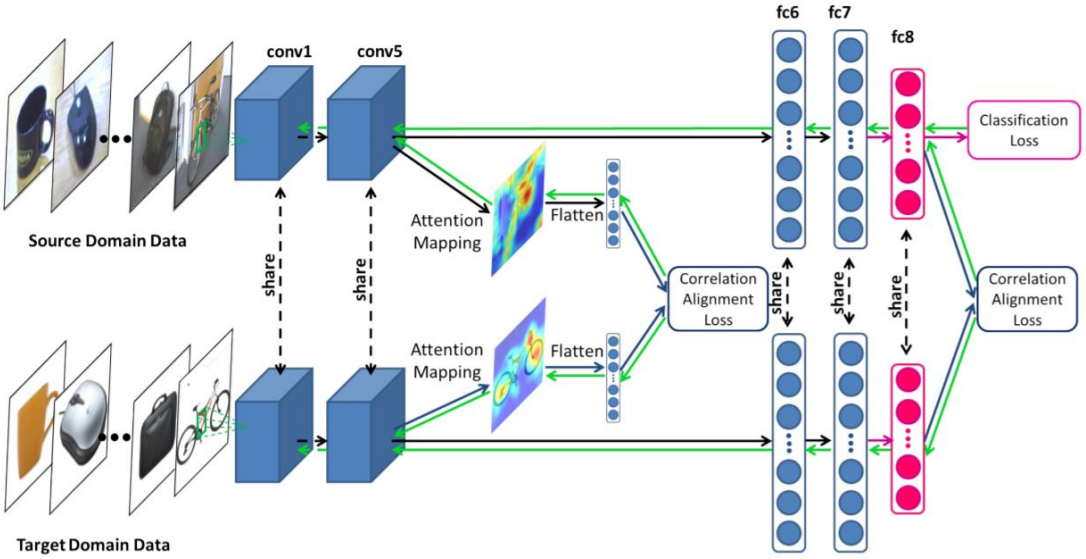


Figure 101. *Deep Unsupervised Convolutional Domain Adaptation. Weights are shared by both source and target domains. Attention mapping is extracted at conv5 (convolutional layer). Correlation Alignment Loss is found at both conv5 and fc8 (fully connected layer).*

With the above described convolutional transfer, DUCDA is proposed. Minimizing the supervised classification loss of labeled source data and unsupervised correlation alignment loss. In the Figure 101, we see the architecture of DUCDA. Where domain adaptation is performed in both convolutional layers and fully connected layers. The multi-layer domain adaptation provides reinforcement to each domain adaptation component. Adaptation in convolutional layers enforces better convolutional representation. Also supports domain adaptation in FC layers While the high-level semantic information from the source in FC layers guide convolutional layers to capture more discriminative patterns of images and reduces the influence of useless background information. DUCDA can also be trained end-to-end.

Algorithm 1 The DUCDA$_{conv5}$ learning algorithm.

Input: Labeled source data: $D_S = \{z_i^s, y_i\}_{i=1}^{N_S}$; Unlabeled target data: $D_T = \{z_j^t\}_{j=1}^{N_T}$;

Output: DUCDA$_{conv5}$ learnt parameters:$\hat{\theta}_{conv5}$ and $\hat{\theta}_{cls}$;

Initialize parameters θ_{conv5} and θ_{cls} with pretrained AlexNet. Re-initialize the weight of fc8 layer with N(0, 0.005).
repeat
  for each source batch B_{S_img} and target batch B_{T_img} **do**
    Calculate $B'_S = \phi(B_{S\_img}; \theta_{conv5})$
    Calculate $B'_T = \phi(B_{T\_img}; \theta_{conv5})$
    Calculate $B_S = Att(B'_S)$
    Calculate $B_T = Att(B'_T)$
    Calculate L_{CLS}
    Calculate $\frac{\partial L_{CLS}}{\partial \theta_{cls}}$
    Update θ_{cls} using SGD
    Calculate $\frac{\partial L_{CLS}}{\partial B'_S} \frac{\partial B'_S}{\partial \theta_{conv5}}$
    Calculate $L_{CAL_{conv5}}$
    Calculate $\frac{\partial L_{CAL_{conv5}}}{\partial B_S} \frac{\partial B_S}{\partial B'_S} \frac{\partial B'_S}{\partial \theta_{conv5}}$
    Calculate $\frac{\partial L_{CAL_{conv5}}}{\partial B_T} \frac{\partial B_T}{\partial B'_T} \frac{\partial B'_T}{\partial \theta_{conv5}}$
    Update θ_{conv5} using SGD
  end for
until *Convergence Or Reach Maximum Iterations*

Figure 102. *Demonstrates the learning algorithm used for DUCDA.*

Adaptation in convolutional layers and FC layers are equally important because features transferability gets worse from convolution 4 to convolution 5. Adaptation in convolutional layers is also important, because FC layers activations are computed based on the convolutional layers activations. If nothing useful is taken from the convolutional layer, a representation incompleteness is created. This incompleteness is continued into the FC layers and cannot be recovered. Adaptation in FC layers only doesn't work. How to solve this? Distill the convolutional layer activations into low dimensional representations through activations-based attention mapping. The knowledge of convolutional layer activations is distilled into dimensional features with appropriate values this has been well adapted in other methods.

Aligning second-order statistics-correlation from source and target distributions have shown meaningful results. The aim of alignment is to transfer correlation from source to target domains. When correlation alignment is adopted discriminative parts of objects are captured well in attention maps. In addition to the fact that attention maps more positively correlated. Making correlation alignment more favorable for adapting attention maps. Correlation Alignment is defined as the distance between the covariances of the vectorized attention maps of conv5 or fc8 features [36].

Experiments were conducted on datasets Office-31 and Office-10 + Caltech-10 datasets. For this dataset testing the top original top layer fc8 was removed. In its place a new FC layer with 31/10 hidden neurons was added. This layer contained the number of categories for Office-31/Office-10+Caltech-10. Weight for fc8 were randomized in the beginning. Learn rate of fc8 was set to 10 times less than the lower levels. Stochastic gradient descent was used, with a momentum of 0.9, weight decay set to 0.0005. Batch sizes of 256 and base learning of 0.001.

Results are shown in the charts below:

Algorithms	A→W	A→D	D→A	D→W	W→A	W→D	Avg
GFK	54.7±0.0	52.4±0.0	43.2±0.0	92.1±0.0	41.8±0.0	96.2±0.0	63.4
TCA	45.5±0.0	46.8±0.0	36.4±0.0	81.1±0.0	39.5±0.0	92.2±0.0	56.9
CNN	61.6±0.5	63.8±0.5	51.1±0.6	95.4±0.3	49.8±0.4	99.0±0.2	70.1
DDC	61.8±0.4	64.4±0.3	52.1±0.8	95.0±0.5	52.2±0.4	98.5±0.4	70.6
DAN_{fc7}	63.2±0.2	65.2±0.4	52.3±0.4	94.8±0.4	52.1±0.4	98.9±0.3	71.1
DAN_{fc8}	63.8±0.4	65.8±0.4	52.8±0.4	94.6±0.5	51.9±0.5	98.8±0.6	71.3
DCORAL	66.8±0.6	66.4±0.4	52.8±0.2	95.7±0.3	51.5±0.3	99.2±0.1	72.1
DAN	**68.5**±0.4	67.0±0.4	**54.0**±0.4	96.0±0.3	**53.1**±0.3	99.0±0.2	72.8
$DUCDA_{conv5}$	62.6±0.6	64.7±0.6	52.4±0.5	96.0±0.5	49.6±0.6	99.4±0.2	70.8
DUCDA	68.3±0.4	**68.3**±0.6	53.6±0.4	**96.2**±0.2	51.6±0.6	**99.7**±0.2	**73.0**

Figure 103. *Accuracy on Office-31 dataset with standard unsupervised adaptation protocol.*

The results of Office-31 so a better average accuracy compared to all other algorithms. Creating a case for convolutional adaptation actually improving fully connected layer adaptation. DUCDA is also close to the best baseline method in three tasks.

Algorithms	A→C	W→C	D→C	C→A	C→W	C→D	Avg
GFK	41.4±0.0	26.4±0.0	36.4±0.0	56.2±0.0	43.7±0.0	42.0±0.0	41.0
TCA	42.7±0.0	34.1±0.0	35.4±0.0	54.7±0.0	50.5±0.0	50.3±0.0	44.6
CNN	83.8±0.3	76.1±0.5	80.8±0.4	91.1±0.2	83.1±0.3	89.0±0.3	84.0
DDC	84.3±0.5	76.9±0.4	80.5±0.2	91.3±0.3	85.5±0.3	89.1±0.3	84.6
DAN_{fc7}	84.7±0.3	78.2±0.5	81.8±0.3	91.6±0.4	87.4±0.3	88.9±0.5	85.4
DAN_{fc8}	84.4±0.3	80.8±0.4	81.7±0.2	91.7±0.3	90.5±0.4	89.1±0.4	86.4
DAN	**86.0**±0.5	**81.5**±0.3	82.0±0.4	92.0±0.3	**92.0**±0.4	90.5±0.2	**87.3**
DCORAL	84.7±0.3	79.3±0.6	82.8±0.5	92.4±0.2	91.1±0.6	91.4±0.6	87.0
$DUCDA_{conv5}$	84.7±0.5	79.4±0.5	**83.0**±0.6	92.7±0.6	85.6±0.6	90.0±0.4	85.9
DUCDA	84.8±0.5	80.2±0.1	82.5±0.6	**92.8**±0.6	91.6±0.6	**91.7**±0.4	**87.3**

Figure 104. *Accuracy on Office-10 + Caltech-10 dataset with standard unsupervised adaptation protocol. For the Office-10 + Caltech-10 dataset testing the average is not the best. But overall it was possible to suggest that the algorithm did failure well compared to its counterparts.*

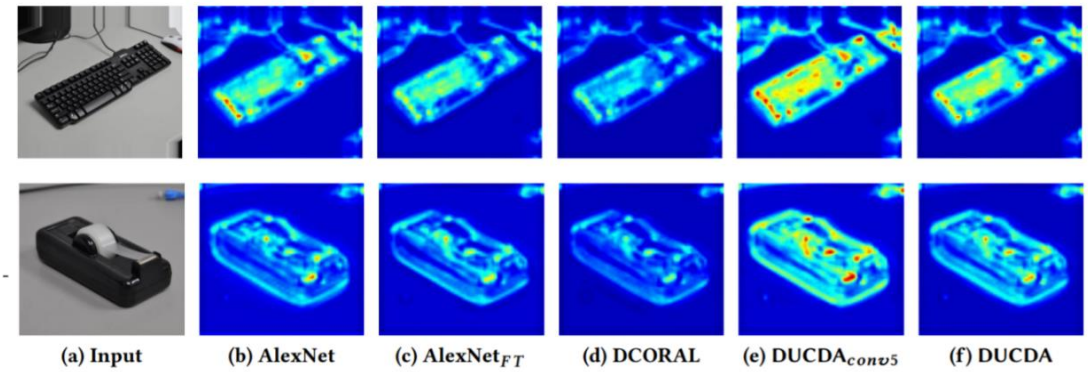


Figure 105. *Attention maps of two samples of target domain DSLR. Each map is extracted from conv5 layers in various networks. The values in DUCDA maps seem larger compared to other attention maps. Unlike other maps, DUCDA maps also seem to cover the whole object.*

Figure 106. *Samples misclassified by DCORAL (red) while correctly classified by DUDCA$_{conv5}$ (green).*

Objectives

Technical Objectives and Requirements

- Visual quality of generated images compared to original images
  - Train architecture and generate images for the **source domain** to establish lower bound for performance
  - Train architecture and generate images for the **target domain** to establish an upper bound for performance
- Compare individually trained datasets generated images to domain adaptation results
- Graph and observe convergence for generator and discriminator loss
- Time for training algorithm (due to time constraints if the algorithm takes more than 5 days to produce meaningful results we will pause our algorithm)
- Aim for high mean classification accuracy

Since research will begin with an attempt to replicate Google's research to later build on, their quantitative results will be the base point for analyzing data produced.

Model	MNIST to USPS	MNIST to MNIST-M
Source Only	78.9	63.6 (56.6)
CORAL [41]	81.7	57.7
MMD [45, 31]	81.1	76.9
DANN [14]	85.1	77.4
DSN [5]	91.3	83.2
CoGAN [30]	91.2	62.0
Our PixelDA	**95.9**	**98.2**
Target-only	96.5	96.4 (95.9)

Figure 107. *Mean classification accuracy (%) for digit datasets. Original research paper results including Google's model results.*

Communication

Mentor

We did have Dr. Boqing Gong as a sponsor for our project; however, he left the university after the fall semester. In our meetings prior to him leaving, we discussed research papers that have aided in the development and design of our architecture. We also discussed and revised our project's overall objective, milestones, code implementation details, and evaluation metrics. Our overall goal for this project was to attempt to develop a new and innovative framework that could potentially match or further improve Google's unsupervised domain adaptation framework.

Team

Our team had two major forms of communication and one form of tracking the progress of our project. Our first form of communication consisted of general face-to-face meetings in the Senior Design laboratory to discuss weekly goals on Mondays. We used Skype to hold video conferences on Saturdays to discuss our weekly goals set on Mondays to make sure that everybody was on the right track and were making progress. To track the progress of our project we used Trello, which allowed us to add specific tasks (cards) to a board and set reminders for our milestones and weekly goals.

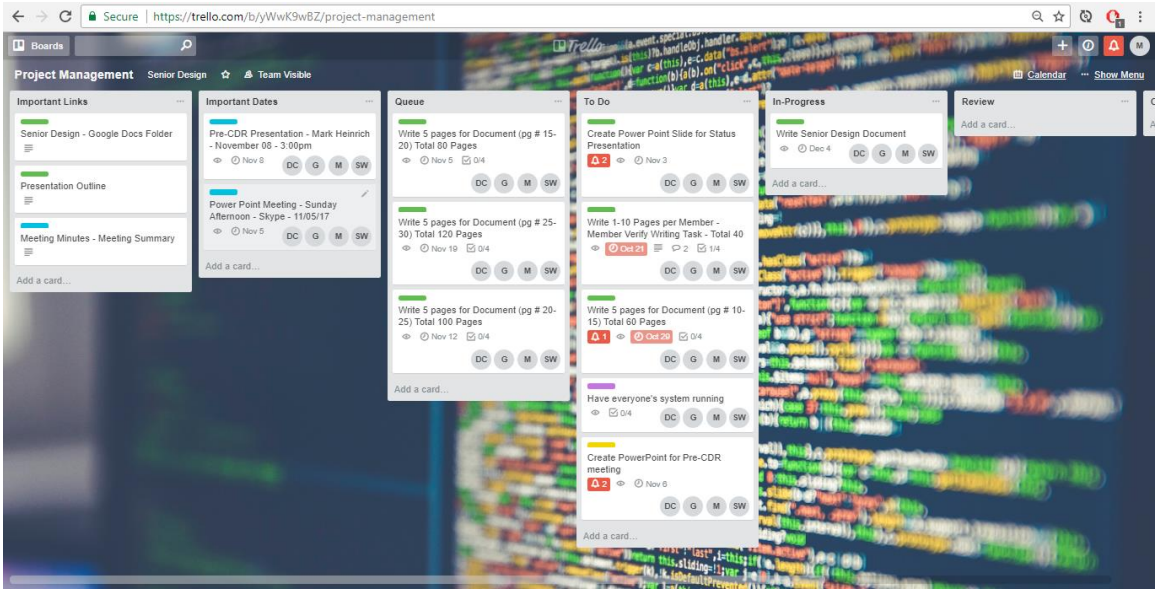


Figure 108. *Our team's Trello board.*

Design

Google's Unsupervised Pixel-Level Domain Adaptation Using GANs

The goal of this project was to improve upon the results of Google's experiment [1]. To achieve this, experiments from Google's research paper needed to be replicated to see if it was possible to reproduce results close to the results mentioned in the paper.

