GENERATIVE ADVERSARIAL NETWORKS & FEATURE MATCHING

Author **Dan Andrei Iliescu**
Supervisor **Prof Gavin Brown**
BSc (Hons) Computer Science
The University of Manchester

May 2018

Abstract

This project is an investigation into a state of the art AI technique for synthetic image generation, called "Generative Adversarial Networks". It is an algorithm that learns to produce realistic images via an adversarial game between two competing learning agents, called a Generator and a Discriminator. I have implemented versions of this algorithm in several architectures and three main flavours and compared their performance to other techniques and amongst themselves. I subsequently focused my investigation on a particular problem with this algorithm, namely its difficulty with generating images with coherent and recognizable objects. I surveyed two approaches to solving this problem, broadly called "feature matching", and developed a solution of my own inspired by these, which improves the quality of produced images in certain cases. In the end, the GAN exhibits impressive results, but also an unpredictable and erratic behaviour, and improvements to the canonical form have so far been limited. I hope that this work will contribute to our understanding of this algorithm.

1 Introduction

Generative Adversarial Networks [4] are an unsupervised learning technique for generating synthetic data. The algorithm receives a set of images belonging to a particular manifold and then learns the essential features of those images so that it can produce new pictures that could plausibly belong to that set. It achieves an understanding of a complex manifold by iteratively refining its hypothesis through the dialectic between the algorithms two constituent learning components: the Generator, which is tasked with producing images, and the Discriminator, which is tasked with differentiating between real images and generated ones.

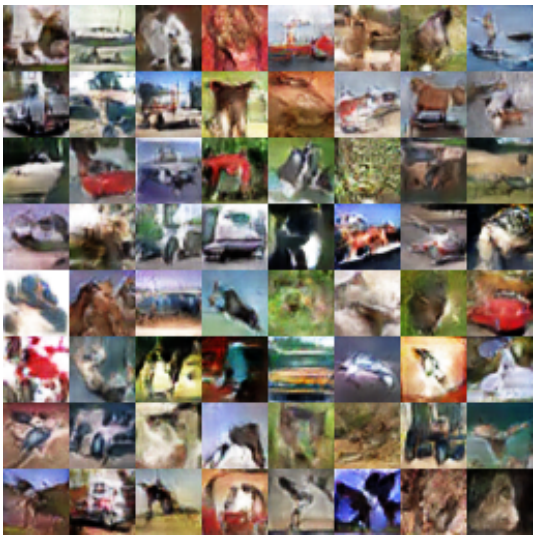

Figure 1: Examples of generated images from my project. DCGAN trained on CIFAR-10.

The GANs strength lies in the high quality of the results it produces, as well as in its conceptual simplicity and flexibility of use. Indeed, in a short timespan of under 4 years since the formulation of this technique, it has been applied to many very diverse areas of utility. Short-term applications include image-to-image translation, data augmentation, semi-supervised learning, data compression, planning for reinforcement learning and sound and image synthesis [9]. The long-term promise of this technique, however, is to revolutionise the way we

conduct research. The whole of scientific discovery and innovation is, essentially, a process of quantifying patterns observed in the natural world for the purpose of constructing artefacts and systems that improve our lives. Such an unsupervised generative model that learns hidden features in data can greatly augment our capacity to identify those patterns and put them to use. We may imagine a future where computers use these algorithms autonomously to derive mathematical models of physical phenomena or to identify genes in DNA sequences.

Currently, however, the study of GANs is still in its early stages, and the technique requires a substantial amount of improvement before it becomes consistent and reliable. Common problems include great difficulties with training and convergence, the lack of a reliable performance metric and problems with capturing complex features of high-variance datasets [2]. One very noticeable problem with even the most performant of architectures is their apparent inability to construct morphologically coherent objects in the images they produce (as we can see in 1, although the images look visually compelling, it is still difficult to identify the objects inside of them). Approaches under the umbrella term feature matching aim to enhance the Generators learning by using features extracted by the Discriminator. In this work, I have observed these problems, surveyed approaches to mitigate some of them and recognized that reliable solutions for many problems are yet to be found.

My contributions in this work are:

1. I have experimented with implementing different kinds of deep learning architectures for the Generator and Discriminator of a GAN and observed the behaviour of the network and the results. On the basis of the quality of results and of the available computational budget, I settled on a network architecture similar to the DCGAN [12].

2. I have built three versions of GANs (DCGAN, WGAN and BEGAN) and a VAE using the same base architecture described above in order to compare the different algorithms fairly. I have also implemented two quantitative evaluation metrics, the Inception Score and the Frechet Inception Distance, in order to perform an automatic evaluation of the images generated by each algorithm. I observed that the GANs obtained better scores than the VAE, with the DCGAN and WGAN achieving the best scores, and that this correlated well with human judgment.

3. I have built a fully convolutional network to classify the generated images of a DCGAN, and then provided visualizations of the labelled activations in the intermediate layers of the GAN using dimensionality reduction techniques, in order to see how well the different components of the algorithm understand the features of the images.

4. I have built two versions of DCGAN which enhance the Generators learning using feature matching (L2FM and DFM), and evaluated their performance visually, automatically using the scores and by using the visualisation tools described above. The improvements to the DCGAN are marginal.

5. I have designed and built my own feature matching algorithm, inspired by the solutions implemented before, but, again, improvements were only marginal.

2 Background and Related Work

Firstly, when talking about generative models, we must distinguish between the practical implementation of the learning mechanisms and the probabilistic interpretation of the associated learning process. While the actual mechanics may seem more intuitive, it is useful to first perceive the problem through the lens of its probabilistic underpinnings, as they give us a more general framework for judging different approaches.

Mathematically, generative models are functions of the form $P_r(X)$ that map data samples X (in our case images) to probability values, in such a way that they assign probabilities proportional to how plausible the samples are. We can imagine images as being points in a multidimensional space, where each coordinate of the point represents a feature of the image (pixel values, usually).

We imagine the data as belonging to a large manifold of points, where the generative model is the probability density function of the manifolds underlying probability distribution, P_r. It is called generative because we implicitly assume that the real data has been generated naturally by sampling from the distribution P_r (which in our case represents the distribution of real images), and if only we found the mathematical expression

for that density function, we would be able to generate new instances of data that are indistinguishable from the real ones.

Because it is often impossible to know with certainty through what process the real data was produced, we compute a parametric probability density function, $P_\theta(X)$, that approximates the real data distribution, Pr. In other words, our goal is to find the parameters of the probability distribution P that most closely resembles the distribution of the real data.

The process of finding the optimal parameters for the distribution P can be described through what is called Maximum Likelihood Estimation, whereby we compute the parameters which maximize the product of the likelihoods of the density $P_\theta(X)$ given each of the learning datapoints we have. We can take the natural logarithm of the total likelihood, since logarithms are monotonous so that our product becomes a sum. This expression is the general measure by which we interpret what different algorithms learn.

Note: The way we normally compute the optimal parameters that maximize a function is through gradient-based methods. This means that we iteratively take the gradient of the function with respect to the parameter, and then follow the gradient with a step size, called the learning rate.

