Generating Calorimetry Signals at the LHCb with Generative Adversarial Networks

The efficacy of generative adversarial networks
in accelerating particle physics simulations

By
Thomas Neuer

University of Zurich[UZH]

Physik-Institut
University of Zurich
Switzerland

A thesis presented for the degree of
Master of Physics MSc.

November 2020

Supervisor:
Prof. Dr. Nicola Serra
Dr. Patrick Owen
Davide Lancierini

Abstract

Today modern particle physics relies heavily on simulation in order to compare new theories with experiments. The most computationally intensive step in the Monte Carlo simulation process is the approximation of particle interactions with the material of the hadronic and electronic calorimeters using the underlying scattering theory. These methods may have problems keeping up with the increased interaction rate of the future High Luminosity Large Hadron Collider (HL-LHC). In this thesis we propose a two step approach for simulating the calorimeter response for full events relying on Generative Adversarial Networks (GAN). Firstly, we approach the full event generation by solving the simpler problem of single track calorimeter response reconstruction with the interaction of single π^+ mesons with the detector material. Secondly, we use the previously trained GAN to produce approximate images of a full event by overlapping track-by-track images. We use these images as input for a second image-to-image translation GAN refining the output and adjusting for imprecise simulations in the first step. We compare this approach with the current state of the art method using Geant4 as well as training a deep convolutional generative adversarial network in a one step approach directly on a 2D representation of the tracker information. This project used the $B \to (D \to K\pi\pi)\mu\nu$ particle interaction as it is currently of great importance at the LHCb hinting at new physics. However, we aimed to keep this work as general as possible to make it applicable to a vast array of problems and other decays. This approach allows for faster full event generation for the future HL-LHC while using fewer computational resources and enables a viable alternative to current state of the art simulation techniques.

Dedication and Acknowledgments

Firstly I would like to thank Prof. Nicola Serra, for making it possible to work on such an exciting and novel approach while providing creative and useful inputs on the methodology along the way. I'd like to thank Davide Lancierini as my direct supervisor for being available at every step of my thesis, providing invaluable guidance on generative adversarial networks, usage of advanced computing resources and the underlying physics necessary to accomplish my goal. Next I want to thank Dr. Patrick Owen and Jonas Eschle, who were always happy to discuss any machine learning techniques and clear up questions concerning the data used in this thesis. Lastly, I want to thank my parents for their consistent support over the last years and for helping me achieve my goals.

Contents

List of Figures

1 Introduction

Event simulation is one the cornerstones of modern particle physics and it is required to compare data from particle accelerator experiments with existing theories gaining new insights into the most fundamental building blocks of our universe. In this thesis I will focus on one particular step of the simulation process.

Even though no deep knowledge of particle physics is required to understand the main conclusions of this project I will describe the Standard Model of Particle physics in the following part. Essential are the next sections explaining the detector of the LHCb experiment at the LHC, especially the physical properties of the hadronic calorimeter (HCAL). These detector specifications determine how the data has been processed before and after using the generative adversarial networks for HCAL response generation. At the end of this chapter I will describe the current state of the art simulation techniques, like Geant4, and their limitations. I will discuss how my approach tries to overcome these limitations via a two step approach translating single tracks to images and using these images to train a second model to simulate the calorimeter response.

In the following chapter the data is described: First a data set containing only single π^+ tracks generated by the Geant4 class `ParticleGun` is described. Both the output of the Geant4 track reconstruction, containing information about the projected x and y coordinates as well as the transverse energy E_T, and the calorimeter simulation are discussed in detail. It is important to deal with the differing LHCb calorimeter granularity in an appropriate way in order to be able to generate single consistent images with the generative adversarial networks.

In the next two chapters I will describe the theory, implementation, advantages and limitations of generative adversarial networks for image generation, detailing which loss functions are appropriate for my task and how they compare with other techniques. The general architectures for the single track propagation and both calorimeter image refinement and tracker image propagation are presented, including specific methods to deal with the failing convergene of the Nash equilibrium and mode collapse.

In the next chapter the results of this thesis are discussed and the images after the first step of translating single tracks to detector images training on a simplified data set containing only π_+ mesons are discussed. Major pitfalls and difficulties during training are presented and the eval-

uation process of the generated images is detailed as no clear and undisputed metric is available. This analysis is repeated for the second step of the project comparing the generated images from a full Geant4 event reconstruction to the ones generated by the final GAN. In a last comparison the usefulness of using two separate generative adversarial networks instead of one is discussed. This direct GAN will translate a 2D representation of the tracker information directly into the calorimeter response without being trained on single track examples.
In the last section the work is summarized in a final conclusion, discussing insights gained during this project, limitations of this work and future goals for using generative adversarial networks for calorimeter simulations, as this work covers by no means all available methods.

1.1 The Standard Model

The Standard Model (SM) of particle physics is a collection of quantum field theories, using local gauge invariance as the fundamental axiom. It describes all known forces, excluding gravity to date, and explains the interaction, creation and propagation of matter via excited discretized fields in space. The dynamics of a system are fully determined by the Lagrangian in eq. (1.1.1) and the principle of least action[1].

$$\mathcal{L}_{SM} = \mathcal{L}_{fermion} + \mathcal{L}_{gauge} + \mathcal{L}_{Yukawa} + \mathcal{L}_{Higgs}, \tag{1.1.1}$$

where

- $\mathcal{L}_{fermion}$: kinetic term for fermions
- $\mathcal{L}_{gauge}$: kinetic term for gauge bosons
- $\mathcal{L}_{Yukawa}$: mass terms for fermions
- $\mathcal{L}_{Higgs}$: Higgs term

A list of all particles contained in the Standard Model is shown in figure 1.1.1. Note that each of the quarks is required to have three identical copies distinguishable only by the charge of the strong force called color, mediated by the gluon. Additionally every particle has a corresponding anti-particle differing in the sign of all quantum numbers. The final ingredient of the Standard Model was discovered experimentally at the LHC in 2012 with the scalar Higgs boson[2], which is a crucial prediction of the Higgs theory. It describes the process of spontaneous symmetry breaking of the SU(2) vacuum state, needed to obtain a consistent description of the massive fermions and vector bosons ($W^{\pm}, Z$), while keeping the gluons and photon massless[3].
So far the Standard model has withstood every experimental test to an unprecedented precision[4]. However, this does not mean that the Standard Model is complete or infallible. On the contrary, there are major physical phenomenon which are not yet incorporated in a concise manner within the Standard Model, like the existence of Dark Matter and Dark Energy[5][6] as well as discrepancies concerning some predictions of the equally successful general theory of relativity[7].

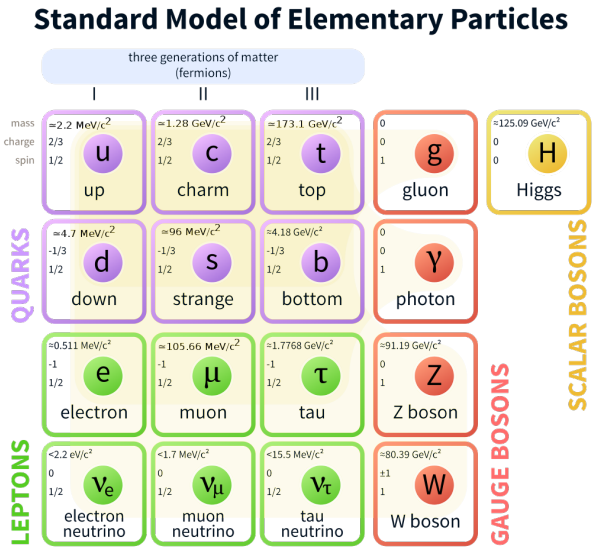


Figure 1.1.1: The Standard Model of particle physics.

1.2 LHC(b)

The Large Hadron Collider (LHC) is a high-energy particle accelerator at CERN in Geneva. The ring-tunnel of the proton-proton synchrotron is situated more than 100 m below the ground. It has a circumference of more than 27 km and includes more than 9500 liquid-helium-cooled superconducting magnets[1]. Figure 1.2.1 shows a sketch of the LHC accelerator chain. As two equally charged particles are collided, the LHC houses two beam pipes, which are kept at an ultra-high vacuum of the order 10^{-10} mbar. Particle beams consist of a few thousand particle bunches, each containing about 10^{11} protons. The center-of-mass energy ($\sqrt{s}$) at the LHC started in 2010 at 900 GeV, but has been increased gradually to the final design energy of $\sqrt{s} = 14$ TeV. The protons are collided at four collision points within the accelerator pipes at a rate of 40 MHz, one collision every 25 ns. The results of the interactions are measured by four big experiments: The two largest detectors, called CMS (Compact Muon Solenoid) and ATLAS (A Toroidal LHC Apparatus), are designed to operate at the energy frontier of particle physics. They validate new physics theories by trying to create real heavy resonances in hard processes with extremely high transverse momentum. Therefore these experiments were designed with a cylindrical symmetry around the interaction point to cover the whole spectrum of relevant particle creations. ALICE (A Large Ion Collider Experiment) has another goal in mind: the study of heavy-ion collisions in order to produce a quark-gluon plasma. Similar conditions are believed to have existed a fraction of a second after the Big Bang before quarks and gluons bound together to form hadrons and heavier particles.

The Large Hadron Collider beauty (LHCb) experiment is the fourth large detector at the LHC and it is mainly concerned with B-meson physics, which is the study of particles consisting of an bottom quark (also known as beauty quark) / anti-bottom quark pair. Due to the large mass of

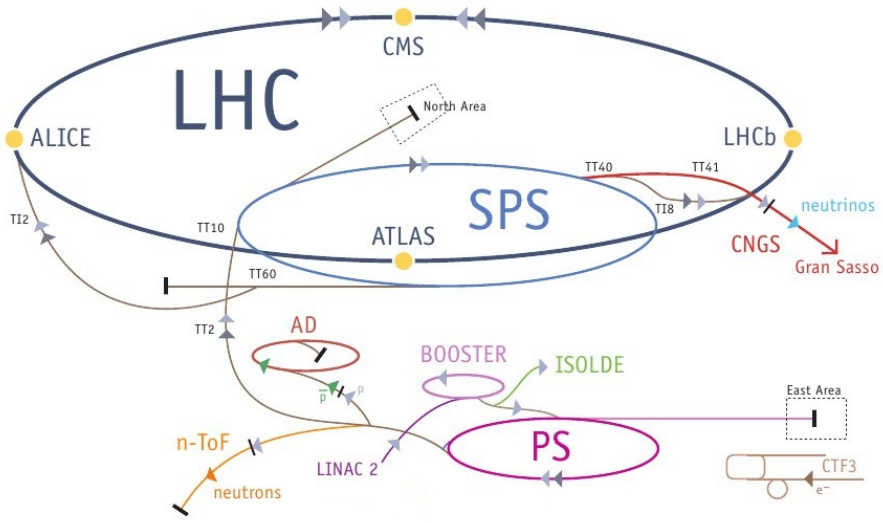
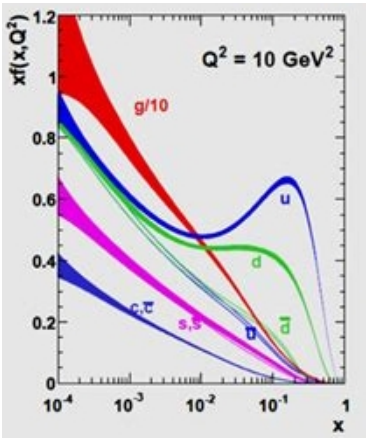
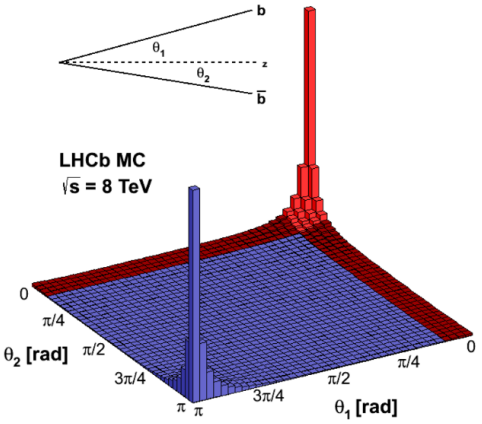


Figure 1.2.1: The CERN Accelarator chain.

the bottom quark a high energetic collision with at least one high energetic parton inside of the proton is needed. As displayed in figure 1.2.2a at these energies the valence quark is likely to



(a) Parton distribution as predicted by the Standard Model[8].



(b) $b\bar{b}$ pair production cross section at $\sqrt{s} = 7\,\text{TeV}$. In red the angular coverage of the LHCb detector is shown and the θ_i are the angles from the beam axis[9].

collide with a low energetic sea quark resulting in an extremely boosted reference frame of the created particles. Therefore the cross-section of the $b\bar{b}$ pair production peaks in both forward and backward direction of the beam axis, see figure 1.2.2b.
Therefore, the LHCb experiment (unlike the other three experiments) was not designed with a cylindrical symmetry around the interaction point, as can be seen in figure 1.2.3. Instead, all resources were focused to build a highly sophisticated detector along one direction of the beam

axis.

Unlike ATLAS and CMS, the LHCb aims to discover physics beyond the Standard Model by investigating effects at the order of suppressed loop level diagrams where effects from new physics become of comparable size to Standard Model predictions. However, due to the suppressed nature of these loop level diagrams a high intensity of primary interactions is required in order to detect enough events for a reliable measurement of the predicted effects. Therefore, the LHCb is said to operate at the intensity frontier in order to discover physics beyond the Standard Model. Its angular coverage in θ is approximately 30 to 300 (250) mrad in the bending (non-bending)

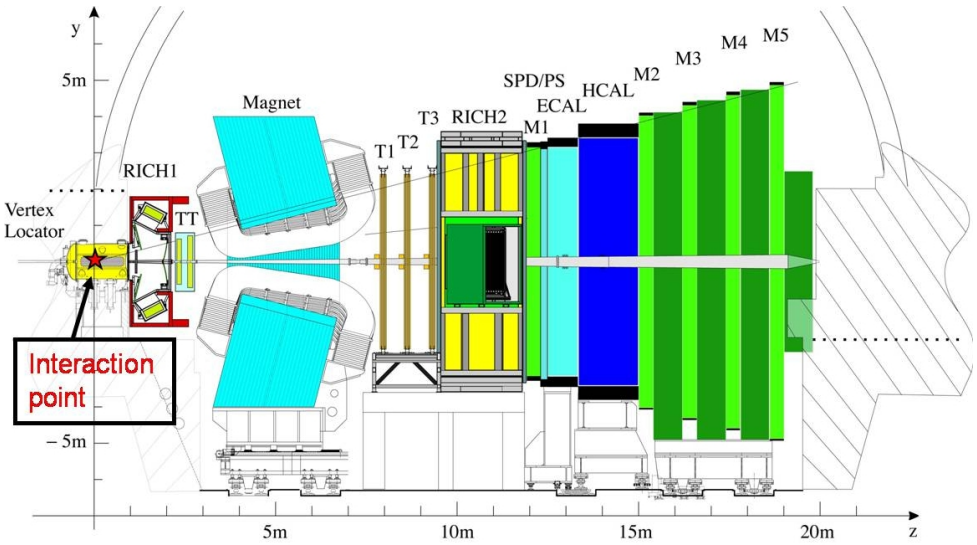


Figure 1.2.3: Schematic side view of the LHCb experiment. Collisions take place at the interaction point and then are measured by the one sided detector arm[9].

plane[9], corresponding to a pseudo-rapidity η between 2 and 4.5, where

$$\eta = -\log\left[\tan\left(\frac{\theta}{2}\right)\right]. \tag{1.2.1}$$

The particles produced at the interaction point in forward direction traverse several detector layers. These detectors measure different properties of the particles, usually with multiple redundancy[9]. The detector consists of a precise tracking system (vertex locator (VELO), Tracker Turicensis (TT), Magnet and straw tube detectors T1-T3), measuring the trajectory and hence momentum component transverse to the magnetic field as well as the charge of the particles. Several types of charged hadrons and leptons are differentiated via a measurement in the two Ring-imaging Cherenkov detectors (RICH 1/2). Candidates for hadrons and charged leptons as well as photons are differentiated using a system of pre-shower (PS) and silicon-pad detectors (SPD), electromagnetic calorimeter (ECAL) and hadronic calorimeter (HCAL). Muons can traverse all calorimeters and are detected in the Muon system (M1 in coincidence with M2-M4).

Of major concern for this project is HCAL which will therefore be described in detail in the

following section. Other important detector components are summarized as well to give a full and self-contained description of the LCHb system.

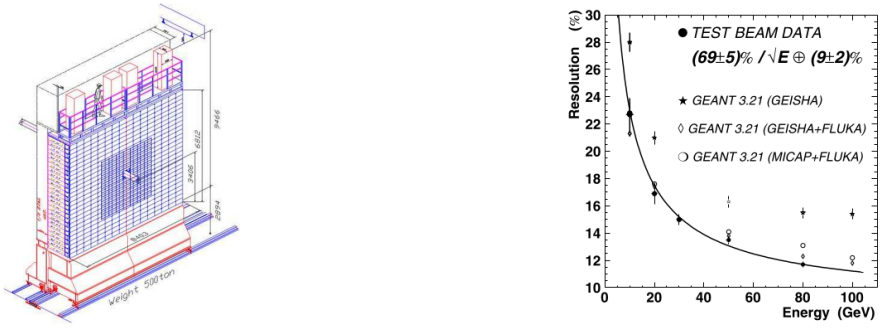
1.2.1 Detector components

Hadronic calorimeter

The hadronic calorimeter at the LHCb experiment is described in great detail in [9] and [10]. It is a sampling iron-scintillator of 5.6 λ_I thickness. It was designed to be part of the L0 hardware trigger by measuring high energy hadronic particles as those are a mark of B-decays. Due to this design requirement it must keep up with the 40 MHz bunch-crossing time at the LHC without dead time. However, an upside is the moderate requirement for the energy resolution for triggering purposes, which is illustrated in figure 1.2.4b and given by

$$\frac{\sigma_E}{E} = \frac{(69 \pm 5)\%}{\sqrt{E}} \oplus (9 \pm 2)\%. \tag{1.2.2}$$

It is located at a distance of 13.33 m from the interaction point and has lateral dimensions of 8.4x6.8 m^2 with a depth of 122 cm. It is shown in figure 1.2.4a. Of great importance for the



(a) The LHCb hadronic calorimeter general view[10]. (b) The LHCb hadronic calorimeter resolution[10].

following work is the split cell granularity in the inner and outer part of the detector which has to be dealt with in image preparation for machine learning algorithms. The resolution in the inner detector is doubled compared to its outer counterpart with cells in the inner detector having dimensions of 131x131 mm^2 while outer cells have 262X262 mm^2. The detector contains 1488 cells, 880 in the inner detector and 608 in the outer one. Note that the inner detector has an opening for the beam pipe so it does not cover all of the interior space. This higher granularity in the inner detector is required because $b\bar{b}$ pair production happens most likely in highly boosted reference frames producing numerous particles close to the beam pipe. In order to be able to differentiate between these tracks a higher resolution is required.

Vertex locator

The Vertex Locator (VELO) of the LHCb experiment consists of a silicon strip detector surrounding the proton-proton collision point[11]. It is designed to precisely measure the particles

trajectories immediately after being produced and determine the vertex they originated from. It consists of 21 modules, each with two silicon microstrip sensors. During data taking, the inner edge of detector is only 8 mm away from the beam, while it is further away during injection of the beams.