Large and well-annotated datasets play a very important role in research, specifically training algorithms, but creating these datasets is a huge problem as it requires a lot of time to manually annotate each image. To overcome this problem, synthetic data (generated images and labels to the images) could be used for the training of the system. Models trained on synthetic data alone do not have satisfactory results. This means they are not able to generalize to real images. This is where the use of unsupervised domain adaptation comes into play. Typically, unsupervised domain adaptation is an attempt to find mapping from representations of a source domain to a target domain or finding domain invariant features. The focus of this research paper was to change and create images from the source domain to look like they were taken from the target domain while maintaining the original content (object) of the image. GANs are used to perform this task and this approach provides a couple of advantages compared to other approaches already implemented. Some of these advantages are discussed below.

Decoupling from the Task-Specific Architecture

In most domain adaptation models, domain adaptation and task specific architectures are integrated, and one cannot alter task specific architectures without re-training the entire domain adaptation process. This model allows the architecture to be altered by getting rid of this constraint.

Generalization Across Label Space

Previous models couple domain adaptation with a specific task and label spaces, which need to be matched in both the source and target domain, but this model can handle cases where the target label space at test time is different from the label space at training time.

Training Stability

The model also allows for a task-specific loss trained on both source and generated images, and pixel similarity regularization, which will help to avoid mode collapse and make training stable

Data Augmentation

Using source images and a noise vector, this model will create many virtual images which are close to the target domain eliminating the issue of datasets not being large enough for training.

Interpretability

Google's model also allows for an output that is a domain adapted image, which is much easier to interpret compared to an output that is a domain adapted feature vector.

The strategy is to focus on object classification and pose estimation, where objects of interest are in the foreground of an input image. Figure 109 (below) shows generated images while focusing on object classification and pose estimation.

(a) Image examples from the Linemod dataset.

(b) Examples generated by our model, trained on Linemod.

Figure 109. *RGBD samples generated from Google's model vs. real RGBD samples from Linemod dataset. In both (a) and (b), the top row is the RGB part of the image and the bottom row is the depth channel.*

Given a labeled dataset in a source domain and an unlabeled dataset in a target domain, the goal of this model is to train a classifier on data from the source domain that generalizes to the target domain. As this model decouples domain adaptation and image classification, the primary function is to generate images from the source, which will look like they were taken from the target domain. After this process of domain adaptation is done, any classifier can then be trained to classify the images as if no domain adaptation has been performed on these images. For

this model, it is considered that both domains have a primary low-level difference like noise, resolution, illumination, and color.

The following represents a labeled dataset from the source domain:

$$X^s = \{x_i^s, y_i^s\}_{i=0}^{N^s}$$

The following represents an unlabeled dataset from the target domain:

$$X^t = \{x_i^t\}_{i=0}^{N^t}$$

The pixel domain adaptation model consists of a generator function, which takes in an input parameter, noise vector, and source image. This results in generated fake image. The generator function is defined as:

$$G(x^s, z; \theta_G) \rightarrow x^f$$

Using the generator function, a new dataset of the fake images of any size could be generated, as defined below:

$$X^f = \{G(x^s, z), y^s\}$$

The first step of this experiment is to create fake images from the source images. This can be achieved by using a generator. The generator needs two inputs to create a fake image: actual real source images and a noise. These fake images will be used later for discrimination between the fake and real target images. The block diagram of the generator looks like Figure 110 (below).

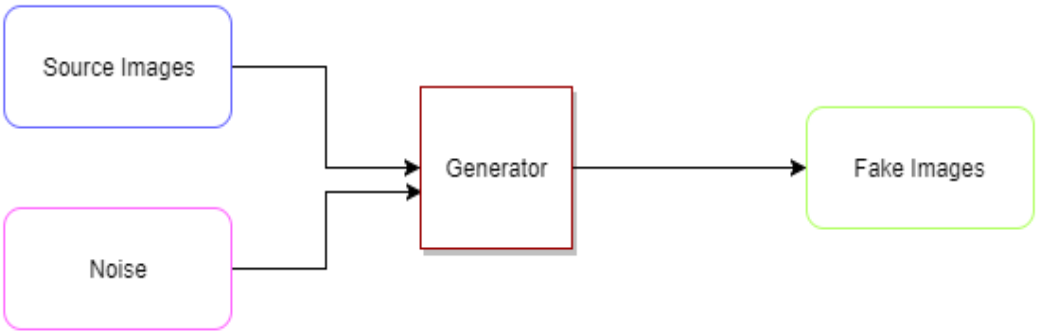


Figure 110. *Block diagram showing inputs and outputs of the generator.*

The section of code in Figure 111 (below) gives us a general idea of how the generator is created and its inputs. The generator is based on a convolutional neural network with residual connections, which maintains the original resolution of images while generating fake images from the source domain and target domain.

```
####################
# Create generator #
####################

with slim.arg_scope(
    [slim.conv2d, slim.conv2d_transpose, slim.fully_connected],
    normalizer_params=batch_norm_params(is_training,
                                        hparams.batch_norm_decay),
    weights_initializer=tf.random_normal_initializer(
        stddev=hparams.normal_init_std),
    weights_regularizer=tf.contrib.layers.l2_regularizer(
        hparams.weight_decay)):
  with slim.arg_scope([slim.conv2d], padding='SAME'):
    if hparams.arch == 'dcgan':
      end_points = dcgan(
          target_images, latent_vars, hparams, scope='generator')
    elif hparams.arch == 'resnet':
      end_points = resnet_generator(
          source_images,
          target_images.shape.as_list()[1:4],
          hparams=hparams,
          latent_vars=latent_vars)
    elif hparams.arch == 'residual_interpretation':
      end_points = residual_interpretation_generator(
          source_images, is_training=is_training, hparams=hparams)
    elif hparams.arch == 'simple':
      end_points = simple_generator(
          source_images,
          target_images,
          is_training=is_training,
          hparams=hparams,
          latent_vars=latent_vars)
```

Figure 111. *Code to create the generator.*

The section of code shown in Figure 112 (below) show the implementation of how to calculate the generator loss. The generator loss aids throughout training so that, in the future, iterations of fake images that are generated have a greater similarity to the target images than in previous iterations. In order to obtain better results and to train the overall network effectively, the loss is not only calculated for the generator, but it also must be calculated for the discriminator.

```
#####################################################################
# Adds a loss which encourages the discriminator probabilities #
# to be high (near one).
#####################################################################

# As per the GAN paper, maximize the log probs, instead of minimizing
# log(1-probs). Since we're minimizing, we'll minimize -log(probs) which is
# the same thing.
style_transfer_loss = tf.losses.sigmoid_cross_entropy(
    logits=end_points['transferred_domain_logits'],
    multi_class_labels=tf.ones_like(end_points['transferred_domain_logits']),
    weights=hparams.style_transfer_loss_weight)
tf.summary.scalar('Style_transfer_loss', style_transfer_loss)
generator_loss += style_transfer_loss

# Optimizes the style transfer network to produce transferred images similar
# to the source images.
generator_loss += _transferred_similarity_loss(
    end_points['transferred_images'],
    source_images,
    weight=hparams.transferred_similarity_loss_weight,
    method=hparams.transferred_similarity_loss,
    name='transferred_similarity')

# Optimizes the style transfer network to maximize classification accuracy.
if source_labels is not None and hparams.task_tower_in_g_step:
  generator_loss += _add_task_specific_losses(
      end_points, source_labels, num_classes,
      hparams) * hparams.task_loss_in_g_weight

return generator_loss
```

Figure 112. *Code for the loss of the generator.*

The discriminator takes two inputs: real target images and generated fake images adapted from the source domain. The goal of the discriminator is to discriminate between real target images and fake images. It gives a probability denoting which image is fake and which one is real. Ideally, after every iteration, the loss should converge more to allow the generator to create better quality images; however, that is not always the case. This continues until a point is reached were the discriminator thinks that the fake images belong to the target domain. This concept is shown below:

$$D(x; \theta_D)$$

The block diagram for the discriminator can be found in Figure 113 (below).

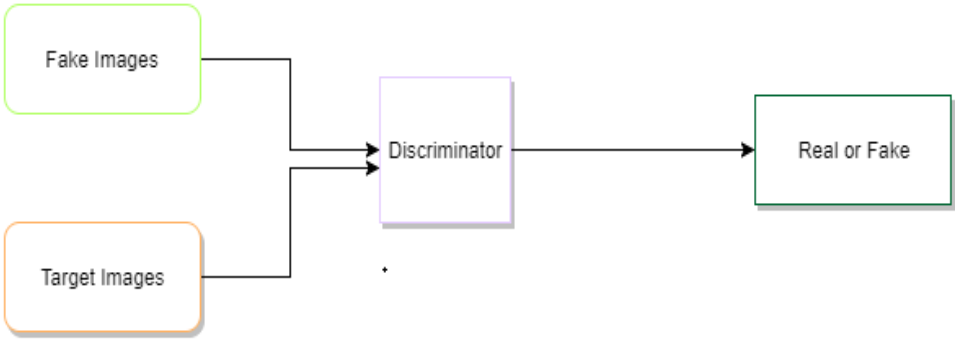


Figure 113. *Block diagram showing inputs and outputs of the discriminator.*

The section of code shown in Figure 114 (below) denotes how the discriminator model is created and how it works. It predicts the domain between the two sets of inputs, which are the target domain and fake images.

```
######################
# Domain Classifier #
######################
if hparams.arch in [
    'dcgan', 'resnet', 'residual_interpretation', 'simple', 'identity',
]:

  # Add a discriminator for these architectures
  end_points['transferred_domain_logits'] = predict_domain(
      end_points['transferred_images'],
      hparams,
      is_training=is_training,
      reuse=False)
  end_points['target_domain_logits'] = predict_domain(
      target_images,
      hparams,
      is_training=is_training,
      reuse=True)
```

Figure 114. *Code for the discriminator and domain classifier.*

The goal is to achieve the minimax optimization in the equation below and push the discriminator to become accurate at distinguishing real and fake images and the generator to generate high-quality domain adapted images using the source domain, which look similar or exactly to the ones in the target domain. To achieve this, an alternation between the discriminator and generator is needed, shown in the following function:

$$\min_{\theta_G,\theta_T} \max_{\theta_D} \propto \mathcal{L}_d(D,G) + \beta \mathcal{L}_t(G,T)$$

α and β are the weights that control the interaction of the losses.

The domain loss functions are shown below:

$$\mathcal{L}_d(D,G) = \mathbb{E}_{x^t}[logD(x^t;\theta_D)] + \mathbb{E}_{x^s,z}[\log\big(1 - D(G(x^s,z;\theta_G);\theta_D)\big)]$$

For the above function, x^s represents the source images and x^t represents the target images.

As discussed earlier in the section of the generator, in the code below in Figure 115, the loss of the discriminator is calculated. The loss of the fake images and real target images are calculated, and then the total loss is returned. In all the loss function calculations, softmax cross-entropy is used.

```
# The domain prediction loss is minimized with respect to the domain
# classifier features only. Its aim is to predict the domain of the images.
# Note: 1 = 'real image' label, 0 = 'fake image' label
transferred_domain_loss = tf.losses.sigmoid_cross_entropy(
    multi_class_labels=tf.zeros_like(end_points['transferred_domain_logits']),
    logits=end_points['transferred_domain_logits'])
tf.summary.scalar('Domain_loss_transferred', transferred_domain_loss)

target_domain_loss = tf.losses.sigmoid_cross_entropy(
    multi_class_labels=tf.ones_like(end_points['target_domain_logits']),
    logits=end_points['target_domain_logits'])
tf.summary.scalar('Domain_loss_target', target_domain_loss)

# Compute the total domain loss:
total_domain_loss = transferred_domain_loss + target_domain_loss
total_domain_loss *= hparams.domain_loss_weight
tf.summary.scalar('Domain_loss_total', total_domain_loss)

return total_domain_loss
```

Figure 115. *Code for the loss of the discriminator.*

The model is augmented with a classifier, which assigns task-specific labels to images. After the generation of the fake images and comparing fake and real images in the discriminator, the next step is to classify the target domain. Before classifying the target images, the classifier needs to be trained. For the training of the classifier, different approaches could be used. The classifier can be trained using only fake images, but doing so will create the need for too many iterations with different initializations since the model would be unstable. Results could be achieved with the above model mentioned, but it is time consuming and difficult. Instead of training the classifier only on the fake images, it could be trained on both real source images and fake images, which will stabilize the model and will require fewer iterations than training the classifier on fake images only.

The classifier function is outlined below:

$$T(x; \theta_T) \rightarrow \hat{y}$$

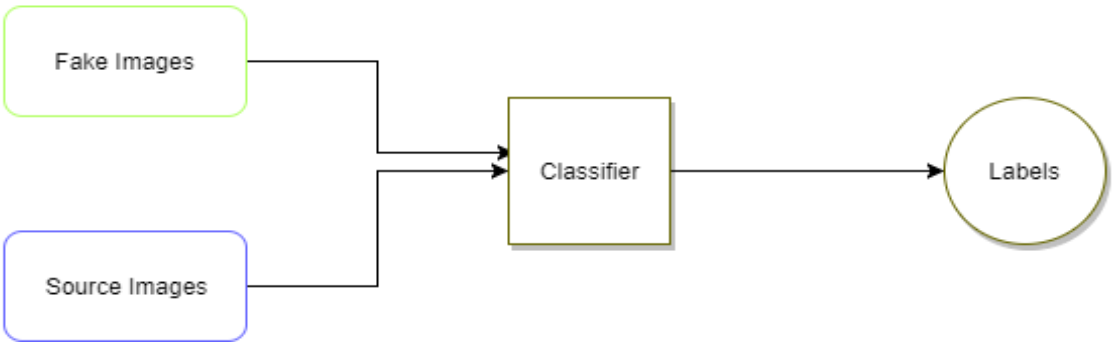


Figure 116. *Block diagram showing inputs and outputs of the classifier.*

Moreover, the source images and fake images are fed into a classifier, which classifies them by assigning labels to the images. The section of code in Figure 117 (below) performs the training of the classifier when it is fed in real source images and fake images.

```
###################
# Task Classifier #
###################
if hparams.task_tower != 'none' and hparams.arch in [
    'resnet', 'residual_interpretation', 'simple', 'identity',
]:
  with tf.variable_scope('discriminator'):
    with tf.variable_scope('task_tower'):
      end_points['source_task_logits'], end_points[
          'source_quaternion'] = pixelda_task_towers.add_task_specific_model(
              source_images,
              hparams,
              num_classes=num_classes,
              is_training=is_training,
              reuse_private=False,
              private_scope='source_task_classifier',
              reuse_shared=False)
      end_points['transferred_task_logits'], end_points[
          'transferred_quaternion'] = (
              pixelda_task_towers.add_task_specific_model(
                  end_points['transferred_images'],
                  hparams,
                  num_classes=num_classes,
                  is_training=is_training,
                  reuse_private=False,
                  private_scope='transferred_task_classifier',
                  reuse_shared=True))
```

Figure 117. *Code for the training of the classifier.*

The task-specific loss function is given below:

$$\mathcal{L}_t(G,T) = \mathbb{E}_{x^s,y^s,z}[-y^{s^T} logT(G(x^s,z;\theta_G);\theta_T - y^{s^T} logT(x^s);\theta_T]$$

For the above function, y^s represents labels for the source images, x^s represents source images, and x^t represents the target images.