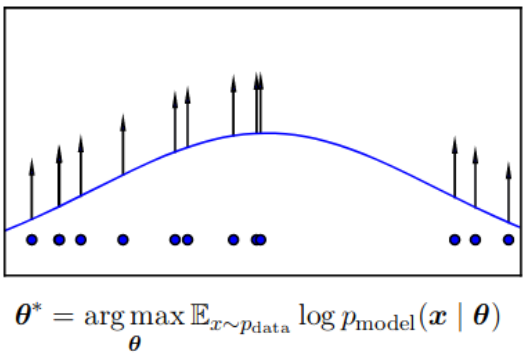

Figure 2: Maximum Log-Likelihood Estimation. To find θ, we push the density up for each data point [5].

Some algorithms perform this estimation explicitly, constructing the density function $P_\theta(X)$ based on learning data and directly maximizing its likelihood by following its gradient uphill. Autoregressive algorithms, like Fully Visible Belief Nets, use the chain rule of probability to decompose the density over the N components (pixels) of datapoint X:

$$P_\theta = \prod_{i=1}^{N} P_\theta(x_i|x_1, ...x_{i-1}) \quad (1)$$

While FVBNs yield good quality results, computation is very slow and sequential, each pixel depending on all pixels that have been computed before. Variational Autoencoders circumvent these computational requirements by approximating an intractable density $P_\theta(X)$ using variational methods. They maximize a tractable lower bound for the density $L_\theta(x) \leq \log P_\theta(x)$. The downside of the VAE is that, because of maximizing a lower bound rather than the likelihood itself, the results appear blurry since they never converge to the true distribution.

Other algorithms, instead of computing an explicit density function, interact only indirectly with the density $P_\theta(X)$ by sampling from it. The GAN is one of them, and works by sampling from a known latent distribution, P_z, (e.g. a normal distribution) and then learning a differentiable mapping, G, from those samples to points on the data distribution, such that $X \sim P_\theta(X) = G(z \sim P_z)$. GANs are considered to produce the best results.

2.1 Generative Adversarial Networks

Now that we have established a mathematical framework with which to understand the class of algorithms to which the GAN belongs, we can delve into the learning mechanisms employed in order to find the function G(z). GANs consist of two learning components:

1. The Generator is a differentiable function $G(z)$ which maps a latent variable to an image. It aims to produce an image belonging to the distribution of real images.

2. The Discriminator is a differentiable function $D(x)$ which maps an image to a confidence value between 1 and 0. It aims to associate a high confidence to real images and a low confidence to images produced by the Generator.

In order to train the model, we let D and G play a minimax game about a loss function. First we initialize D and G randomly. Then, iteratively, we sample a batch $x \sim P_r$ of real data and a batch $z \sim P_z$ of latent variables. We pass the latent variables through the G in order to produce a batch of images, which we then feed to D to obtain a "fake score", $D(G(z))$. Separately, we feed the real data batch to D and get a "real score", $D(x)$.

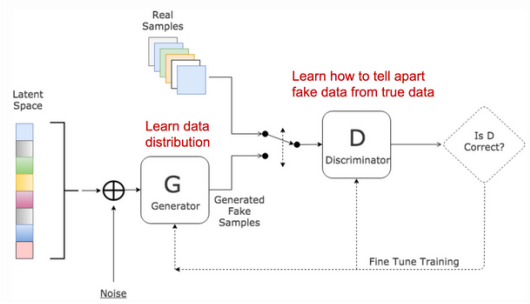

Figure 3: The basic structure of a GAN [15].

We then compute a loss function based on the real and fake scores that aims to quantify how well D can differentiate the data. The final step is to use gradient-descent-like algorithm (usually Adam) in order to update D and G following the gradient of the score with respect to each function, so that D will increase the score and G will decrease it.

$$\min_G \max_D E\left[\log D(x)\right] + E\left[1 - \log D(G(z))\right] \quad (2)$$

The loss is simply the cross-entropy between Ds predictions and the correct labels for the classification task. The left and right terms measure how well D can identify real data and generated data, respectively. This back and forth game between the two agents forces them to improve, and ideally ends when they reach a Nash equilibrium (that is, the loss function reaches a saddle point).

A key observation is what happens when D is ideal. A perfect discriminator would output 1 for real data and 0 for fake data:

$$D'(x) = \frac{P_r(x)}{P_r(x) + P_\theta(x)} \quad (3)$$

If we plug this into the loss function, the loss becomes the Jensen-Shannon Divergence, a well-known distance metric between distributions.

Note on divergences: There are many metrics that measure the distance between distributions. The most well known of these is the Kullback-Leibler divergence. When we perform Maximum Likelihood Estimation, we are actually trying to minimize the KL divergence between the real distribution and the generated one:

$$KL(P_r|P_\theta) = E_x\left[\log \frac{P_r(x)}{P_\theta(x)}\right] \quad (4)$$

If we inspect it, we can see that when there is a point of real data that our generated distribution does not cover, the divergence skyrockets to infinity. This is called a false negative, and the divergence is not well adapted to quantify it, since it measures a ratio. The Jensen-Shannon divergence does not have this problem. It is equal to:

$$JS(P_r|P_\theta) = KL\left(P_r|\frac{P_r + P_\theta}{2}\right) + KL\left(P_\theta|\frac{P_r + P_\theta}{2}\right) \quad (5)$$

By the use if the mean distribution, we moderate the impact that false negatives or positives have on the value. It is possible that the use of this divergence as opposed to the KL divergence is one of the reasons for the GANs success in comparison to other models, since it penalizes false positives and false negatives equally.

2.2 Problems with the GAN at first glance

Despite its enormous power, we must discuss a few theoretical limitations with the GAN (before we get to more empirical ones), since addressing them is the motivations for the many of the variations and improvements made to the canonical GAN form.

1. Firstly, there is no guarantee that a Nash equilibrium can be reached by using a gradient-descent-based optimization algorithm [2]. In fact, we can think of many simple examples of two-argument functions whose saddle point cannot be reached by jumping steps along its gradient with respect to each of the parameters. Let x and y be our two players and $f(x, y) = xy$ is our loss function that x and y try to minimize and maximize, respectively. Using gradient-descent, we update the values such that $x := xhy$ and $y := y + hx$ where h is the learning rate. Even though the function has a saddle point at $(0, 0)$ we see the optimisation spiral the functions out of control.

2. The second problem is that of low-dimensional supports [2]. A support for a probability distribution is a manifold of points where the distribution has values greater than zero. This refers to the observation that, even though we model our distributions in an n-dimensional space (one dimension for each pixel), the distributions themselves have far fewer dimensions, since very

Algorithm 1 The canonical Generative Adversarial Network

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

Figure 4: Original GAN algorithm [4].

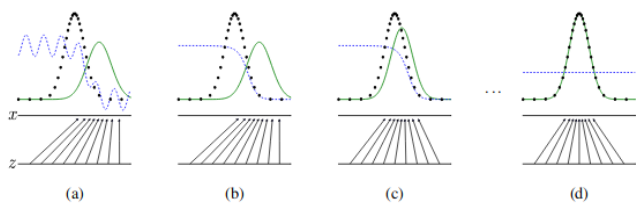

Figure 5: The black dots are $P_r(x)$, the green line is $P_\theta(x)$ and the blue line is $D(x)$. We see that D converges to its ideal value in b) and pushes G towards the real distribution. d) is the Nash equilibrium [4].