Muon system

The muon stations M1-M5 serve the purpose of detecting high transverse momentum particles including a measurement of the particles position[12]. M1 is stationed before the calorimeter system for a better transverse momentum (p_T) resolution while the other stations are in the outer regions of the LHCb. Mostly muons are able to penetrate through the electromagnetic and hadronic calorimeter as they are minimally ionizing particles (MIPs) over a wide range of momenta. Therefore they leave the cleanest signal of all elementary particles in the detectors. The spatial resolution is diminished the further away the detector is positioned from the interaction point, because more area can be covered with a detector of lower granularity.

Magnet

The dipole magnet consists of two saddle shaped aluminium coils inside an iron yoke and is operated at environment temperature. The field created by the magnet has its main component along the y-axis. In the z-direction the field has a large gradient, resulting in a field of less than 2 mT inside the RICH envelopes but an overall bending power of 4 Tm. The magnet deflects charged particles in the positive or negative x-direction (coordinate system as defined in figure 1.2.3) allowing the determination of the charge of a passing particle based on the observed curvature. To minimize systematic effects, the polarity of the magnetic field can be reversed. Data taken with the two different polarities are assigned to the "magnet-up" or the "magnet-down" data set, depending on the direction of the field's y-component.

Trigger

The Trigger is not a specific part of the LHCb detector but has a hardware and multiple software stages. The hardware trigger is called level zero trigger (L0) and it reduces the rate of events from 40 MHz to about 1 MHz[13]. It evaluates information from the calorimeters and Muon system. The trigger fires if one or more pre-defined conditions, called trigger lines, are satisfied. At this rate the whole detector information can be read-out and used in the software stage. If at least two muons in a collision pass the trigger conditions, the event is saved and further processed.
In the next step, the software trigger, called High Level Trigger (HLT), implements at first a partial event reconstruction in order to reduce the rate to 43 kHz (HLT1)[14].

1.3 Current simulation techniques

Simulation is among the cornerstones of modern particle physics. It is used along every step of the way to the discovery of new physics. It encompasses event generation via particle interaction, propagation through the detector material, track reconstruction and more. The first of these tasks is used to simulate new particle physics theories which can be compared to real data taken from the experiments at the LHC or other particle accelerators. If there is a notable difference between the distributions of the two which can not be explained away the theory is deemed to be wrong or at least incomplete and has to be adjusted or even discarded altogether.
As previously stated, the Standard model is the most precise model to date in physics. Therefore, new physics has to be at least compatible with the Standard Model and we expect new effects either at higher energy scales or as corrections to suppressed loop-level diagrams. In order to detect these small discrepancies a high precision on the measurements of potential new effects is needed both for data from experiments and for simulations applying theory to generate predictions for the investigated particle interactions. This is especially true at the LHCb investigating rare decays. To reduce the data error an upgrade to the LHC, called the High Luminosity LHC (HL-LHC) is planned and it should be able to increase the luminosity by a power of ten[15]. In order to increase the number of generated simulations we either could increase the amount of computing resources assigned to this process or we need to find new ways to increase the simulation efficiency at least for certain parts of the event generation and reconstruction chain. The current state of the art technique for simulating particle propagation through the detector material is performed in Geant4. Currently, about 50-70% of LHC's worldwide computing power is used for full detector simulations corresponding to billions of CPU hours[16]. The most computationally demanding task of the whole simulation process is the propagation of the particle showers in the (hadronic) calorimeter and it can take seconds or even minutes per event on modern machines[17][18]. Therefore the current approach is not feasible after the high luminosity upgrade in order to keep the simulation error far below the data error. Therefore this work aims to develop a novel algorithm to reduce the amount of time needed to generate the calorimeter response in the Geant4 event reconstruction.

1.3.1 GEANT4

Geant4[19] is the current state of the art particle physics simulation program using Monte Carlo methods to model the interaction of particles with material. It is the sophisticated, flexible and highly configurable standard for detailed full detector reconstruction. It allows the user to configure the geometry, material and shape of a detector while emulating most of the physical phenomenon seen in real detectors. It still is one of the most cited papers in all of nuclear physics and the second most cited paper by all of CERN[20]. It encompasses digitization, hit management, visualization and the modeling of specific detector efficiencies with a stunning width and depth of utility.

Geant4 reconstructs events via the propagation of particles through material in small iterative

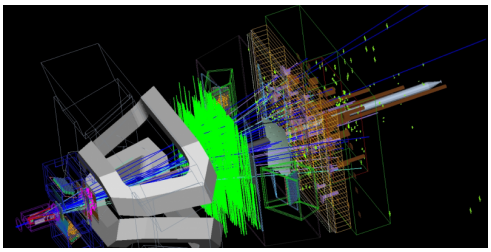


Figure 1.3.1: Exemplary output of a Geant4 reconstructed event[21].

steps calculating probabilistic distributions over common physical phenomenon in detectors like pair production, bremsstrahlung, multiple scattering, etc. This process naturally gets slower for more complex events where the need for the propagation for dozens of particles arises, many of which cascade into a shower of multiple particles in the calorimeter, called jets. All these particles tracks are then propagated and monitored leading to an exponential increase in training time until they fall below a certain energy threshold.
Even though Geant4 provides a great amount of utility by producing track information along every step of its calculation for later reconstruction purposes, this detailed approach becomes unfeasible in rare event and high luminosity particle physics where millions of events are needed to reach the desired precision of modern experiments. This drawback motivated this project and aims to remedy it.

1.4 Using GANs

This project is concerned with the substitution of the Geant4 calorimeter propagation step involving dozens of particles contained in the resulting jets with an approximate method simulating the calorimeter response directly without going through intermediate shower steps. This should drastically reduce the amount of time and computing resources needed in the simulation process as the calorimeter simulation done by Geant4 is the largest sub-process of the full event reconstruction. With the increasing demand generated by the HL-LHC and the advent of fast and efficient generation algorithms it has now become necessary for this approach to be tested in practice.
The currently most popular algorithm for data generation is the generative adversarial network proposed in 2014 by Ian Goodfellow[22]. Since then research focused on the improvement and extension of the original algorithm has moved forward at an expeditious pace with new architectures being proposed frequently. The necessary theory and architectures will be described in chapters 3 and 4.
Since its popularization over the last five years the world of particle physics, especially the domain of simulation, has not remained oblivious to this new development and multiple research papers have been published using generative adversarial networks for calorimeter response sim-

ulation of single tracks[23–27] and particle jets[28] or for the generation of events in the form of four-vectors[29–31]. It was also successfully used in other HEP domains[32–35].
I propose in this work the reconstruction of a full event using a two step approach: First we reproduce the work of the previously mentioned papers by implementing our own generative adversarial networks transforming tracking information from single particles into a 2D calorimeter representation. A data set generated by ParticleGun (Geant4) is used containing π^+ track information and the corresponding calorimeter image. After fitting this model to a reasonable degree the GAN is used to generate a single image from a full event by overlapping images from single tracks and slightly processing them in a deterministic way. We utilize data generated by Geant4 for the particle decay $B \to (D \to K\pi\pi)\mu\nu$ which is of special interest at the LHCb[36]. We expect these images to be of rather low quality mainly due to three reasons:

- No information about particle identification may be used as the calorimeter is part of the L0 trigger. Hence we cannot fit a (separate) model for each particle type but use only π^+ in the first step. So this network only has the chance to learn the true distribution for these kinds of particles. In the full event other particles will be present as well. We chose π^+ as the default as they are the most numerous particles measured in the HCAL.

- Some charged particles will not leave a signal in the hadronic calorimeter even though they are measured by the tracker. Mainly this concerns muons which in most cases penetrate the ECAL as well as the HCAL. Our models, however, can not use information about particle identification.

- Some neutral particles won't leave information in the tracking system but are visible in the HCAL. This mainly concerns neutrons which have to be added in the second refinement step as the first network only can generate a calorimeter response using tracker input.

We fit the model in such a way that data could later be used to train this second approach. This is important as we know that Geant4 is not a perfect simulation technique and has its own drawbacks. Even though this is not the scope of this work, the second step could be used training only on real data and reproduce the detector efficiency for particle interactions at the LHCb detector better than Geant4 which often requires reweighing with control channels.
For comparison we introduce another approach to the problem generating the calorimeter image for the full event directly from a 2D representation of the tracker information without using two separate networks trained on two different data sets.
So in total this thesis discusses three generative adversarial network approaches:

1. `Single track propagation` (STP): A single vector containing tracker information provided by Geant4 is translated into a calorimeter image.

2. `Calorimeter image refinement` (CIR): An approximation to the calorimeter image is generated with the SiTT and it is refined to match the final full event calorimeter image.

3. `Tracker image propagation` (TIP): The information about the tracks in the full event is condensed into a single image which is then translated to the final full event calorimeter image.

2 Data sets

2.1 Single track π^+ data set

The data set for the first part was generated by the Geant4 `ParticleGun` class and was made available as a root tuple. It consisted of over 100 branches describing many of the important properties which can be measured by the LHCb detector for the signal particles as well as the underlying event. This again demonstrates the utility and verbosity provided by Geant4. This work aims to be applicable to full event data in an efficient way using Geant4 tracker reconstruction information and the resulting algorithm should be able to work as part of the L0 trigger as the real calorimeter does. Therefore no information about particle identification is used. By respecting these constraints we decided to take three branches as input for the first network translating tracker information into a first calorimeter image for a single track:

- `x_projections`: The x coordinate as projected using information from tracking stations just before the impact into the calorimeter.
- `y_projections`: The y coordinate as projected using information from tracking stations just before the impact into the calorimeter.
- `real_ET`: The transverse energy of the particle calculated from tracker information using its total energy and its pseudo-rapidity.

The branch for the transverse energy instead of the total energy is used because high transverse energy tracks are an indication of B-meson physics. Therefore the L0 trigger fires when a certain threshold of transverse energy is exceeded, see section 5.1.1 for more details. Hence, it is an important quantity in the simulated response and transverse energy is used throughout this thesis.
The tracker information can conveniently be visualized as a 2D image via conversion of the x and y coordinates into grid indices to match the pixel-calorimeter detector at the LHCb. The grid limits are determined by the geometry of the calorimeter as the resulting 2D image represents the pion directly before its impact into the calorimeter. By design, the outer section of the

calorimeter consists of 26x32 pixels, which has to be doubled to 52x64 pixels to match the inner calorimeters granularity as described further down this text. See the left and middle part of figure 2.1.1 for a visual representation. Note that algorithms exist to translate tracker images

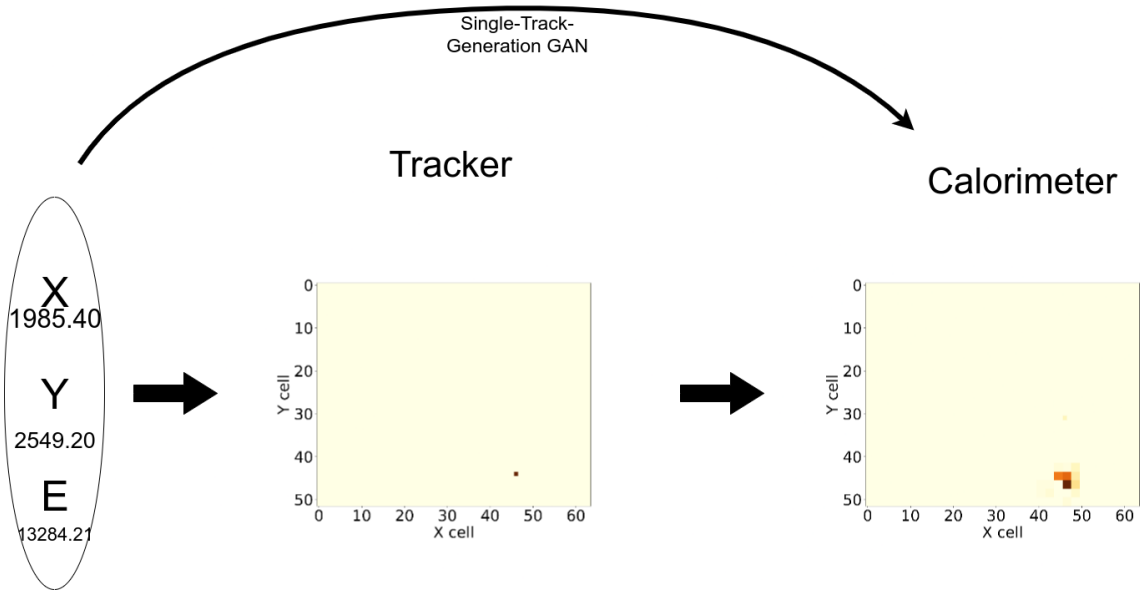


Figure 2.1.1: Conversion of the tracker information into a tracker image. On the right hand side is the calorimeter image as generated by Geant4, which needs to be replicated by our algorithm.

directly into calorimeter images. Such algorithms are used in the second step of this work as they are suitable for image-to-image translation. However, we refrained from using the approach at this point because they consist of zeros except for one pixel storing all the information of the image. We don't expect a lot of spatial correlation in these "images", so we decided to use a more efficient non-convolutional network for this part of the project and introduce algorithms respecting spatial correlation at a later point.
In the following two sections the preprocessing of the tracker and calorimeter data is described.

Tracker

Neural networks (including generative adversarial networks) are not scale invariant, meaning that the scale of the input is relevant and influences the quality of the results. While there is no clear recipe to find the perfect functions for preprocessing the data a safe method is the standardization of the values. This means subtracting the mean μ_i of all values x_i of branch i, called feature in machine learning, and dividing by it's standard deviation σ_i, see equation 2.1.1. The advantage of this method is that it leaves the shape of a distribution of values invariant, cancels out the units the feature might have and brings all features on the same scale by representing a value as its numbers of standard deviations from its mean.

$$\tilde{\boldsymbol{x}}_i = \frac{\boldsymbol{x}_i - \mu_i}{\sigma_i} \tag{2.1.1}$$

Both the `x_projections` as well as the `y_projections` were preprocessed in this way. While the same could have been done for the `real_ET` it was divided by the maximum possible value measured in the calorimeter, 6.120 GeV corresponding to the saturation limit in the silicon cells of the calorimeter. The same scaling is done later for the calorimeter response, so that both the tracker input energy as well as the calorimeter response are on the same scale. In theory, the energy measured in the tracker and calorimeter should match closely. However, both LHCb components measure the energy with some error. As the calorimeter is part of the L0 trigger one of its main requirements is to keep up with the collision frequency of 40 MHz. Therefore a lower resolution was accepted and tracker energies may vary greatly from calorimeter energies. The distribution of original features for a subset of the data is shown in figure 2.1.2 with their relevant statistical properties of the distributions.

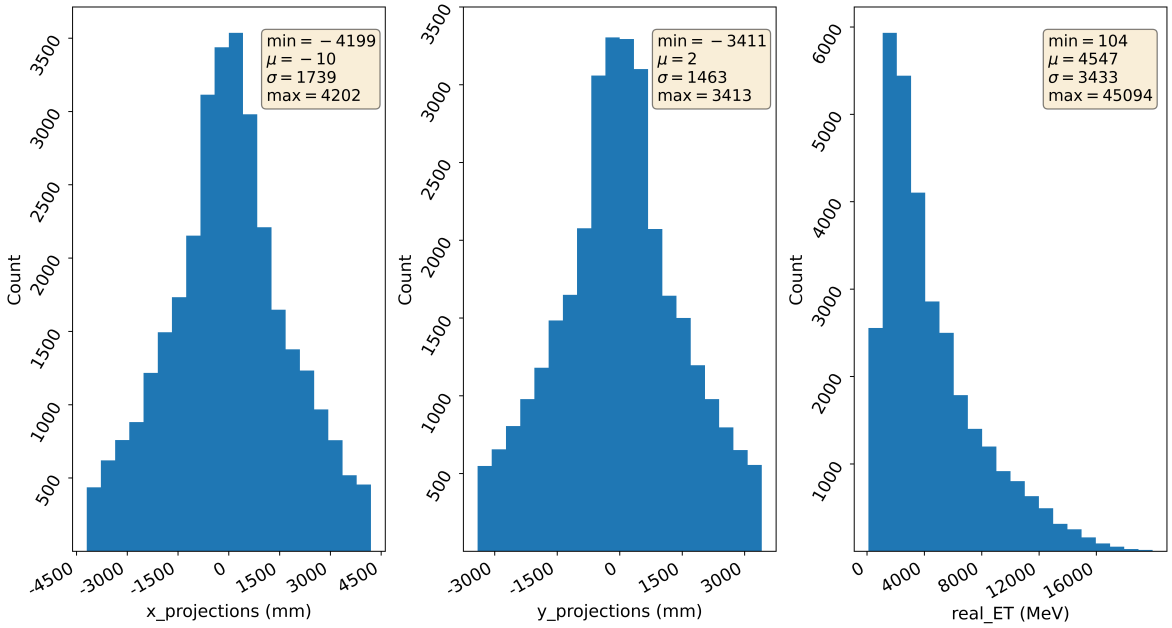


Figure 2.1.2: Distribution of input features for the first generative adversarial network.

Calorimeter

The target calorimeter images created by Geant4 are preprocessed in a way that is largely determined by the physical setup of the LHCb detector. Due to the boosted nature of the $b\bar{b}$ pair production process the decay products enclose a relatively small angle with respect to the beam pipe. Therefore a higher granularity close to the beam pipe is desirable. As described in section 1.2.1 the calorimeter essentially is split into an inner and outer part. The inner part has twice the granularity compared to the outer one meaning that the individual cells in the inner tracker have a cross-sectional area which is only one forth of the outer ones. This allows for a more precise measurement close to the beam pipe area where the more interesting particles pass by.

In this work the calorimeter is considered to be a two dimensional entity measuring only one value, the transverse energy of a particle. Therefore it can be represented as a two dimensional image consisting of N_yxN_x pixels representing the energy deposit at a certain coordinate point on the detector surface. From the Geant4 output we have two images per event, one from the outer calorimeter with lower granularity and one from the inner detector. Naturally, in most cases one of the two images is empty. For example if the Pion impacts in the outer calorimeter there is no signal in the inner one, and vice versa. For this thesis, however, we want to have one final image per event to make training with a generative adversarial network simpler. We therefore have to combine the inner calorimeter image with the outer one in a meaningful way, but before being able to do this they have to have the same granularity. This is achieved by dividing every cell in the outer detector into four cells, effectively doubling the number of cells in x and y direction. This is visualized in figure 2.1.3 showing on the left the pixel-wise mean over all images in the inner detector with $N_x = 32$ and $N_y = 28$ with a cavity for the beam pipe in the center. Excluding the inner beam pipe this results in 871 active cells which is very close to the 880 described in [10], the difference coming from inefficient or defect cells. The central

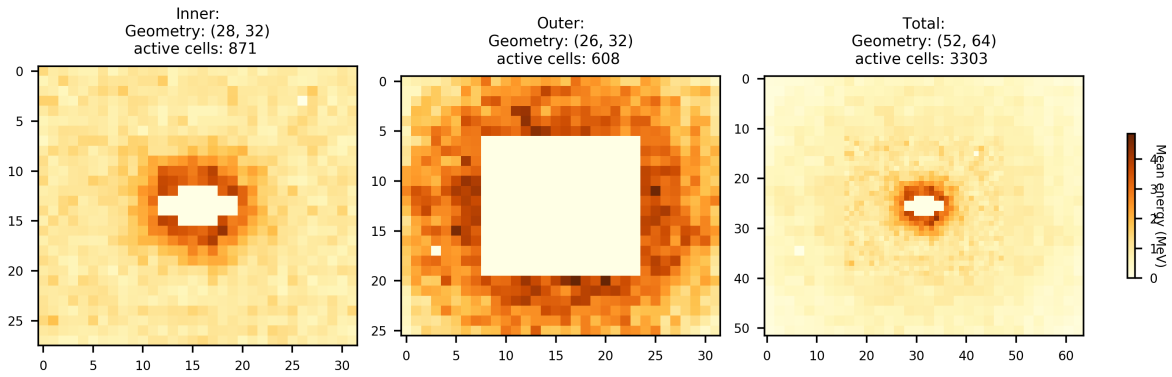


Figure 2.1.3: Average over all images in inner detector, outer detector before adjusting its granularity and final calorimeter images with adjusted outer part.

image of figure 2.1.3 visualizes the same information for the outer detector before its granularity is increased, so it consists of $N_x = 32$ and $N_y = 26$ pixels having 608 active cells. The empty part in the center is of course filled with the image from the inner detector after doubling the dimensions of the outer one. This results in the right image of figure 2.1.3 with $N_x = 64$ and $N_y = 52$. The outer part is now fairly opaque due to the fact that every pixel energy was divided by four to achieve the desired granularity.