The piece of code in Figure 118 (below) performs the softmax cross-entropy on images and calculates the loss for the classifier.

```
def _add_task_specific_losses(end_points, source_labels, num_classes, hparams,
                              add_summaries=False):
  """Adds losses related to the task-classifier.

  Args:
    end_points: A map of network end point names to `Tensors`.
    source_labels: A dictionary of output labels to `Tensors`.
    num_classes: The number of classes used by the classifier.
    hparams: The hyperparameters struct.
    add_summaries: Whether or not to add the summaries.

  Returns:
    loss: A `Tensor` representing the total task-classifier loss.
  """
  # TODO(ddohan): Make sure the l2 regularization is added to the loss

  one_hot_labels = slim.one_hot_encoding(source_labels['class'], num_classes)
  total_loss = 0

  if 'source_task_logits' in end_points:
    loss = tf.losses.softmax_cross_entropy(
        onehot_labels=one_hot_labels,
        logits=end_points['source_task_logits'],
        weights=hparams.source_task_loss_weight)
    if add_summaries:
      tf.summary.scalar('Task_Classifier_Loss_Source', loss)
    total_loss += loss
```

Figure 118. *Code for the loss of the classifier.*

The whole architecture with the inputs, outputs, number of layers, and name of the layers in the generator and discriminator is shown in Figure 119 (below).

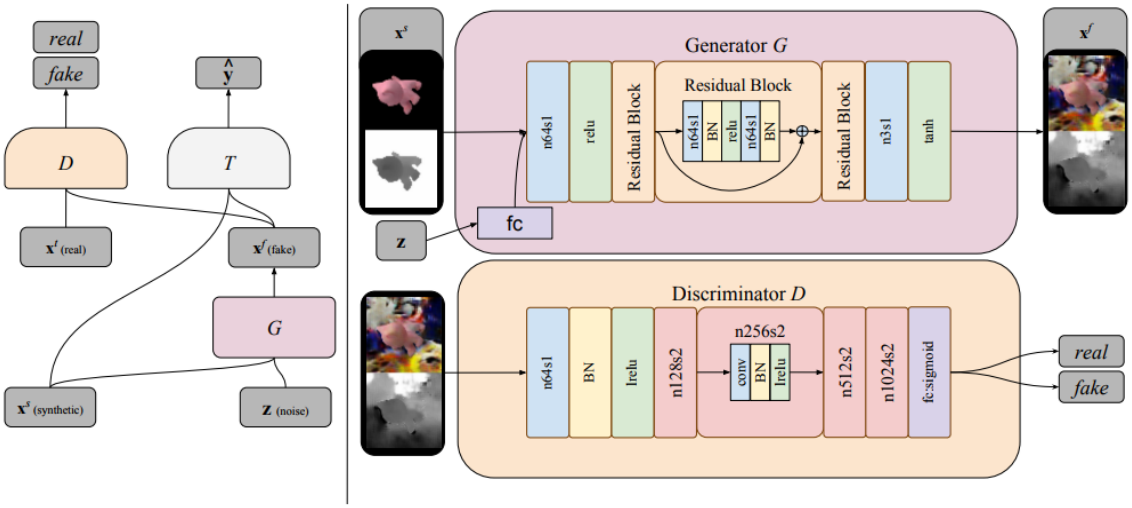


Figure 119. *On the left of the figure, the high-level overview of the overall architecture is shown. On the right, components of the generator and discriminator are explained. A convolution with stride 1 and 64 channels is indicated as n64s1 in the image. lrelu denotes leaky ReLU nonlinearity. BN denotes the batch normalization layer and FC denotes a fully connected layer.*

This method is evaluated on MNIST, MNIST-M, USPS and LineMod datasets. Qualitative evaluation is done by visually inspecting generated images to examine the underlying pixel adaptation process from the source to target domain. Quantitative evaluation is done by comparison of performance of previous models with source only and target only datasets which did not use any domain adaptation process.

MNIST to USPS

The 10 image classes (digits from 0-9) from the MNIST dataset are used as source images and the same classes (digits from 0-9) from the USPS dataset are used as target images. For source only validation, 50,000 images out of 60,000 images from the MNIST dataset are used as training images and the remaining 10,000 images are used to validate Source-only experiments. For the USPS dataset, 6,562 training images, 729 validation images, and 2,007 test images are used.

MNIST to MNIST-M

MNIST-M is the variation of the MNIST dataset which is used for unsupervised domain adaptation. MNIST-M contains MNIST images which have colors inverted with the background. MNIST is the source domain and MNIST-M is the target domain. 10,000 out of the 59,001 MNIST-M training examples are used.

Synthetic Cropped LineMod to Cropped LineMod

The Cropped LineMod dataset contains images of one of eleven small objects from the LineMod dataset centered on a colored background. The Synthetic Cropped LineMod datasets consists of the same object models centered on a black background. The synthetic cropped dataset is treated as the source domain and the cropped dataset is treated as the target domain. The model is trained on 109,028 rendered source images, 9,673 real images which are target images, 1,000 images for validation, and 2,655 for testing purpose. This scenario is used for both classification and pose-estimation. In this case, the classifier outputs a 3D pose estimation in the form of a quaternion vector which is a little different than its generic output.

$$T(x;\theta_T) \rightarrow \{\hat{y}, \hat{q}\}$$

The loss of the classifier will also change to the equation below, where the first and second terms are the same as previous loss, but the third and fourth terms are the log of a 3D rotation metric for quaternions.

$$\mathcal{L}_t(G,T) = \mathbb{E}_{x^s,y^s,z}[-y^s \log \hat{y}^s - y^{s^T} log\hat{y}^f + \mathcal{E}\log\left(1 - \left|q^{s^T}\hat{q}^s\right|\right) + \mathcal{E}\log(1 - \left|q^{s^T}\hat{q}^f\right|)]$$

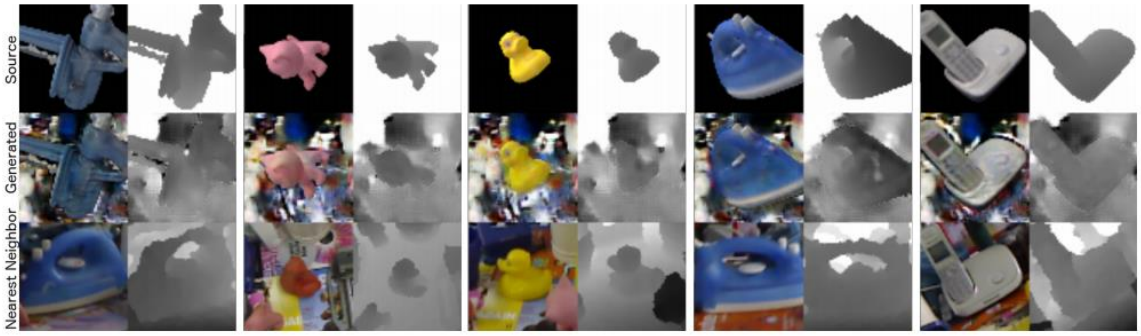


Figure 120. *The results of the model's ability to generate samples when trained on the Synthetic Cropped LineMod dataset to adapt to the Cropped LineMod dataset are shown in figure below. The top row shows source RGB and depth image pairs from Synthetic Cropped LineMod. The middle row shows adapted images from source images and noise vector. The bottom row shows the nearest looking image from the target domain to the generated adapted image.*

All implementation is done using TensorFlow and training is done using an Adam optimizer. The generator is a residual neural network which maintains the resolution of the original image. The noise vector is fed to a fully connected layer which transforms it to a channel which is connected to the input image as an extra channel. The discriminator is a convolutional neural network and the number of layers depends on the image resolution. The first layer is a stride 1x1 convolution followed by a stacking stride of 2x2 convolution until resolution is reduced to 4x4. The number of filters in all layers of G and the first layer of D is 64.

The tables below show the mean classification accuracy for digit datasets. The “Source Only” and “Target-only” rows are the results when no domain adaptation is used. 1,000 labeled target images were used as a validation dataset. The new created architecture uses combinations of source and target datasets. The results of the new architecture are first compared to “Source Only” and then compared to “Target-only”. These represent a lower bound and an upper bound on performance, respectively. Additionally, the new architecture is able to reduce the mean angle error for adaptation of the Synthetic Cropped LineMod dataset to the Cropped LineMod dataset, which is more than half accurate than the previous state-of-the-art architecture.

Model	MNIST to USPS	MNIST to MNIST-M
Source Only	78.9	63.6 (56.6)
CORAL [41]	81.7	57.7
MMD [45, 31]	81.1	76.9
DANN [14]	85.1	77.4
DSN [5]	91.3	83.2
CoGAN [30]	91.2	62.0
Our PixelDA	**95.9**	**98.2**
Target-only	96.5	96.4 (95.9)

Figure 121. *Mean classification accuracy for digit datasets.*

Model	Classification Accuracy	Mean Angle Error
Source-only	47.33%	89.2°
MMD [45, 31]	72.35%	70.62°
DANN [14]	99.90%	56.58°
DSN [5]	**100.00%**	53.27°
Our PixelDA	99.98%	**23.5°**
Target-only	100.00%	6.47°

Figure 122. *Classification accuracy and pose error for Synthetic Cropped LineMod to Cropped LineMod.*

Model–RGB-only	Classification Accuracy	Mean Angle Error
Source-Only–Black	47.33%	89.2°
PixelDA–Black	94.16%	55.74°
Source-Only–INet	91.15%	50.18°
PixelDA–INet	96.95%	36.79°

Figure 123. *Mean classification accuracy and pose error using different backgrounds from the source domain.*

Conclusion

A state-of-the-art method for unsupervised domain adaptation is proposed which outperforms previous work in unsupervised domain adaptation. For the Synthetic Cropped LineMod to Cropped LineMod experiment, this model more than halves the error of pose estimation.

General Model

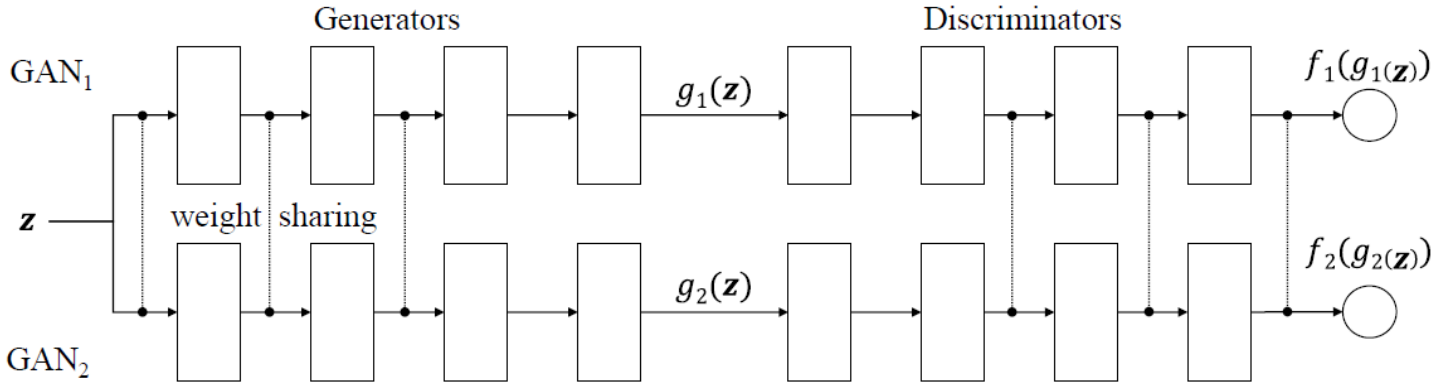


Figure 124. *Diagram of framework architecture. The design developed was chosen based on results found during research. The architecture is that of the CoupledGANs [23]. GAN_1 and GAN_2 are each replaced by the WGAN [25] for our CoWGAN model or the InfoGAN [10] for our CoIGAN model.*

Model Architectures

CoWGAN

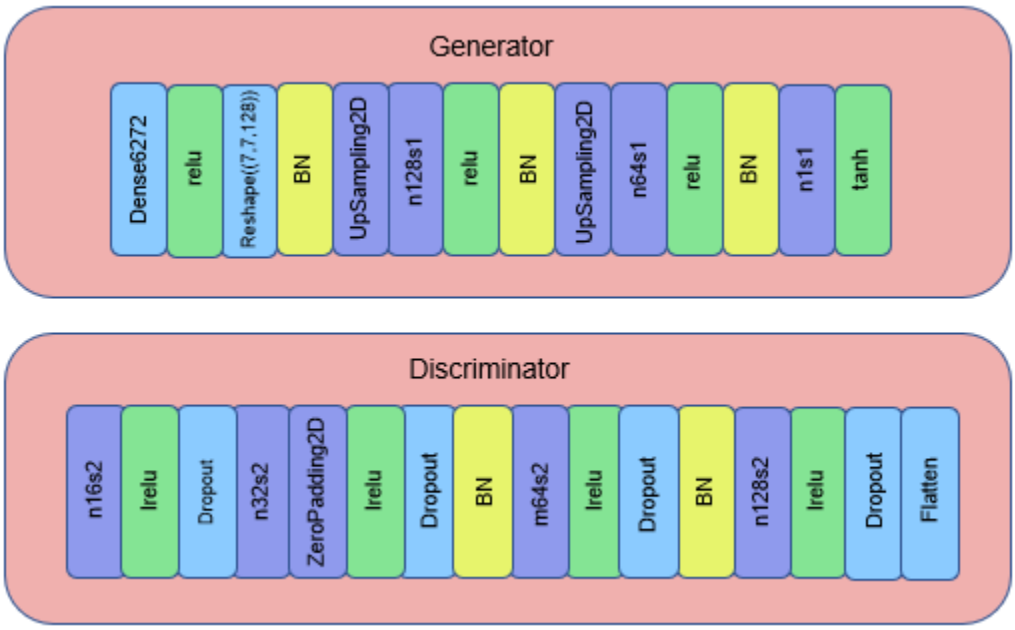


Figure 125. *WGAN generator and discriminator used for the CoWGAN and partial weight sharing.*

Our first model consists of two Wasserstein GANs coupled together. In Figure 124, GAN_1 and GAN_2 are each replaced by a WGAN where the generator and critic of the WGAN are used instead of GAN_1's and GAN_2's generator and discriminator. The loss for the model is also changed to be the Wasserstein distance. The architectures for the WGAN generator and discriminator (critic) used for partial weight sharing are shown in Figure 125.

For the weight sharing of CoWGAN, each generator consists of batch normalization layers, Rectified Linear Units (ReLU) and tanh activation layers, a couple of core layers, and three convolutional layers. Each discriminator consists of batch normalization layers, LeakyReLU activation layers, flatten and dropout core layers, and four convolutional layers.

The generators also contain additional layers where no weight sharing occurs. In order, the additional layers are a dense layer, LeakyReLU layer, batch normalization layer, dense layer, tanh activation layer, and reshape layer.