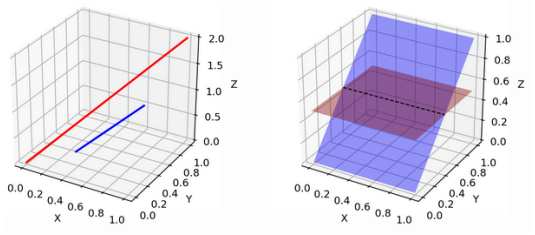

Figure 7: Left: Low dimensional supports, very likely to be disjoint. Right: High dimensional supports often intersect. [15]

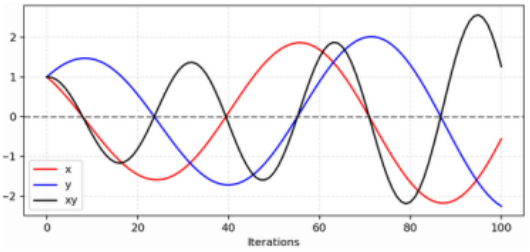

Figure 6: The function spirals outward even though the saddle point is at 0. [15]

few of all the combinations of pixels actually resemble real images. If we have two distributions of low-dimensional support in a very high dimensional space is that they are very likely to be disjoint.

The problem with having disjoint supports is that the Jensen-Shannon Divergence, which is the loss function of the GAN when the Discriminator is ideal, outputs a constant value when the distributions are completely disjoint, and has no gradient. Lets consider an example, suppose we have two distributions, P and Q:

Because we have false negatives, the KL divergence becomes infinite; this is unhelpful. The JS divergence is $\log 2$. This is also unhelpful because the value does not depend on . This means that even if we make small improvements to the distribution, the divergence will not change, which means that the gradient is zero. This is a real problem, and many times it happens that GANs do not train at all because of disjoint initial distributions.

3. The problem of vanishing gradients can exist even when the distributions are not disjoint, simply because of the numerical limitations of the computer. When the Discrimina-

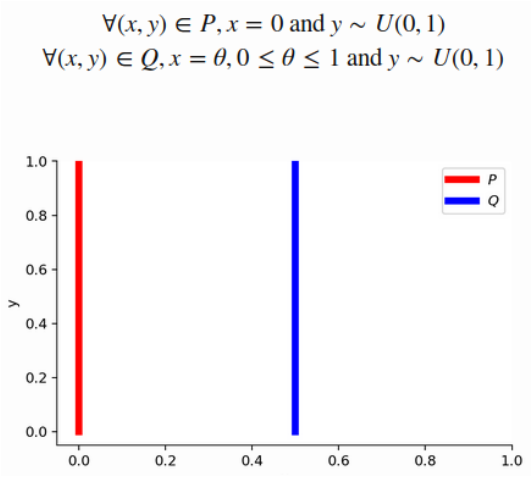

Figure 8: There is no overlap when > 0. [15]

tor becomes too confident in its predictions, the gradient in most places tends to 0. While not theoretically reaching 0, it can become so small so as to be represented as 0 by the computer. The Generator cannot train with a gradient of 0, and this is why G and D have to be carefully balanced so that none of them becomes much more powerful than the other.

4. Mode collapse is a more subtle error which is very common but not fully understood. Briefly, it describes cases when the Generator behaves very conservatively and learns to reproduce only a few modes of the real data. It is usually also associated with very low quality of produced images. It is a pathological case of the adversarial game, where instead of converging on an equilibrium, the two agents play a cat-and-mouse game that always misses the true distribution. There are many theories and approaches on how to minimize collapse.

5. A problem associated with the whole class of generative models is the lack of a proper automatic evaluation metric. Generative models are, in themselves, our attempt to understand the very complex nature of real data (e.g. to understand what features of an image make it look realistic). As such, outside human judgment, the only way to judge a generative model is through the use of another, more powerful generative model. This is an imperfect solution since the metrics will have their own problems, which makes them not fully reliable. In the experiments section, we will discuss two algorithms for such evaluation.

2.3 Wasserstein GAN

The Wasserstein GAN [1] is a variation of the canonical GAN model that addresses the problem of disjoint supports through replacing the loss function of the generator from an approximation of the Jensen-Shannon divergence to an approximation of the Wasserstein distance. Also known as the Earth-Mover distance, it is a measure of divergence between two probability distributions that intuitively represents the effort required to reshape one distribution into the other.

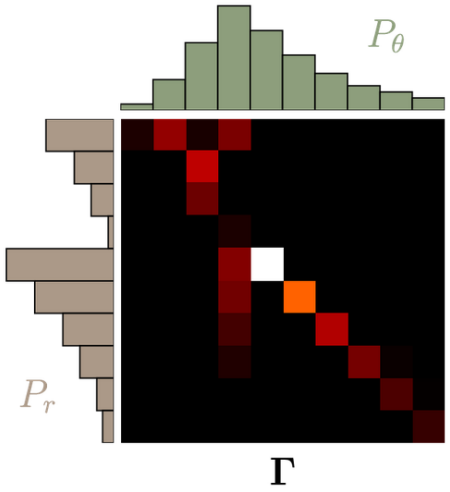

Figure 9: Optimal transport plan for changing P_r into P_θ. [7]

If we imagine the two distributions as being piles of earth, then the EM distance is equal to the amount of earth that has to be displaced in order to reshape one pile into the other, multiplied with the distance the earth has to be moved. The formula for the distance is:

$$W(P_r, P_\theta) = \inf_{\gamma \sim \prod(P_r, P_\theta)} E_{(x,y)\sim\gamma}[||x - y||] \quad (6)$$

where $\prod(P_r, P_\theta)$ is the set of all possible joint probability distributions between P_r and P_θ.

A joint distribution $\gamma \sim \prod(P_r, P_\theta)$ represents a transport plan where $\gamma(x, y)$ is the amount of mass that has to be moved from point $x \sim P_r$ to point $y \sim P_\theta$. This means that the sum of masses from all x that are transported to y is equal to the total mass of y itself (or vice versa).

$$\sum_x \gamma(x, y) = P_\theta(y), \sum_y \gamma(x, y) = P_r(x) \quad (7)$$

If, for any two points x and y, the total mass transported is $\gamma(x, y)$ and the distance transported is $||x - y||$, the expected cost averaged across all pairs is

$$\sum_{x,y} \gamma(x, y)||x - y|| \;=\; E_{(x,y)\sim\gamma}[||x - y||] \qquad (8)$$

If we take the minimum of this value across all transport plans, we obtain exactly the value of the Wasserstein distance.

This divergence metric is an improvement on the Jensen-Shannon divergence because it gives us an indication not only of how different the two distributions are, but also of how far apart they are.

To see this, we can take the same example from 8, where the KL divergence is infinity and the JS divergence is $\log 2$. The EM distance for the same example is equal to $|\theta|$, so is proportional to the distance between the two distributions. This means that the loss has a constant gradient that pushes G towards the real distribution and converges.