Note that for the training of the generative adversarial networks the complete images were further preprocessed by dividing the images by the maximum energy possibly measured ($E_{\text{max}} = 6.12\,\text{GeV}$) to scale them to the interval [0, 1]. This means that the outer calorimeter values are in the interval [0, 0.25]. We also padded empty rows to the top and bottom to achieve a more symmetric image which is advantageous for convolutional neural networks which often halve and double the image sizes. During training of the first step algorithm the images were padded with

two rows at the top and bottom resulting in an image with dimensions $N_x = 64$ and $N_y = 56$. The second part relies heavily on convolutional and pooling layers favoring the bisection of images. Therefore 6 rows were padded at the top and bottom achieving images with a square cross-sectional area of $N_x = 64$ and $N_y = 64$.
The most important metrics later used for the evaluation of the generated image quality are shown in figure 2.1.4. The functions are described in detail in section 5.1. They indicate common

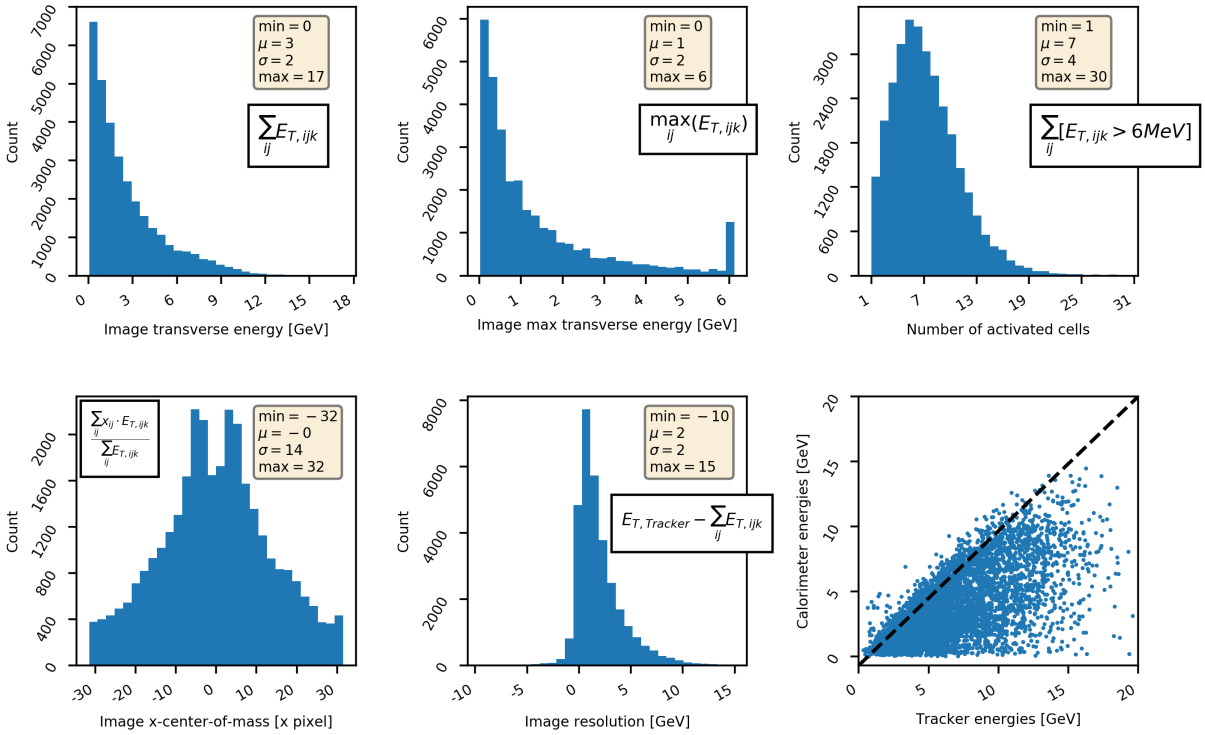


Figure 2.1.4: Distribution of relevant metrics for the output images and the function applied to a single image for the first generative adversarial network. Every function is applied on image k by aggregating over pixels i in the horizontal (x) direction and pixels j in the vertical (y) direction.

important characteristics in the resulting calorimeter response like measured energy in all cells, number of fired cells and center of mass for the energy measurements.

2.2 $B \to D\mu\nu$ data set

The root tuple for the describing the process $B \to (D \to K\pi\pi)\mu\nu$ can be considered an extension of the π^+ data set to a full event and the same Geant4 Monte Carlo procedure was applied to the tracks. The most important difference to the single pion data set is the presence of multiple tracks per event. The signal particles are the two π particles, the Kaon and the muon. These will later only become relevant for the calculation of the HCAL trigger-on-signal (HTOS) and trigger-independent-of-signal (HTIS) for evaluating images in the second part of this work. We

will not use this information for the training of any generative adversarial network as we hope that it will pick up on the importance of these particles by itself and this information is not yet available for the L0 trigger.
Much of the preprocessing and scaling is done in the same way as described in the previous section, so it is not mentioned here again. However, the input data for this step will be processed further by using the results from the first part. The generative adversarial network trained on the π^+ data set should be able to translate single track information (x, y, E_T) into a reasonable calorimeter response. This translation will then be applied to the full event generating as many calorimeter responses as there are tracks in a given event. These images are then summed up on a per-pixel basis. Note that we limited the detector response generated in the first step to the interval [0, 1] by using an appropriate activation function in the last layer of the generating network. After summation single pixels might take values greater than one. So every value larger than unity will be cut off, as this is the physical detector limit. Additionally, the outer detector should not be able to measure a value larger than 0.25 due to the granularity increasing procedure explained in the previous section. So a separate cut is performed for the corresponding pixels. See figure 2.2.1 for a visual representation of the process. These images will then be

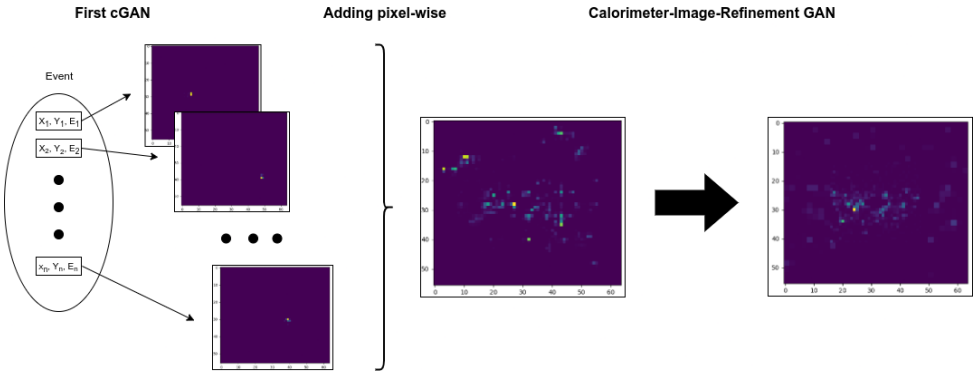


Figure 2.2.1: Visualization of the total generation pipeline in a two step process.

used for the second calorimeter image refinement network which will learn to refine these images by hopefully removing the muons from the total calorimeter response and inserting neutrons at appropriate positions as well as improving the general distributions of the output.
This two step approach was finally compared to a more direct approach by transforming the event information directly into a 2D representation by matching the predicted x and y projections onto a grid of $\{0, 1, 2, ..., 64\}$x$\{0, 1, 2, ..., 64\}$ and giving these pixels the values of the corresponding energy. These images were then divided pixel-wise by the maximum amount of energy measured per pixel in the calorimeter (6.12 GeV) to bring both images on the same scale, see figure 2.2.2 for reference.

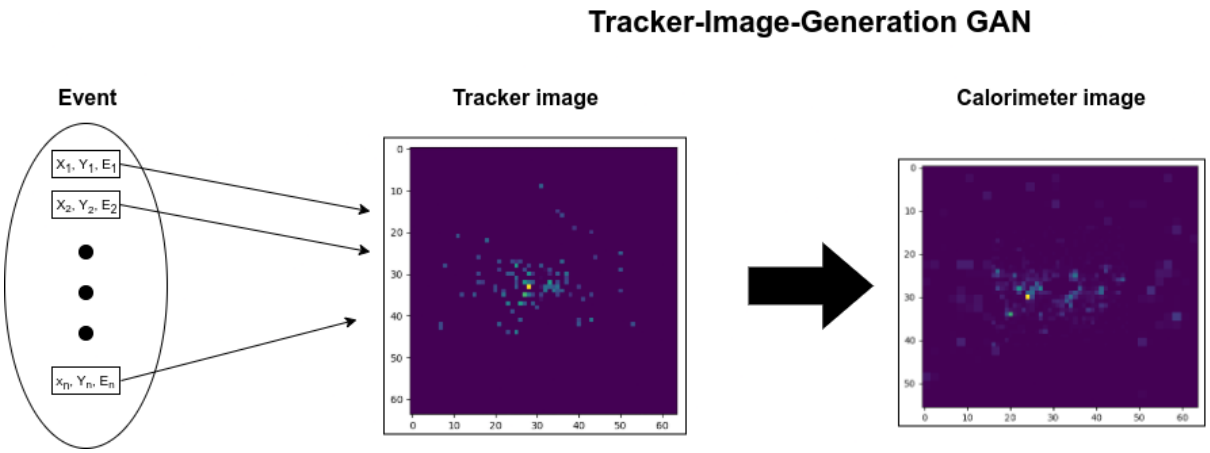


Figure 2.2.2: Visualization of the direct approach by converting a 2D representation of the tracker information directly into a calorimeter image.

3

Theory of image generation with GANs

In this chapter we describe the theory underlying all generative adversarial networks for image generation from input vectors or images. It is paramount that this transformation is not a deterministic one as this would not reflect the underlying uncertainty about the complicated interaction processes happening between the particles and the detector material. As a condition, every algorithm has to have a stochastic component in order to simulate the variability in the true data distributed as P_r.
In general there are two possibilities to approximate a true probability density P_r for a given data set:

1. Directly learn the probability density function P_r. The Maximum Likelihood approach is designed to do that providing a rich theoretical framework for computing estimates and confidence intervals. However, sometimes this is infeasible especially in the high-multidimensional setting because no reasonable assumptions about the output distribution can be made and optimization becomes very inefficient.

2. Alternatively we approximate P_r by P_g via a function g transforming an existing distribution $Z \sim P_z$ into P_g, where $P_g = g(Z)$. Z is usually a common distribution like the Gaussian or Uniform.

The first approach sounds reasonable as it is well based in statistical theory. However, the original GAN paper showed that maximizing the likelihood function is asymptotically equal

to minimizing the Kullback-Leibler divergence[22]:

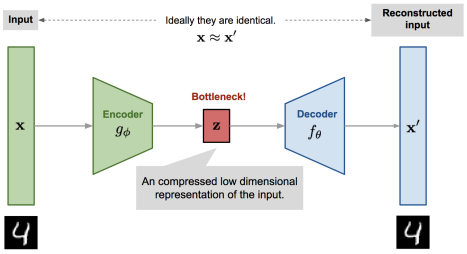
$$\begin{aligned}
\hat{\theta}_{ML} = \lim_{m\to\infty} \operatorname*{argmax}_{\theta\in\mathbb{R}^d} \frac{1}{m}\sum_{i=1}^{m} \log P_\theta(x_i^{(i)}) &= \operatorname*{argmax}_{\theta\in\mathbb{R}^d} \int_x P_r(x)\log P_\theta(x)dx \qquad \text{(limiting case)} \\
&= \operatorname*{argmin}_{\theta\in\mathbb{R}^d} -\int_x P_r(x)\log P_\theta(x)dx \qquad \text{(taking min of negative)} \\
&= \operatorname*{argmin}_{\theta\in\mathbb{R}^d} \int_x P_r(x)\log P_r(x)dx - \int_x P_r(x)\log P_\theta(x)dx \qquad \text{(add constant w.r.t } \theta) \\
&= \operatorname*{argmin}_{\theta\in\mathbb{R}^d} \int_x P_r(x)\log \frac{P_r(x)}{P_\theta(x)}dx \\
&= \operatorname*{argmin}_{\theta\in\mathbb{R}^d} KL(P_r||P_\theta) \qquad \text{(limiting case)}
\end{aligned}$$

In reality, this exact maximization problem is often very difficult in practice, especially for very high dimensional data as is regularly encountered for image generation tasks. Additionally, the Kullback-Leibler divergence has some undesirable properties, especially for the generation of highly correlated data, discussed in a later chapter. Due to these reasons most modern image generation techniques as well is this project focus on the second approach explained in the following two sections.

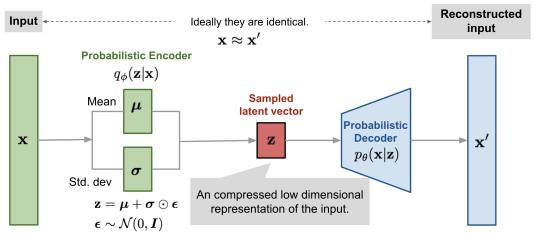
We will start with an explanation of the variational autoencoder which was proposed earlier than any generative adversarial network[37]. It is important to understand the basics of this algorithm because it will be part of the final network in the second calorimeter image refinement as well as tracker image propagation algorithm and it is educational to compare its loss function with the GAN objective function to fully appreciate these methods.

3.1 Variational Autoencoders

An (non-variational) autoencoder is a basic machine learning technique useful for image denoising, image compression and feature extraction[38, 39]. It consists of two different neural networks which can be considered a non-linear function mapping from an input space to any output space. The first, called the encoder, usually compresses the input into a lower dimensional space while the second, called the decoder, reproduces the original image from this compressed space[40]. The general structure is shown in figure 3.1.1a. However, the low dimensional space can't be used to generate images by itself, as the algorithm does not learn to interpolate between different points in space. A variational autoencoder is designed to get rid of this problem and interpolate a probability distribution between several points to serve as a generative model[42]. The goal of the variational autoencoder is for the encoder to minimize the difference between an approximate distribution $q_\phi(z|x)$ (with parameters ϕ) and a true latent distribution $p_\psi(z|x)$ (with parameters ψ). The latent variable z follows the prior distribution $P_z(z)$. After training we can sample from this distribution to generate meaningful samples by passing it to the decoder imitating the real data distribution $x \sim P_r(x)$ with the generating distribution $x' \sim P_g(x'|z;\theta)$. Assume that a true encoding distribution exists given by $p_\psi(z|x)$, but it is intractable and can't



(a) Architecture of the autoencoder[41].



(b) Architecture of the variational autoencoder[41]. The function $p_\theta(x|z)$ corresponds to $P_g(x|z;\theta)$

be computed from data. Therefore we need to approximate the true distribution with an encoder distribution $q_\phi(z|x)$ from which we can sample. We therefore minimize the Kullback-Leibler (KL) divergence as[43]

$$\begin{aligned}
\mathrm{KL}(q_\phi(z|x)||p_\psi(z|x)) &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p_\psi(z|x)}\right] \qquad (3.1.1)\\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log q_\phi(z|x) - \log p_\psi(z|x)\right] \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log q_\phi(z|x) - \log P_g(x|z) - \log P_z(z) + \log P_r(x)\right] \\
\Rightarrow \mathrm{KL}(q_\phi(z|x)||p_\psi(z|x)) - \log P_r(x) &= -\mathbb{E}_z\left[\log P_g(x|z)\right] + \mathbb{E}_z\left[\log q_\phi(z|x) - \log P_z(z)\right] \\
&= -\mathbb{E}_z\left[\log P_g(x|z)\right] + \mathrm{KL}(q_\phi(z|x)||P_z(z)) \\
&\equiv -\mathrm{ELBO}(\theta,\phi) \\
\Rightarrow \mathrm{ELBO}(\theta,\phi) &= \log P_r(x) - \mathrm{KL}(q_\phi(z|x)||p_\psi(z|x)) \leq 0 \qquad (3.1.2)
\end{aligned}$$

This is the so-called evidence lower bound (ELBO). The name stems from the fact, that the KL divergence is larger or equal to 0. Therefore $\mathrm{ELBO}(\theta,\phi) \leq \log P_r(x)$. So the corresponding function is a lower bound for $\log P_r(x)$, also known as the evidence or the log-likelihood. By maximizing the ELBO we are maximizing the probability to generate real data samples due to the fact that if $\mathrm{ELBO}(\theta,\phi) \to 0 \Rightarrow \log P_r(x) \to 0 \Rightarrow P_r(x) \to 1$. This means that we can maximize the ELBO instead of minimizing the original KL, which would have been intractable. However, in machine learning we want to deal with objective functions being positive and having a lower bound of zero. This leads naturally to the following loss function by multiplying the ELBO by minus one.

$$L_(\theta,\phi) = -\mathbb{E}_{z\sim q_\phi(z|x)}[\log P_g(x|z)] + \mathrm{KL}(q_\phi(z|x)||P_z(z)) \geq 0 \qquad (3.1.3)$$

The first term is the reconstruction likelihood and the second term penalizes differences between the learned distribution $q_\phi(z|x)$ and the prior distribution $P_z(z)$. It acts as a natural regularizer and measures the amount of information lost when approximating P_z with q_ϕ.

Assuming a normal density for the generating distribution results in

$$P_g(x|z) = \frac{1}{\sqrt{(2\pi)^k|\Sigma_1|}} \exp\left(-\frac{1}{2}(x-\mu_\phi(z))^T\Sigma_1^{-1}(x-\mu_\phi(z))\right) \tag{3.1.4}$$
$$\Rightarrow \log P_g(x|z) \propto (x-\mu_\phi(z))^T\Sigma_1^{-1}(x-\mu_\phi(z)),$$

which penalizes via the square error if the output x is different from the input $\mu_\phi(z)$. The Kullback-Leibler divergence prevents the distribution $q_\phi(z|x)$ to collapse to a single point which is a prominent problem in high quality image generation called mode collapse. We will discuss this problem in greater detail at a later point.
Let's discuss what would happen if either part would not be present in the loss.

1. Without the KL-divergence term, the algorithm will not learn the prior distribution and we have no means of controlling the second moment. It will converge to zero. The resulting algorithm is basically an autoencoder which only models z around a very specific point in latent space but does not interpolate and does not correspond to $P_z(z)$.

2. Without the regularization term the algorithm will not converge to the data distribution. The distribution $q_\phi(z|x)$ is likely to follow the prior distribution but the numbers are not reconstructed as nothing penalizes bad reconstruction. Minimizing only the Kullback-Leibler divergence practically means in this case doing a non-parametric maximum likelihood estimation of the prior distribution.