CoIGAN

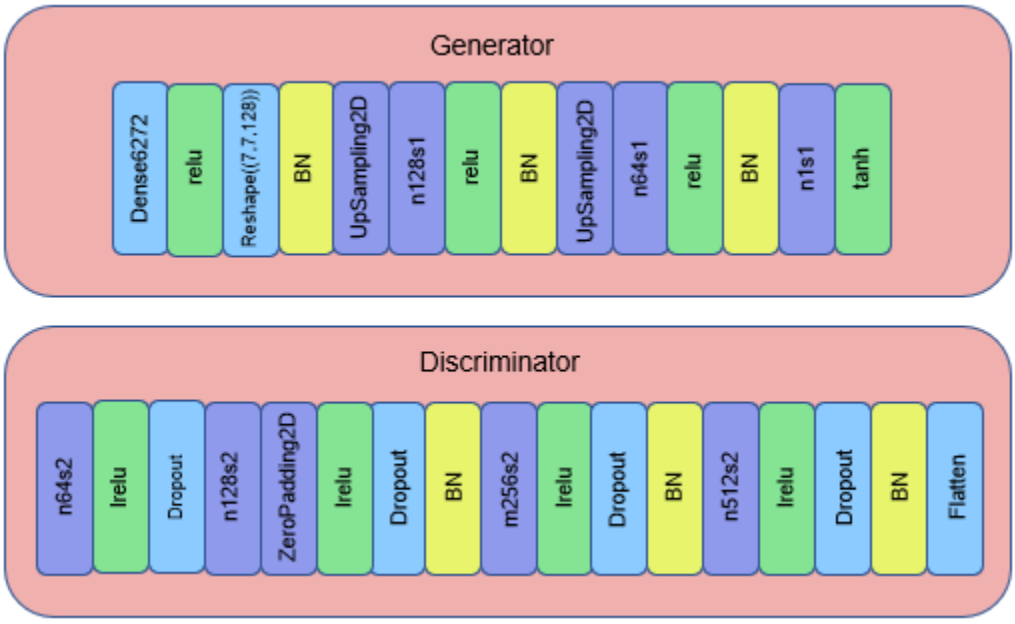


Figure 126. *InfoGAN generator and discriminator used for the CoIGAN and partial weight sharing.*

Our second model consists of two InfoGANs coupled together similar to the WGANs in the CoWGAN. Instead of replacing GAN_1 and GAN_2 each with a WGAN, we use the InfoGAN generator and discriminator. Although, the loss for the model stays the same as the CoGAN (binary cross-entropy). The architectures for the InfoGAN generator and discriminator (critic) used for partial weight sharing are shown in Figure 125.

For the weight sharing of CoIGAN, each generator consists of batch normalization layers, Rectified Linear Units (ReLU) and tanh activation layers, a couple of core layers, and three convolutional layers. Each discriminator consists of batch normalization layers, LeakyReLU activation layers, flatten and dropout core layers, and four convolutional layers.

The generators also contain additional layers where no weight sharing occurs. In order, the additional layers are a dense layer, LeakyReLU layer, batch normalization layer, dense layer, tanh activation layer, and reshape layer.

The overall design of the CoIGAN is similar to the CoWGAN except for the number of filters used for the convolutional layers in the discriminator and an additional batch normalization layer in the discriminator. Additionally, each model uses different losses.

Datasets

MNIST

MNIST [38] is a commonly used dataset for training image processing systems and in machine learning. It was created and first used by Corinna Cortes, a research scientist at Google Labs in New York, and Christopher J.C. Burges, a researcher at Microsoft Research in Redmond. It is a subset of images from Special Database 1 (SD-1) and Special Database 3 (SD-3) of the NIST dataset. Originally, the creators thought that SD-3 was the best for training and SD-1 was the best for testing. They eventually decided that they should mix the two sets since it should not matter which one is used for training or testing; the results should still be precise. For the training set of the new database, they chose 30,000 images from SD-1 and 30,000 images from SD-3. For the testing set, they chose 5,000 images from SD-1 and SD-3 each. Additionally, they changed the image size from 20 x 20 pixels to 28 x 28 pixels. To accomplish this, they found the center of the pixels in each original image and put that spot at the center of the 28 x 28 area. The resulting dataset is the MNIST database.

In general, MNIST consists of 70,000 images of handwritten digits. There are 60,000 for the training set and 10,000 for the testing set. Each set contains labels for the images where their values correspond to the digit in each figure.

Regarding Google's pixel domain adaptation experiment, MNIST was used as a source domain. For this reason, we also used it as our source domain for training and testing our models.

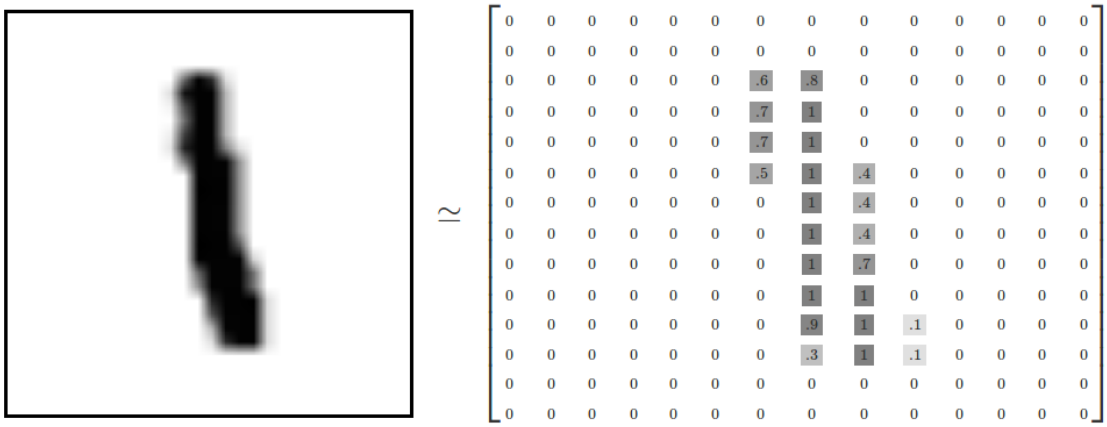


Figure 127. *An example of an image from the dataset and how it looks in the 28 x 28 pixel field.*

MNIST-M

The MNIST-M dataset [1] is a variation of the MNIST dataset meant for unsupervised domain adaptation. It contains MNIST digits mixed with random color patches from the Berkeley Segmentation Data Set and Benchmarks 500 (BSDS500) [40]. To make MNIST-M, each digit from MNIST was used as a binary mask and then was inverted in conjunction with the background images from BSDS500. It can be easily created by downloading BSDS500 and running a python script on MNIST.

```
curl -O http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/BSR/BSR_bsds500.tgz
python create_mnistm.py
```

Figure 128. *Code and script for turning MNIST into MNIST-M.*

MNIST-M was used as one of the target domains for Google's experiment and was used to replicate the experiment and evaluate our own GAN implementations. Additionally, we used the script above to generate the dataset.

Figure 129. *Set of images from the MNIST-M dataset.*

USPS

The USPS dataset [43, 44] contains 8-bit grayscale images of handwritten digits. Each image is a number from 0 to 9 and is 16 x 16 pixels in size. Originally, they were obtained by scanning the digits from envelopes by the United States Postal Service. The digits were binary and, since they were of different sizes and positions, were deslanted and normalized to a 16 x 16 pixel field. Overall, there are 7,291 images in the training set, 2,007 images in the testing set, and 10 classes of images.

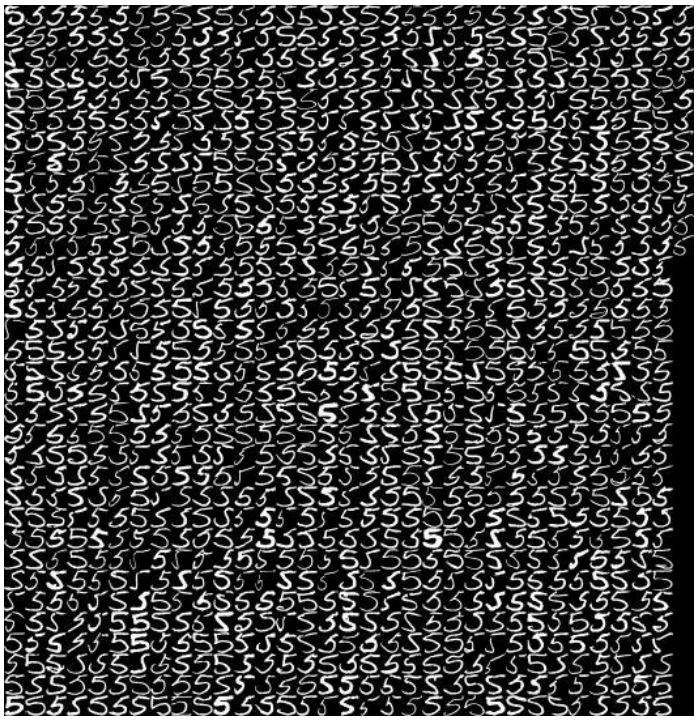


Figure 130. *Images of the digit 5 from the USPS dataset*

Google chose USPS as a second target domain, using 6,562, 729, and 2,007 images for training, validation, and testing, respectively. We used USPS and the same division of images for our project.

Synthetic Cropped LineMod and Cropped LineMod

The LineMod dataset [1, 46] is a set of RGB images of multiple textureless objects in a cluttered setting. The positions of the objects, including depth, vary between images. To get the Cropped LineMod dataset, Google took the dataset and cropped the images so that there was solely one object in the center with an RGB background. They used only 11 of the available objects which were identified as duck, iron, phone, holepuncher, lamp, cam, driller, can, ape, benchviseblue, and cat. The Synthetic Cropped LineMod dataset consisted of CAD models of the same Cropped LineMod objects. The objects were set in different poses with a black background. For evaluation of their model, Google had Synthetic Cropped LineMod and Cropped LineMod as a source and target domain, respectively.

Since the datasets were not freely available, Synthetic Cropped LineMod and Cropped LineMod were not included in our evaluation of our models and in the replication of Google's experiment.

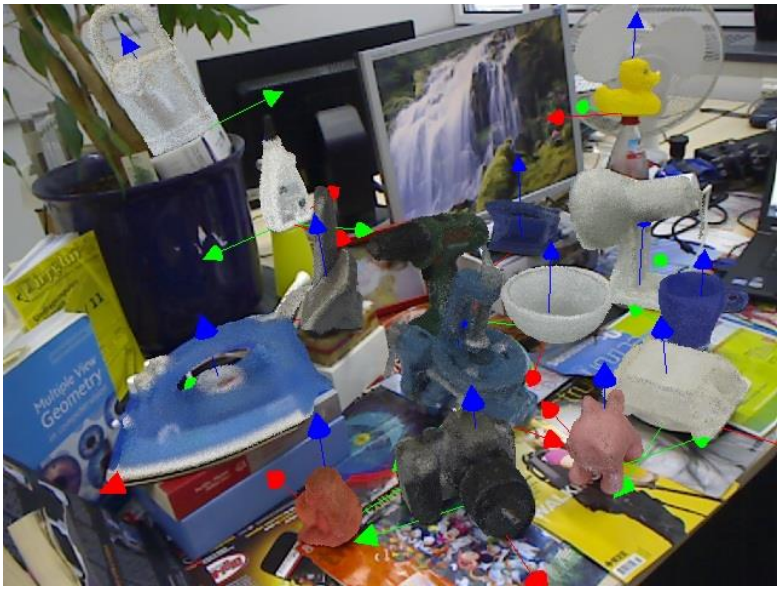


Figure 131. *Image from the LineMod dataset. The objects are positioned close to each other against a colored background and are augmented with their 3D models and coordinate systems.*

Languages and Libraries

Python

Figure 132. *Python logo.*

Python is a high-level, multi-paradigm programming language which was released in 1991 for general use. It is the most popular among data scientists and machine learning developers, supporting the use of deep learning models, packages, and frameworks. Examples of these include SciKit-Learn, SciPy, Pandas, and NumPy. Python is also easy to learn, easily readable, and has a math-like syntax, which make it appealing for the development of machine learning systems. For these reasons, we decided to use Python as our programming language. We used the free 4.2.0 version of Anaconda, an open source distribution of Python that is easy to install and contains a large number of packages.

TensorFlow

Figure 133. *TensorFlow logo.*

In order to develop and evaluate our models, we looked at the machine learning software libraries Keras, Torch, Theano, and TensorFlow. We quickly decided against Torch because it uses the scripting language Lua and we would like to stick to libraries written in Python. We then narrowed it down to Keras and TensorFlow because of our group's over familiarity with the two. Keras is a high-level framework that is easy to learn, highly modular, extensible, and is a wrapper to TensorFlow and other libraries. On the other hand, while TensorFlow is a low-level framework and has a sharp learning curve, it provides more functionalities and flexibility, and offers more control over our models than Keras. Additionally, it has a specialized debugger for visualization of its data flow graphs and a large active community.

After assessing both libraries, we chose to use both since we were able to have a deeper understanding of the models we created and had as much functionality available to us as possible. We used TensorFlow version 1.4 and updated it as needed. We also used some of the deep learning libraries available for TensorFlow such as TF-Slim and TFLearn.

Keras

Figure 134. *Keras logo.*

Although we initially had decided to use Tensorflow as our main library for machine learning, we quickly learned the usability of Keras. Keras is a python based high-level neural network API. It can be used on top of Tensorflow and adds additional user-friendly methods to machine learning. Keras' goal is to enable fast experimentation. It allows easy and fast prototyping with both convolutional and recurrent networks support. Keras also has the capability to run both on CPU and GPU. Since our group had access to both a CPU only server and GPU enabled desktop, this allowed us to run the our experiments on both machines.

NumPy

Figure 135. *NumPy logo.*

Since we dealt with a lot of data and inputs, we needed to be able to perform manipulations on them sufficiently fast. For this reason, we used the NumPy library which is a Python package for data analysis. It contains broadcasting functions, integration tools for other code, and an N-dimensional array object. We primarily be used its array object to deal with data. The array is similar to arrays of other languages (contains elements indexed by positive integers), but it is more like a mathematical object and has greater functionality. Vector and matrix operations, such as elementwise arithmetic and matrix multiplication, can be performed on it. Additionally, the basic arithmetic operations can be done on arrays of different sizes by broadcasting the smaller array so that the arrays have compatible shapes. Overall, the functionality of the NumPy array and the speed at which we can manipulate it were useful for designing our GAN models.

```
import numpy as np

# Creating a NumPy array (1D array, 2D array, and 3D array with all 1's)
arr1 = np.array([1,3,5,7,9])
arr2 = np.array([(2,4,6,8,10),(1,3,5,7,9)])
arr3 = np.ones((5,2))

# Checking properties of a NumPy array (check the dimensions and number of elements in the array)
arrayDimensions = arr2.shape
arrayElements = arr2.size

# Performing mathematical operations on NumPy arrays (elementwise and matrix operations)
# Broadcasting occurs in the elementwise addition and multiplication below
elemAddition = np.add(arr1, arr2)
elemMultiplication = np.multiply(arr1, arr2)
matrixProduct = np.dot(arr2, arr3)
```

Figure 136. *Example code for NumPy arrays and functions.*

Results

Replication of Google's Experiment

In order to validate Google's results and alleviate concerns over whether it is possible to achieve such high classification accuracy in unsupervised domain adaptation and even end to end unsupervised domain adaptation, we trained and tested Google's GAN twice with the MNIST and MNIST-M datatsets as the source and target domain, respectively. We also wanted to do MNIST to USPS domain adaptation with Google's GAN, but were unable to because Google did not provide the exact code and USPS classifier that they used.