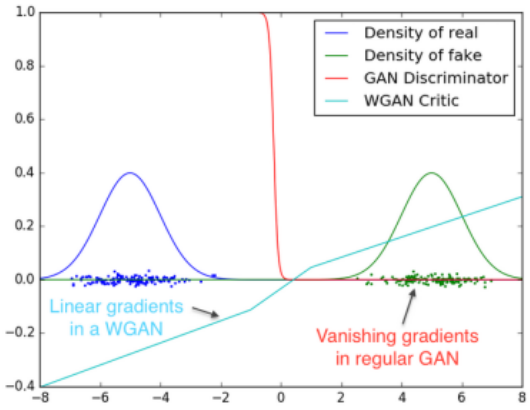

Figure 10: Comparison between the Generator losses of the standard GAN and WGAN when the distributions are disjoint. The GANs gradient vanishes, while the WGANs is non-zero. [15]

The Wasserstein distance exactly is intractable, but we can compute an approximation of it by using the Kantorovich-Rubinstein duality. The result becomes

$$W(P_r, P_\theta) = \max_{\omega\in\Omega} E_{x\sim P_r}\left[f_\omega(x)\right] - E_{x\sim P_\theta}\left[f_\omega(x)\right] \qquad (9)$$

where f_ω is a parametric function which represents the discriminator, defined by parameter ω with the property that f is 1-Lipschitz. Intuitively, this means that the slope of F never surpasses 1. To enforce this rule, the original WGAN clipped the weights of the discriminator function, but that has been shown to impede learning. The WGAN-GP [6] adds a gradient penalty to the loss in order to enforce the rule but avoid weight clipping. The loss of the WGAN-GP becomes:

$$L = W(P_r, P_\theta) + \lambda E_{x\sim P_z}\left[(||\nabla_x f(x)||_2 - 1)\right] \qquad (10)$$

where λ is the coefficient of the gradient penalty, and the function f is no longer artificially constrained.

2.4 Boundary Equilibrium GAN

Another approach that aims to stabilise the GAN training is the BEGAN [3], which uses an autoencoder as the discriminator in order to tackle the problem of balancing G and D. An autoencoder is a learning agent comprised of two components, the encoder, which maps the data to a lower-dimensional latent code, and the decoder, which seeks to reconstruct the original data from the resulting code (the Variational Autoencoder is the most well known of these). The autoencoder learns its parameters by minimizing the reconstruction loss, which is usually the squared error between the reconstructed images and the original ones:

$$L(x) \;=\; ||x - D(x)||^2 \qquad (11)$$

The novelty of the BEGAN lies in its generator loss, which is a measure of the divergence of the reconstruction losses of the real and generated distributions, rather than the divergence between the distributions themselves. The BEGAN uses the Wasserstein distance as a measure of this divergence. Another important contribution, which gives the name of this algorithm, is the idea of balancing the learning effort dedicated to the generator with the one dedicated to the discriminator. This is done by the introduction of a new hyperparameter quantifying the equilibrium.

$$\gamma \;=\; \frac{E_{z\sim P_z}\left[L(G(z))\right]}{E_{x\sim P_r}\left[L(x)\right]} \qquad (12)$$

In our model, the discriminator has two competing goals: auto-encode real images and discriminate real from generated images. The γ term lets us balance these two goals by giving us an indication of which loss is stronger. Consequently, the

losses to be minimized in each learning step become:

$$\begin{aligned} L_D &= L(x) - k_t L(G(z_D)) \\ L_G &= L(G(z_G)) \\ k_{t+1} &= k_t + \lambda_k(\gamma L(x) - L(G(z_G))) \end{aligned} \tag{13}$$

The term k balances how much emphasis is put on the real and fake loss during training and is updated according to the formula described above. The λ is simply a learning rate for k.

The BEGAN also has a measure of convergence:

$$M = L(x) + |\gamma L(x) - L(G(z))| \tag{14}$$

which decreases as the reconstruction of D becomes more faithful, and as the reconstruction errors of the real and fake data converge.

2.5 Deep Learning

So far we have seen how GANs use different functions (generators and discriminators) in order to learn distributions of data from the natural world. These functions are actually implemented as Deep Neural Networks, which are learning algorithms that have proven great representational power and have revolutionized the field of Machine Learning.

The core component of a DNN is the layer function, which consists of an affine transformation followed by a non-linear activation function. The affine transformation is basically a matrix multiplication followed by the addition of a bias. This means that the function receives as argument a vector of N feature values, multiplies that vector with a matrix of MxN weights, then adds the resulting vector to a vector of M bias terms. The weights and bias terms are the parameters that the network adjusts when training. The activation function takes the result of the affine transformation and applies some kind of simple non-linearity to it (modulus, hyperbolic tangent etc.) in order to break the linearity of the layer. The layers are stacked together so that the activation of one layer becomes the input for the next layer. It is precisely this alternation of affine and non-linear transformations that adds complexity to the modelling power of the network with each added layer. The final layer function is:

$$a^{(k+1)} = \sigma\left(W^{(k+1)}a^{(k)} + b^{(k+1)}\right) \tag{15}$$

where a are the activation vectors of the previous and current layer, W and b are the weights matrix and bias vector of the current layer, respectively and σ is the non-linearity function. Each layer has different weights and biases, in order to enhance the network representational power.

To train such a network, we need to distribute the gradient of the loss function to each of the parameters that we want to optimize, in order for them to adjust their values accordingly. A systematic way to do this is through backpropagation, whereby we calculate the gradients with respect to the parameters in the final output layers, and then work our way backwards through the computational graph of the network, computing the gradients with respect to the current layer from the gradients of the previous layer. The parameters then perform gradient descent:

$$W := W - \alpha\frac{\delta L}{\delta W} \tag{16}$$

where α is the learning rate.

The neural networks described so far are known as fully-connected layers, since there exists a variation on this basic structure, called convolutional layers, which give their name to Convolutional Neural Networks. These layer functions are specifically designed for images, although they can be used for many other purposes, and yield significantly more representational power to DNNs by making some assumptions about the data they work with.

In fully connected layers, there is no difference between the same features at different locations in the images or different features at the same location, they are all encoded as separate values in the vector of activations. Fully connected layers do not make assumptions about the nature of the data they are working with, so they will try to establish relations between any two values related to a specific piece of data. This means that, in order to extract 32 features belonging to an image of $64x64$ pixels with 3 channels, we would need a matrix of $(64, 64, 3, 32), (64, 64, 3) = 4,831,838,208$ weights, a huge number.

Because we know what the nature of our data is, we are able to separate between locational data (pixels) and features. As such, instead of encoding our activations as a vector, we encode them as a cube, where each slice is a representation of the local values of a particular feature. Instead of a matrix multiplication, we convolve each slice of the cube (each feature) with a different kernel, a smaller matrix of weights meant to perform local linear transformations in the image. Since the features we want to extract from the image

are independent of the location, we will use the same kernel for an entire slice (say a 3x3 kernel). A convolutional layer of this type would need $(3, 3, 32), (64, 64, 3) = 3,538,944$ weights to extract 32 features from a 64x64 image, a much more manageable number than with fully connected layers.

Because of this enormous efficiency compared to fully-connected layers, convolutional layers can extract a much greater number of features from an image. This is why CNNs have prompted a boom in vision-related AI technology.

In the case of GANs (and VAEs), the learning agents (Generator, Discriminator, Encoder, Decoder, etc.) are deep convolutional neural networks. The exact specifications of the networks depend on the implementation and are part of this investigation.

3 Experiments

In this section I will describe the various networks I have built and experimented with in the first part of my project, and provide a comparison between the chosen architectures and flavours. I will also discuss the automated metrics used to evaluate the results. The experiments are inspired by [10].

3.1 Building a GAN

The first part of my investigation was to implement various deep learning architectures for the Generator and Discriminator of the GAN, and observe the results. For the implementation, I used Python 3 (since it is the lingua franca of Deep Learning Models) in conjunction with Tensorflow, a framework especially designed for Deep Learning algorithms. I ran experiments on my GPU in order to speed up parallelisable operations.

The architecture that I chose as a starting point comes from the Deep Convolutional GAN, which has proven extremely popular due to its versatility and straightforward nature. In this paradigm, the Generator and Discriminator share the same basic structure, with 4 convolutional layers.