While equation 3.1.3 is useful from a theoretical point of view it is still not feasible to implement it into a specific algorithm. We must be able to explicitly compute the KL divergence between the true and imitating latent distribution. This is were one of the drawbacks of variational autoencoders comes into play: We must choose a multivariate normal distribution for both of them in order to calculate the KL divergence.
We will show that the Kullback-Leibler divergence for two multivariate normal densities is given by[44]:

$$\mathrm{KL}(p(x)||q(x)) = \frac{1}{2}\sum_k \left[\Sigma_{\phi,(kk)}(x) + \mu_{\phi,(k)}(x)^2 - 1 - \log\Sigma_{\phi,(kk)}(x)\right]. \tag{3.1.5}$$

We start with the density of a multivariate normal random variable given by

$$X \sim \mathbb{N}(\mu_1, \Sigma_1) \Rightarrow p(x) = \frac{1}{\sqrt{(2\pi)^k|\Sigma_1|}} \exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\right). \tag{3.1.6}$$

This leads to the log-likelihood

$$\log p(x) = -\frac{k}{2}\log 2\pi - \frac{1}{2}\log|\Sigma_1| - \frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1). \tag{3.1.7}$$

We use this to calculate the KL divergence of two normally distributed random variables via

$$\begin{aligned}\mathrm{KL}(p(x)||q(x)) &= \sum_x p(x)(\log p(x) - \log q(x)) \\ &= \sum_x p(x)\left[\frac{1}{2}\log\frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) + \frac{1}{2}(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2)\right].\end{aligned} \tag{3.1.8}$$

These terms can be simplified drastically. First we compute the second term.

$$\begin{aligned}
\sum_x p(x)\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) &= \mathbb{E}_{x\sim p}\left[\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\right] \\
&= \mathbb{E}_{x\sim p}\left[tr\left(\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\right)\right] \quad \text{(use } x^TAx \text{ is scalar)} \\
&= tr\left(\mathbb{E}_{x\sim p}\left[\frac{1}{2}(x-\mu_1)(x-\mu_1)^T\Sigma_1^{-1}\right]\right) \quad (tr \text{ is cyclic and switched with } \mathbb{E}) \\
&= tr\left(\mathbb{E}_{x\sim p}\left[(x-\mu_1)(x-\mu_1)^T\right]\Sigma_1^{-1}\frac{1}{2}\right) \quad \text{(Linearity of expectation)} \\
&= tr\left(\Sigma_1\Sigma_1^{-1}\frac{1}{2}\right) \quad \text{(Def. of covariance matrix)} \\
&= \frac{1}{2}tr[I_k] = \frac{k}{2} \qquad (3.1.9)
\end{aligned}$$

Now consider the third part of equation 3.1.8.

$$\begin{aligned}
\sum_x p(x)\frac{1}{2}(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2) &= \mathbb{E}_{x\sim p}\left[\frac{1}{2}\left[(x-\mu_1)+(\mu_1-\mu_2)\right]^T\Sigma_2^{-1}\left[(x-\mu_1)+(\mu_1-\mu_2)\right]^T\right] \\
&= \mathbb{E}_{x\sim p}\left[\frac{1}{2}(x-\mu_1)^T\Sigma_2^{-1}(x-\mu_1) + \cancel{(x-\mu_1)^T\Sigma_2^{-1}(\mu_1-\mu_2)}^{\mathbb{E}(x)=\mu_1} + \frac{1}{2}(\mu_1-\mu_2)^T\Sigma_2^{-1}(\mu_1-\mu_2)\right] \\
&= tr(\frac{\Sigma_2^{-1}\Sigma_1}{2}) + \frac{1}{2}(\mu_1-\mu_2)^T\Sigma_2^{-1}(\mu_1-\mu_2) \qquad (3.1.10)
\end{aligned}$$

Putting it all together results in the KL divergence given above

$$\mathrm{KL}(p(x)||q(x)) = \frac{1}{2}\left[\log\frac{|\Sigma_2|}{|\Sigma_1|} - k + tr(\Sigma_2^{-1}\Sigma_1) + (\mu_2-\mu_1)^T\Sigma_2^-1(\mu_2-\mu_1)\right] \qquad (3.1.11)$$

If our latent variable distribution $q(x) = P_z(z)$ is taken to be a standard Gaussian with mean zero and variance matrix equal to the identity and $p(x)$ is substituted to be dependent on parameters ϕ and be similar to the notation for the variational autoencoder $q_\phi(x)$ the KL divergence becomes

$$\mathrm{KL}(q_\phi(x)||\mathbb{N}(0,1)) = \frac{1}{2}[tr(\Sigma_\phi(x)) + \mu_\phi(x)^T\cdot\mu_\phi(x) - k - log|\Sigma_\phi(x)|]. \qquad (3.1.12)$$

Note that most of the time we choose our latent variable dimensions to be uncorrelated, then Σ_ϕ is diagonal and the expression becomes[44]

$$\begin{aligned}
\mathrm{KL}(q_\phi(x)||\mathbb{N}(0,1)) &= \frac{1}{2}\left[\sum_k \Sigma_{\phi,(kk)}(x) + \sum_k \mu_{\phi,(k)}(x)^2 - \sum_k 1 - \log\prod_k \Sigma_{\phi,(kk)}(x)\right] \\
&= \frac{1}{2}\sum_k\left[\Sigma_{\phi,(kk)}(x) + \mu_{\phi,(k)}(x)^2 - 1 - \log\Sigma_{\phi,(kk)}(x)\right].
\end{aligned}$$

An improvement with respect to later computational performance can be made by substituting $\Sigma_\phi(x)$ with $\log\Sigma_\phi(x)$ to get rid of the square root of the $\Sigma_\phi(x)$ (see eq. 3.1.16) in the transformation below. So in practice one inserts $\exp\Sigma_\phi$ when needed. Our final loss function now

reads

$$\mathcal{L}(\theta,\phi) = -\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log P_g(x|z)\right] + \frac{1}{2}\sum_k \left[\exp\Sigma_{\phi,(kk)}(x) + \mu^2_{\phi,(k)}(x) - 1 - \Sigma_{\phi,(kk)}(x)\right] \tag{3.1.13}$$

Another difficulty during training arises due to the randomness in the latent space and the inability of gradient descent to back-propagate such a stochastic process. First look at the derivatives with respect to θ.

$$\nabla_\theta\mathcal{L}(\theta,\phi) = -\mathbb{E}_{z\sim q_\phi(z|x)}\left[\nabla_\theta \log P_g(x|z)\right] = \frac{1}{m}\sum_{i=1}^{m}\nabla_\theta \log P_g(x|z^{(i)}) \tag{3.1.14}$$

This works without trouble but the second step for the optimization of ϕ is harder because $\nabla_\phi\mathbb{E}_{q_\phi(z|x)}[f(z)] \neq \mathbb{E}_{q_\phi(z|x)}[\nabla_\phi f(z)]$. So we need to find a way to re-express the expectation in such a way that the parameters ϕ are no longer within the distribution. Fortunately this can be done rather easily if we continue to assume a normal distribution[45].
If

$$\epsilon \sim \mathbb{N}(0,1) \tag{3.1.15}$$

$$\Rightarrow g_\phi(\epsilon,x) = \mu_\phi(x) + x\cdot\Sigma^{1/2}_\phi(x) + \epsilon = z \sim \mathbb{N}(\mu_\phi(x),\Sigma_\phi(x)). \tag{3.1.16}$$

If we sample ϵ from a standard normal distribution and transform via this equation, the parameters are no longer a function of ϕ and we can use the back-propagation algorithm. There are

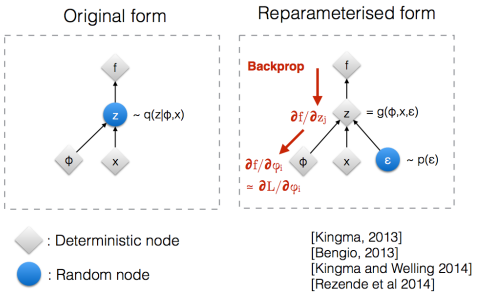


Figure 3.1.2: Random node before and after the reparametrization trick[46].

mainly three problem with variational autoencoders. First we need to assume that every distribution is Gaussian for the simplifications to work. However, in most cases this is reasonable as the Gaussian distribution has a lot of desirable properties and in the absence of any information about the latent space it is as reasonable as any other distribution. Another disadvantage is the usage of the KL divergence itself as it has some undesirable properties when q and p are not overlapping discussed in section 3.3.5. A third drawback is the implementation of a pixel-wise loss function in the form of the binary cross-entropy for the logarithmic term in in equation 3.1.13. This neglects the spatial correlation when generating images as every pixel of the generated image is evaluated by comparing it to the corresponding pixel in the input image but not to its surrounding image.

3.2 Why (c)GANs

Some of the drawbacks of the variational autoencoder are obvious when looking at the derivation of the loss function. It is rather restrictive in it's assumptions about the latent and data distribution by forcing them to follow a Normal distribution. Furthermore, the pixel-wise penalization of the generated image empirically leads to smeared images as no consideration is placed on the correlation of neighboring pixels. Generative adversarial networks eliminate both of these problem by introducing a second network responsible for giving feedback to the generating function[22, 47].
A GAN, shown in figure 3.2.1, uses two neural networks. The first is called the generator and

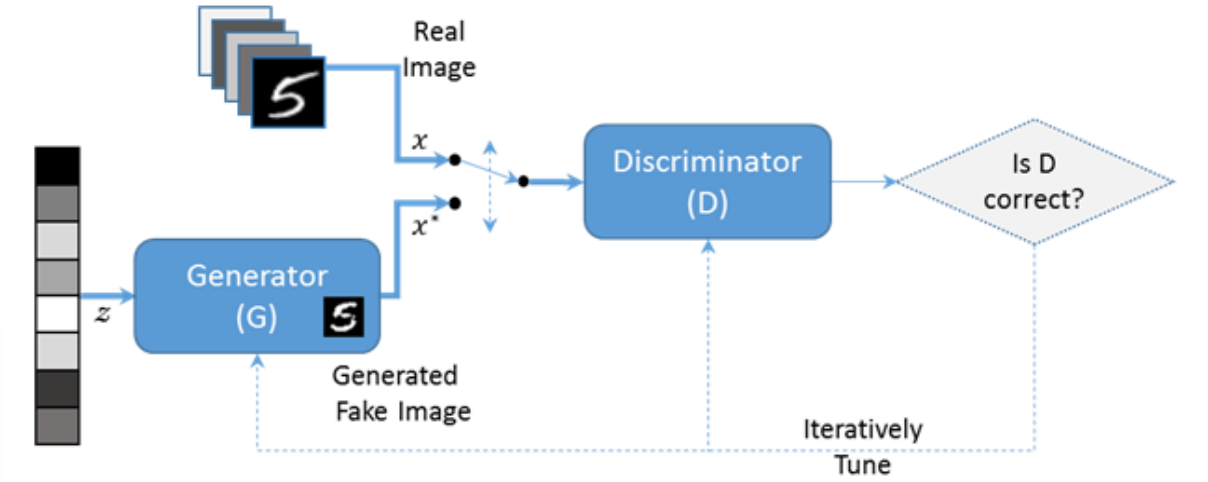


Figure 3.2.1: Architecture of the generative adversarial network[48].

is responsible for transforming a latent distribution $P_z(z)$ into an approximation $P_g(x|z)$ of the real data distribution $P_r(x)$. Note that there is no restriction on the nature of the latent distribution and any valid distribution is possible. It aims to fool the second network called the discriminator. This network tries to discriminate real samples $x \sim P_r(x)$ from generated images $x' \sim p(x'|z)$ and gives feedback to the generator about possible improvements.
These networks work in a completely unsupervised way meaning no information about labels for the images is needed for the algorithm to train. While this has its own advantages we have no control over the type of image generated when transforming the sampled variable $z \sim P_z$ with the generator $G(z;\phi)$ into P_g. In the case of the famous `mnist` data set this means that when sampling a random variable z it is unclear whether the produced image will be a zero or nine or anything in between. This is undesirable for this thesis as we want to generate a specific type of image for every track. If the tracker generates the information that a particle will hit in the upper right corner of the calorimeter this information should influence the generation of the generated calorimeter image. We can provide this information using conditional GANs. We will derive all loss functions for the unconditional case, however, they are derived in an absolutely

equivalent way when conditioning on the input. The main difference is that the label has to be provided once during the generation process and as well when passing the true and generated images into the discriminator, see figure 3.2.2.

This technique works for almost any GAN architecture and certainly for every architecture

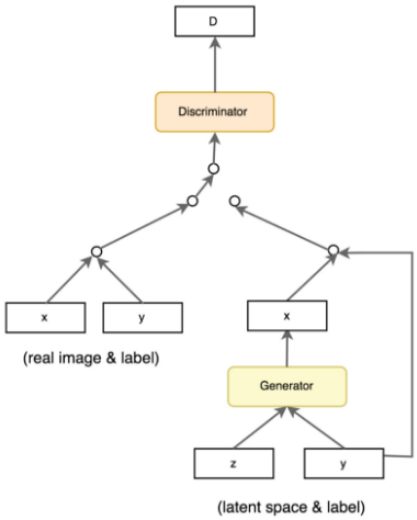


Figure 3.2.2: Modification made to Vanilla GAN in order to achieve a conditional GAN.

used in this thesis.

Even though the principle method of generative adversarial networks sounds simple the technicalities are rather complex as the solution of this minmax-problem is called a Nash equilibrium in game theory. It is the solution to a problem where one player tries to maximize one objective function while it's adversarial aims to minimize a variant of this objective. No reliable optimizer exists for this problem and even in the most simplistic scenarios algorithms may not converge[49]. Therefore it is important to fully grasp how the loss functions are computed and what their theoretical properties are. In the next section we will explore this subject in great detail.

3.3 The loss function

GANs are neural networks training in an adversarial manner in order to mimic some true distribution P_r with P_g. The two building blocks of GANs are:

1. Generator G: Trained on data coming from a true distribution P_r by taking some random density $P_z(z)$ and produce a distribution P_g which is close to P_r according to some measure of closeness. So $G(z;\phi) \sim P_g \approx P_r$.

2. Discriminator D: Differentiates between real and generated samples given input from the real and approximating distributions. Usually the discriminator has a single output node and aims for the case of $D(x;\theta) = 1$ and $D(G(z;\phi);\theta) = 0$.

3.3.1 Jensen-Shannon Divergence

The loss function used in the origanl GAN paper is a direct result from statistical decision and maximum likelihood theory in the form of the binary cross entropy, where y denotes the true label of a sample and $\hat{y}$ the estimation from the discriminator.

$$l(x, y) = y \cdot \log(D(x;\theta)) + (1-y) \cdot \log(1 - D(x;\theta)) \quad \text{derived from} \tag{3.3.1}$$

$$L(x, y) = D(x;\theta)^y \cdot (1 - D(x;\theta))^{(1-y)} \qquad y = \begin{cases} 1 & \text{if } y \text{ from } P_r \\ 0 & \text{if } y \text{ from } P_g \end{cases} \tag{3.3.2}$$

The discriminator aims to classify the inputs correctly, so it needs to maximize the above expression[22]:

$$\max \log D(x;\theta) + \log\left[1 - D(G(z;\phi);\theta)\right] \tag{3.3.3}$$

On the other hand the generator tries to fool the discriminator with $D(G(z;\phi);\theta) = 1$. Therefore the generator minimizes the above loss for generated images

$$\min \log\left[1 - D(G(z;\phi);\theta)\right] \tag{3.3.4}$$

with the unique solution $D(G(z;\phi);\theta) = 1$. The full loss function is then given by the min-max problem

$$\min_{\phi}\max_{\theta} \mathcal{L}(\theta, \phi) = \min_{\phi}\max_{\theta} \left[\log D(x;\theta) + \log\left[1 - D(G(z;\phi);\theta)\right]\right] \tag{3.3.5}$$

which is given in the original paper more correctly for a full data set as

$$\min_{\phi}\max_{\theta} \mathcal{L}(\theta, \phi) = \min_{\phi}\max_{\theta} \mathbb{E}_{x\sim P_r(x)}\left[\log D(x;\theta)\right] + \mathbb{E}_{z\sim P_Z}\left[\log\left[1 - D(G(z;\phi);\theta)\right]\right] \tag{3.3.6}$$

resulting in the adversarial game.
We now investigate the loss of the GAN given in equation(3.3.6). Assume we have a fixed generator, then the optimal prediction a Discriminator can make is given by

$$\begin{aligned} D_G^* &= \underset{D}{\operatorname{argmax}} \quad \mathcal{L}(D, G) \\ &= \underset{D}{\operatorname{argmax}} \quad \mathbb{E}_{x\sim P_r(x)}\left[\log D(x)\right] + \mathbb{E}_{z\sim P_Z}\left[\log\left[1 - D(G(z))\right]\right] \\ &= \underset{D}{\operatorname{argmax}} \quad \int P_r(x) \log D(x) dx + \int P_Z(z) \log\left[1 - D(G(z))\right] dz \end{aligned} \tag{3.3.7}$$

We express $z \sim P_z$ of the latent variable in terms of the transformed random variable $x \sim P_g$ with $x = G(z;\phi)$, then

$$P_g(x) = P_z(G^{-1}(x))\frac{dG^{-1}(x)}{dx} = P_z(z)\frac{dz}{dx}. \tag{3.3.8}$$

This results in

$$\begin{aligned} D_G^* &= \underset{D}{\operatorname{argmax}} \quad \int P_r(x) \log D(x) dx + \int P_g(x) \log\left[1 - D(x)\right] dx \\ &= \underset{D}{\operatorname{argmax}} \quad \int P_r(x) \log D(x) + P_g(x) \log\left[1 - D(x)\right] dx \end{aligned} \tag{3.3.9}$$

So the optimal discriminator is given by

$$\begin{aligned} \frac{d}{dD(x)} \quad & \left[P_r(x) \log D(x) + P_g(x) \log\left[1 - D(x)\right]\right] \stackrel{!}{=} 0 \\ \Rightarrow & \frac{P_r(x)}{D_G^*(x)} - \frac{P_g(x)}{1 - D_G^*(x)} = 0 \\ & D_G^* = \frac{P_r(x)}{P_r(x) + P_g(x)} \end{aligned} \tag{3.3.10}$$

Of course this is only a theoretical result as the true distribution $P_r(x)$ is not known in real world examples and one has to find an approximate solution via iterative optimization algorithms. It is still educational to continue with the generator to understand the processes during training. Now we try to calculate the optimal G for a given D^*. It is desirable that this solution is unique and satisfies $P_g(x) = P_r(x)$[50].

$$G^* = \underset{G}{\operatorname{argmin}} \quad \mathcal{L}(D_G^*, G) \tag{3.3.11}$$

$$\begin{aligned} &= \underset{G}{\operatorname{argmin}} \int P_r(x) \log D_G^*(x) + P_g(x) \log\left[1 - D_G^*(x)\right] dx \\ &= \underset{G}{\operatorname{argmin}} \int P_r(x) \log \frac{P_r(x)}{P_r(x) + P_g(x)} + P_g(x) \log \frac{P_g(x)}{P_r(x) + P_g(x)} dx \\ &= \underset{G}{\operatorname{argmin}} \int P_r(x) \log \frac{P_r(x)}{P_r(x) + P_g(x)} + P_g(x) \log \frac{P_g(x)}{P_r(x) + P_g(x)} \\ &\qquad + \log(2) P_r(x) - \log(2) P_r(x) + \log(2) P_g(x) - \log(2) P_g(x) dx \\ &= \underset{G}{\operatorname{argmin}} \int -\log(2) \cdot (P_r(x) + P_g(x)) + P_r(x) \cdot \log \frac{P_r(x)}{(P_r(x) + P_g(x))/2} \\ &\qquad + P_g(x) \cdot \log \frac{P_g(x)}{(P_r(x) + P_g(x))/2} dx \\ &= \underset{G}{\operatorname{argmin}} -\log(2) \cdot \int P_r(x) + P_g(x) dx \\ &\qquad + \int P_r(x) \cdot \log \frac{P_r(x)}{M} + P_g(x) \cdot \log \frac{P_g(x)}{M} dx, \quad \text{with } M(x) = \frac{P_r(x) + P_g(x)}{2} \\ &= \underset{G}{\operatorname{argmin}} -2 \cdot \log(2) + \mathrm{KL}(P_r(x) || M) + \mathrm{KL}(P_g || M) \\ &= \underset{G}{\operatorname{argmin}} -\log(4) + 2 \cdot \mathrm{JS}(P_r(x), P_g(x)) \end{aligned} \tag{3.3.12}$$

The unique solution of this optimization is given by

$$P_g(x) = P_r(x), \quad \text{with } V(D_G^*, G^*) = -\log(4) \tag{3.3.13}$$

So ideally the generative adversarial networks minimizes the Jensen-Shannon divergence which has improved properties compared to the Kullback-Leibler divergence, investigated in greater detail in section 3.3.5.
In practice the generator performs usually badly at the start of training when the generator produces samples which are extremely far from the true distribution. This leads to $D(G(z)) \approx 0$. If the above function $\mathcal{L}(D, G)$ is investigated it becomes apparent that at this position the slope of the generator loss $\log(1 - D(G(z)))$ is close to zero, which results in very slow training. If the above loss is re-expressed as two maximization processes

$$\max_{\theta} \left[\log D(x;\theta) + \log\left[1 - D(G(z;\phi);\theta)\right]\right] \tag{3.3.14}$$

$$\max_{\phi} \log D(G(z;\phi);\theta), \quad \text{where } \log D(G(z;\phi);\theta) \overset{D(G(z;\phi);\theta)\to 1}{\longrightarrow} 0, \tag{3.3.15}$$

then the resulting sets of losses lead to much better properties during optimization especially at the start of training, when the gradients need to be large to get results within a reasonable quality at a faster pace, see figure 3.3.1.