The hyperparameters for the GAN were the same as mentioned in Google's research paper [1]. Training lasted for 500,000 iterations and checkpoints of the model were saved every few hundred iterations. After training, we tested the model and observed the results on Tensorboard, a set of visualization tools for TensorFlow. The classification accuracies for each run and the average classification accuracy can be seen in Figure 137.

Google GAN	Classification Accuracy
Run 1	94.9%
Run 2	98.8%
Average	96.9%

Figure 137. *Classification accuracies for the replication of Google's GAN in MNIST to MNIST-M domain adaptation.*

Based on these results, the average classification accuracy of 96.9% is fairly close to the 98.2% classification accuracy obtained by Google. It is fair to say then that a statistical averaging of the classification accuracies of a multitude of runs would results in a classification accuracy almost identical to that of Google's. This signified that it is possible to achieve results similar to Google and gave us hope that we could achieve a decent classification accuracy for end to end unsupervised domain adaptation.

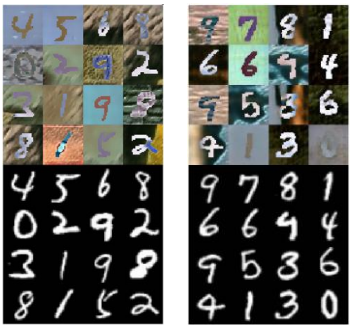


Figure 138. *MNIST and MNIST-M images generated during testing of Google's GAN.*

CoWGAN

The training of CoWGAN was done four different times, each with various learning rates and batch sizes, on the MNIST dataset. The first three runs had a batch size of 16 and runs 2 and 4 shared the same learning rate of 0.0001. The learning rates and batch sizes for each run are listed below:

Run 1: Learning rate = 0.00005, batch size = 16
Run 2: Learning rate = 0.0001, batch size = 16
Run 3: Learning rate = 0.0002, batch size = 16
Run 4: Learning rate = 0.0001, batch size = 32

Overall, each run did not produce great results. Most of the images were not discernable even in later training iterations. Run 2 produced the best results at around the 3,000th iteration, but images became unrecognizable as the model kept training. Because of these poor results, we abandoned the CoWGAN model to develop our CoIGAN model.

Figure 139. *Progression of image generation during CoWGAN training. Digits are discernable in the middle image; however, image quality reduces as training progresses.*

COIGAN

We trained our CoIGAN model to adapt from MNIST to MNIST, MNIST to MNIST-M, and MNIST to USPS for 30,000 iterations. Each experiment used an image batch size of 32, a learning rate of 0.0002, and beta value of 0.5.

Domain Adaptation	Source Domain Classification Accuracy	Target Domain Classification Accuracy
MNIST to USPS	59.3%	59.1%
MNIST to MNIST-M	70.4%	88.0%

Figure 140. *Source and target domain classification accuracies for two of the domain adaptation experiments on CoIGAN.*

MNIST to MNIST Image Generation

For the MNIST to MNIST domain adaptation experiment, the best visual results could be seen at 30,000 iterations. As can be seen, the image generation suffers; however, some numbers can be discerned. For this experiment, the classification accuracy is 52.87%, which is approximately the performance of most clustering algorithms.

