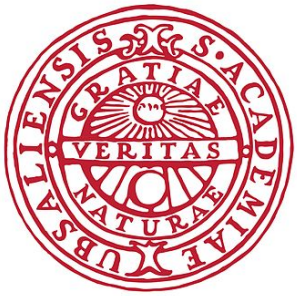


UPPSALA UNIVERSITET

Generative Adversarial Networks
Öppen fördjupningskurs i inbyggda system 1TE722

Daniel Hjelm 980406-2454
Daniel.Hjelm.0958@student.uu.se

June 4, 2022

Abstract

The aim of this project is to familiarize and study the Generative Adversarial Networks (GAN) in order to be prepared for a internship at Deutsches Elektronen-Synchrotron (DESY) in Zeuthen, Germany where the GAN framework will be implemented to generate realistic noise for a neutrino detector. From the analysis, it is clear that there are both basic and advanced GAN versions capable of delivering extraordinarily impressive generated data, ranging from images of faces to footprints of cosmic-ray airshowers. One of the GAN versions, the Wasserstein GAN, have in previous and similar studies proved to be a successful candidate for the DESY project but have strong competition in other GAN versions as well as other methods such as the Variantional Autoencoder. Whether the Wasserstein GAN or any other model will be able to generate realistic noise will be investigated further at DESY in the autumn but one thing is sure and that is that the GAN family is unprecedented framework and deserves all the hype it gets.

Contents

1 Introduction

1.1 Background

During the last couple of years, a construction of a neutrino detector called The Radio Neutrino Observatory in Greenland, or in short RNO-G, have been started. As the name suggest, the detector aims to detect ultra high energetic neutrinos. A neutrino is an elementary particle with spin $\frac{1}{2}$ that only participate in gravity and weak interactions. Detecting neutrinos, as for all measurements, comes with unwanted noise which might interfere with the measurements. However, this noise can be useful when studying neutrinos and the idea of using the noise is where this project originates from.

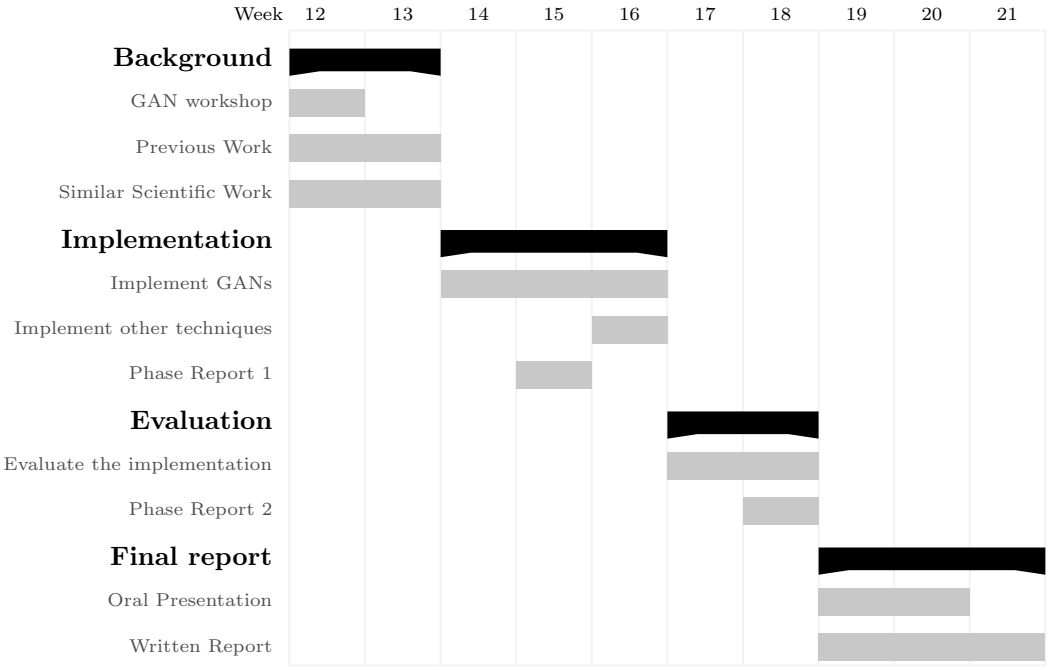
In the autumn, a project stretching over two months will take place at Deutsches Elektronen-Synchrotron (DESY) in Zeuthen, Germany. The project is managed by astroparticle physicist and professor Anna Nelles and aims to architect and implement a Generative Adversarial Network (GAN) to generate realistic noise for RNO-G. This is of huge importance since realistic noise can be used to simulate actual neutrino events. At the present time, the noise in the simulations is added by collecting data from the signals containing only noise, setting huge demands on processing and storage of large amounts of data. If a GAN can successfully create realistic noise, less data and time will be used, resulting in a more efficient simulation process which is essential for future research about neutrinos.

1.2 Purpose of the project

In order to be prepared for the project in the autumn, the purpose of this project is to familiarize with and study GANs as well as other concepts needed for the internship at DESY. In that way, a better and more promising result can be achieved at DESY which hopefully can result in helping further work and research in detecting neutrinos.

1.3 Plan

The project has been carried out according to the time schedule displayed in the Gantt chart below.



2 Generative Adversarial Network

Generative Adversarial Network, in short GAN, is a set of deep learning generative models originally published at Université de Montréal by Ian Goodfellow et al. back in 2014 [1]. The architecture of the GANs consist of two neural networks called generator and discriminator competing against each other in sort of a game where one's loss is the others gain. Despite their young age, GANs are used in numerous of application across several fields of research. As an example, GANs have been used in art where they are a central role in creating AI art as well as inpainting paintings and photographs [2]. GANs have also made an appearance in science where researches have used GAN models to improve astrophysical images [3] and to model high energy jet formation [4]. Despite the rapid increase of usage and need of GANs over the last years, concerns about the model and similar models have been raised. One of the main concerns is about producing fake videos, images and photographs, possibly used for nefarious purposes. At the website thisxdoesnotexist.com [5] one can see super realistic images of everything between cats to rentals created by GANs which hopefully are not used for wrong reasons.