Note that the loss function of the generator does not involve the data in any way, only the

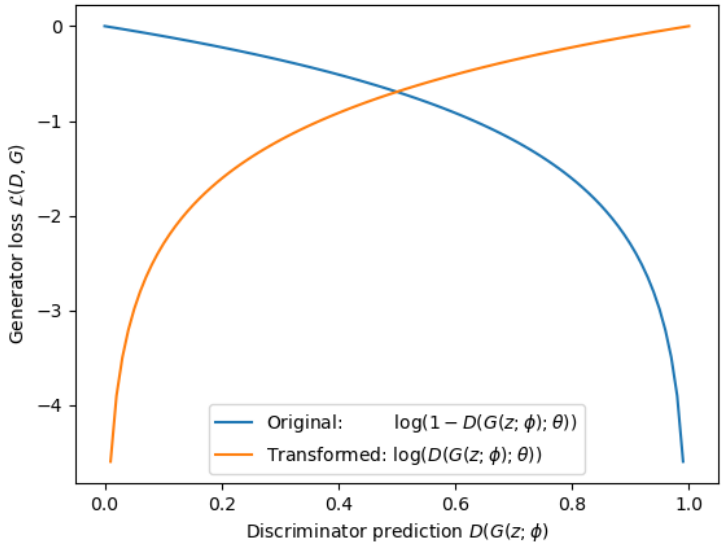


Figure 3.3.1: Functional form of the two formulations for the loss of the generator. At the beginning of training when the generated image quality is low the gradient of the original loss function (blue) is close to zero, while it is large for the transformed loss (orange).

output of the discriminator is taken into account. This is one of the reasons that make GANs resistant to overfitting.

3.3.2 Kullback-Leibler Divergence for GANs

With the new step implemented the connection to the Jensen-Shannon divergence is lost and it is theoretically unclear what is actually minimized. However, the numerical advantages and

speed-up in learning are worth the transformation[51].
Another alternative to the transformation described in 3.3.15 is the sum of the original and updated loss:

$$\min \log 1 - D(G(z)) \text{ and } \max \log D(G(z)) \tag{3.3.16}$$

$$\rightarrow \max \mathcal{L}(D, G) = \max \log \frac{D(G(z;\phi);\theta)}{1 - D(G(z;\phi);\theta)} \tag{3.3.17}$$

It can be shown that this actually minimizes the Kullback-Leibler divergence $\mathrm{KL}[P_g|P_r]$, not to be confused with the Maximum Likelihood approach $\mathrm{KL}[P_r|P_g]$. Note, however, as this is the `logit`(π) function which ranges from minus infinity to infinity in contrast to many other loss functions. This needs to be taken into account in its implementation as a proper scaling factor might be necessary.
The prove that equation 3.3.17 minimizes the KL divergence is easy because it was shown that the ideal and converged solution for an optimal discriminator between P_r and P_g is given by:

$$D^*(x) = \frac{P_r(x)}{P_r(x) + P_g(x)} \tag{3.3.18}$$

We can then show that if after the discriminator optimization the discriminator D is close to the Bayes-optimal D*

$$\begin{aligned}
\max \mathcal{L}(D^*, G) &= \max \mathbb{E}_{x \sim P_X} \log \frac{D^*(x)}{1 - D^*(x)} \\
&= \max \mathbb{E}_{x \sim P_X} \log \frac{\frac{P_r(x)}{P_r(x) + P_g(x)}}{1 - \frac{P_r(x)}{P_r(x) + P_g(x)}} \\
&= \max \mathbb{E}_{x \sim P_X} \log \frac{P_r(x)}{P_g(x)} \\
&= \min \mathbb{E}_{x \sim P_X} \log \frac{P_g(x)}{P_r(x)} = \min \mathrm{KL}[P_g(x)|P_r(x)]
\end{aligned} \tag{3.3.19}$$

This completes the proof, as minimizing this KL divergence results in maximizing the proposed loss function.

3.3.3 Generalization of the JS divergence

A generalization of the Jensen-Shannon divergence with parameter π $(0 \leq \pi \leq 1)$ exists, which covers both the standard Jensen-Shannon divergence mentioned above as well as both the forward and backward Kullback-Leibler divergence[51]:

$$\mathrm{JS}_\pi(P_r|P_g) = (1-\pi)\mathrm{KL}(P_r|\pi P_r + (1-\pi)P_g) + \pi \mathrm{KL}(P_g|\pi P_r + (1-\pi)P_g) \tag{3.3.20}$$

This results in

- $\pi = 0$: $\mathrm{JS}_0[P_r|P_g] = \mathrm{KL}(P_r|P_g)$

- $\pi = 0.5$: $\mathrm{JS}_{0.5}(P_r|P_g) = 0.5 \cdot \left(\mathrm{KL}(P_r|\frac{P_r+P_g}{2}) + \mathrm{KL}[P_g|\frac{P_r+P_g}{2}]\right) = \mathrm{JS}(P_r, P_g)$
- $\pi = 1$: $\mathrm{JS}_1(P_r|P_g) = \mathrm{KL}(P_g|P_r)$

In figure 3.3.2 the difference between these decisions is visualized in a practical example. Why the losses behave like shown in the figure will be clear after reading chapter 3.3.5.

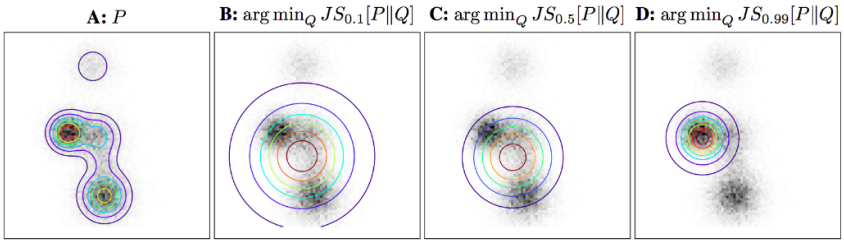


Figure 3.3.2: From left to right: 1) True data distribution 2) KL(P|Q)=JS_0 (over-generalizing) 3) standard Jensen-Shannon divergence $\mathrm{JS}_0.5$ and 4) KL(Q|P)=JS_1 (under-generalizing)[52].

3.3.4 The Wasserstein distance

This section will describe the last loss function used for the single track propagation of this thesis. After this section we will discuss their differences in a much more practical approach and investigate which advantages one loss might have over another.
The loss of the Wasserstein GAN[53] is derived from transport theory[54], often called earth-mover distance:

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \int \gamma(x,y)||x-y||dxdy = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||], \tag{3.3.21}$$

where $\gamma(x,y)$ is a transport plan, which indicates how much mass is transferred from $x \sim P_r$ to $y \sim P_g$. It's connection to probability theory is especially clear for discrete distributions where the densities are called probability mass functions assigning single scalar probabilities to every point in the sample space. Two examples of transport plans for a simple problem are given in figure 3.3.3.
Two conditions have to be fulfilled:

- $\int_y \gamma(x,y)dy = P_r(x)$. The amount of mass transported from x to all y's must be equal to the previous mass at x.
- $\int_x \gamma(x,y)dx = P_g(y)$. The amount of mass delivered to point y from all x's must equal the mass of the posterior distribution at this point.

This concept can naturally be generalized to the continuous case as we can understand our central problem of transforming P_g into P_r as a transportation of scalar values (densities) without

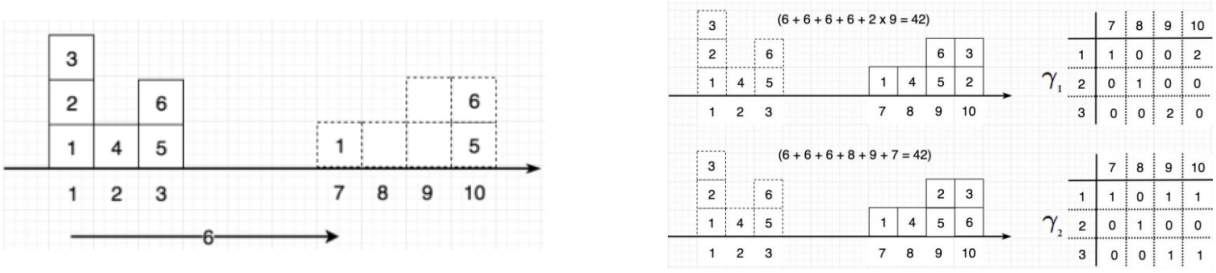


Figure 3.3.3: Two examples of a possible transport plan to solve a simple problem in transport theory[55].

changing the total mass ($\int p(x) = 1$).
The average cost is

$$\int_x \int_y \gamma(x,y)||x-y||dydx = \mathbb{E}_{(x,y)\sim\gamma}[||x-y||]. \tag{3.3.22}$$

Optimal transport problems are a special case of linear programming problems since both the function to be optimized and the constraints are linear functions of the transportation map. The theory behind linear programming dates back to the beginning of the last century and is one of the cornerstones of mathematical optimization. One of the most fundamental results of linear programming is that any linear problem has a dual problem. The solution of the dual problem provides an upper bound to the solution of the original (primal) problem. Fortunately, it turns out that in the case of optimal transport the solution of the dual problem does not simply provide a bound but is indeed identical to the solution of the primal problem. Furthermore, the dual formulation of the optimal transport problem is the starting point for adversarial algorithms and the Wasserstein GAN. The dual formulation of an optimal transport divergence is given by the following formula[56]:

$$f_c^*(P_r, P_g) = \sup_{f\in L_c}\left[\int f(x)P_r(x)dx - \int f(y)P_g(y)dy\right] \quad \text{(Kantorovich-Rubenstein duality)}, \tag{3.3.23}$$

where L_c is the set of functions whose growth is bounded by c:

$$L_c = \{f : \mathbb{R}^n \to \mathbb{R} | f(x) - f(y) \leq c(x,y)\},$$

It is far from obvious why this expression is equivalent to the primal expression. However, the formula itself has a rather intuitive interpretation. Clearly, if P_r is equal to P_g, the difference of their expected values will be zero regardless of f and consequently the divergence will vanish. Normally P_r and P_g will differ over some or even most regions of their domain. In this case the divergence is obtained by finding the function f that maximizes this difference in terms of its expected value. In other words, f acts like a discriminator extracting the features that maximally differentiate P_r from P_g. Note that without any constraints on f any small difference in the distributions can be magnified arbitrarily and the divergence would be infinite. However,

a main difference between a discriminator D in vanilla GANs and f is the un-boundedness of the function f. Hence this function is called a critic to differentiate it from its vanilla GAN counterpart.
Equation 3.3.23 can be written as

$$W(P_r, P_g) = \sup_{||f||_L \leq 1} \mathbb{E}_{x \sim P_r}[f(x)] - \mathbb{E}_{x \sim P_g}[f(x)] \tag{3.3.24}$$

where the K-Lifschitz function f never exceed a slope of K. Note that every K-Lifschitz function is a 1-Lifschitz function if we divide it by K. The loss for the optimization of the Wasserstein network can now be expressed in terms of the parameters θ[54]:

$$\begin{aligned} \max_{\theta, ||f||_L \leq K} \quad \mathbb{E}_{x \sim P_r}[f_\theta(x)] - \mathbb{E}_{x' \sim P_g(\phi)}[f_\theta(x')] &\leq \sup_{||f||_L \leq K} \mathbb{E}_{x \sim P_r}[f_\theta(x)] - \mathbb{E}_{x' \sim P_g(\phi)}[f_\theta(x')] \\ &= K \cdot W(P_r, P_g). \end{aligned} \tag{3.3.25}$$

Note that on the left hand side of this equation only the parameters θ of a fixed function f are optimized while on the right hand side the optimization is performed over the space of all K-Lifschitz functions. For optimization purposes, we are not required to know what the specific value of K is. It's enough to know that it exists, and that it's fixed throughout training process. Gradients of f_θ will be scaled by an unknown K, but they'll also be scaled by the learning rate, so K will get absorbed into the hyper-parameter tuning. The left hand side of equation 3.3.25 is the objective function for the critic (discriminator) of the Wasserstein network aiming to separate both distributions as well as possible.
If the family of functions $\{f_\theta\}_{\theta \in \Theta}$ contains the true supremum, the distance is exact. This is highly unlikely, but it is known that neural networks act as universal function approximators[57]. In generative models the goal is to find $P_g = G(z; \phi)$ such that P_g approximates P_r. Given a fixed function for the critic f^*_θ we can get the optimal generator with:

$$\nabla_\phi W(P_r, P_g) = \nabla_\phi \left(\mathbb{E}_{x \sim P_r}[f_\theta(x)^*] - \mathbb{E}_{z \sim Z}[f^*_\theta(G(z; \phi))] \right) \tag{3.3.26}$$

$$= -\mathbb{E}_{z \sim Z}[\nabla_\phi f^*_\theta(G(z; \phi))] \tag{3.3.27}$$

The invocation of the Lifschitz constraint is difficult in practice. The original paper proposes the algorithm shown in 1. It clips the weights at small values in the hope of approximately fulfilling the constraint. However, this is a very crude method and improvement exists. [58] proposes to introduce a L_2 norm between the gradients and the identity. This has the practical drawback that the explicit computation of the gradients take along time and training slows down greatly. This project still used the latter approach as some hours of training are worth the more exact fulfillment of the Lifschitz constraint.

3.3.5 Practical comparison

In this chapter we are taking a closer look at the different loss functions described in the previous section. These loss functions are responsible in some part for the advantages and drawbacks of

Algorithm 1: WGAN algorithm[53]

Input: α (learning rate, 0.0001). c (clipping parameter, 0.01). m (batch size, 64). n_{critic} (iterations of discriminator, 5). w_0 (initial weights). θ_0 (initial generator parameters)

while θ *has not converged* **do**
  for t=0,...,n_{critic} **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim P_r$ a batch from real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch from prior distribution.
    $\text{grad}_w \leftarrow \nabla_w[\frac{1}{m}\sum_{i=1}^{m} f_\theta(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_\theta(g_\theta(z^{(i)}))]$
    $w \leftarrow w + \alpha \cdot RMSProp(w, \text{grad}_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch from prior distribution.
  $\text{grad}_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_\theta(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot RMSProp(\theta, \text{grad}_\theta)$
end

each of these methods. To summarize we are comparing

1. Variational Autoencdoers using
   Kullback-Leibler Divergence:
$$\text{KL}(P_r||P_g) = \int_x P_r(x) \log \frac{P_r(x)}{P_g(x)} dx$$

2. Generative adversarial networks using
   Jensen-Shannon Divergence:
$$\text{JS}(P_r, P_g) = \frac{1}{2}\left[\text{KL}(P_r(x)||M(x)) + \text{KL}(P_g(x)||M(x))\right], \quad \text{with } M = \frac{P_r + P_g}{2}$$

3. Wasserstein generative adversarial networks using
   Wasserstein distance:
$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||] = \inf_{\gamma \in \Pi(P_r, P_g)} \int_{x,y} \gamma(x, y)||x - y|| dy dx,$$
$$\text{with } \int_y \gamma(x, y) dy = P_r(x) \text{ and } \int_x \gamma(x, y) dx = P_g(y)$$

Kullback-Leibler divergence

First we discuss the KL divergence given by

$$\text{KL}(P_r(x)||P_g(x)) = \int_x P_r(x) \log \frac{P_r(x)}{P_g(x)} dx.$$

This function is minimized by $P_g(x) = P_r(x)$, but some major drawbacks are around the corner. First, how is it justified to use this specific form of the divergence instead of reverse KL divergence given by

$$\mathrm{KL}(P_g(x)||P_r(x)) = \int_x P_g(x) \log \frac{P_g(x)}{P_r(x)} dx.$$

This divergence is also minimized by $P_g(x) = P_r(x)$. However, both losses focus on different things during optimization and usually result in different transformation functions $G(z;\phi)$. The forward KL divergence explodes if for some points x in the domain of the two distributions it holds that $P_r > 0$ and $P_g \approx 0$. So minimizing this loss leads to a dispersed P_g covering the tails of P_r, see figure 3.3.4 for a visual explanation. In practice this often leads to diffuse images over the whole domain of P_r but without covering the probability modes of P_r.
In contrast the reversed KL divergence tends to infinity if $P_r \approx 0$ and $P_g > 0$ over some domain $x \in X$. So minimizing this function leads to a very localized P_g not stretching over the whole viable domain of P_r. In practice this leads to a few good classes of images but not the whole output space is represented. Note that due to its properties, the reverse KL divergence is often

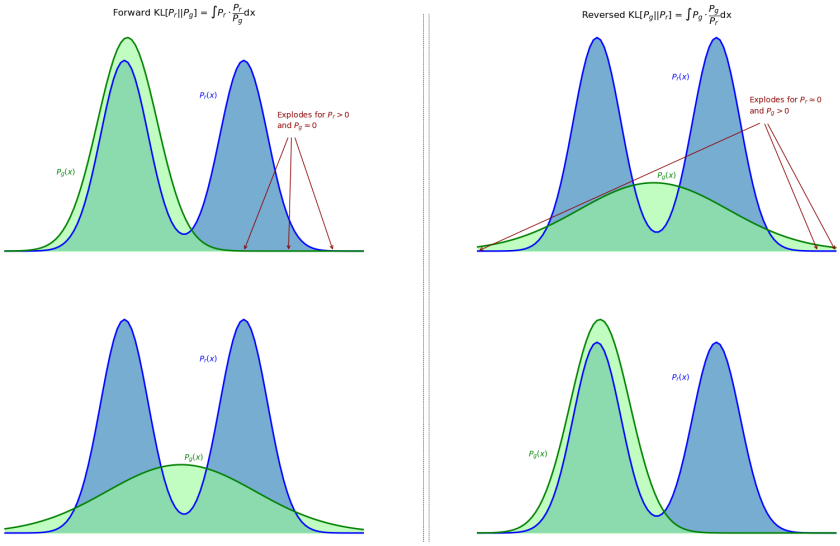


Figure 3.3.4: In blue the true distribution of the data $P_r(x)$ is shown, in green the learned approximation P_g. Left: forward KL-loss explodes when $P_r > 0$ and $P_g \approx 0$, so it creates a stretched P_g covering the tails of P_r. Right: Reverse KL loss explodes when $P_g > 0$ and $P_r \approx 0$, so it creates a localized P_g keeping under the tails of P_g.

called the mode seeking divergence as it focuses the largest mode of P_r. Therefore this loss tends to increase the problem of mode collapse and it prefers to place low probability wherever the data P_r is not likely.
The forward KL is sometimes called the mean seeking divergence due to its centralizing property. It prefers to place high probability at all points in the sample space where their are likely occurrences of the data. In summary the reverse KL divergence usually produces high quality

image with low variance while the forward KL generates images with high variance but sometimes lower quality.
A problem for both KL divergences arises when the two probability distributions P_r and P_g barely share a common support. In this case both losses diverge and the computation of gradients is not possible. This commonly happens during the first few steps of training when the generating distribution P_g is far different than the data distribution P_r.