The Discriminator performs downsampling on the image in order to map it to a prediction. The approach of choice for downsampling is to use a stride in the kernel, which means that the kernel is applied at certain intervals on the locations of the activations. A stride of 2 means that each subsequent layer activations will have half the height and width of the current one. There are other approaches, such as pooling, but they have been shown empirically to be less effective since they lose positional information. I used a kernel size of 4x4, a multiple of the stride, in order to avoid aberrations. Another dimension of the network is the number of filters per convolution. The DCGAN architecture employs a fixed number of filters in the first layer (usually 64), and then doubles that number with each subsequent layer.

The Generators architecture is the inverse to that of the Discriminator. In order to perform upsampling, we need to use the kernel with a stride of $\frac{1}{2}$, so that the next layer will have twice the height and width of the current one. This is usually called a "deconvolution", although it is technically still a convolution, but with a stride smaller than 1.

Another feature of the DCGAN architecture is the use of batch-normalization in the layers. This means that the resulting activations of a layer are normalized according to the batch statistics before they are fed to the next layer. A batch is simply a group of data examples which are processed by the network simultaneously so as to take advantage of the parellelisation opportunities presented by the GPU. Batch-normalisation is useful because it adjusts the activations of a particular example in relation to the other examples, so the network has knowledge of a whole small distribution rather than a single example. This process has been shown empirically to improve the performance of the GAN.

In my experiments, I have used the Adam Optimizer, a gradient-descent based algorithm, since it has been empirically shown to obtain good results and is the preferred optimiser of most of the experts in the field.

I have used a normal distribution with a mean of 0 and variance of 1 as a latent distribution from which to sample the latent variable z. Although there is not much empirical evidence to prefer a normal distribution over, say, a uniform one, I believe it makes more mathematical sense to choose a normal distribution. Suppose we want to generate images of ducks. We want to be able to sample from the middle of the distribution, which would produce the "quintessential" exemplar of a duck, but we also want to be able to sample (with decreasing probability), from outliers of the distribution, which would produce increasingly irregular ducks (possibly blending into geese or other animals). Every real distribution has this feature whereby there exists examples which are closer to the archetype than others, and the normal distribution is the simplest one which satisfies these

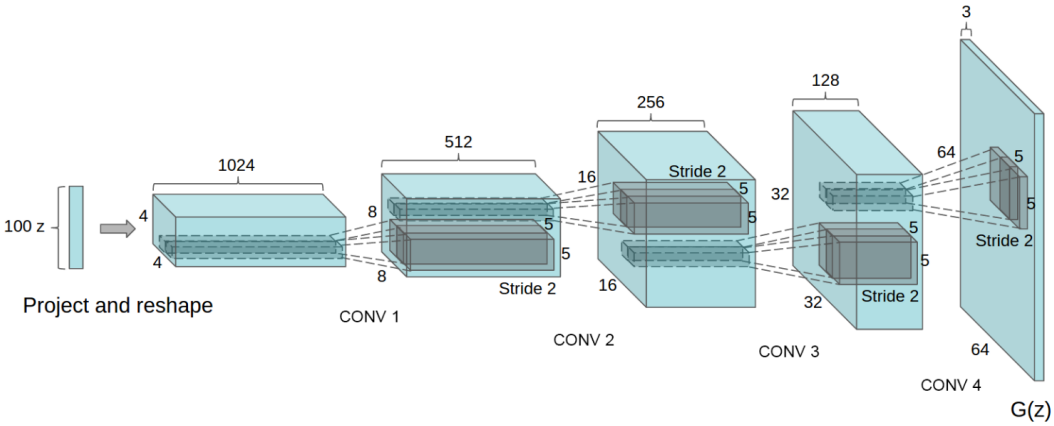

Figure 11: The basic DCGAN architecture for the Generator [12].

requirements.

3.2 The Datasets

For this project, I have chosen to work with two image datasets, MNIST and CIFAR-10. They are both datasets whereby the images are divided into classes (10 for each set) and all the classes are disjoint. We do not use the class information for training our algorithms, since we want our models to be completely unsupervised, but it is very useful for evaluation purposes, since we can check how well the models identify the classes inside the datasets. MNIST is a dataset of 60,000 28x28x1 images of handwritten digits. This dataset is quite regular, and serves as a first milestone for every network. It is also useful as the class information is very clear, so it makes the understanding of the networks easy to judge.

CIFAR-10 is a dataset of 60,000 32x32x3 photographs of the real world, where the subjects of the photographs belongs to either of ten categories (aeroplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck). This dataset is more varied and complex, and shows the limitations of the current generative models, since its distribution is much harder to capture.

3.3 The Training of a GAN

Every GAN that I have trained follows roughly the same pattern. Initially the results start as random noise, and then gradually they move towards more definite blobs of colour. After the blobs have been "decided", the network starts adding detail. This is when we can start telling more and more about the class information of the generated images. Unfortunately, for most GANs, the only way to quantify the training process is through subjective judgment, since the losses of D and G tell us nothing.

Figure 12: Examples of images from the CIFAR-10 and MNIST dataset.

3.4 The Architecture

The first experiments I performed consisted of varying the structure of the deep networks that comprised the Discriminator and Generator. The experiments and conclusions are:

1. Varying the number of layers in the architecture influences strongly the quality of samples. Deeper architectures produce crisper images with stronger colours 15.

2. Varying the number of filters in the convolutional layers also influences the quality of the generated images, particularly the level of detail. The more filters the layers have, the more details in the resulting images 16.

For a fair comparison, all subsequent algorithms have been implemented using 4 convolutional layers and 64 filters on the lowest layer.

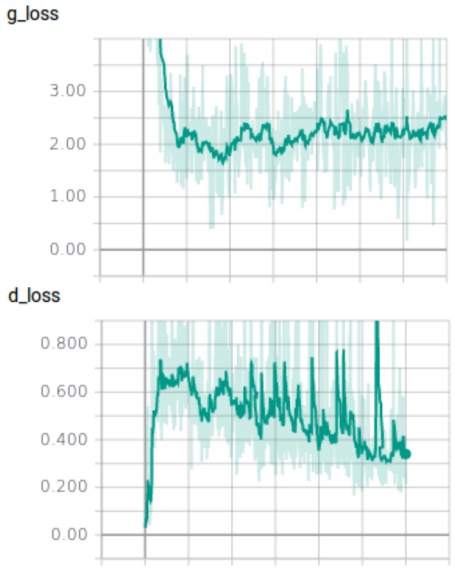

Figure 13: The losses of D and G during training of the DCGAN. They do not correlate with the increasing quality of samples.

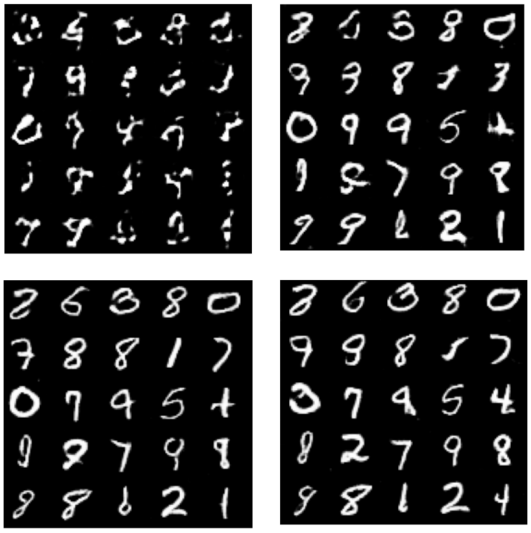

Figure 14: DCGAN training on MNIST. Top-left: 1 epoch, top-right: 3 epochs, bottom-left: 6 epochs, bottom-right: 9 epochs. The quality is clearly improving.