2.1 Architecture

As previously mentioned, the architecture of the GANs consist of two neural networks called generator and discriminator. To describe the relationship between the two networks within the GAN on a high level, one can use an analogy including an art forger and an art expert [6] as seen in Figure 2.3. The art forger, representing the generator, tries to create realistic images whilst the expert (discriminator), receives these images as well as true authentic images, and wish to to tell if they are real and fake. Both the forger and the expert are trained together, essentially competing in a game where they try to beat the other. The competition drives both models to improve and after generations of training the art forger will, hopefully, be able to create images that are realistic enough so that the discriminator can not tell forged and authentic images apart. As a result, we have a forger (generator) creating super realistic images not even an expert (discriminator) can tell are fake.

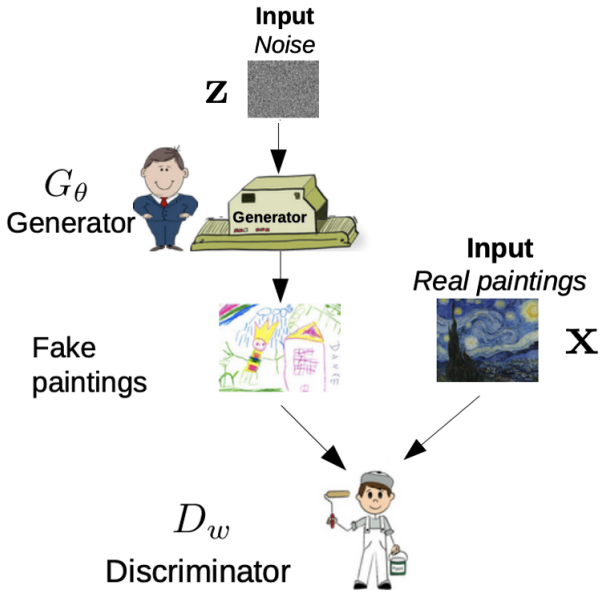


Figure 2.1: A high-level representation of a GAN consisting of an art forger and expert [7].

A more detailed and low level description of the architecture is displayed in Figure 2.2 below. The GAN works by the principle that the generator $\mathcal{G}$ takes some kind of noise $\mathbf{z}$ as input from a random probability distribution $\mathbf{p(z)}$. This noise is then mapped to a distribution by the generator to generate a sample $\mathbf{x}$' which is then fed to the discriminator $\mathcal{D}$. In addition to the fed data from $\mathcal{G}$, the discriminator takes real data $\mathbf{x}$ from the distribution of real data $p_{data}(\mathbf{x})$. The discriminator then predicts if the data is real or fake using an activation function such as the Sigmoid activation function and outputs a number between 0 and 1 where 0 represent fake and 1 real data.

2.2 Training

Since the GANs consists of two different networks, the training have to be done in a smart way. Generally for neural networks, the weights in the network is updated to reduce the loss or error

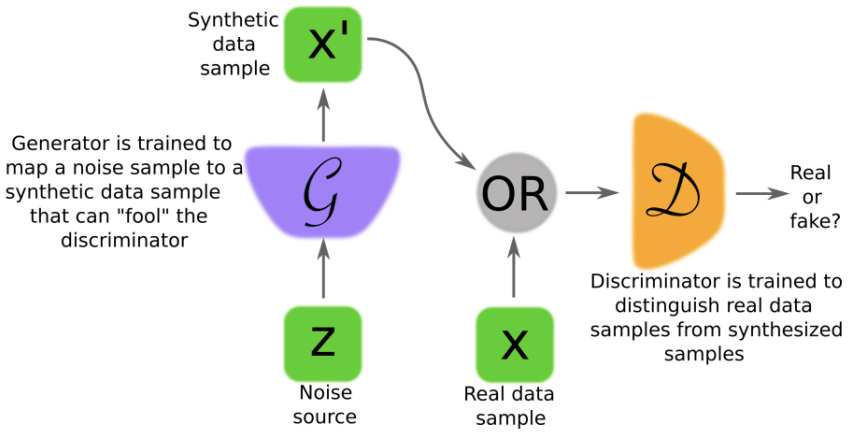


Figure 2.2: The architecture of the GAN consisting of a generator and discriminator [6].

of the output. In a GAN however, the loss is not directly connected to the generator but the loss is achieved through the discriminator's prediction. Hence, the discriminator must be included in the training of the generator since the generator learns from the discriminators feedback. However, the discriminator should not change during the training of the generator since it would make it hard, if not impossible, for the generator to hit a moving target. As a result, the generator and the discriminator have two completely different training processes, both of which are included in the training of the entire GAN.

As previously mentioned, the discriminator takes fake generated data from the generator and real data as input. Hence, the discriminator can be seen as a simple classifier trying to distinguish between real and fake data. As a result, the training of the discriminator is quite simple and can be summarized into three steps:

1. The discriminator classifies real and fake data.
2. The loss for the discriminator penalizes the discriminator for misclassifying.
3. The weights of the discriminator is updated using backpropagation.

An important note is that the discriminator is connected to two loss functions: its own loss function and the generator's. Therefore it is important that it uses the right loss function at the right training sequence, namely discriminator loss for discriminator training and generator loss for generator training.

The training procedure of the generator can be explained in the following steps:

1. Generator samples random noise **Z** from random probability distribution **p(z)**.
2. Generator produces a output sample **X'** and feed it to the discriminator.
3. Discriminator classifies **X'** and the generator loss for the classification is calculated.

4. Use backpropagation through both generator and discriminator to find gradients and update generator weights according to the gradients.

Now that the generator and discriminator training processes are set, the entire GAN can be trained by using alternating training:

1. Discriminator train for one or several epochs.
2. Generator train for one or several epochs.
3. Continue with previous steps to train the generator and discriminator.

This alternating, back and forth, game that it is to train these networks can also be described mathematically by defining the value function:

$$V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{p_{data}(x)} \, log\mathcal{D}(x) + \mathbb{E}_{p_g(x)} \, log(1 - \mathcal{D}(\mathcal{G}(z)) \tag{2.1}$$

where $\mathbb{E}$ is the expected value, p_{data} is the distribution of the data and p_g is the generated probability distribution for the generator sample. This value function is then used in a two-player minimax game:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{G}, \mathcal{D}) \tag{2.2}$$

where $\mathcal{D}$ aims find the maximum probability to select the right label for generated samples and real data whilst $\mathcal{G}$ aims to minimize $V(\mathcal{G}, \mathcal{D})$ to create the smallest difference between generated and real data [6]. The optimal point of Eq. 2.2 is called Nash equilibrium and is achieved when both the discriminator and the generator "knows" the other's strategy and realize there is no gain in changing strategy [8].