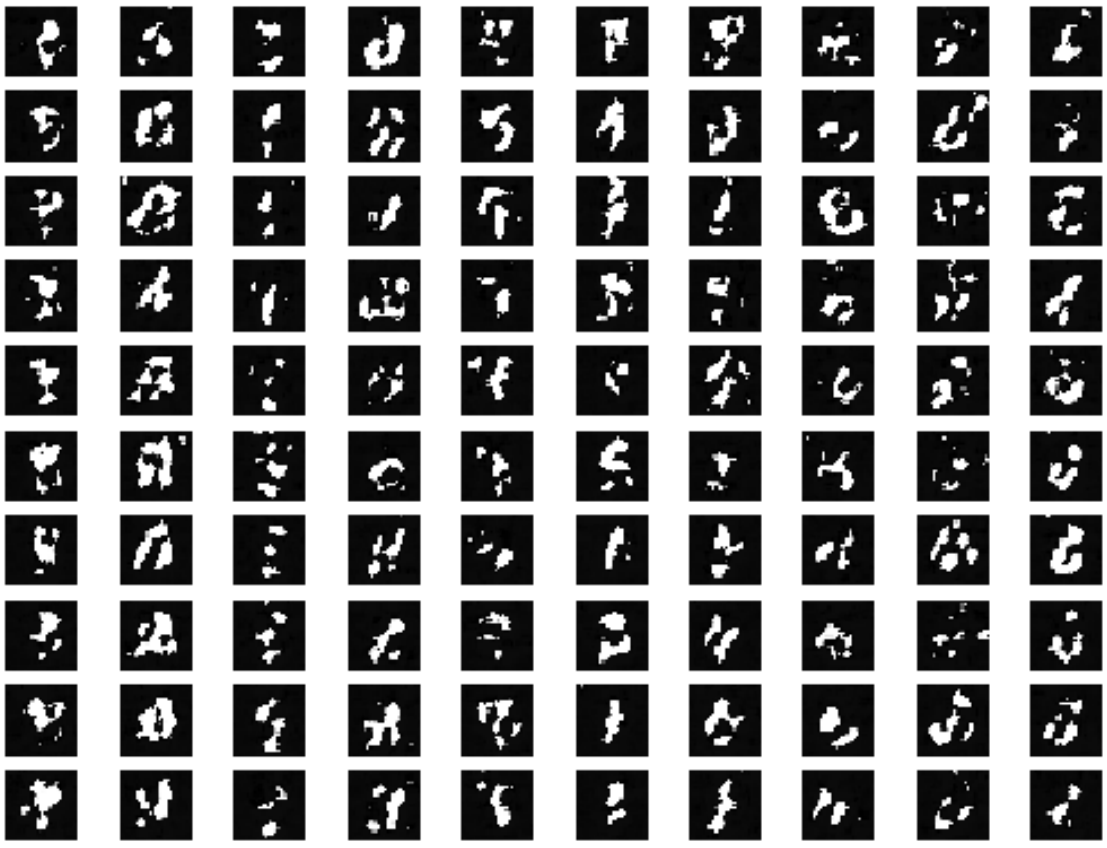


Figure 141. *MNIST generated images at early stages of CoIGAN training.*

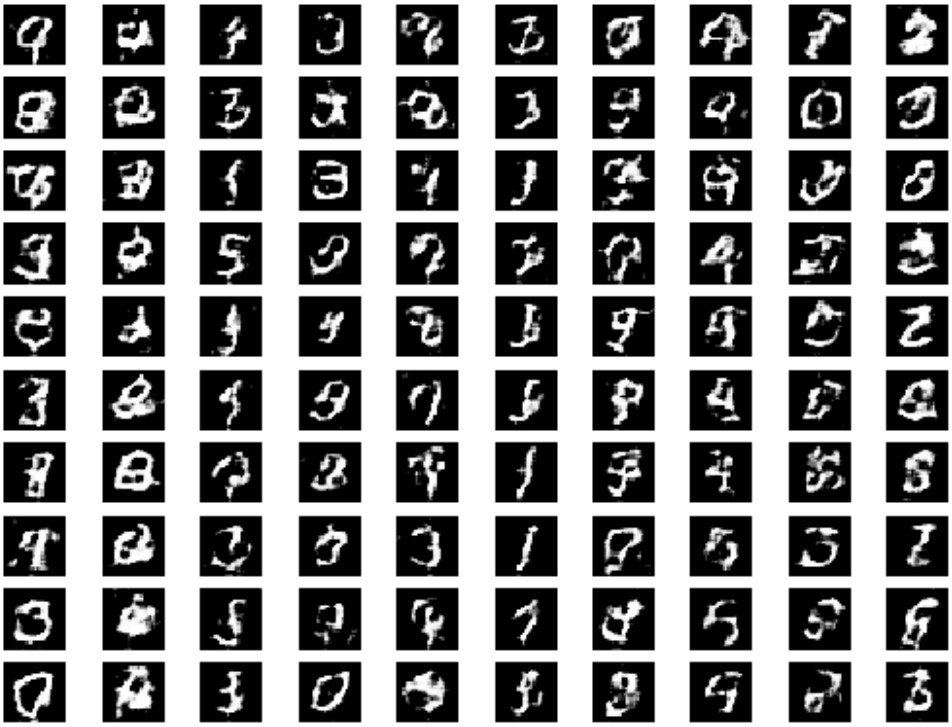


Figure 142. *MNIST generated images at middle stages of CoIGAN training.*

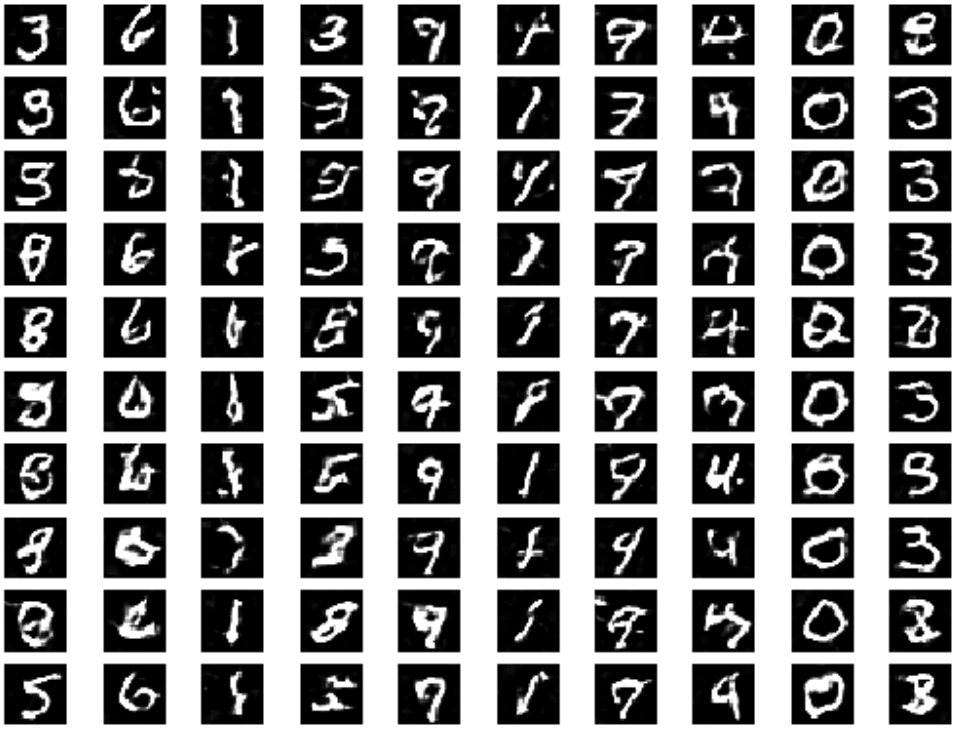


Figure 143. *MNIST generated images at optimal stages of CoIGAN training.*

MNIST to MNIST Graphs

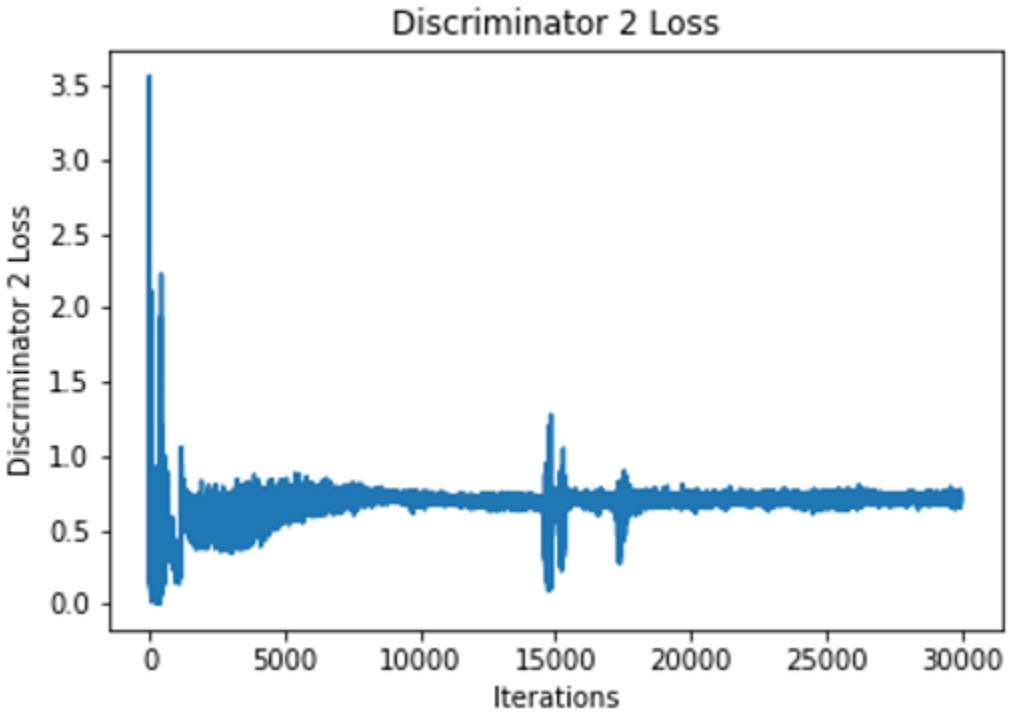


Figure 144. *Graph of the discriminator losses for the MNIST target domain.*

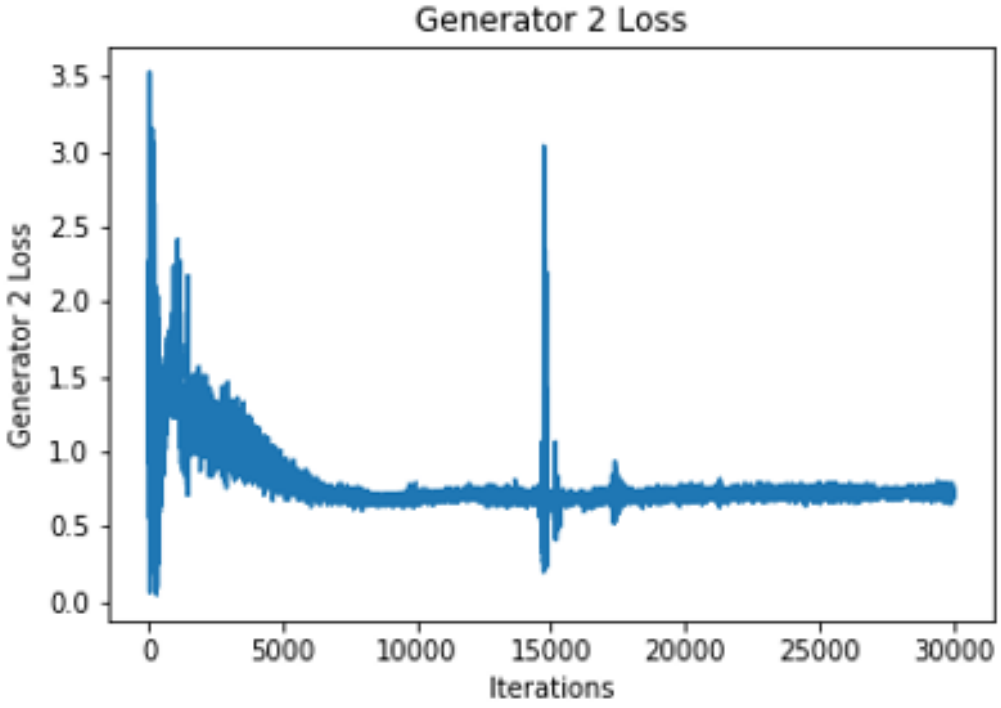


Figure 145. *Graph of the generator losses for the MNIST target domain.*

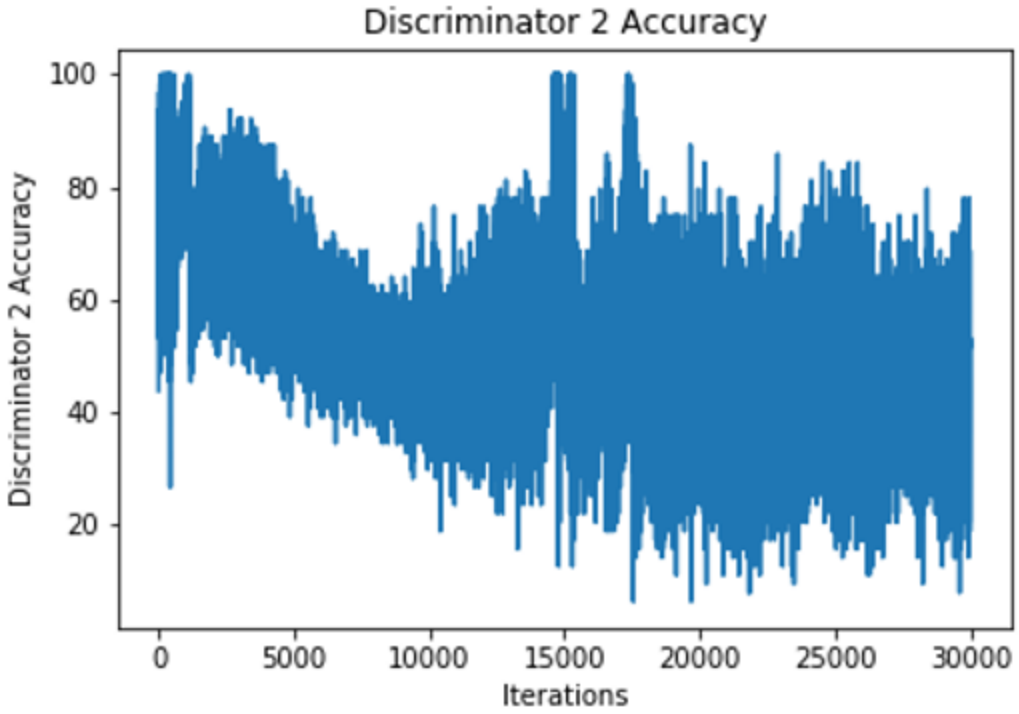


Figure 146. *Graph of the classification accuracies for the MNIST target domain.*

MNIST to USPS Image Generation

For the MNIST to USPS domain adaptation experiment, the best visual results could be seen at 24,000 iterations. Compared to the previous experiment (MNIST to MNIST), this particular experiment is not as successful. However, the classification accuracy for this experiment is 59.07%, which is higher. One thing to note is that the image generation and the classification accuracy are mutually exclusive.

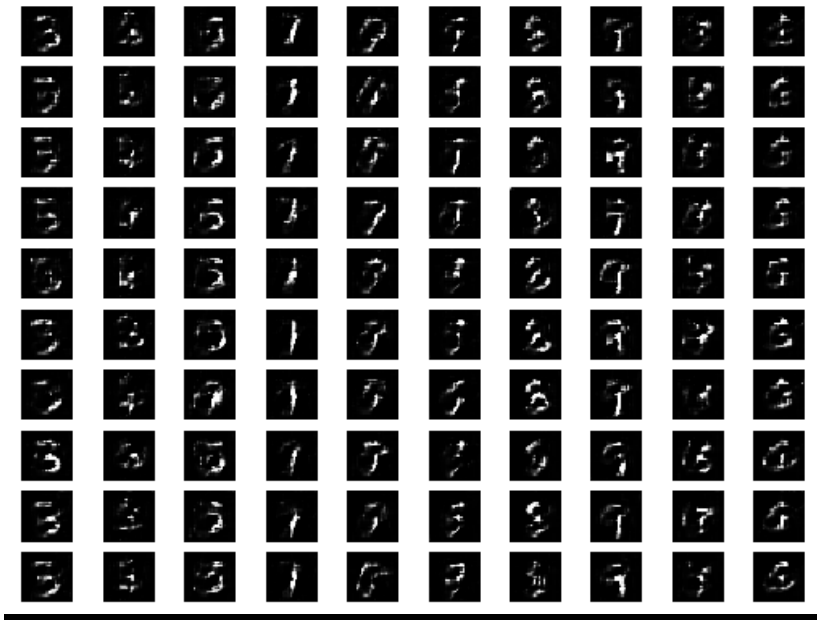


Figure 147. *USPS generated images at early stages of CoIGAN training.*

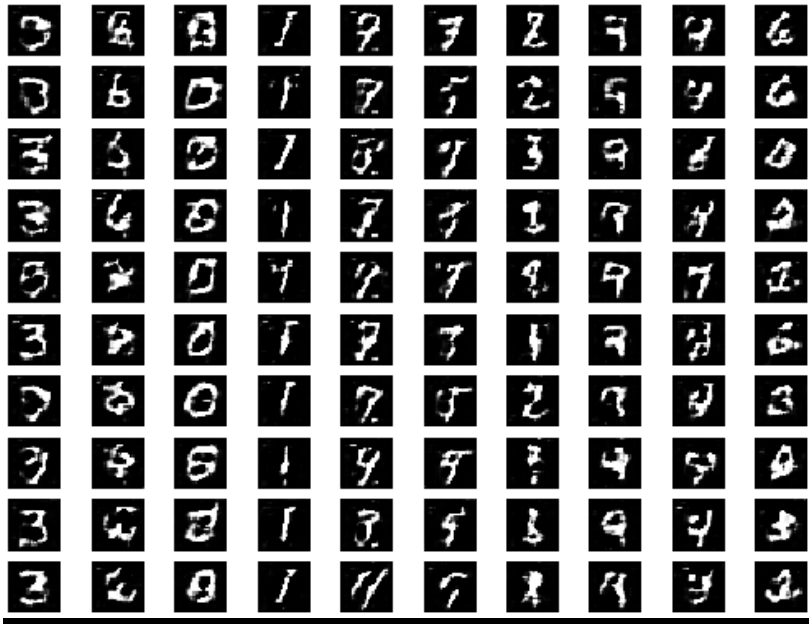


Figure 148. *USPS generated images at middle stages of CoIGAN training.*

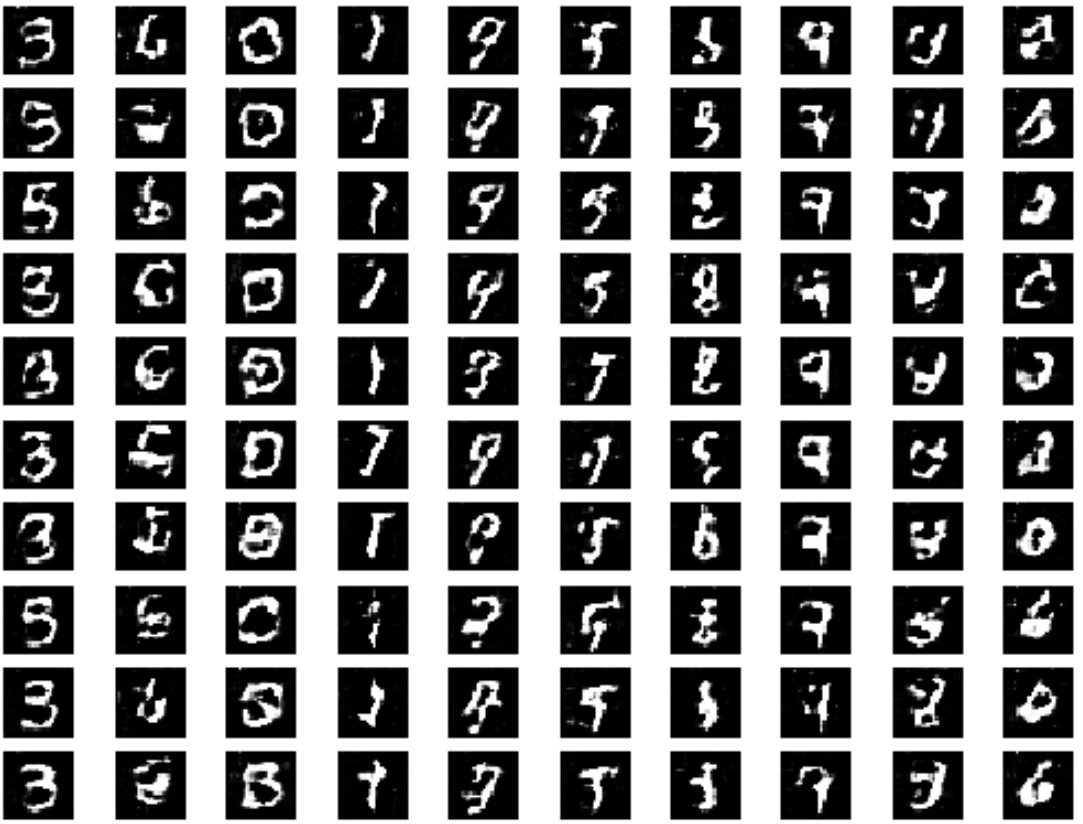


Figure 149. *USPS generated images at optimal stages of CoIGAN training.*

MNIST to USPS Graphs

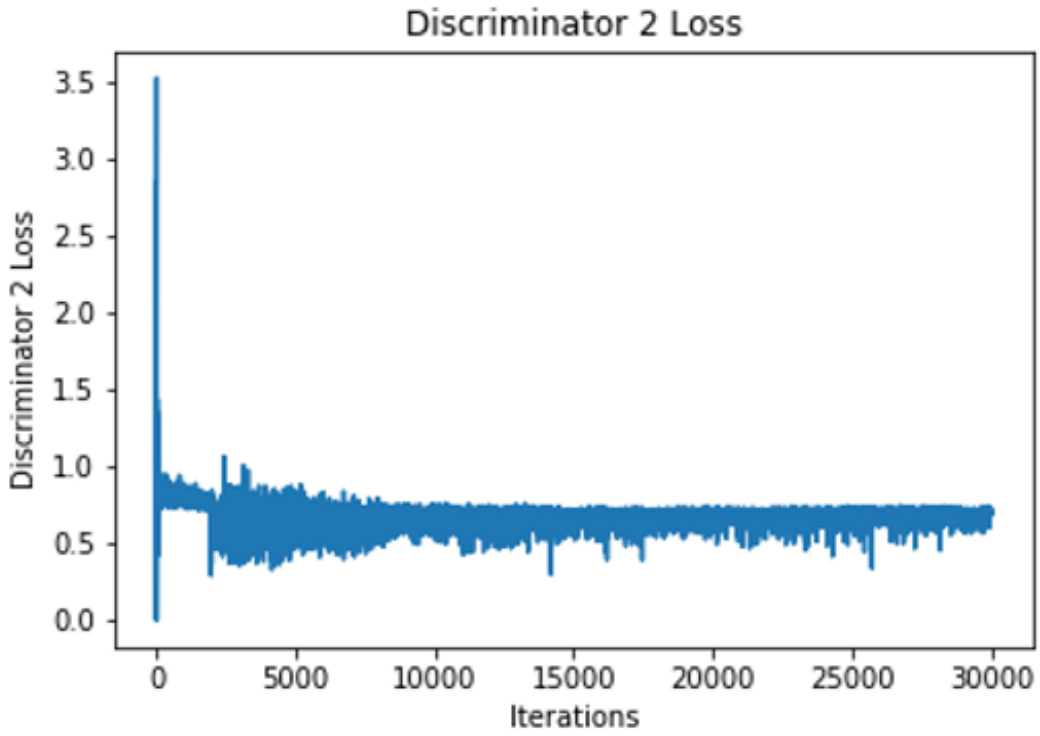


Figure 150. *Graph of the discriminator losses for the USPS target domain.*

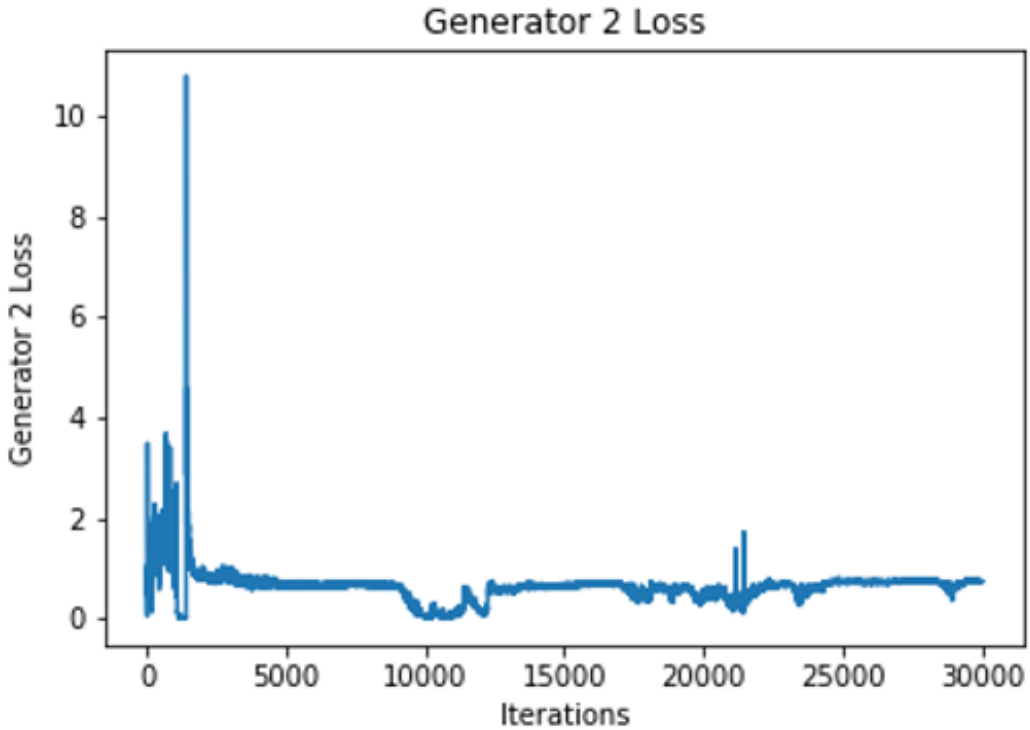


Figure 151. *Graph of the generator losses for the USPS target domain.*

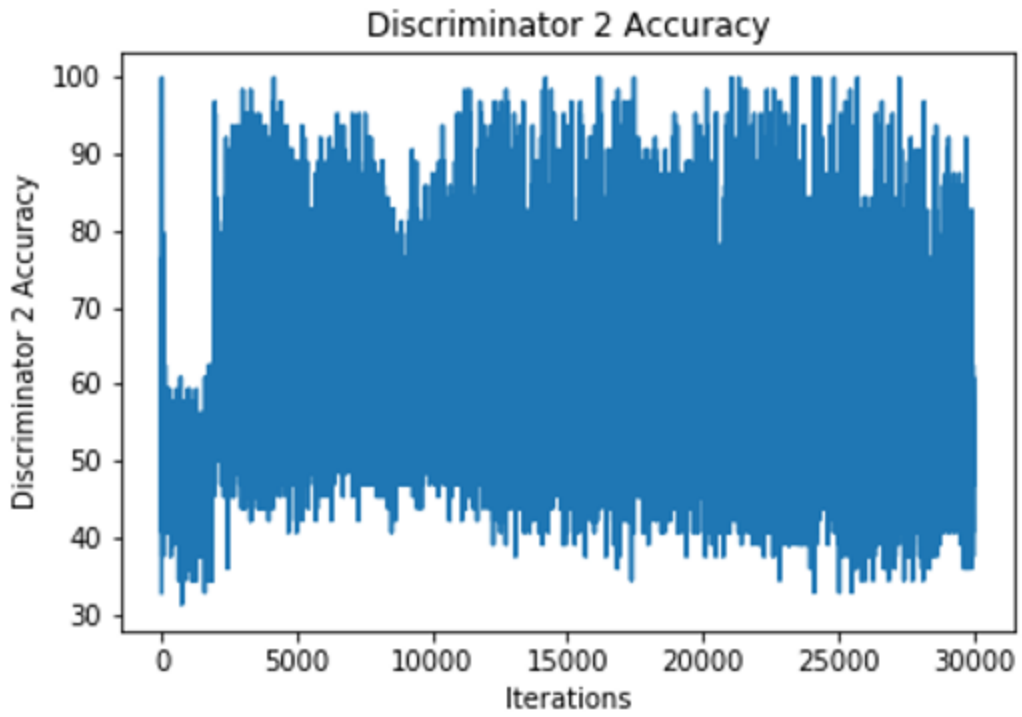


Figure 152. *Graph of the classification accuracies for the USPS target domain.*

MNIST to MNIST-M Image Generation

For the MNIST to MNIST-M domain adaptation experiment, the best visual results can be seen at 25,000 iterations. Again, some numbers can be seen, but the image generation could be better improved. This experiment yielded the worst visual results compared to the other datasets.