3.5 The Algorithms

Here we have a direct comparison between all the algorithms presented so far (DCGAN, WGAN, BEGAN, VAE), on both datasets 17.

The first observation is the stark difference in quality on the CIFAR-10 dataset between the DCGAN and WGAN on one side, and the BEGAN and VAE on the other. It seems that generative models using autoencoders have good results on low-variance datasets but produce very blurry images on high-variance ones. This may be because the lower bound of the likelihood gets increasingly lax as the dataset becomes more varied.

The second observation is that the WGAN does not seem to present a significant improvement to the DCGAN, which is using the traditional GAN cost. It seems that, once the problem of disjoint supports is surpassed, the WGAN and DCGAN converge to a similar solution. A difference would be that the WGAN images have a more fragmented aspect.

Although there is variance among GANs, we can see that all the GANs perform better than the VAE on both datasets. This is further supported by the results of the automated evaluations I have ran. Finally, we see that, even though the quality of the generated images differ from algorithm to algorithm, none of them actually seem to capture the shape of any object in the dataset. There are many reasons for this, mostly because object information is much more diverse, scarce and complex than simpler image statistics that make the images feel real. Nevertheless, feature matching is an approach that aims to tackle exactly this issue, and we will discuss it shortly.

3.6 Evaluation Metrics

Quantitative evaluation of GANs is known to be extremely difficult and largely unreliable, and it is so for a number of reasons:

GANs exist, ultimately, to produce images that humans identify as realistic or satisfactory. We may try to construct an objective metric by which to approximate the quality of the produced images, but it will always be secondary to someones subjective opinion, since the utility of the metric itself is tested against human judgment. Not only are the current metrics much too simplistic to compete with human perception, but even some hypothetical metric, however complex, would be unreliable precisely in the ways in which it differs from the human judgment with which it is trying to compete.

Furthermore, should a new metric be invented that is reliably better then all of the others created so far, that metric could instantly be used as the new loss function for which the GAN optimizes, thereby improving the performance of all GANs, but, at the same time, leaving us again with no metric by which to judge the performance of individual architectures.

As such, automatic quantitative evaluation of GANs exists in a niche, that of evaluating many

more images than a human ever could, with an architecture more complex than that of the GAN itself. Although it may seem artificial to judge the performance of an algorithm by use of a more complex version of a similar algorithm (after all, all that extra computing power could have just gone into making a more powerful GAN), it is still instructive to perform this experiment in order to compare the performance of GAN flavours and architectures.

Both measures that I used in my project are based on the Inception network, a classifier network trained on real images that is known to extract features very effectively in its upper layers.

1. The Inception Score [14] is one measure of the quality of generated samples, which aims to quantify how clearly the images can be classified into categories and how evenly distributed those categories are. We run the trained classifier on the generated images and record the output in order to calculate the conditional probabilities $p(y|x)$, where y is a label and x is an image. We want the images to be clearly separable into classes, so $p(y|x)$ should be low entropy, and all classes to appear in roughly the same amounts, so $p(y)$ should be high entropy. The measure that rewards both characteristics is:

$$IS(G) = \exp\left(E_{x\sim P_g} KL(p(y|x)|p(y))\right) \quad (17)$$

This measure correlates well with human judgment as long as the images look "nice". There exist many kinds of adversarial examples, but they are generaly unlikely and easy to spot.

2. The Frechet Inception Distance [8] is meant as an improvement to the Inception Score. It refers to the activations in an intermediate layer inside the Inception network, and it calculates the Frechet distance between the distribution of activations when the network is fed real data, and the distribution of activations when the network is fed generated data. The expression for this distance is:

$$||m - m_w||^2 + Tr(C + C_w - 2(CC_w)^{\frac{1}{2}}) \quad (18)$$

I have used the two measures in parallel in order to judge the models more holistically.

3.7 Results

The results of the automated evaluation are largely in agreement amongst themselves and with human judgment. DCGAN performs best, followed by WGAN, BEGAN and VAE. This indicates that there is a correlation between the metrics and the subjective criteria we use to judge images.

	IS	FID
DCGAN	**6.01**	**59.6**
WGAN	5.99	65.8
BEGAN	3.64	136.2
VAE	3.21	158.3

4 Visualisations

In order to understand more about how GANs learn and represent the information internally, we need a way to visualize the activations of the layers inside the network. Because the activations are strongly multi-dimensional, I have built a dimensionality reduction algorithm that combines two well known techniques, PCA and t-SNE, in order to bring the activations from a multi-dimensional space into a two dimensional space without losing too much of the relevant data.

Principal Component Analysis is a reliably statistical tool for dimensionality reduction. Briefly, given a set of points, PCA finds the system of N orthogonal coordinates that explains as much of the variance in the data as possible. N is the dimension of the space to which we want to map our data. Mathematically, PCA is an orthogonal transformation whereby the datapoints are multiplied with a matrix to produce their coordinates in the new, lower-dimensional space. Although a powerful technique in its own right, we use the PCA to weed out unimportant components in our data so that we remain with 50 principal components. The explaining power of the 50 principal components also gives us an indication of the entropy of the data (if 50 principal components explain a low total variance, that means the data is compact and so the entropy is high).

We then pass the 50 extracted principal components through a t-distributed Stochastic Neighbour Embedding [11]. This is a newer, more advanced technique that aims to reduce the dimensionality of the set while preserving the relative distances between points. It does this by generating datapoints in a two dimensional space and

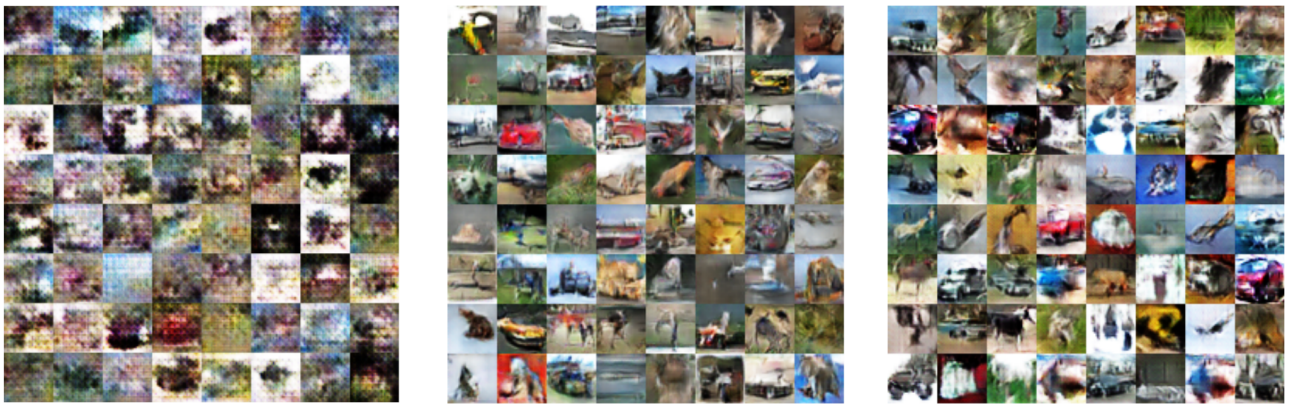

Figure 15: DCGAN on CIFAR-10. Left: 3 conv layers, Centre: 4 conv layers, Right: 5 conv layers. Quality increases with number of layers.