2.3 Challenges for GANs

Generative Adversarial Networks have, just like other generative models, some common challenges, problems and failure modes. None of these problems have been fully solved but are part of active research. One of the problems which was pointed out by M. Arjovsky and L. Bouttou [9] are vanishing gradient, a common problem for neural networks. Vanishing gradients appears during training and prevents the network's weights to be updated between iteration due to the update being proportional to the vanishingly small gradients. This phenomena may appear in a GAN if the discriminator is close to optimal and do not serve the generator with enough feedback to improve. A suggested solution to the problem is to change the loss function to the Wasserstein loss which intended design is to prevent vanishing gradients [10] (more on Wasserstein GAN in Section 2.4.4). Goodfellow et al. suggested another approach where the loss function is modified so that the generator tries to maximize $log(\mathcal{D}(\mathcal{G}(z))$ rather than minimizing $log(1 - \mathcal{D}(\mathcal{G}(z))$ which proved to strengthen the gradients, especially in early stages of training [1].

Commonly when you implement a GAN, you want to produce a great mixture of outputs. As an example, when generating images of cars you want a different car for every random input. This is not always the case for GAN but the generator might learn to generate a feasible output and then continue to generate only that output. That is, the generator finds the output that seems to the best fit for feeding the discriminator. When the generator finds one, or possible a small pool of outputs, to feed to the generator repeatably, the discriminator should develop a strategy to reject

these outputs. However, the discriminator might get stuck in a local minima and can't find this optimal strategy. As a result, the generator finds the most plausible set of outputs for this stuck discriminator and optimizes for this particular discriminator. This type of failure for a GAN is commonly referred to as mode collapse and the Wasserstein is one proposed solution to it since it manages the problem with vanished gradients. Another attempt to remedy was proposed by Metz et al. [11] and is called Unrolled GANs. This type of GAN uses a loss function that includes future discriminators and hence the generator can't only focus on optimizing its outputs for the current one.

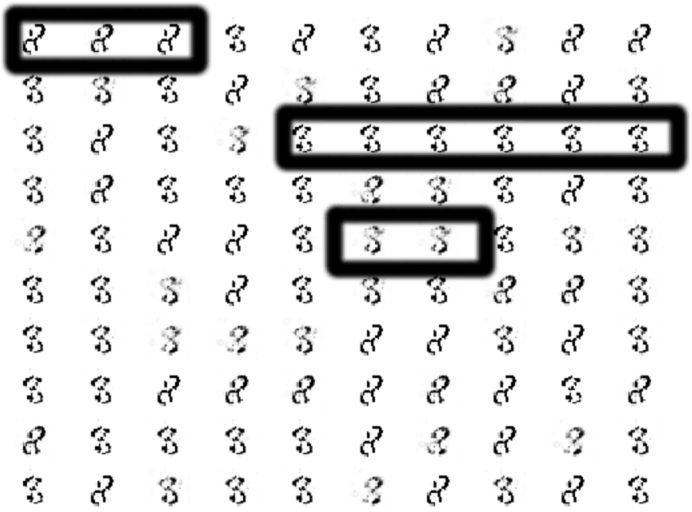


Figure 2.3: An example of mode collapse for a Conditional GAN where the GAN generates the same output several times [12].

As training proceed, the generator will hopefully improve and the performance of the discriminator will deteriorate since it will be harder to differentiate between fake and real data. If the training succeeds completely, the generator will be so good that the discriminator can't tell the difference and simply flips a coin when making its prediction. This effect presents a problem with the convergence of the GAN framework. Since the discriminator can't tell real and fake data apart, the feedback fed back to the generator will become less and less relevant. At some point it will become entirely random feedback which might confuse the generator during training, possibly tearing down the performance of the generator completely. An example of how a convergence failure for a GAN, with associated loss and accuracy function, is displayed in Figure 2.4 and 2.5 below.

The convergence of a GAN is seldom described as stable but rather a fleeting state. A common approach used by researchers to improve the convergence is to use some form of regularization. M. Arjovsky and L. Bouttou [9] tried to add random noise to the discriminators input whilst K. Roth et al. implemented a regularization scheme including a added penalty on the weighted gradient-norm of the discriminator [13]. Both approaches proved to improve the stability of the GAN but the problem is still investigated further in several fields of research.

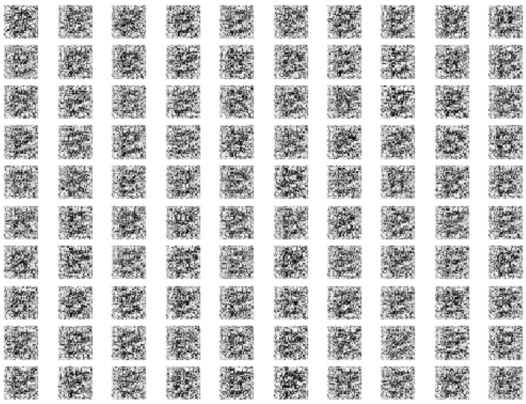


Figure 2.4: Sample of generated images of handwritten number 8 from a GAN that has failed to converge [12].

Figure 2.5: Plots of the accuracy (left) and loss (right) of a GAN that has failed to converge [12].

2.4 GAN versions

During the last couple of years, researchers have found new techniques to improve the GAN framework and create new versions of the GAN to add to the GAN family. In the coming subsections, a sample of all the GAN versions will be presented to give a taste on the different approaches researches have used to improve the GAN as well as to give a sense of the possibilities with the GAN framework and what great assets they actually are.

2.4.1 Original GAN

The original Generative Adversarial Network presented by by Ian Goodfellow et al. [1], also called original or first GAN, uses a simple fully connected neural network architecture for both the discriminator and generator. This network suffers from the challenges discussed in Section 2.3 and can be implemented for relatively simple tasks such as generating 1D data or simple images like the hand written digits in the MNIST dataset [14].

2.4.2 Convolutional GAN

Thanks to the nature of convolutions, changing from full-connected networks to convolutional neural networks within the GAN should be perfect fit when handling image as data. However, experiment performed on the natural image dataset CIFAR-10 [15] suggested that the training was more difficult than anticipated. As a result, A. Radford et al. presented a set of convoultional networks called Deep Convolutional Generative Adversarial Networks, or in short DCGAN [16]. The DCGAN

networks rely on strided, and in some cases fractionally-strided, convolutions allowing the networks to learn both up- and down-sampling operators during training. These operators are key when mapping from an image space to a lower dimensional space such as the latent space and from the image space to for example the discriminator thanks to their ability to manage the change in location and sampling rates [6]. All in all, the convolutional, and notably DCGAN, have proven to be more powerful than the original version, especially when handling images.