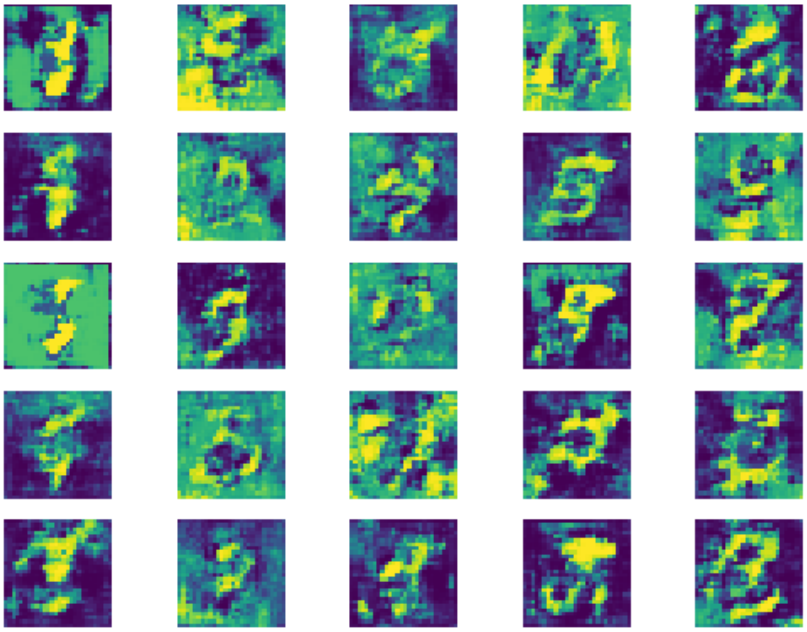


Figure 153. *MNIST-M generated images at early stages of CoIGAN training.*

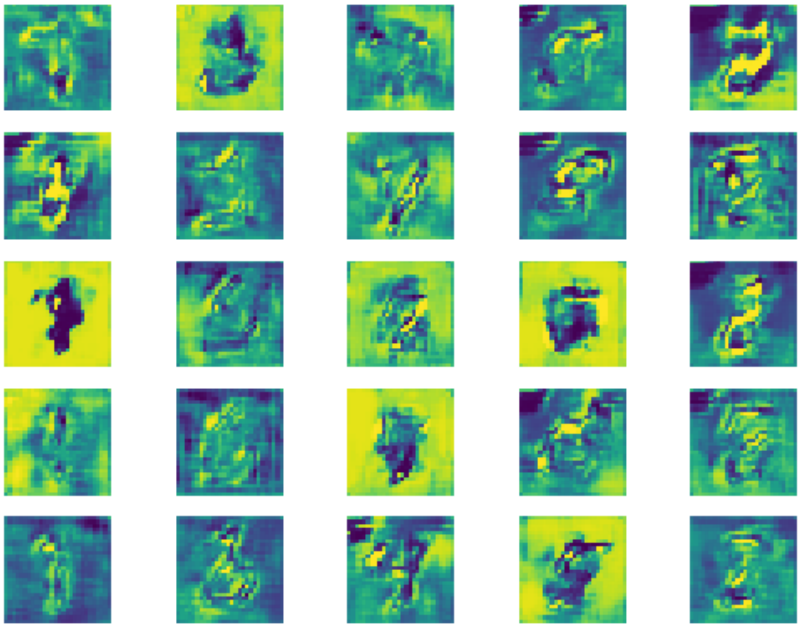


Figure 154. *MNIST-M generated images at middle stages of CoIGAN training.*

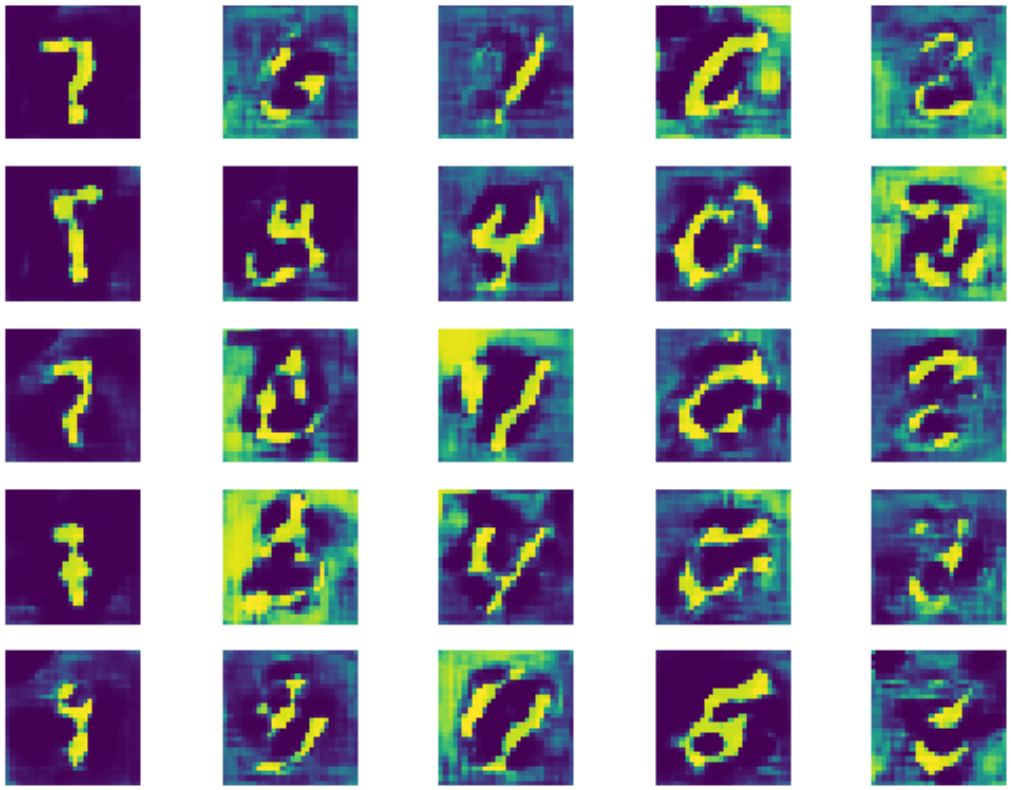


Figure 155. *MNIST-M generated images at optimal stages of CoIGAN training.*

MNIST to MNIST-M Graphs

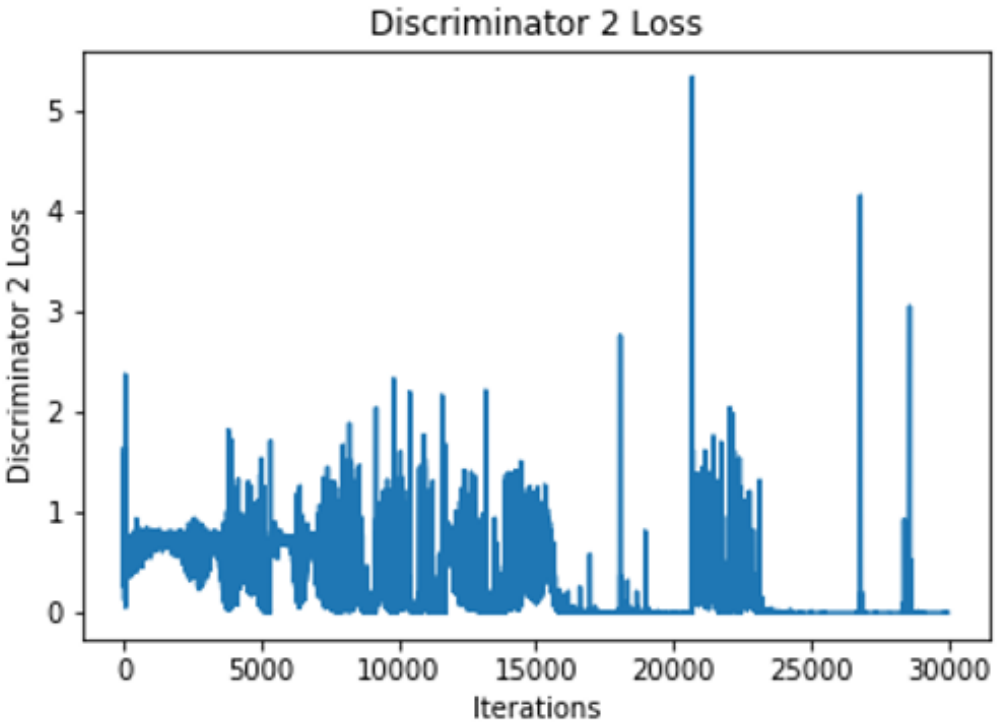


Figure 156. *Graph of the discriminator losses for the MNIST-M target domain.*

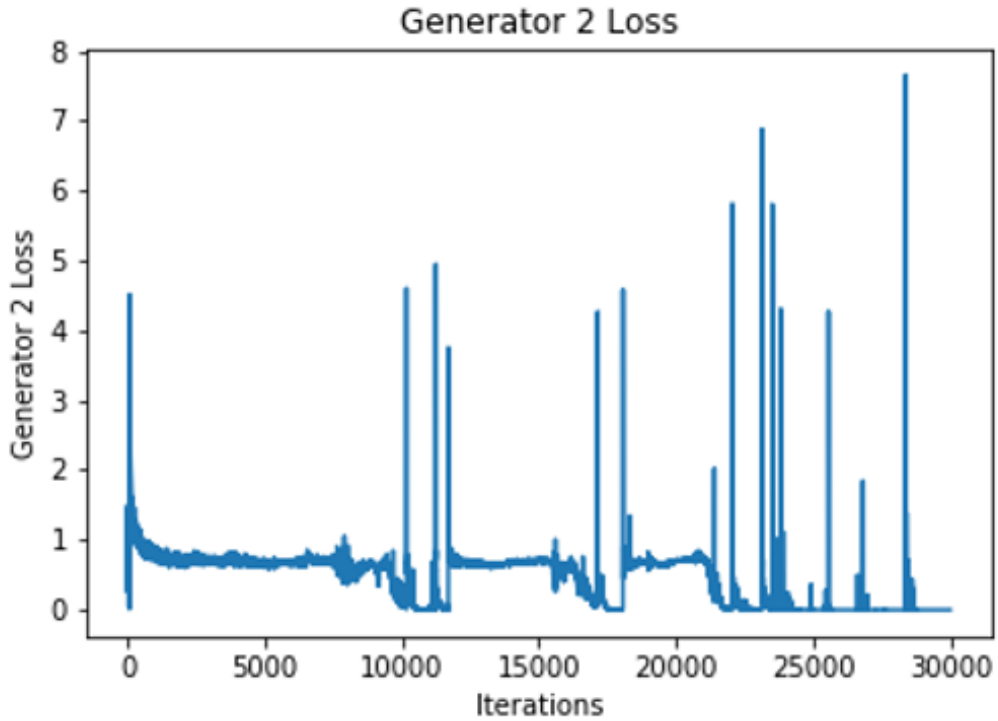


Figure 157. *Graph of the generator losses for the MNIST-M target domain.*

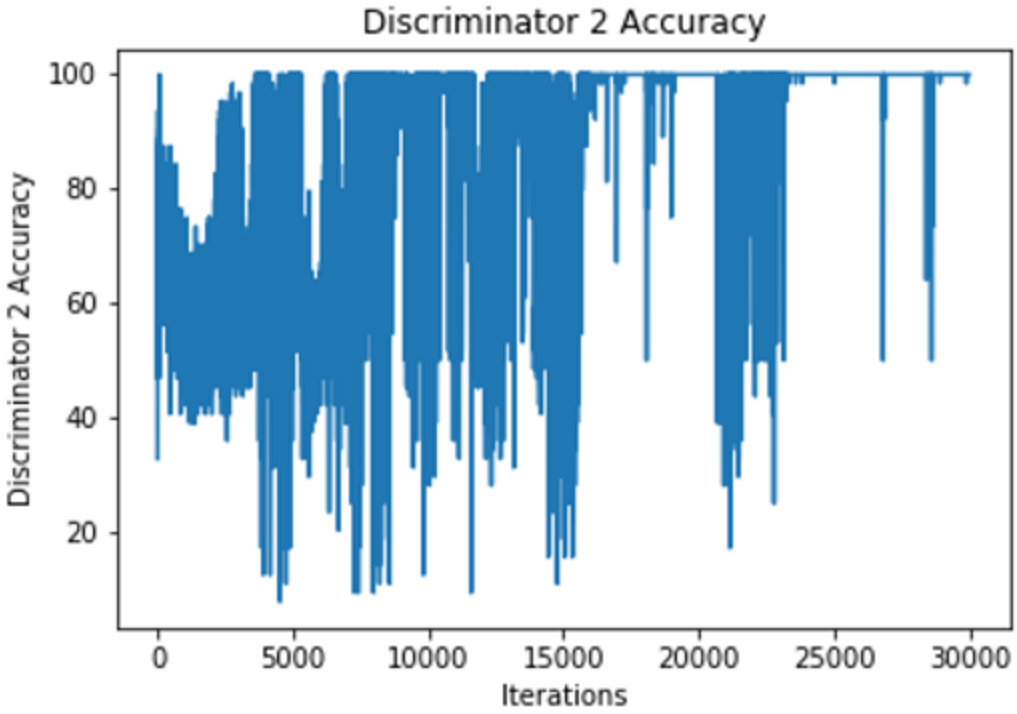


Figure 158. *Graph of the classification accuracies for the MNIST-M target domain.*

CoPixGAN

Since the MNIST to MNIST-M domain adaptation experiment using CoIGAN did yielded barely visible numbers, we decided to create a new architecture. This new architecture was formed by simply coupling two of Google's PixelDA GANs, which is not a full unsupervised end-to-end model, but it does incorporate unsupervised learning. With this architecture, we were able to yield very high-quality images and successfully adapt colored images to a target domain. The best visual results can be seen at 1164 iterations. With this approach, the number of iterations to train a domain adaptation model is decreased.