Figure 16: DCGAN on CIFAR-10. Left: 32 filters, Centre: 64 filters, Right: 128 filters. Quality increases with number of filters.

Figure 17: DCGAN, WGAN, BEGAN, VAE on CIFAR-10 and MNIST.

then adjusting the distances between the points so that the distribution of closeness between pairs of points in the lower dimensional representation is similar to the one in the higher dimensional space. It learns this similarity by minimizing the KL divergence between the two distributions. We use the t-SNE in order to transform our 50 principal components into 2 dimensions, so that the activations may be visualized in a diagram.

The final piece in the visualization tool is a Deep Convolutional Classifier that I implemented in order to assign labels to the generated data. This way, we can trace the classes through the layers of the GAN and identify which activations produced which classes. The Classifier that I have built is a fully convolutional network with 9 convolutional layers, that achieves an accuracy of 98% on MNIST.

Observations:

1. Class information is not very visible in the early layers of the generator 18, as the classes have a great deal of overlap. On one hand, this is unsurprising, since the latent distribution is normal and class differentiation requires a number of transformations before it can become apparent. On the other hand, the generated data 19 seems to have more class information than conv2, which means that the early layers of the Generator do not understand the data distribution, but rather try to imitate it. This may be why GANs seem unable to produce coherent objects in their images: the early Generator layers do not have a grasp on high level features in the data.

2. Class information becomes much more visible in the higher Discriminator layers than it is in the data itself. This indicates that the Discriminator encodes a feature extractor. The real data is more clearly separable from the generated data in the activations of the higher layers of the Discriminator. Perhaps if we tried to pull the distribution of the activations of the generated data towards the distribution of the activations of the real data, this would stimulate the class differentiation in the generator and, thus, produce images with better defined objects. This is the starting point for the idea of Feature Matching.

5 Feature Matching

Feature Matching [14] means, essentially, adding a loss to the Generator that quantifies the divergence between the activations of the real data in a higher layer of the Discriminator, and the activations of the generated data in such a layer.

$$||f(x) - f(G(z))|| = \sqrt{\sum_{i}^{N}(f(x_i) - f(G(z_i)))^2} \tag{19}$$

The initial model proposed an error loss for this divergence where f is the hidden layer in the Discriminator (in our case, conv3). An improvement of this technique came with Denoising Feature Matching [13], whereby the simple loss is replaced with a Denoising Autoencoder tasked with learning the distribution of the activations of the real data. This means that, in this paradigm, the Discriminator stays the same, the Denoising Autoencoder learns the distribution of real activations, and then the Generator optimizes for the reconstruction error of the DAE when it is fed activations from generated data.

$$\begin{aligned} &\min_{\theta_r} E_{x\sim P_r}\left[||\Phi(x) - r(C(\Phi(x)))||^2\right] \\ &\min_{\theta_G} E_{z\sim P_z}\left[\lambda_d||\Phi(x) - r(C(\Phi(x)))||^2\right] \end{aligned} \tag{20}$$

The first term is the reconstruction loss for the DAE on the activations of the real data. The second term is the loss for the Generator, it is then added to the standard cost of the Generator in order to enhance the learning. Denoising Feature Matching is reported to produce significant improvements to the performance of the GAN.

5.1 My version of Feature Matching

Inspired by feature matching, I attempted to build my own solution to the problem of object incoherence in generated images. Starting from observation 1. in the previous section, I designed and implemented a version of the GAN meant to increase class awareness in the early layers of the Generator by stimulating the Generator to mimic the Discriminators hidden layers. To this effect, I implemented an additional discriminator, called F, which is meant to separate between the activations of conv3 in the Discriminator and the activations of conv2 in the Generator. The losses for this new GAN become:

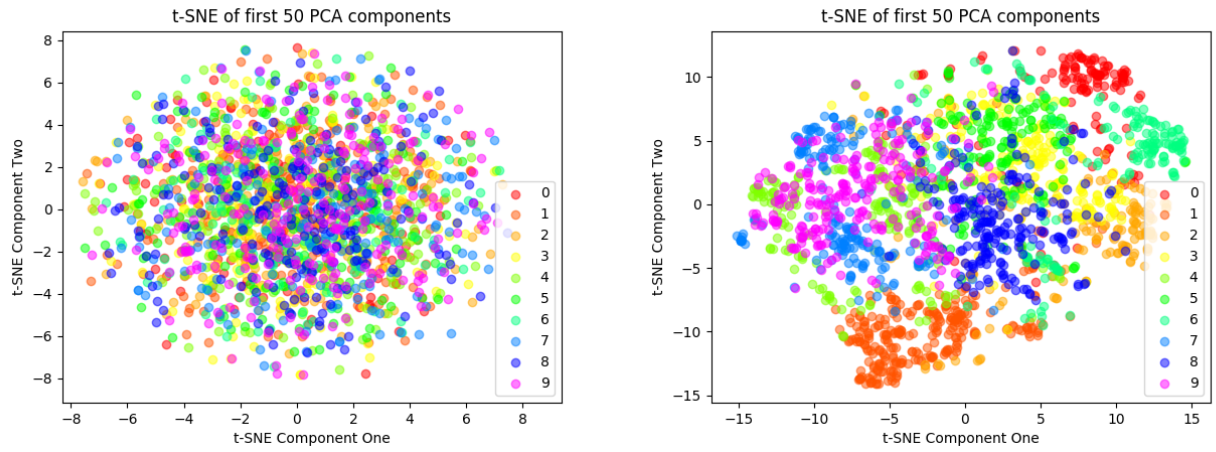

Figure 18: Left: P_z, Right: conv2 G layer.

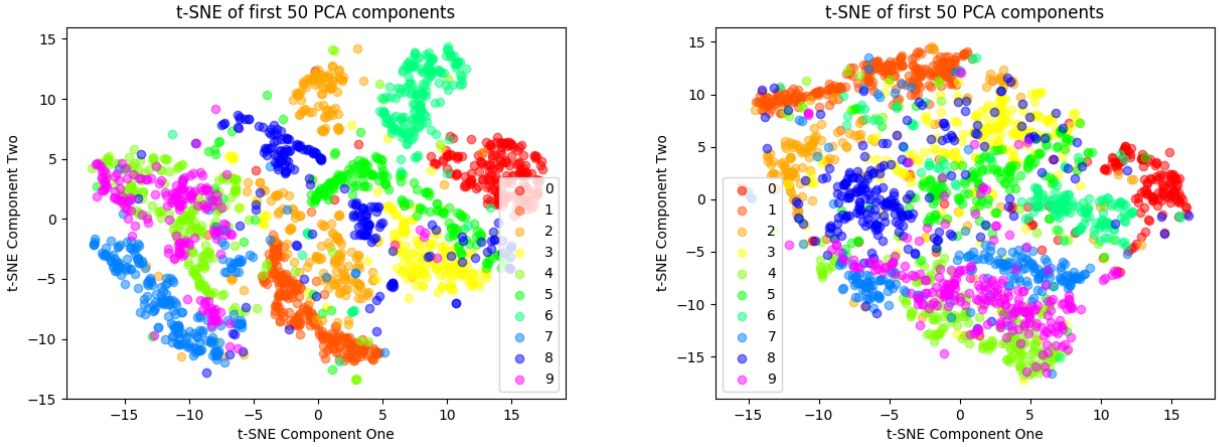

Figure 19: Left: Real MNIST images. Right: DCGAN generated MNIST images.