2.4.3 Conditional GAN

The conditional GAN, in short cGAN, is a version of the original GAN having the capability to take input which becomes a condition for both the discriminator and the generator [17]. This makes us as user able to control the generation of data which can be helpful when for example dealing with imbalanced data set where it is possible to generate more samples of the underrepresented class. For instance, when using the MNIST data set, certain digits can be generated or in the case of CIFAR-10, specific images of for instance horses can be produced.

The original GAN can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information i [17]. This extra information i could be any kind of auxiliary information, such as class labels. The conditioning is done by feeding i into the both the generator and discriminator as additional input layer. As a result, the mathematical formulation of the value function 2.1 is reformulated to:

$$V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{p_{data}(x)} log\mathcal{D}(x|i) + \mathbb{E}_{p_g(x)} log(1 - \mathcal{D}(\mathcal{G}(z|i)) \tag{2.3}$$

There are several variants of cGANs that have been published in the last years. X. Chen et al. [18] presented in their paper the Information Maximizing GAN, in short InfoGAN, which by introducing control variables allows the user to control the features of generated of images such as thickness and style in the case of studying the MNIST dataset. The Auxiliary Classifier GAN (AC-GAN) on the other hand, requires the discriminator to not only predict the realness of the images but also the class/source of the image. As result, the training is stabilized and allows generation of high resolution and quality images whilst learning to represent the latent space independently from the label of the class [12].

2.4.4 Wasserstein GAN

In 2017 Arjovsky, et al. [10] introduced the Wasserstein GAN, WGAN for short. The WGAN architecture aims to find an alternative approach to train the generator in order to find a better approximation of the distribution of the training data. In the WGAN the discriminator is changed to a critic which instead of giving classification or a prediction of the probability of a sample being real or fake, it "scores" the sample's realness or fakeness. The motivation for this is to train the generator to seek the minimum distance between the distribution of the training set and the distribution generated by the generator. This measure is commonly called Wasserstein-1 which is an approximation of the Earth-mover (EM) distance and is defined as:

$$W(P_r, P_g) = \inf_{\gamma \in \prod(P_r, P_g)} \mathbb{E}_{(x,y)\sim y}[||x - y||] \tag{2.4}$$

where $\prod(P_r, P_g)$ is the set of all joint distributions $\gamma(x, y)$ whose marginals are the real probability distribution P_r and the generated probability distribution P_g. The term $\gamma(x, y)$ can be thought of

as the mass needed to be moved to transform P_r into P_g and the EM-distance as the calculated cost of the optimal path. Since the infimum in Eq. 2.4 is highly intractable, Arjovsky, et al. got rid of it using the Kantorovich-Rubinstein-duality [10] which gives the following equation:

$$\sup_{||f||_{Lip\leq 1}} \mathbb{E}_{x\sim P_r}[f(x)] - \mathbb{E}_{x\sim P_g}[f(x)] \tag{2.5}$$

This approach is an approximation of the EM-distance since it imposes a limitation on f to be 1-Lipschitz-continuous. The 1- Lipschitz-continuous condition can be expanded to K-Lipschitz-continuous by changing $||f||_{Lip\leq 1}$ to $||f||_{Lip\leq K}$. To make the most out of the K-Lipschitz-constraint, it is added to the loss function of the discriminator, making it penalize gradients which are not equal to 1. As a result, the gradient penalty loss function is defined as:

$$GP(x_r, x_g) = \mathbb{E}[f(x_r)] - \mathbb{E}[f(x_g)] - \lambda\,\mathbb{E}[(||\nabla f(\hat{u})||_1 - 1)^2] \tag{2.6}$$

where x_r are real data samples, x_g are generated samples, λ scales the magnitude of the penalty and the mixture term $\hat{u}$ is defined as:

$$\hat{u} = \epsilon x_r - (1-\epsilon)x_g \tag{2.7}$$

where ϵ is uniformly sampled from a distribution with values between 0 and 1. Using this approximate Wasserstein-distance approach as the loss function for the WGAN helps with stabilization and makes the model less sensitive during training. In some cases more importantly also relates the discriminator loss to the quality of the images generated by the generator.

2.4.5 Image-to-image translation GANs

Image-to-image translation is the controlled process of converting an image from a certain domain to another where the target is to map a given input image to an output image. For instance, the process of converting a black and white image to a color image is an image-to-image translation. This is often a very demanding and hard problem which commonly depend on specially designed loss functions and models for the given data set.

The Pix2Pix GAN model presented by P. Isola et al. [19] in 2016, is designed for image-to-image translation. The Pix2Pix architecture is an extension of the cGAN where the condition is a given input image. The generator is built using the U-net architecture [20] and does not take an input from the latent space like the original GAN but the needed randomness instead originates from dropout layers in the model. In addition, the discriminator focuses on a patch scale, trying to classify the realness of every patch of an image. The Pix2Pix objective function can be formulated as:

$$\arg\min_{\mathcal{G}}\max_{\mathcal{D}} L_{cGAN}(\mathcal{G},\mathcal{D}) + \lambda L_{L1}(\mathcal{G}) \tag{2.8}$$

where L_{cGAN} is the same equation as 2.3 and $L_{L1}(\mathcal{G})$ is the L_1 distance defined as:

$$L_{L1}(\mathcal{G}) = \mathbb{E}_{x,y,z}[||y - \mathcal{G}(x,z)||_1] \tag{2.9}$$

The Pix2Pix GAN have seen success in several application, ranging from converting thermal images to color photographs to transforming hand sketches to images.

Another image-to-image translation GAN is the Cycle-Consistent Adverserial Network, also referred to as CycleGAN [21]. Usually when training a neural network for image-to-image translation a huge number of paired samples is required which can be expensive and also difficult to prepare. However, the CycleGAN uses a technique including not one but two generator and discriminators that does not require paired samples but is trained unsupervised using a set of input images that might not even be related to the output domain. In Figure 2.6, the architecture of the CycleGAN is illustrated, displaying the relationship between the parts of the model and the cycle-consistency.