Figure 159. *Various generated images of the source domain MNIST and target domain MNIST-M during CoPixGAN training.*

Administrative Content

Budget and Financing

Since Google's domain adaptation model and all of our models had to be trained with substantially large datasets, it was required that each member of the team had to have access to a computer with a GPU that had considerable processing power. Additionally, the GPU had to have time to run the training algorithms without being interrupted unless saving the state of the models was available. TensorFlow, one of the Python software libraries we used to run our deep learning algorithms, recommends having an NVIDIA GPU that has CUDA Compute Capability 3.0 or higher. In the case that any of this was not available to a team member at any point, we determined that we should have a virtual server provided by UCF or pay for Amazon Web Services' elastic GPUs.

US East (Ohio) and US East (N. Virginia)

Elastic GPU Size	GPU Memory	Price
eg1.medium	1 GiB	$0.050/hour
eg1.large	2 GiB	$0.100/hour
eg1.xlarge	4 GiB	$0.200/hour
eg1.2xlarge	8 GiB	$0.400/hour

Figure 160. *AWS GPU prices based on GPU size for the Eastern United States.*

Fortunately, we were given access to a UCF server with multiple CPUs and Alienware computers containing NVIDIA GeForce GTX 1080 Ti graphics cards. We therefore were able to run our models relatively quickly, avoid paying for AWS GPUs, and keep the cost of our project at $0.

Gantt Chart

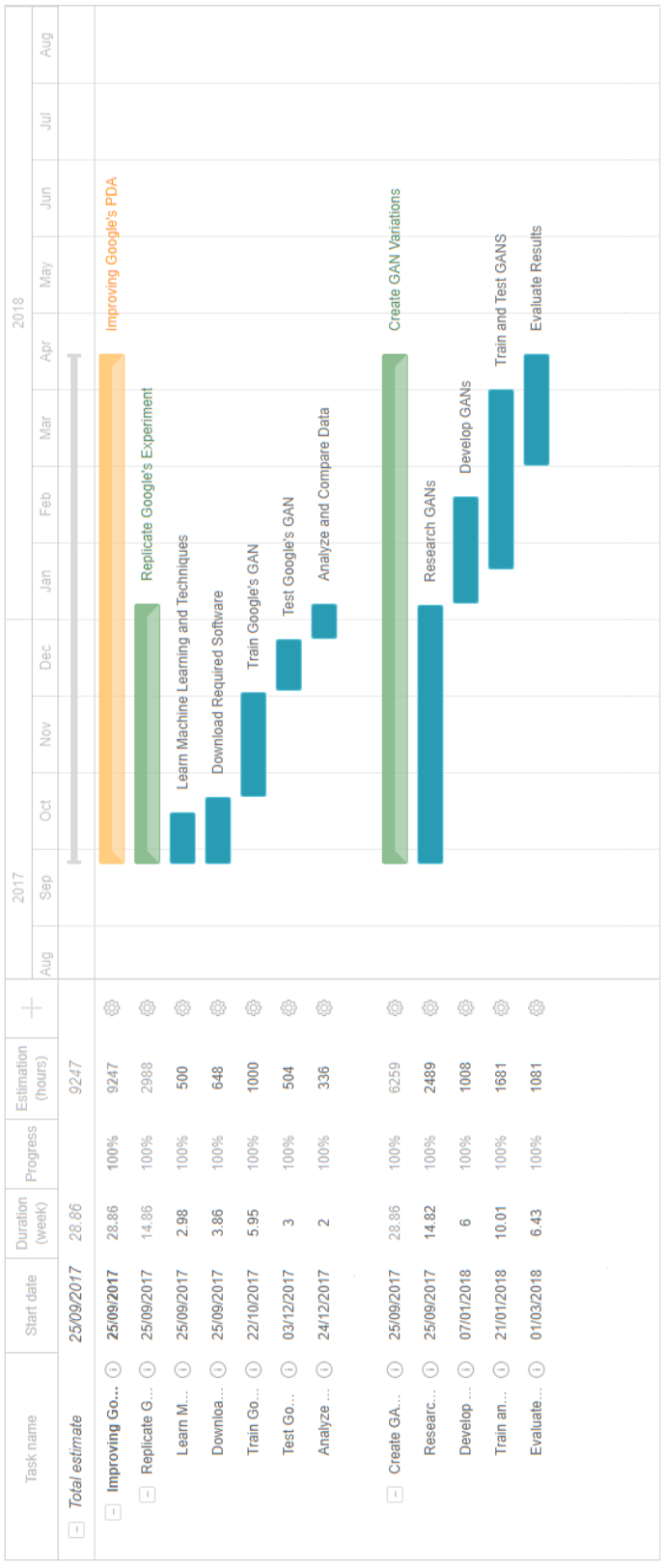


Figure 161. *Gantt chart.*

Pert Chart

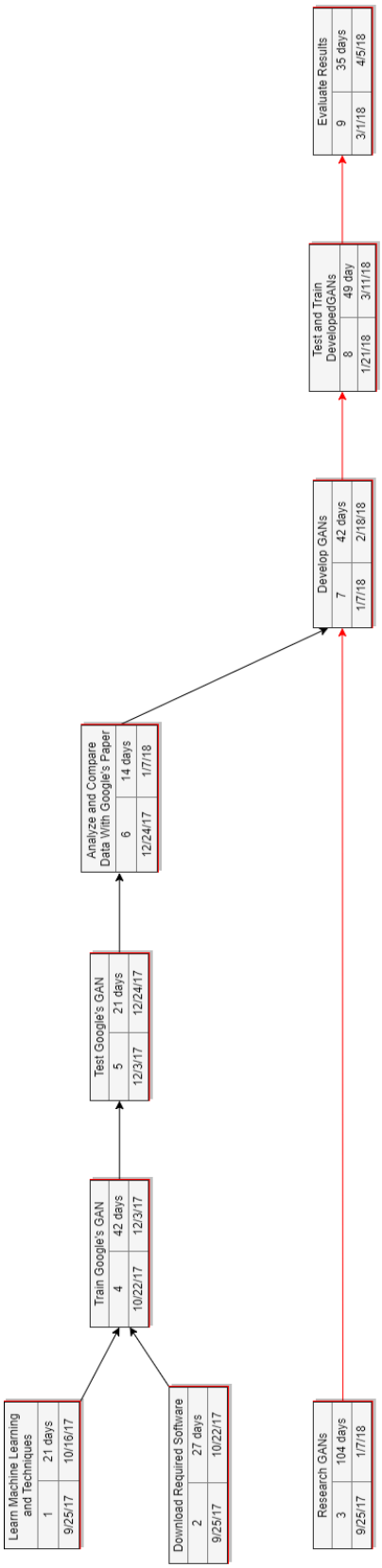


Figure 162. *Pert chart.*

Conclusion

Our proposed domain adaptation end-to-end unsupervised models, CoIGAN and CoWGAN, were able to adapt from source to target domain. As mentioned, our models used partial weight sharing, as opposed to Google's research model, which did not use any type of coupling or weight sharing. Additionally, Google calls their method unsupervised learning, even though they use labels in the source domain. Our model does not use labels in any of the domains, hence why it can be speculated that our image generation is not of high quality and the classification is not as good as Google's model. However, in terms of an end-to-end unsupervised domain adaptation model, this model has achieved remarkably astounding results. The results and observations made show that our weight sharing models, specifically CoIGAN, may have some promising results that should be further explored. In future work, we may have to restrict the number of layers that are shared to minimize and eradicate confusion in the model. Other improvements can also be made by tweaking hyperparameters within our model.

Since our main models CoIGAN and CoWGAN had some difficulties in image generation, we decided to try a new model called CoPixGAN as a bonus exploratory method, where we coupled two of Google's PixelDA GANs in an unsupervised model that fully shared weights. With this new method, we were able to see that high quality images and classification match that of Google's method.

All in all, this research has shown us how GANs can significantly impact image generation, classification, and domain adaptation as a whole.

References

1. D. Dohan, et al. “Unsupervised Pixel–Level Domain Adaptation with Generative Adversarial Networks*.” arXiv preprint arXiv:1612.05424v2*. 2017.

2. M. Bejiga, A. Zeggada, A. Nouffidj, and F. Melgani. “A Convolutional Neural Network Approach for Assisting Avalanche Search and Rescue Operations with UAV Imagery.” *Remote Sensing*. 2017.

3. K. Kamnitsas, C. Baumgartner, C. Ledig, V. Newcombe, J. Simpson, A. Kane, D. Menon, A. Nori, A. Criminisi, D. Rueckert, and B. Glocker. “Unsupervised Domain Adaptation in Brain Lesion Segmentation with Adversarial Networks.” *arXiv preprint arXiv: 1612.08894*. 2016.

4. I. Goodfellow, et al. “Deep Learning.” *MIT Press*. 2016. http://deeplearningbook.org

5. “Generative Models.” *OpenAI*. 30 November 2017. https://blog.openai.com/generative-models/

6. I. Goodfellow, et al. "Generative Adversarial Nets." *Advances in Neural Information Processing Systems*. 2014.

7. I. Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks." *arXiv preprint arXiv:1701.00160*. 2016.

8. A. Radford, L. Metz, and S. Chintala. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.” *arXiv preprint arXiv:1511.06434*. 2016.

9. M. Mirza and S. Osindero. "Conditional Generative Adversarial Nets." *arXiv preprint arXiv:1411.1784*. 2014.

10. X. Chen, et al. “InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets.” *Advances in Neural Information Processing Systems*. 2016.

11. M. Chen, et al. “Marginalized Denoising Autoencoders for Domain Adaptation.” *arXiv preprint arXiv:1206.4683*. 2012.

12. S. Clinchant, G. Csurka, and B. Chidlovskii. “A Domain Adaptation Regularization for Denoising Autoencoders.” *The 54th Annual Meeting of the Association for Computational Linguistics*. 2016.

13. P. Vincent, et al. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion." *Journal of Machine Learning Research*, 11, pages 3371-3408. 2010.

14. E. Tzeng, et al. "Adversarial Discriminative Domain Adaptation (Workshop Extended Abstract)." *Workshop Track – ICLR 2017*. 2017.

15. T. Unterthiner, et al. "Coulomb GANs: Provably Optimal Nash Equilibria via Potential Fields." *arXiv preprint arXiv:1708.08819*. 2017.

16. X. Huang, et al. "Stacked Generative Adversarial Networks." *arXiv preprint arXiv:1612.04357*. 2016.

17. H. Venkateswara, et al. "Deep Hashing Network for Unsupervised Domain Adaptation." *arXiv preprint arXiv:1706.07522*. 2017.

18. J. Zhao, M. Mathieu, and Y. LeCun. "Energy-based Generative Adversarial Network." *arXiv preprint arXiv:1609.03126*. 2016.

19. P. Russo, et al. "From Source to Target and Back: Symmetric Bi-directional Adaptive GAN." *arXiv preprint arXiv:1705.08824*. 2017.

20. K. Saito, Y. Ushiku, and T. Harada. "Asymmetric Tri-training for Unsupervised Domain Adaptation." *arXiv preprint arXiv:1702.08400*. 2017.

21. S. Sankaranarayanan, et al. "Generate to Adapt: Aligning Domains using Generative Adversarial Networks." *arXiv preprint arXiv:1704.01705*. 2017.

22. Y. Ganin and V. Lempitsky. "Unsupervised Domain Adaptation by Backpropagation." *arXiv preprint arXiv:1409.7495v2*. 2015.

23. M. Liu and O. Tuzel. "Coupled Generative Adversarial Networks." *arXiv preprint arXiv:1606.07536v2*. 2016.

24. O. Sener, et al. "Unsupervised Transductive Domain Adaptation." *arXiv preprint arXiv:1602.03534v3*. 2016.

25. M. Arjovsky, S. Chintala, and L. Bottou. “Wasserstein GAN.” *arXiv preprint arXiv:1701.07875*. 2017.

26. A. Irpan. “Read-through Wasserstein Gan.” *Sorta Insightful*. 26 Ferbruary 2017. https://www.alexirpan.com/2017/02/22/wasserstein-gan.html

27. P. Joshi. “What is Manifold Learning?” *Perpetual Enigma*. 21 June 2014. https://prateekvjoshi.com/2014/06/21/what-is-manifold-learning/

28. G. Qi. “Loss-Sensitive Generative Adversarial Networks on Lipschitz Densities.” *arXiv preprint arXiv:1701.06264*. 2017.

29. D. Pfau and O. Vinyals. “Connecting Generative Adversarial Networks and Actor-Critic Methods.” *arXiv preprint arXiv:1610.01945*. 2017.

30. T. Salimans, I. Goodfellow, W. Zarmeba, V. Cheung, A. Radford, and X. Chen. “Improved Techniques for Training GANs.” *arXiv preprint arXiv:1606.03498*. 2016.

31. F. Shaikh. “Introductory Guide to Generative Adversarial Networks (GANs) and Their Promise!” *Analytics Vidhya*. 15 June 2017. https://www.analyticsvidhya.com/blog/2017/06/introductory-generative-adversarial-networks-gans/

32. F. Kratzert. “Understanding the Backward Pass Through Batch Normalization Layer.” *Flaire of Machine Learning*. 12 February 2016. https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html

33. A. Kumar, P. Sattigeri, and P. T. Fletcher. “Improved Semi-supervised Learning with GANs using Manifold Invariances.” *arXiv preprint arXiv:1705.08850*. 2017.

34. J. Donahue, P. Krahenduhl, and T. Darrell. “Adversarial Feature Learning.” *arXiv preprint arXiv:1605.09782*. 2017.

35. D. Berthlost, T. Schumm, and L. Metz. “BEGAN: Boundary Equilibrium Generative Adversarial Networks.” *arXiv preprint arXiv:1703.10701*. 2017.

36. Z. Junbao, S. Wang, W. Zhang, and Q. Haung. “Deep Unsupervised Convolutional Domain Adaptation.” *ISBN: 978-1-4503-4906-2*. 2017.

37. K. Bousmalis. “Code for Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks.” https://github.com/tensorflow/models/tree/master/research/domain_adaptation

38. Y. LeCun, C. Cortes, and C. Burges. “The MNIST Database of Handwritten Digits.” http://yann.lecun.com/exdb/mnist/

39. “MNIST for ML Beginners.” *TensorFlow*. 2 November 2017. https://www.tensorflow.org/get_started/mnist/beginners

40. P. Arbelaez. “Contour Detection and Image Segmentation Resources.” *Berkeley University of California Computer Vision Group*. January 2013. https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html#bsds500

41. C. Mellina. “Domain-Adversarial Training of Neural Networks in Tensorflow.” https://github.com/pumpikano/tf-dann

42. N. Inoue. “MNIST-M Loader for Chainer.” https://github.com/naoto0804/chainer-mnist-m

43. H. Riemenschneider. “Yet Another Computer Vision Index to Datasets (YACVID) – Details.” 12 March 2013. http://riemenschneider.hayko.at/vision/dataset/task.php?did=96

44. “Info.” https://web.stanford.edu/~hastie/ElemStatLearn/datasets/zip.info.txt

45. S. Roweis. “Data for MATLAB Hackers.” https://cs.nyu.edu/~roweis/data.html

46. S. Hinterstoisser. “Chair for Computer Aided Medical Procedures & Augmented Reality.” *Technical University of Munich*. http://campar.in.tum.de/Main/StefanHinterstoisser