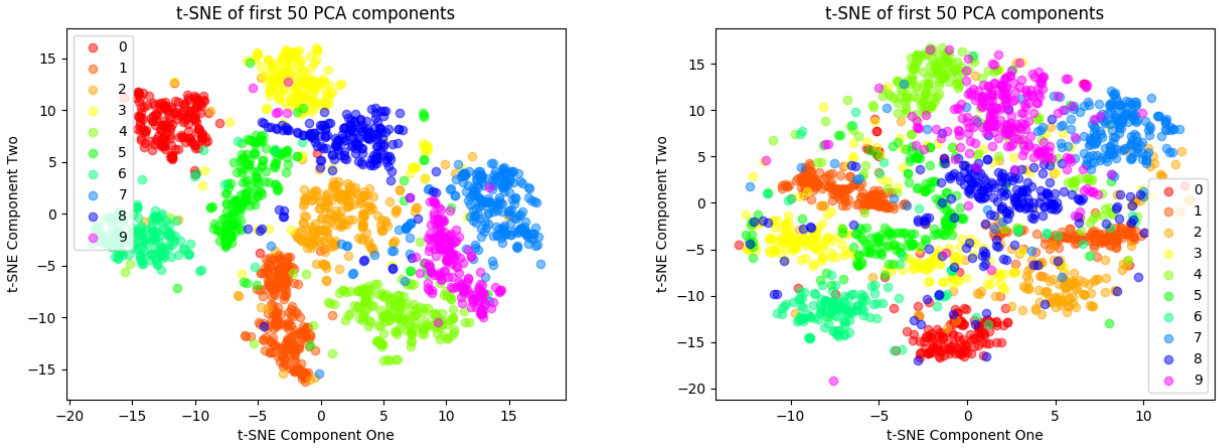

Figure 20: DCGAN on MNIST. Left: D conv3 layer on real images, Right: D conv3 layer on generated images.

$$
\begin{aligned}
L_D &= E_x\left[\log(D(x))\right] + E_z\left[\log(1 - D(G(z)))\right] \\
L_F &= E_x\left[\log(F(D_f(x)))\right] + E_z\left[\log(1 - F(G_f(z)))\right] \\
L_G &= -(L_D + L_F)
\end{aligned} \tag{21}
$$

Where G_f and D_f are the hidden layers in the network.

The hope is that this additional discriminator F would bridge the gap between the Generator and the Discriminator, and that the Generator could learn from the inner states of the Discriminator.

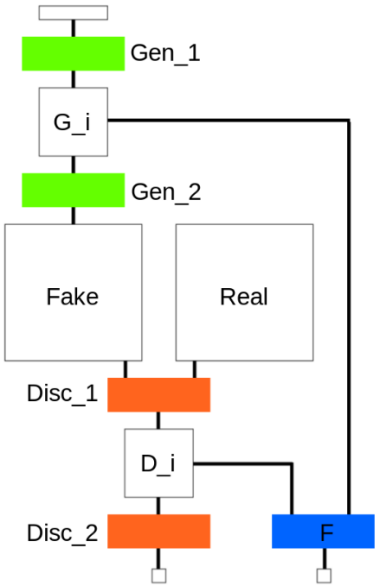

Figure 21: A sketch of the architecture of my version of Feature Matching.

5.2 Results

Unfortunately, when running the two feature matching algorithms, there doesnt seem to be any significant improvement on the DCGAN 22. While the samples look good, this is far from the promised increase in object coherence.

The automated evaluation metrics tell a similar story, where DFM and my implementation achieve slightly higher Inception Scores than the DCGAN, but also perform sightly worse on the Frechet Inception Distance.

There could be a number of reasons for these results. The first is that, due to hardware limitations, none of the algorithms achieved their full potential and, therefore, it is difficult to judge their performance. The second is that Feature Matching may not be such a robust solution as previously thought, and there exist such cases when applying the additional loss function to the Generator does not noticeably improve the results.

	IS	FID
DCGAN	6.01	**59.6**
WGAN	5.99	65.8
BEGAN	3.64	136.2
VAE	3.21	158.3
DFM	6.06	73.9
mine	**6.12**	64.4

6 Conclusion and Future Work

In conclusion, the Generative Adversarial Network is a powerful tool for synthetic image generation, but many fundamental issues with it still remain unsolved. In this work, I have observed how varying the specifications of the network architecture noticeably influences the quality of the results, I have compared the performance of different generative algorithms and found that no version consistently outperforms the standard one, I have implemented visualization tools for seeing inside of the GAN layers and have implemented some feature matching techniques that did not constitute a significant improvement on the standard GAN.

The future of the GAN models probably lies in further investigation into making the networks learn high level features more readily. This could be done through techniques such as feature matching, minibatch discrimination or progressive growing, or could instead focus on compiling datasets more suitable for learning coherent objects. This must be coupled with an increasingly growing understanding of the mathematics behind the GAN architecture and its training, which, as of now, remains elusive.

References

[1] Arjovsky, M., Chintala, S., Bottou, L. Wasserstein GAN. arXiv:1701.07875v3 [stat.ML] 6 Dec 2017.

[2] Arora, S., Zhang, Y. Do GANs actually learn the distribution? An empirical study. arXiv:1706.08224v2 [cs.LG] 1 Jul 2017.

[3] Berthelot, D., Schumm, T., Metz, L. BEGAN: Boundary Equilibrium Generative Adversarial Networks. arXiv:1703.10717v4 [cs.LG] 31 May 2017.

Figure 22: Left: DCGAN, Centre: DFM, Right: mine.

[4] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y. Generative Adversarial Networks. In Advances in neural information processing systems, pp. 26722680, 2014a.

[5] Goodfellow, I. NIPS 2016 Tutorial: Generative Adversarial Networks. arXiv:1701.00160v4 [cs.LG] 3 Apr 2017.

[6] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A. Improved Training of Wasserstein GANs. arXiv:1704.00028v3 [cs.LG] 25 Dec 2017

[7] Herrmann, V. Wasserstein GAN and the Kantorovich-Rubinstein Duality. 24 Feb 2017.

[8] Heusel, M., Rmsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In Advances in Neural Information Processing Systems 30 (NIPS 2017).

[9] Isola, P., Zhu, JY., Zhou T., Efros, A. Image-to-Image Translation with Conditional Adversarial Networks. In Computer Vision and Pattern Recognition 2016.

[10] Lucic, M., Kurach, K., Michalski, M., Gelly, Sylvain., Bousquet, O. Are GANs Created Equal? A Large-Scale Study. arXiv:1711.10337v3 [stat.ML] 20 Mar 2018.

[11] van der Maaten, L., Hinton, G. Visualizing Data using t-SNE. Journal of Machine Learning Research 9 (2008) 2579-2605.

[12] Radford, A., Metz, L., Chintala, S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. ICLR 2016.

[13] Warde-Farley, D., Bengio, Y. Improving Generative Adversarial Networks With Denoising Feature Matching. In International Conference on Learning Representations 2017 (Conference Track).

[14] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X. Improved techniques for training gans. In Advances in Neural Information Processing Systems, pp. 22342242, 2016.

[15] Weng, L. From GAN to WGAN. 2017