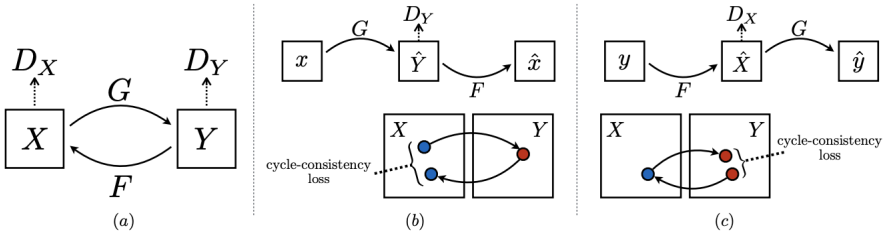


Figure 2.6: (a) Model containing two mapping functions G : X → Y and F : Y → X, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y , and vice versa for D_X and F. Two cycle consistency losses is introduced to regularize the mapping, capturing the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: x → G(x) → F(G(x)) ≈ x, and (c) backward cycle-consistency loss: y → F(y) → G(F(y)) ≈ y [21].

Defining the cycle-consistency loss as:

$$L_{cyc} = \mathbb{E}_{p_{data}(x)}[||F(G(x)) - x||_1] + \mathbb{E}_{p_{data}(y)}[||G(F(y)) - y||_1] \tag{2.10}$$

one can state the complete value function as:

$$L(G, F, D_x, D_y) = L_{GAN}(G, D_x, x, y) + L_{GAN}(F, D_y, x, y) + \lambda L_{cyc}(G, F) \tag{2.11}$$

where L_{GAN} is the original value function defined in Eq. 2.1 and λ controls the relative importance of the two objectives. As a result, the CycleGAN aims to solve:

$$\arg\min_{G,F} \max_{D_x, D_y} L(G, F, D_x, D_y) \tag{2.12}$$

The CycleGAN is an impressive and powerful framework that have achieved breathtaking results in numerous of application including translating images of horses to zebras and translating artistic style. An example of the artistic style translation can be seen in Figure 2.7 below.

2.4.6 BigGAN

In 2018 A. Brock et al. introduced BigGAN, a GAN designed to combine a set of the most successful practices that work generally for GANs to generate class-conditional images [22]. The BigGAN takes both points from the image class and the latent point and as the names suggests, scales up the batch size and uses more model parameters. As a result, the images generated end up having both high-quality and high-resolution. The framework have seen lots of success in several applications, ranging for generating high-resolution photographs of dogs to high-quality images of rocket launches.

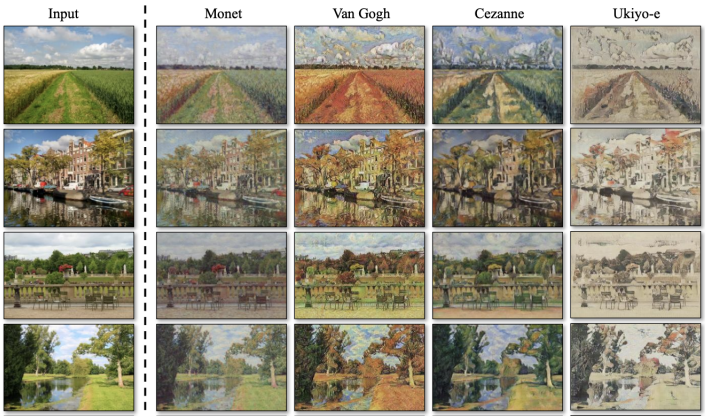


Figure 2.7: Examples of results from the CycleGAN transferring a photograph into styles of famous painters [21].

2.4.7 Progressive Growing GAN

As the name suggest, Progressive Growing GAN's architecture is starting with low resolution representation of the data and then progressively grows the size until the desired size is achieved. This is established by incrementally adding layers which form finer and finer detail during the training process. In the original paper "Progressive Growing of GANs for Improved Quality, Stability, and Variation" published in 2017 by T. Karras et al. [23], the training was both stabilized and speed-up, allowing them to generate super realistic high-resolution images. A set of these realistic images is displayed in Figure 2.8 below.


Figure 2.8: Examples of 1024 X 1024 images generated by the Progressive Growing GAN [23].

2.4.8 StyleGAN

In most research papers, the focus has been on improving the discriminator to be able to optimize the efficiency of the generator model. There are not many published papers where the improvement have been focused on the generator but in T. Karras et al. paper "A Style-Based Generator Architecture for Generative Adversarial Networks" [24] they propose extensive changes to the generator. They call the model StyleGAN and the proposed changes include a network mapping points between the latent space and an intermediate space, the usage of the intermediate latent space able to control

style of the output image as well a possibility to use noise to introduce variation in the generator. In Figure 2.9 the difference between a traditional generator and the Style-based generator is displayed, illustrating how much more advanced the latter is.

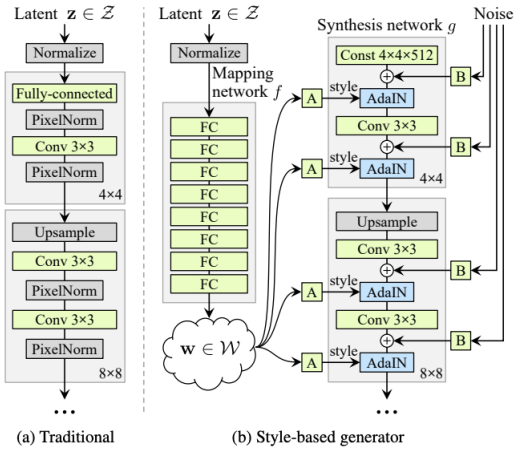


Figure 2.9: Example on how the architecture differs between a traditional and a Style-based generator [24].

The StyleGAN is capable of generating unprecedented super-realistic images of faces, cars and animals and is also able to control the style and detail of the images. A small taste of the powerful StyleGAN model can be seen in Figure 2.10 below.

3 Alternative models and techniques

In this section, several alternative models and techniques similar to the GAN is presented. Just like the GAN model, these models have their advantages and disadvantages and can be seen as the main competitors to the GAN framework.

3.1 Variational autoencoder

In D. P. Kingma and M. Welling's paper "Auto-Encoding Variational Bayes" [25] they present an autoencoder suited for a generative process. The so-called variational autoencoder (VAE) uses a regularised training pattern to assure that the latent space have desirable properties. Similarly to the regular autoencoder, the variational model consists of a encoder and a decoder which aims to find the minimum error between the input data and the encoded-decoded data. However, instead of encoding the input as an single input, it is encoded as a distribution over the latent space, providing the needed regularisation during training. VAEs have become one of the most prominent models used for unsupervised learning of complex distribution, seeing success in generation data such as house numbers, CIFAR images and physical models [26].