Jensen-Shannon divergence

The Jensen-Shannon divergence removes some of the disadvantages arising in the case of the KL divergence. First, it's a symmetric loss function treating $P_r(x)$ and P_g on equal footing:

$$\mathrm{JS}(P_r(x), P_g(x)) = JS(P_g(x), P_r(x))$$

One of the main advantages of the JS loss is, that it gets rid of the divergent behavior of the KL loss for two distributions with disjoint supports (especially present during the first few steps of training). In this case the JS divergence is finite and given by

$$\begin{aligned}
\mathrm{JS}(P_r(x)||P_g(x)) &= \frac{1}{2}\left(\mathrm{KL}(P_r(x)||M(x)) + \mathrm{KL}(P_g(x)||M(x))\right) \\
&= \frac{1}{2}\left[\int_x P_r(x)\log\frac{P_r(x)}{\frac{P_r(x)+\cancelto{0}{P_g(x)}}{2}}dx + \int_x P_g(x)\log\frac{P_g(x)}{\frac{\cancelto{0}{P_r(x)}+P_g(x)}{2}}dx\right] \\
&= \frac{1}{2}\left[\int_x P_r(x)\log 2dx + \int_x P_g(x)\log 2dx\right] \\
&= \log(2)
\end{aligned}$$

So this loss converges to a fixed value for distributions which do not share a common support. Unfortunately in this case the gradients converge to zero, as shown in figure 3.3.5. So at the start of training the progress might be extremely slow or not start at all if the two distributions are very far from each other or the discriminator starts to dominate. It is therefore of utmost importance to balance the strengths of the two networks in a delicate way. As in real life, it is easier to spot a fake painting than generating said painting. It is therefore advised to shift the balance of power more in favor of the generator, making it more complex and possibly increasing its learning rate compared to the discriminator. It is not recommended to train the discriminator until convergence as would be desirable from theory but to alternate quickly between updating the generator and discriminator even if it means that the generator updates will be rather imprecise.

Wasserstein distance

The Wasserstein or Earthmover distance can be considered an improvement compared to the JS divergence as it removes its stagnating property at large differences of distributions. The main

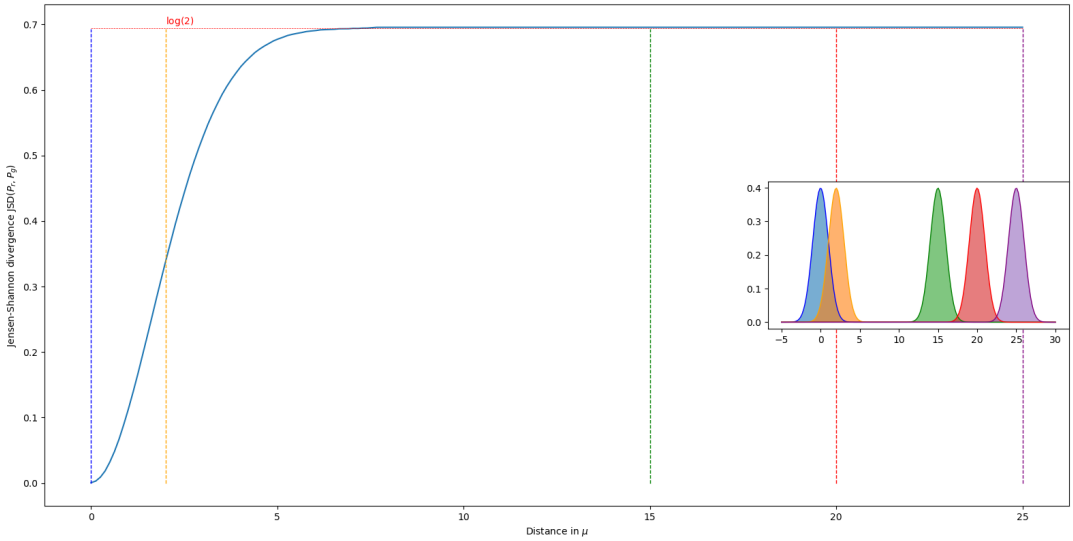


Figure 3.3.5: Stagnation of a Jensen-Shannon divergence for two distributions that are far away slows down training drastically. This occurs frequently at the beginning of training[59].

difference is that it does not measure the distance in y direction over the domain $x \in X$, but it measures directly the distance in x, see figure 3.3.6. The problem with the JS divergence is that for distributions which do not share a common support almost everywhere the loss flattens to $\log 2$ as described in the previous section. So at some point the loss function saturates and therefore training at the start is slow. However, the Wasserstein distance increases when the distance in x direction between two distributions increases, so it does not saturate and training should be fast from the beginning. Also balancing the critic and generator is easier in theory as it is no problem to train the Wasserstein GAN to convergence.

These losses are now compared for a toy example. Consider the following situation of two distributions over $\mathbb{R}^2$ Let the true distribution be $(0,\, y)$, with $y \sim U(0,1)$. Consider the family of distributions P_g, with $P_g = (\theta, y)$ and $y \sim U(0,1)$. We want an optimization algorithm which converts $P_g \to P_0$, which means the metric should support the optimization $\theta \to 0$. If we calculate some of the different divergence measures, we get

- Kullback-Leibler divergence and reverse Kullback-Leibler divergence

$$D_{KL}(P_0||P_g) = \int_{x,y} P_0(x,y)\log\left(\frac{P_0(x,y)}{P_g(x,y)}\right)dxdy$$

$$D_{KL}(P_0||P_g) = D_{KL}(P_g||P_0) = \begin{cases} +\infty & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases}$$

- Jensen-Shannon divergence: Use $M = (P_0 + P_g)/2$.

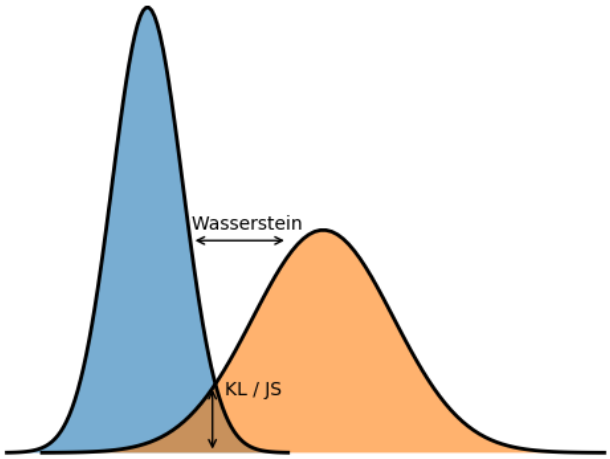


Figure 3.3.6: Different approaches for Wasserstein vs KL and JS divergence.

We have that

$$D_{JS}(P||Q) = \frac{1}{2}(D_{KL}(P||M) + D_{KL}(Q||M)) = \begin{cases} \log(2) & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \\ , \end{cases}$$

because

$$D_{KL}(P_0||M) = \int_{x,y} P_0 \log\left(\frac{P_0}{(P_0+P_g)/2}\right) = \int_{x,y} P_0 \log(2) dxdy = \log(2)$$

- Wasserstein distance: In this case the best way to transfer one distribution to the other is via a straight line, so the cost is given by

$$W(P_0, P_g) = |\theta|$$

So the Wasserstein distance is the only metric allowing for the correct optimization in this simplified situation. For both the KL and JS divergence the cost function is either infinite or constant, which results at best in vanishing gradients.
Of course, this is a constructed example with disjoint supports, but similar things happen in low dimensional manifolds in high dimensional space. Another argument for the Wasserstein distance is the following:
Let P_r be a fixed distribution. Let Z be a random variable with $Z \sim P_z$. Let $G(z;\phi)$ be a deterministic function parameterized by ϕ, and let $P_g = G(z;\phi)$. It holds that[60],

1. if G is continuous in ϕ, so is $W(P_r, P_g)$.

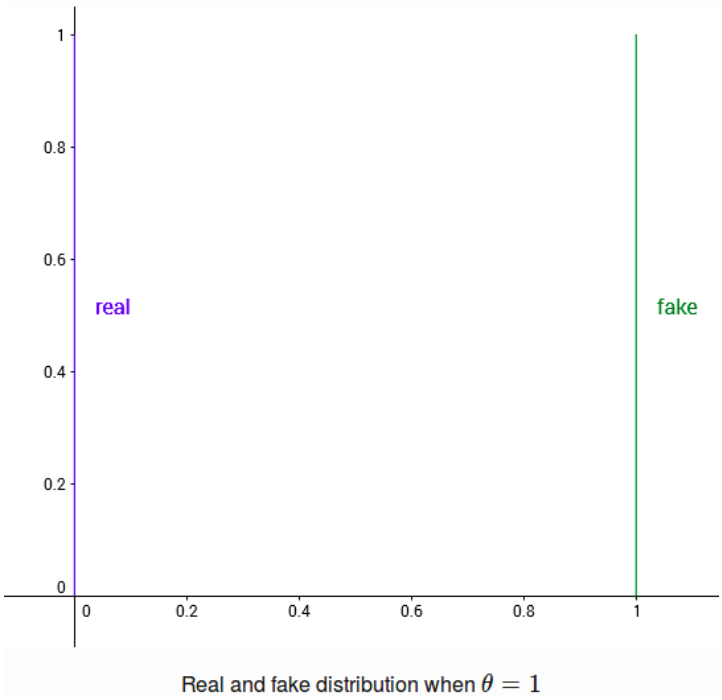


Figure 3.3.7: Toy example for computing different loss functions.[60]

2. if G is sufficiently nice, then $W(P_r, P_g)$ is continuous everywhere, and differentiable almost everywhere.

3. Statements 1-2 are false for the Jensen-Shannon divergence $\mathrm{JS}(P_r, P_g)$ and both the KL divergences.

Another theorem states: Let P be a distribution, and $(P_n)_{n\in\mathbb{N}}$ be a sequence of distributions. Then the following holds true[60].

1. The following statements are equivalent.

   - $\delta(P_n, P) \to 0$ with δ the total variation distance: $\delta(P_n, P) = 0$ if $P_n = P$ and $\delta(P_n, P) = 1$ if $P_n \neq P$.
   - $\mathrm{JS}(P_n, P) \to 0$.

2. $\mathrm{KL}(P_n \| P) \to 0$ or $\mathrm{KL}(P \| P_n) \to 0$ imply the statements in (1)

3. The following statements are equivalent.

   - $W(P_n, P) \to 0$
   - $P_n \to P$, where $\to$ represents convergence in distribution for random variables.

4. The statements in (1) imply the statements in (3)

Together, this proves that every distribution that converges under either KL, reverse-KL, total variance or JS divergences also converges under the Wasserstein divergence. It also proves that a small Wasserstein distance truly corresponds to a small difference in distributions.
To close this section we compare the Wasserstein GAN to the Vanilla GAN proposed in the original paper[61]:

1. In the case of Vanilla-GANs the discriminator maximizes and generator minimizes the loss

$$\frac{1}{m}\sum_{i=1}^{m}\log(D_\theta(x^{(i)})) + \frac{1}{m}\sum_{i=1}^{m}\log(1 - D_\theta(G_\phi(z^{(i)}))). \tag{3.3.28}$$

   This can be shown to be proportional to the Jensen-Shannon divergence.
   In contrast, the f_θ in WGANs are called critics, as they are not required to be in the interval $[0, 1]$. Instead they approximate the Wasserstein distance between latent and data distribution directly via regression and the loss

$$\frac{1}{m}\sum_{i=1}^{m} f_\theta(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_\theta(G_\phi(z^{(i)})). \tag{3.3.29}$$

2. In GANs the discriminator is more often than not too strong and one needs to alternate between D and G before convergence is reached. This is not the case for Wasserstein GANs and f_θ should be trained to convergence to get accurate estimates of $W(P_r, P_g)$. Then the gradient $\nabla_\theta W(P_r, P_g)$ will be more accurate for the generator.

3. Batch normalization and complicated convolutional structures are usually not needed in WGANs.

4. According to the original paper of the GAN architecture the best optimizer to use is the Adam optimizer, while the best one for WGANs is the RMSProp optimizer.

5. Empirically, mode collapse is not such a urgent problem in WGANs

3.4 Limitations

In this part we will discuss some of the difficulties and problems when training generative adversarial networks. In the next section we will describe methods usually applied to support the training process.

Mode collapse

Mode collapse is one of the main problems when training GANs. The discriminator differentiates only between real and generated images, but is not concerned with the labels of these images (in fact a Vanilla GAN is an unsupervised learning method). So it is often easier for a GAN to learn to produce only one single perfect generated image, instead of learning a variety of outputs and generalizing to the whole data set[62], see figure 3.4.1.

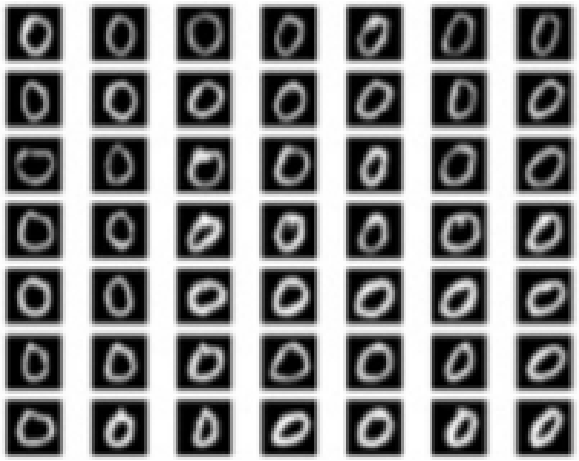


Figure 3.4.1: Mode collapse in the mnist data set.

Nash equilibrium

In its foundation the combined generator-discriminator loss of a GAN is a min-max optimization problem, well known from game theory. One player tries to maximize the objective function while the other aims to minimize the same function. The state where both players achieve this balance without an improvement in either direction is called the Nash equilibrium. However, even if such a point exists in theory which is barely ever guaranteed in a real world example, there are common pathological examples where an optimization algorithm cannot find this equilibrium[63]. Consider the following example of a cost function:

$$\min_x \max_y V(x, y) = x \cdot y \tag{3.4.1}$$

The Nash equilibrium is at $x = y = 0$. This is the only state where neither player can gain or lose when their opponent is changing its value. This problem should be easily solved with any optimization algorithm. However, when using gradient descent the values of x and y will oscillate around their Nash equilibrium without ever reaching it. On the contrary, the cost function will diverge after some time[64], see figure 3.4.2.

Balancing

One of the hardest tasks when choosing the hyper-parameters of a specific GAN algorithm is the balancing of network complexity for generator versus the discriminator / critic. Ad-hoc, it is not clear which combination of networks work at all. A generator performing extremely well with a smaller discriminator might collapse completely or fail to converge when using a larger discriminator. In this work we found that in most cases, the discriminator should have about 1/10 to about 1/2 of the parameters contained in the generator, but even more drastic differences worked under certain circumstances.

Additionally the learning rate might be chosen differently for both networks to control the value of their gradients. In this work the learning rate of the discriminator was often chosen to be one magnitude lower compared to the generator.

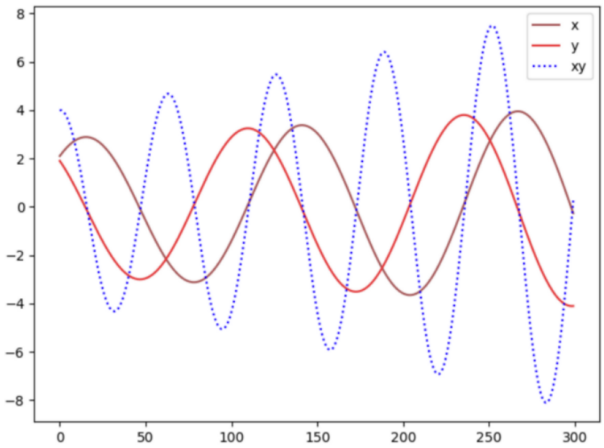


Figure 3.4.2: Example of a failed convergence when computing the Nash equilibrium[63].

Hyper-parameter tuning

Since its inception generative adversarial networks have come a long way. Today there exist countless variations of the original algorithm with new loss functions and architectures arising frequently. It is often hard to decide which loss function and therefore which general structure to use. However, a study published in 2018 tried to determine if there is an obvious best algorithm to use in a general case[65]. What they found is that the most important part about generative adversarial networks is rarely the loss function itself but the tuning of the numerous hyper-parameters. Among those are learning rate, optimization algorithms, general architecture, batch size, training time, size of the latent dimension and many more. By choosing the correct values for each of these parameters might result in a high quality model regardless of the specific loss function being minimized. However, this is no easy task as well as there is and infinite amount of possibilities to try out with every point in this subspace taking a rather long time to be trained.

4 Implementation of GANs

In this section specific algorithms for either single track propagation or calorimeter image refinement are given using the theory from the previous section. Generative adversarial networks consist of two neural networks. In the conditional case the generator can be considered a non-linear function G mapping from the random and conditional subspace to the output with

$$\begin{aligned} G : \mathbb{R}^n \times \mathbb{R}^k &\to \mathbb{R}^m \\ (z, y) &\to x' = G(z, y; \phi) \end{aligned} \tag{4.0.1}$$

and the discriminator is a non-linear function D predicting the label $\hat{l}$ from the output space of the generator and the conditional space:

$$\begin{aligned} D : \mathbb{R}^m \times \mathbb{R}^k &\to \mathbb{R}^1 \\ (x, y) &\to \hat{l} = D(x, y; \theta) \end{aligned} \tag{4.0.2}$$

Every (generative adversarial) network consists of mainly three different building blocks

- **The architecture**: This determines the number of layers used in the network, the number of nodes per layer, the activation function, usage of batch normalization and dropout, etc. It is crucial for the network to be powerful enough to capture the complexity of the data but at the same time is not powerful enough to overgeneralize to the training data without extracting the most important attributes of the data.

- **The loss**: The loss determines what measure of discrepancy between the true and predicted distribution is chosen. The most popular loss functions include the Kullback-Leibler divergence, Jensen-Shannon divergence and Wasserstein divergence. The choice of loss determines how learning behaves in different stages of the process and the quality of the output images.

- **The optimizer**: The optimizer is the implementation of the procedure to achieve the most advantageous loss. Most of the time in machine learning the loss function is highly non-convex and it is infeasible to determine the global minimum of a function. However, even

the localization of local minima is not always easy due to the high-dimensional nature of the optimization space, so different algorithms lead to differing speed and behavior of convergence. Popular choices are variations of the stochastic gradient descent (SGD) algorithm, like the Adam optimizer or RMSProp optimizer.

4.1 Single track propagation

For the single track propagation part of this thesis calorimeter images were created by concatenating the random subspace $z \sim P_z$ with the conditional space y and propagating those through a mostly densely connected network. It was then reshaped into a 2D image format and de-convolutional layers were used to create images of the desired resolution. Different architectures, loss functions and optimizers were tested in order to find a viable solution for the given problem. Note that the implementation of a Vanilla GAN as proposed by the original paper is not fundamentally different from a Wasserstein GAN. The difference mostly consists of a few lines of code exchanging the loss function minimized by the optimizer.