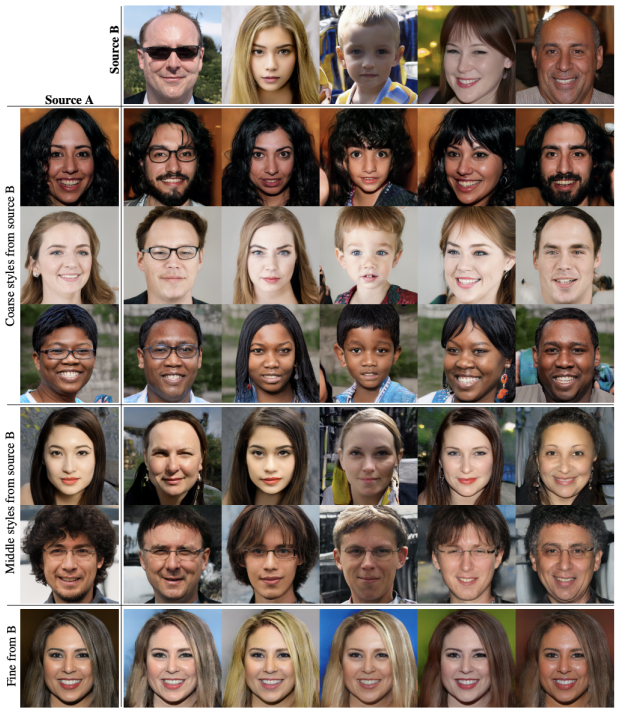


Figure 2.10: In this image there are two sets of images Source A and B generated by the StyleGAN. The rest of the the images are generated from the sources by taking a certain subset of style from one of the sources and the rest of the other one. Using styles from the coarse spatial resoultions $(2^4 - 2^8)$ changes high-level styles such as hairstyle, face shape and pose. Styles from the middle resolution $(2^{16} - 2^{32})$ corresponds smaller detail changes such as hair and facial and styles from the finest resolution $(64^2 - 1024^2)$ changes for instance micro structures and color schemes [24].

3.2 Flow-based generative model

The flow-based generative model is a machine learning model using normalizing flows that is used for generation of data. Normalizing flows is a method used in statistics that relies on the change-of-variable law of probabilities to transform simple distributions of data into complex ones. This straightforward likelihood modeling comes with pros such as that the negative log-likelihood function can be used as loss function since it can be computed directly. The flow-based generative model have seen success across several fields, including audio [27] and video generation [28].

3.3 Energy based model

The energy-based model, or in short EBM, is probabilistic generative model originating in statistical physics. The model is based on an energy function describing the probability of states and has since it first made an entrance back in the 80s emerged to a well developed and extended model. The EBM have been used in applications such as anomoly detection [29] and image modeling [30].

4 Implementation

In this section, implementation of the GAN framework will be demonstrated. The neural networks are created and evaluated using Python with Tensorflow and Keras and the code is openly available in the following Github repository: `https://github.com/DHjeelm/OpenProjectAboutGANs`

4.1 Generating handwritten digits with original GAN

The original GAN created to generate handwritten digits uses fully connected neural network architecture for both the discriminator and generator. Both the networks have three hidden layers with 256, 512 and 1024 neurons each but differ in the output layer. The size of the output layer of the generator is simply the resolution of the digits, in this case is 28 by 28 pixels. For the discriminator, the output layer has 1 neuron with a Sigmoid activation function in order to binary classify the likelihood of a sample being fake or real. The GAN can successfully generate handwritten and the result is displayed in Figure 4.1 below.

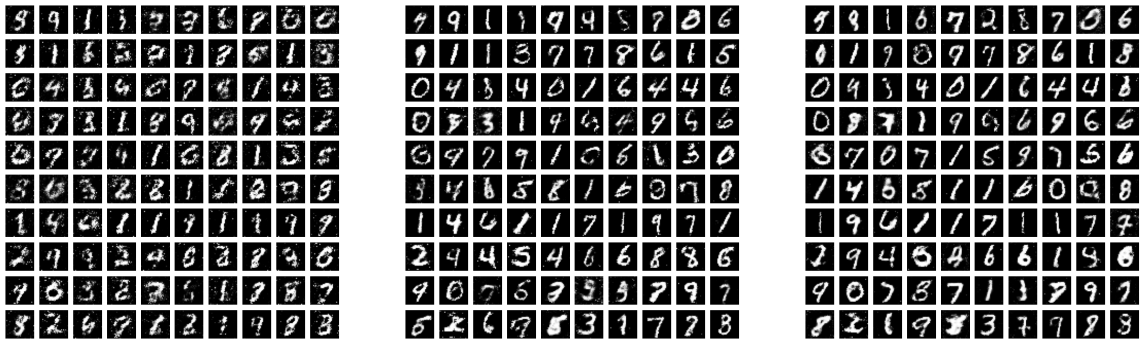


Figure 4.1: Images of generated handwritten digits after 1, 10 and 20 epochs of training.

4.2 Generating fashion images with DCGAN

The architecture of this model relies on the concepts of DCGANs presented in Section 2.4.3 to generate 28 by 28 pixel images of fashion items such as dresses, heels and sneakers. Both the networks in the GAN relies on for example 2D convolutional layers, upsamling and batch normalization which are essential for the generation of the images. An example of a set of generated images is displayed in Figure 4.2 below.

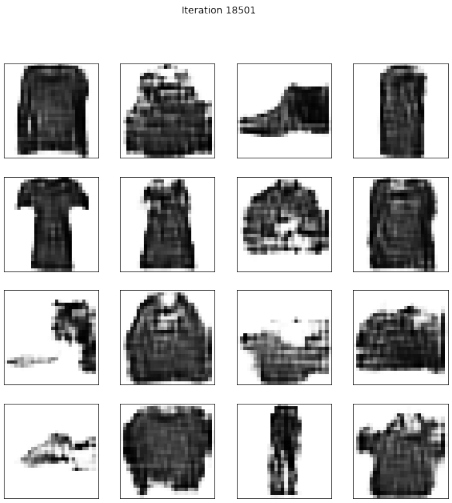


Figure 4.2: A set of fashion images generated by the DCGAN.

4.3 Generating fashion images with Variational Autoencoder

Just as the DCGAN, the Variational Autoencoder (VAE) can generate fasion garments. The VAE consist of an encoder bulit upon dense and convolutional layers which are connected through a sampling bottleneck layer to the decoder. A sample of generated images from the VAE is displayed in Figure 4.3 below:

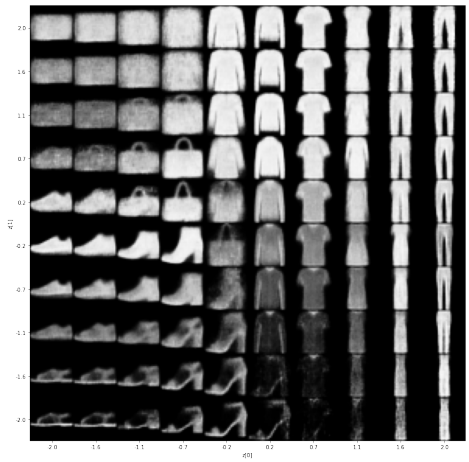


Figure 4.3: A set of fashion images generated by the VAE.

4.4 Conditionally generating fashion images with cGan

Coming back to the generation of fashion images from the previous sections, the cGan can be used to use a conditionally generate these images. As an example, the class label can be used as condition

for the model which results in the user having control on which type of fashion item is created. This is displayed in Figure 4.4 below where each column represent a different type of fashion garment.

Figure 4.4: A set of conditional generated fashion images where each column represent a different type of fashion garment.

4.5 Generating footprints of cosmic-ray induced airshowers using Wasserstein GAN

In M. Erdmann et al. paper "Generating and refining particle detector simulations using the Wasserstein distance in adversarial networks" [31] they try to generate footprints of cosmic-ray induced airshowers, as for instance measured by the Pierre Auger Observatory. In this case we simplify the experiment and without deep-diving into the physics behind the airshower, a random footprint from the provided dataset looks like this:

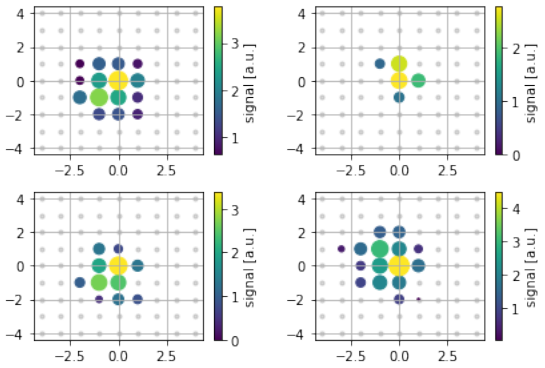


Figure 4.5: Example footprint of an airshower from the dataset.

Using the Wasserstein loss to create a WGAN together with a DCGAN architecture for both the generator and the discriminator, the model successfully generates new sample retaining the characteristic of the cosmic-ray induced airshowers as shown in Figure 4.6 below.

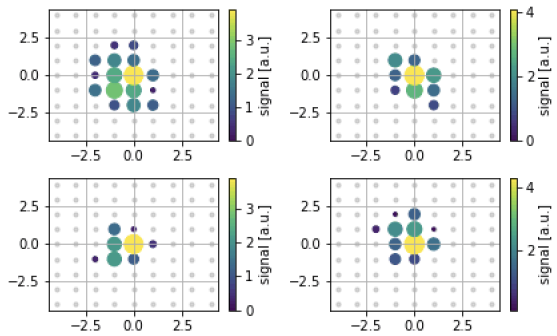


Figure 4.6: Example of generated airshower from the WGAN.

4.6 Image translation of satellite images to Google Maps using Pix2Pix GAN

Using the *maps* data set presented in P. Isola's et al. paper [19] and implementing the Pix2Pix GAN in Keras with the U-Net architecture for the generator and the patch focused discriminator, it is possible to translate satellite images into Google Maps images. To train the Pix2Pix is a demanding process, taking several hours to train for 10 epochs on a modern computer but converges slowly to giving the expected output. A plot of how the translated images looks like after 10 and 100 epochs is displayed in Figure 4.7 and 4.8 below.

Figure 4.7: Source, generated and expected image of the image translation from satellite image to Google Maps images using the Pix2Pix GAN trained for 10 epochs.

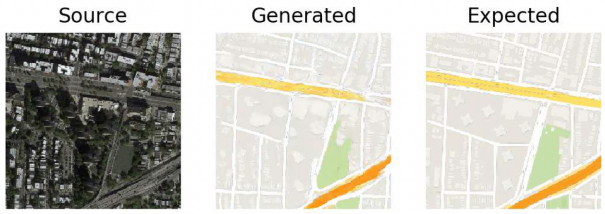


Figure 4.8: Source, generated and expected image of the image translation from satellite image to Google Maps images using the Pix2Pix GAN trained for 100 epochs [19].

4.7 Translate images of horses to zebras with CycleGAN

One of the interesting application of the CycleGAN is presented in P. Isola's et al. paper "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks" [21] was what they called object transfiguration. This was demonstrated in the paper as examples of transforming for instance apples to oranges and horses to zebras. The latter can be achieved by constructing a CycleGAN containing two deep convolutional neural networks as discriminators and a residual network architecture for the two generators. Due to the training taking too long to finish on a standard laptop, the resulting images can not be presented. However, an example of this transformation from horse to zebra taken from the same research paper mentioned above is displayed in Figure 4.9 below.

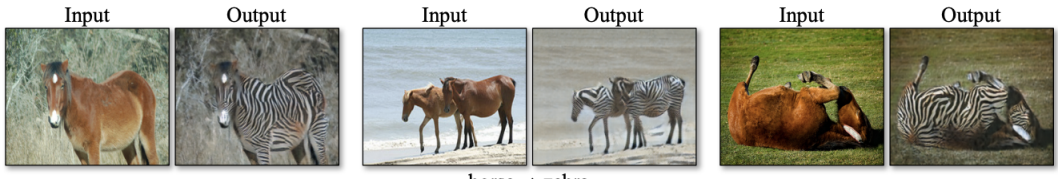


Figure 4.9: Example of horses translated to zebras using the CycleGAN [21].

5 Evaluation

When analyzing the previous and similar work together with the implementation of the GAN models, it is clear that is should be possible to generate realistic noise for the RNO-G. It is however not an easy task and requires a great deal of work with both the architecture of the GAN as well as the data. As of right now, the data used at DESY is not available since the supervisor of the project explained that there is a lot of physics subtleties in the data that one needs to correctly consider for training the GAN. As a result, it only makes sense to use the data when working at DESY in order to avoid trying to work with something that actually isn't super relevant for the project.