4.2 Image-to-image translation

In this chapter the algorithms trained in the second part of this thesis will be detailed. As described earlier we are concerned with techniques for calorimeter image refinement after generating images with the first network by applying the algorithm on each track separately and adding these images, see again figure 2.2.1 for reference. The second network is used to refine images needed mainly due to three reasons:

1. The first network trains on a π^+ data set and hence does not inherently know how protons, kaons or muons should interact with the hadronic calorimeter.

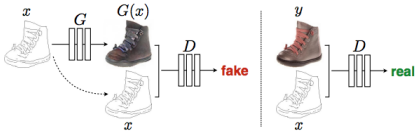
2. All charged particles measured by the tracking system are translated with the conditional single track propagation GAN even though some of them will most likely not show up in the hadronic calorimeter, like electrons and muons.

3. Neutral particles are not measure in the tracking system but might be visible in the calorimeter, like the neutron.

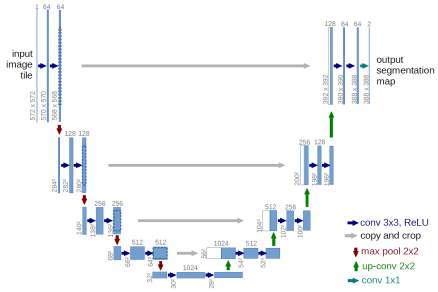
So to summarize the second network needs to be able to remove certain tracks, add new tracks and refine existing tracks.
The theory involved is much the same as for the generative adversarial networks discussed in the previous section. Many of the following algorithms are extended by more auxiliary networks, special architectures for the generator or discriminator and additional terms to the loss function penalizing more types of errors.

4.2.1 Pix2Pix

The Pix2Pix architecture[66] was one of the first proposed image-to-image translation algorithms in the domain of generative adversarial learning. The general architecture is shown in figure 4.2.1a.
The paper introduced two new ideas for improving model training and image quality. First it introduced the PatchGAN architecture for the discriminator greatly increasing the image quality of the generator. Secondly it introduced the U-NET architecture for the generator, see figure 4.2.1b.
The loss function proposed is



(a) Architecture of the Pix2Pix algorithm[66].



(b) U-Net architecture of the generator[66].

$$\mathcal{L}_{\text{Pix2Pix}}(G, D) = \mathcal{L}_{GAN}(G, D) + \mathcal{L}_1^x(G), \text{ with} \tag{4.2.1}$$
$$\mathcal{L}_{GAN}(G, D) = \mathbb{E}_{x\sim P_r(x)} \log D(x;\theta) + \mathbb{E}_{z\sim P_z(z)} \log 1 - D(G(z;\phi);\theta)$$
$$\mathcal{L}_1^x(G) = ||x - G(z;\phi)||_1$$

The authors of the original paper found that including a random noise vector did not help diversifying the output and all stochastic properties of the algorithm were lost. Of course, for some cases the random output is not a necessary condition and this algorithm might work well. In our case, however, we explicitly want to model the random quantum mechanical processes happening in the calorimeter so we don't expect this algorithm to outperform the more tailored solutions to our problem.

4.2.2 VAE-GAN

Let's recall the loss function for the variational autoencoder

$$\mathcal{L}(\theta, \phi) = -\mathbb{E}_{z\sim q_\phi(z|x)} \left[\log P_g(x|z)\right] + KL(q_\phi(z|x)||p_\theta(z)). \tag{4.2.2}$$

The problem with the variational autoencoder is its restrictiveness in the choice of distributions for z and x. This is necessary for the computation of the Kullback-Leibler divergence in the loss. A second disadvantage is the usage of KL divergence itself as it has some undesirable properties when P_g and P_r are barely overlapping, see section 3.3.5 for a detailed discussion. Additionally, the output constructed by the decoder is evaluated by a pixel-wise loss function,

often the cross-entropy or the $L2$-Norm. However, this loss is insufficient for the task of image creation, as it works in stark contrast to human perception of images. Therefore we need a new function to learn an appropriate loss function.
So our goal is to get rid of these disadvantage with an adversarial structure, seen in figure 4.2.2. The generator has two purposes in this architecture. First, it acts as an encoder for the VAE

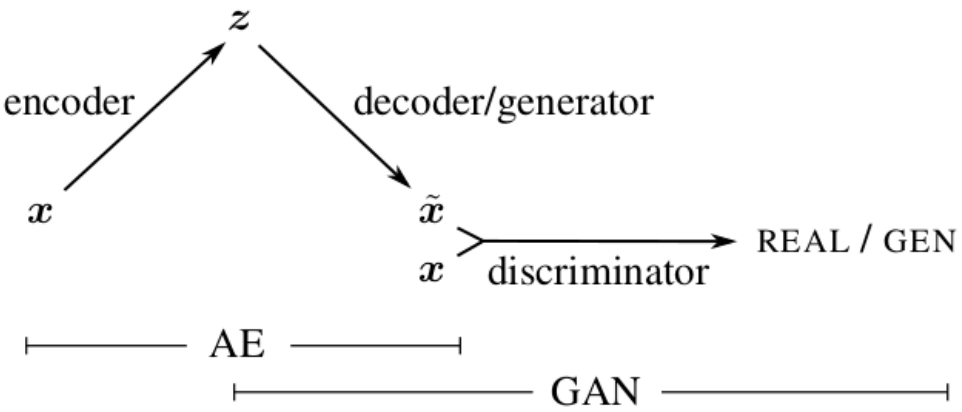


Figure 4.2.2: Architecture of VAE-GAN[37]

structure, but secondly also serves as the generator in the GAN part. Note that in this case $P_z(z)$ can be any distribution without restrictions, but is mostly chosen to be $N(0, I)$.
The algorithm works in two phases:

1. Reconstruction: The variational autoencoder part tries to reconstruct the image as faithfully as possible.

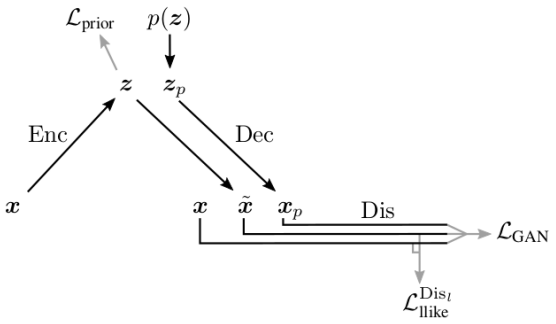
2. Regularization: The GAN imposes JS divergence between $P_r(x)$ and $P_g(z)$. In this step the decoder acts as the generator.

The loss for this architecture[37] is given by

$$\begin{aligned}
&\mathcal{L}_{\text{VAEGAN}}(G, D, E) = \mathcal{L}_{GAN}(G, D) + \mathcal{L}_{VAE}(E) + \mathcal{L}_1^x(G, E), \text{ with} \\
&\mathcal{L}_{GAN}(G, D) = \mathbb{E}_{x\sim P_r(x)} \log D(x;\theta) + \mathbb{E}_{z\sim P_z(z)} \log 1 - D(G(z;\phi);\theta) \\
&\mathcal{L}_{VAE}(E) = \text{KL}(q_\psi(z|x)||P_z(z)) \\
&\mathcal{L}_1^x(G, E) = \mathbb{E}_{z\sim q_\psi(z|x)}||x - G(z;\phi)||_1
\end{aligned} \tag{4.2.3}$$

Note that the last loss term involves the output of the lth layer of the discriminator network. This could potentially be the last layer, but the authors found that using the second to last layer produces a better output. An explanation might be that the last scalar output does not capture enough of the complexity to be used by the autoencoder. This method can be used for other architectures as well and is called feature matching, see section 4.3. Often a Gaussian distribution is assumed for this layer, so a mean squared error loss is used which is its log likelihood function.

The authors of the original paper[37] used a slightly different approach as they found better performance. It takes two fake input images, once as generated from the encoder $E(x;\psi)$ and once as sampled from $P_z(z)$, see figure 4.2.3. The proposed loss function now is slightly altered



Algorithm 1 Training the VAE/GAN model

$\boldsymbol{\theta}_{\text{Enc}}, \boldsymbol{\theta}_{\text{Dec}}, \boldsymbol{\theta}_{\text{Dis}} \leftarrow$ initialize network parameters
repeat
  $\boldsymbol{X} \leftarrow$ random mini-batch from dataset
  $\boldsymbol{Z} \leftarrow \text{Enc}(\boldsymbol{X})$
  $\mathcal{L}_{\text{prior}} \leftarrow D_{\text{KL}}(q(\boldsymbol{Z}|\boldsymbol{X})\|p(\boldsymbol{Z}))$
  $\tilde{\boldsymbol{X}} \leftarrow \text{Dec}(\boldsymbol{Z})$
  $\mathcal{L}_{\text{llike}}^{\text{Dis}_l} \leftarrow -\mathbb{E}_{q(\boldsymbol{Z}|\boldsymbol{X})}\left[p(\text{Dis}_l(\boldsymbol{X})|\boldsymbol{Z})\right]$
  $\boldsymbol{Z}_p \leftarrow$ samples from prior $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
  $\boldsymbol{X}_p \leftarrow \text{Dec}(\boldsymbol{Z}_p)$
  $\mathcal{L}_{\text{GAN}} \leftarrow \log(\text{Dis}(\boldsymbol{X})) + \log(1 - \text{Dis}(\tilde{\boldsymbol{X}})) + \log(1 - \text{Dis}(\boldsymbol{X}_p))$
  // Update parameters according to gradients
  $\boldsymbol{\theta}_{\text{Enc}} \overset{+}{\leftarrow} -\nabla_{\boldsymbol{\theta}_{\text{Enc}}}(\mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{llike}}^{\text{Dis}_l})$
  $\boldsymbol{\theta}_{\text{Dec}} \overset{+}{\leftarrow} -\nabla_{\boldsymbol{\theta}_{\text{Dec}}}(\gamma\mathcal{L}_{\text{llike}}^{\text{Dis}_l} - \mathcal{L}_{\text{GAN}})$
  $\boldsymbol{\theta}_{\text{Dis}} \overset{+}{\leftarrow} -\nabla_{\boldsymbol{\theta}_{\text{Dis}}}\mathcal{L}_{\text{GAN}}$
until deadline

Figure 4.2.3: Proposed architecture with additional fake image (left) and implementation details (right)[37].

to

$$\mathcal{L}_{\text{VAEGAN}}(G,D,E) = \mathcal{L}_{GAN}(G,D,E) + \mathcal{L}_{VAE}(E) + \mathcal{L}_1^x(G,E), \text{ with} \tag{4.2.4}$$
$$\mathcal{L}_{GAN}(G,D,E) = \mathbb{E}_{x\sim P_r(x)} \log D(x;\theta) + \mathbb{E}_{z\sim P_z(z)} \log 1 - D(G(z;\phi);\theta) + \mathbb{E}_{z\sim q_\psi(z|x)} \log 1 - D(G(z;\phi);\theta)$$
$$\mathcal{L}_{VAE}(E) = \text{KL}(q_\psi(z|x)||P_z(z))$$
$$\mathcal{L}_1^x(G,E) = \mathbb{E}_{z\sim q_\psi(z|x)}||x - G(z;\phi)||_1$$

A conditional form of the algorithm exists, seen in figure 4.2.4. This algorithm can then be used for image-to-image translation purposes and is introduced as such in [67].

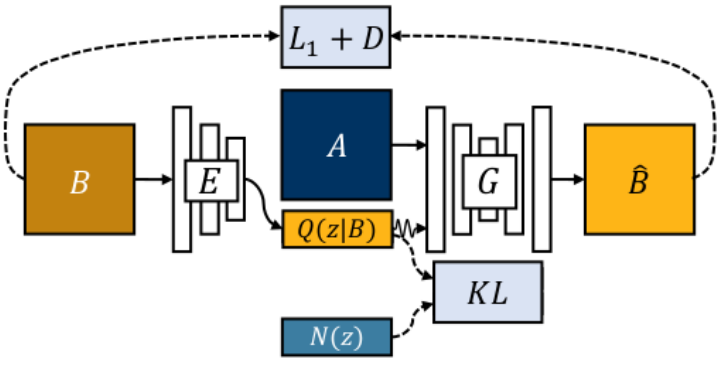


Figure 4.2.4: Architecture for the conditional variant of the VAE-GAN[67].

4.2.3 Conditional latent regressor GAN

The conditional latent regressor GAN, or cLR-GAN for short, is another image to image translation algorithm. However, it mainly introduces the usage of an auxiliary network, designed as an encoder $E(x;\psi)$, which maps a generated output back to its random latent space. This borrows from the idea of Info-GAN[68] and can in principal be used for any network structure. It was specifically designed to tackle the problem of mode collapse[69, 70] because if the generator tries to fool the discriminator by producing one "perfect" example the reconstruction of this image will result in the same latent space vector. This will increase the reconstruction loss in the latent space and is unfavorable for the generator. The architecture is presented in 4.2.5.

A key difference to the VAEGAN is that in this case the encoder produces a point estimate of

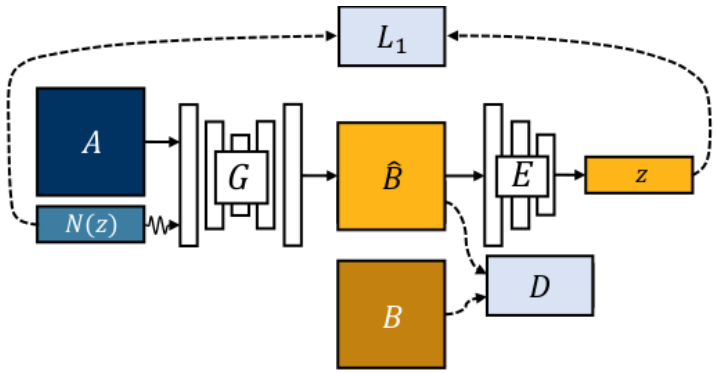


Figure 4.2.5: Architecture of the conditional latent regressor GAN[67].

the latent dimension vector z instead of trying to predict a Gaussian distribution.
The total loss function is given as

$$\begin{aligned}
&\mathcal{L}_{\text{cLR-GAN}}(G,D,E) = \mathcal{L}_{GAN}(G,D) + \mathcal{L}_1^z(G,E), \text{ with} \\
&\mathcal{L}_{GAN}(G,D) = \mathbb{E}_{x\sim P_r(x)} \log D(x;\theta) + \mathbb{E}_{z\sim P_z(z)} \log 1 - D(G(z;\phi);\theta) \\
&\mathcal{L}_1^z(G,E) = \mathbb{E}_{z\sim q_\psi(z|x)} ||z - E(G(z;\phi);\psi)||_1
\end{aligned} \tag{4.2.5}$$

4.2.4 BicycleGAN

The BicycleGAN combines the ideas of the cVAE/GAN and the cLRGAN[67], by using both cycles of these networks. The schematic architecture is shown in figure 4.2.6.

This network utilizes a rather complex loss function combining the GAN loss with two L_1 reconstruction loss terms in both image and latent space latent as well as a KL divergence term. All these terms are weighted against each other to maximize flexibility with the drawback of

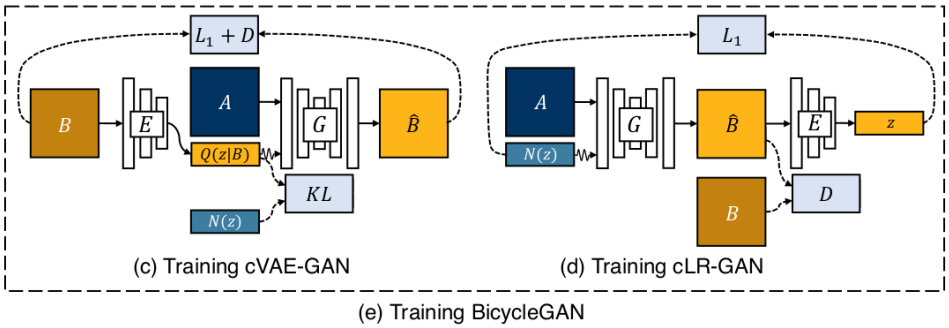


Figure 4.2.6: Architecture of the BicycleGAN[67].

increased hyper-parameter space. The full loss function is given by

$$\begin{aligned}\mathcal{L}_{\text{BiCylce}}(G,D,E) &= \mathcal{L}_{GAN}(G,D,E) + \lambda_x \mathcal{L}_1^x(G,E) + \lambda_z \mathcal{L}_1^z(G,E) \\ &\quad + \lambda_{VAE}\lambda \mathcal{L}_{VAE}(E), \text{ with} \\ \mathcal{L}_{GAN}(G,D,E) &= \mathbb{E}_{x\sim P_r(x)} \log D(x;\theta) + \mathbb{E}_{z\sim P_z(z)} \log 1 - D(G(z;\phi);\theta) \\ &\quad + \mathbb{E}_{z\sim q_\psi(z|x)} \log 1 - D(G(z;\phi);\theta) \\ \mathcal{L}_1^x(G,E) &= \mathbb{E}_{z\sim q_\psi(z|x)} ||x - G(z;\phi)||_1 \\ \mathcal{L}_1^z(G,E) &= \mathbb{E}_{z\sim q_\psi(z|x)} ||z - E(G(z;\phi);\psi)||_1 \\ \mathcal{L}_{VAE}(E) &= \text{KL}(q_\psi(z|x)||P_z(z))\end{aligned} \tag{4.2.6}$$

4.3 Improving components

In this chapter we will discuss some of the most popular techniques for preventing some of the points of failures discussed in the previous sections. For many of these techniques it is not 100% known why they work so well, but many practitioners are content with the fact that they help preventing mode collapse and improve image quality.