Although no actual data is available, it is possible to analyze and predict which of the GAN variants that should be considered for the DESY project. As discussed in Section 2.3, there are some challenges with the GAN framework which must be considered and in most cases handled to be able to train a GAN successfully. Hence, the models that consider these problems such as the DCGAN and Wasserstein, are more promising to work than for example the original GAN. The Wasserstein GAN have previously been used for noise generation in the Simon Hillman's Bachelor's theis "Simulating background noise of radio neutrino detectors with a Generative Adversarial Network (GAN)" [32]. Hillman's network could learn to generate some characteristic features of the noise signal and scored high in several metrics. The network was however not perfect and missed some essential features in the signal. Therefore, Hillman suggested to for instance increase number of training epochs, test more hyperparameters and implement more complex networks. Thus, a model combining the techniques in the DCGAN with the Wasserstein GAN framework could improve the result even further, hopefully being able to generate realistic noise signals.

The more advanced GAN variants, including for instance the StyleGAN and Pix2Pix GAN, does not seem to be relevant due to them being overly specific for certain type of data which in most cases

is image data. However, they present some interesting architectures and techniques which might be applicable for generating the noise data. Models that are more promising for implementation are the alternative models presented in Section 3. Although they have not explicitly been used for noise generation, they have proved to be successful in other context of data generation. Whether these models will be implemented on the given problem is yet to be decided by the supervisor but could otherwise be an interesting extension to the project.

6 Conclusion and further work

The purpose of this project was to familiarize and study the Generative Adversarial Network family. Based on the analysis conveyed, it can be concluded that the GAN family is a strong and powerful framework for generating data ranging from 1D functions to high-resolution and super realistic images. Whether it will be possible to generate realistic noise for the neturino detector is hard to answer but the networks, especially the Wasserstein GAN, proves promising results for a successful generation.

As for further work, the internship in DESY in the autumn is a clear continuation of this project and will start of where this projects ends. Moreover, it would be interesting in testing out and implement more GAN variants as there are countless of versions which all have specilazed for a specific application.

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[2] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, "Generative image inpainting with contextual attention," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[3] K. Schawinski, C. Zhang, H. Zhang, L. Fowler, and G. K. Santhanam, "Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit," *Monthly Notices of the Royal Astronomical Society: Letters*, p. slx008, Jan 2017. [Online]. Available: http://dx.doi.org/10.1093/mnrasl/slx008

[4] L. de Oliveira, M. Paganini, and B. Nachman, "Learning particle physics by example: Location-aware generative adversarial networks for physics synthesis," *Computing and Software for Big Science*, vol. 1, no. 1, Sep 2017. [Online]. Available: http://dx.doi.org/10.1007/s41781-017-0004-6

[5] K. Hora, "This x does not exist," Available at https://thisxdoesnotexist.com/.

[6] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, jan 2018. [Online]. Available: https://doi.org/10.1109%2Fmsp.2017.2765202

[7] M. Erdmann, J. Glombitza, G. Kasieczka, and U. Klemradt, *Deep Learning for Physics Research*. WORLD SCIENTIFIC, 2021. [Online]. Available: http://deeplearningphysics.org

[8] T. Ferguson, *A Course In Game Theory*. World Scientific Publishing Company, 2020. [Online]. Available: https://books.google.se/books?id=knr1DwAAQBAJ

[9] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," 2017. [Online]. Available: https://arxiv.org/abs/1701.04862

[10] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*. PMLR, 2017, pp. 214–223. [Online]. Available: https://arxiv.org/pdf/1701.07875.pdf

[11] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled generative adversarial networks," 2016. [Online]. Available: https://arxiv.org/abs/1611.02163

[12] J. Brownlee, *Generative Adversarial Networks with Python: Deep Learning Generative Models for Image Synthesis and Image Translation*. Machine Learning Mastery, 2019. [Online]. Available: https://books.google.se/books?id=YBimDwAAQBAJ

[13] K. Roth, A. Lucchi, S. Nowozin, and T. Hofmann, "Stabilizing training of generative adversarial networks through regularization," 2017. [Online]. Available: https://arxiv.org/abs/1705.09367

[14] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[15] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[16] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015. [Online]. Available: https://arxiv.org/abs/1511.06434

[17] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014. [Online]. Available: https://arxiv.org/abs/1411.1784

[18] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," 2016. [Online]. Available: https://arxiv.org/abs/1606.03657

[19] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," 2016. [Online]. Available: https://arxiv.org/abs/1611.07004

[20] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015. [Online]. Available: https://arxiv.org/abs/1505.04597

[21] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," 2017. [Online]. Available: https://arxiv.org/abs/1703.10593

[22] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," 2018. [Online]. Available: https://arxiv.org/abs/1809.11096

[23] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," 2017. [Online]. Available: https://arxiv.org/abs/1710.10196

[24] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," 2018. [Online]. Available: https://arxiv.org/abs/1812.04948

[25] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013. [Online]. Available: https://arxiv.org/abs/1312.6114

[26] C. Doersch, "Tutorial on variational autoencoders," 2016. [Online]. Available: https://arxiv.org/abs/1606.05908

[27] W. Ping, K. Peng, K. Zhao, and Z. Song, "Waveflow: A compact flow-based model for raw audio," 2019. [Online]. Available: https://arxiv.org/abs/1912.01219

[28] M. Kumar, M. Babaeizadeh, D. Erhan, C. Finn, S. Levine, L. Dinh, and D. Kingma, "Videoflow: A conditional flow-based model for stochastic video generation," 2019. [Online]. Available: https://arxiv.org/abs/1903.01434

[29] S. Zhai, Y. Cheng, W. Lu, and Z. Zhang, "Deep structured energy based models for anomaly detection," 2016. [Online]. Available: https://arxiv.org/abs/1605.07717

[30] Y. Du and I. Mordatch, "Implicit generation and generalization in energy-based models," 2019. [Online]. Available: https://arxiv.org/abs/1903.08689

[31] M. Erdmann, L. Geiger, J. Glombitza, and D. Schmidt, "Generating and refining particle detector simulations using the wasserstein distance in adversarial networks," 2018. [Online]. Available: https://arxiv.org/abs/1802.03325

[32] S. Hillman, "Simulating background noise of radio neutrino detectors with a generative adversarial network (gan)," 2021.