Minibatch discrimination

One of the major problems while training any GAN is mode collapse. The networks might learn to produce one single high quality image regardless of the input to fool the discriminator. One possible remedy is to provide the discriminator with information about a whole batch of true and generated images. It should then recognize that it is fairly unlikely that a set of almost identical images come from the true data distribution P_r[71].
Unfortunately it is not always straight forward to use minibatch discrimination from a theoretical point of view. We will show in the following paragraph that the usage of minibatch discrimination on a Kullback-Leibler divergence (as used by a VAE or generalized JS Divergence from GAN, see section 3.3.2) does not change the underlying loss function and hence its interpretation remains unchanged. However, it still may improve training by stabilizing the algorithm and reduce the

variance. For the classical Jensen-Shannon divergence or Wasserstein loss the implications of using minibatch discrimination are not as clear.
Minibatch discrimination is interested in the joint probability distribution of multiple examples

$$P^{(N)}(x_1, ..., x_N) = \prod_{n=1}^{N} P(x_n). \tag{4.3.1}$$

The resulting algorithm will minimize the divergence

$$\mathrm{KL}^{(N)}[P|Q] = \mathrm{KL}[P^{(N)}|Q^{(N)}]. \tag{4.3.2}$$

Assume now for simplicity a minibatch of size $N = 2$:

$$\begin{aligned}
\mathrm{KL}^{(2)}[P|Q] &= \mathrm{KL}[P^{(2)}|Q^{(2)}] \\
&= \mathbb{E}_{x_1\sim P, x_2\sim P} \log \frac{P(x_1)P(x_2)}{Q(x_1)Q(x_2)} \\
&= \mathbb{E}_{x_1\sim P, x_2\sim P} \log \frac{P(x_1)}{Q(x_1)} + \mathbb{E}_{x_1\sim P, x_2\sim P} \log \frac{P(x_2)}{Q(x_2)} \\
&= \mathbb{E}_{x_1\sim P} \log \frac{P(x_1)}{Q(x_1)} + \mathbb{E}_{x_2\sim P} \log \frac{P(x_2)}{Q(x_2)} \\
&= 2\mathrm{KL}[P|Q]
\end{aligned} \tag{4.3.3}$$

So in full generality we have[72]

$$\mathrm{KL}^{(N)}[P|Q] = \mathrm{KL}[P^{(N)}|Q^{(N)}] = N \cdot \mathrm{KL}[P|Q]. \tag{4.3.4}$$

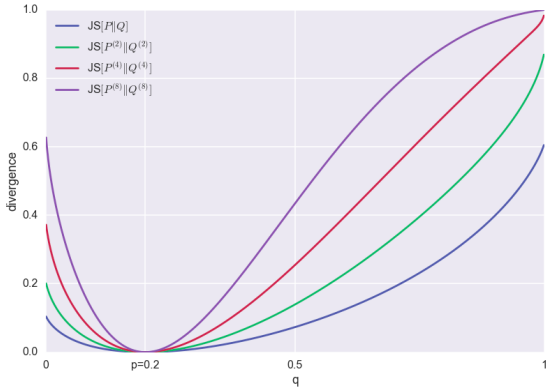
Changing the discriminator to differentiate between minibatches leaves the loss function unaltered and one still performs approximate gradient descent on a Kullback-Leibler divergence.
The same thing does not hold true for the Jensen-Shannon divergence:

$$\mathrm{JS}[P^{(N)}|Q^{(N)}] \leq N \cdot \mathrm{JS}[P|Q]. \tag{4.3.5}$$

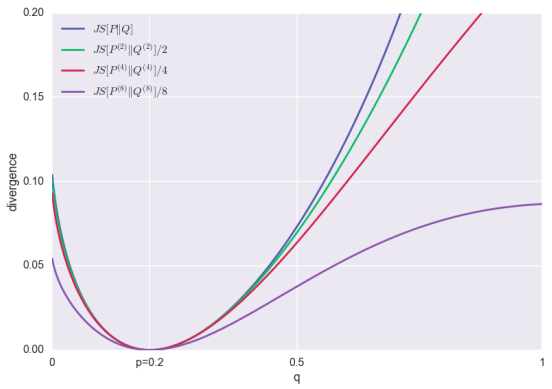
In fact for fixed P and $JS[P^{(N)}|Q^{(N)}]/N$ is strictly non-increasing. Plotting the normalized and un-normalized JS Divergence for two Bernoulli distributions with one fixed parameter p=0.2 and varying q results in[72]. [73] suggests that minibatch discrimination does not improve or change the outcome of the algorithm when using ADAM as optimizer.

Feature matching

Feature matching is a method to modify the generator loss in order to remove the explicit Nash process where two participants try to modify the same loss function. Instead of minimizing the usual loss function the generator aims to match explicit features of a real image. This sometimes helps to make the training process more stable and prevent mode collapse[61].
This is most easily accomplished by matching an intermediary layer f of the discriminator



(a) Un-normalized JS divergence for two Bernoulli distributions with fixed probability parameter for one of them[72].



(b) Normalized JS divergence for two Bernoulli distributions with fixed probability parameter for one of them[72].

between true and generated samples, see figure 4.3.2. Normally the generator loss is changed to the L_2 norm squared of the second-to-last layer between true and fake samples.

$$\mathcal{L}_{G,fm} = ||\mathbb{E}_{x \sim p_{data}} f_l(x) - \mathbb{E}_{z \sim p_z(z)} f_l(G(z))||_2^2 \tag{4.3.6}$$

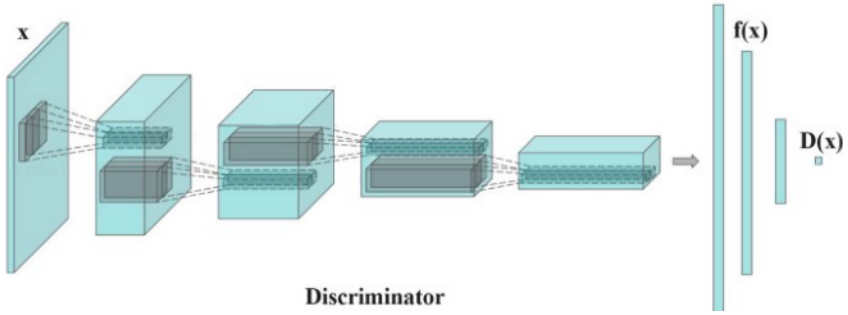


Figure 4.3.2: Feature matching loss to prevent the explicit Nash process[61].

PatchGAN

The PatchGAN is again a method to improve the differentiation power of the discriminator. It was originally introduced for the Pix2Pix[66] architecture explained in section 4.2.1. Instead of predicting labels for true or fake images as a whole a PatchGAN produces a multidimensional output predicting the authenticity of a whole NxN input section of the original image. In theory, this should help generating more realistic images over the whole image space as every NxN patch is checked for plausibility.

Other

Other techniques include

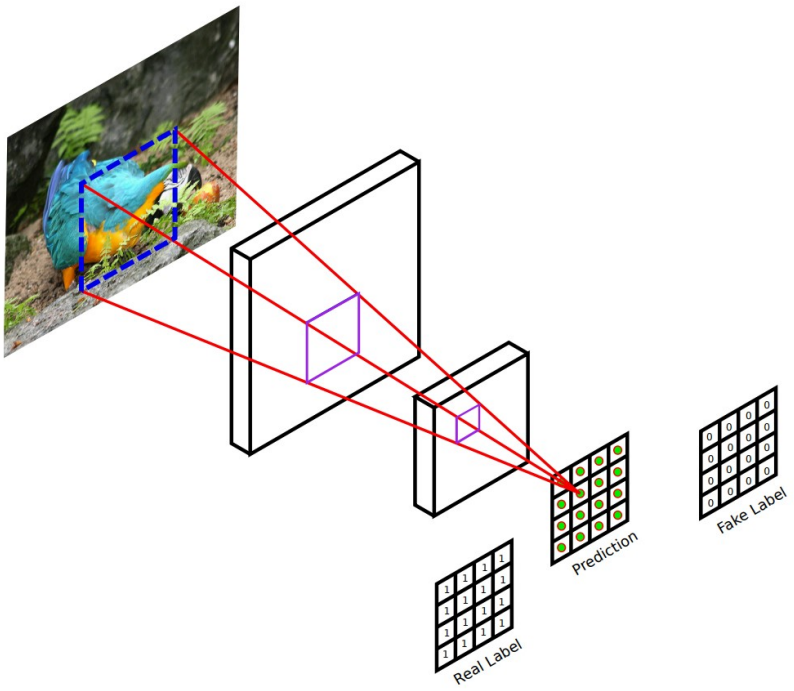


Figure 4.3.3: General architecture of a PatchGAN. The important difference to a conventional discriminator is the multi-dimensional output layer[74].

- Spectral normalization for WGAN to enforce Lifschitz continuity with less computational resources than Gradient Penalization. It can also be used for other networks to help with exploding and vanishing gradients[75].
- Different number of steps and learning rates for discriminator and generator.
- (Virtual) batch normalization and dropout to reduce overfitting[76][71].
- Stopping the loss to saturate by forcing the discriminator to map real images to a label < 1 called one-sided label smoothing[71].
- Residual blocks help to increase the depth and complexity of the generator by concatenating earlier layers of the networks with layers further down the learning path[77].

We refer to the dedicated papers at this point to not further prolong this explanatory chapter.

5 Results

In this chapter I will review the results of this thesis. First I will discuss how the resulting images were evaluated and by which standards they were measured. After that I will discuss applying these metrics the quality of image generation for the conditional generative adversarial networks. After their potency to create images in the calorimeter to a reasonable degree is established I discuss the results of the calorimeter-refinement task. In the end I'll compare the results to direct tracker image propagation algorithm.

5.1 Evaluation metrics

There are no obvious metrics to analyze the image quality for an arbitrary problem. However, as our problem of creating a HCAL response is rooted in physics some desirable physical properties can be used which should be present in the images to describe the given situation. The images will be evaluated using the following metrics, where (i, j) denotes the the pixel in the ith row and jth column of the kth image:

- Total energy per image:

$$E_k = \sum_{ij} E_{T,ijk} \tag{5.1.1}$$

- Maximum energy per image:

$$E_{\max,k} = \max_{ij} E_{T,ijk} \tag{5.1.2}$$

- Number of activated cells defined by pixels containing more than 6MeV of energy (cell trigger minimum):

$$N_k = \sum_{ij} I[E_{T,ijk} > 6MeV] \tag{5.1.3}$$

  where I is the indicator function being one if the condition inside the bracket is true and zero otherwise.

- The energy resolution by subtracting the total energy per image from the tracker input energy, called *real_ET* in the root tuple:

$$r_k = \texttt{real\_ET}_k - E_k \tag{5.1.4}$$

Additionally, I will investigate in the first part of the single track propagation the center of energy given by

$$\mathrm{COE}_k = \frac{\sum_{ij} E_{T,ijk} \cdot i}{\sum_{ij} E_{T,ijk}}. \tag{5.1.5}$$

For the full event with multiple tracks this metric loses its meaning. Instead HTOS and HTIS rates are introduced, which are not sensible in the first part of the project where only signal particles without an underlying event are present.

5.1.1 HTIS / HTOS

The hadronic trigger-on-signal (HTOS) and trigger-independent-of-signal (HTIS) are two important quantities measured in the calorimeter. Its is defined by sliding a 2x2 window over all adjacent detector pixels, corresponding to a convolutional operation with kernel size equal to two and a stride of one. If the sum of these four pixels exceeds a threshold value of 3.2 GeV the L0-trigger is fired and the image is used for further processing. It is of great interest if these pixels correspond to a signal particle or tracks of the underlying event. On the left column of figure 5.1.1 tracker images with three green squares are shown. Their centers mark the three signal particles of this particular process (K, π^+, π^-) obtained after particle identification through the full event reconstruction by Geant4. The side length of the squares is 7 pixels.
In the center image of figure 5.1.1 the corresponding calorimeter image is shown with the triggered 2x2 cells. Not all images in the data set triggered due to the hadronic calorimeter so there are images which contain no red squares.
On the right of figure 5.1.1 the center image of the calorimeter is overlapped with the green rectangles of the left image. There are three possibilities for an image triggered in the calorimeter:

1. All red squares intersect with one or more green squares → HTOS
2. No red squares intersect with any green square → HTIS
3. At least one red square intersects with a green square and at least one red square does not → HTOS & HTIS

Note that the granularity in the outer detector is halved compared to the inner detector still four adjacent cells are summed up. Therefore the trigger cells in the outer detector are four times larger than in the inner one, see figure 5.1.1.

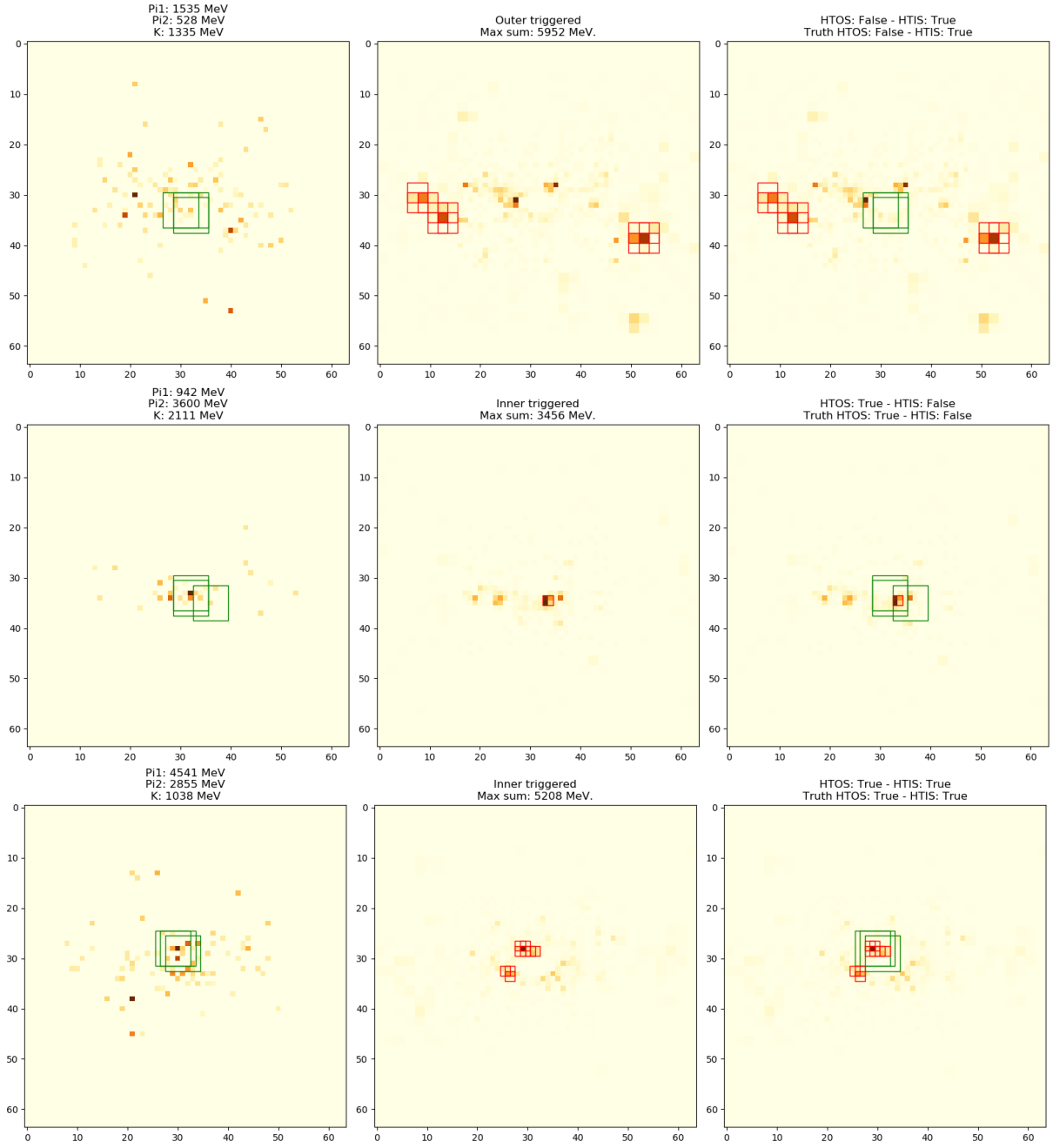


Figure 5.1.1: The left column shows a tracker image with a frame around the three signal particles. The middle column shows a calorimter image with a frame around the triggered group of cells. The right image overlays the second image with the green frames of the first image. If there is an overlap between the red and green rectangles it is considered HTOS, else TIS. Both are possible at the same time if multiple groups of cells get triggered in the calorimeter (see bottom image).

5.2 Single track propagation

This part focuses on the evaluation of the single track propagation algorithm. As previously explained there are a lot of hyper-parameters to adjust and a lot of regularization methods to choose from. A table of hyper-parameters is shown in appendix B. A subset of configurations was chosen by random sub-sampling of the parameter space. All networks were evaluated and compared by using the metrics described in the beginning of this chapter. Note that a lot of networks performed rather well and the final decision to choose exactly one of those is rather subjective. However, it is not of the utmost importance to have the objectively best network at this stage. Firstly, because their probably does not exist an objectively "best" network as all of

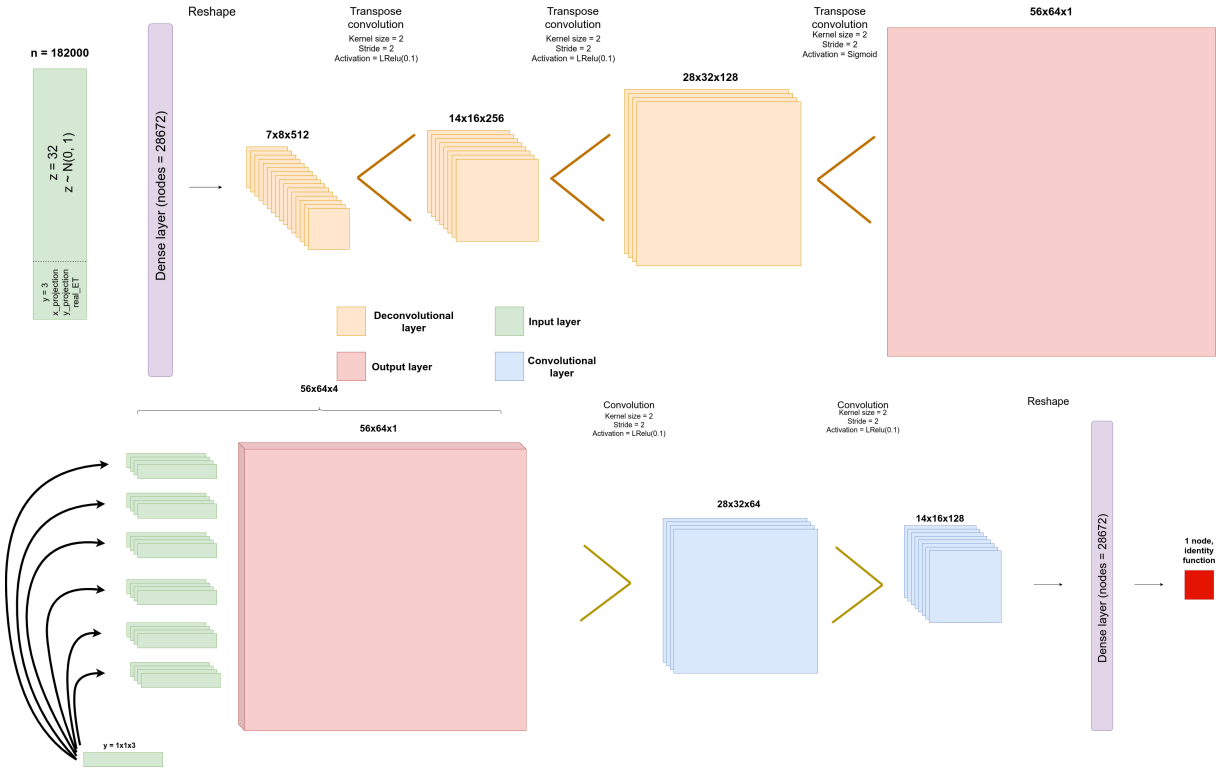


Figure 5.2.1: Conditional generative adversarial network for single track propagation and its set of hyper-parameters. The generator is shown in the upper half and the critic is displayed in the lower half of the figure. The network was trained with the Adam optimizer with a learning rate of 0.0001. For every generator training step the critic was trained for five steps. As suggested, no batch normalization or drop out has been used due to the Wasserstein loss with gradient penalization.

them have some (dis)advantages. Secondly, because the output of this network should only be an approximation serving as input for the second calorimeter image refinement algorithm. The final networks and its parameters are shown in figure 5.2.1 and details are given in appendix C.1. It was trained using the Adam optimizer with a learning rate of 0.0001 for both the generator and the critic. A batch size of 32 was chosen even though no significant drop in image quality

was observed for another choice of this parameter. As the Wasserstein loss was used the critic was trained five times for every step in the generator optimization. The latent dimension of the random input vector z was chosen to be 32 in order to keep the number of parameters relatively low but still allowing for great complexity in the encoding of the image variation.
This network can now be used to generate calorimeter images by passing a vector of size three with components (x, y, E_T) into the generator containing the x-projection, y-projection and transverse energy as predicted by the tracker. Six examples are shown in figure 5.2.2. On the left are the Geant4 generated images, on the right the result after passing the vector into the conditional generative adversarial network. A first visual inspection shows promise in this

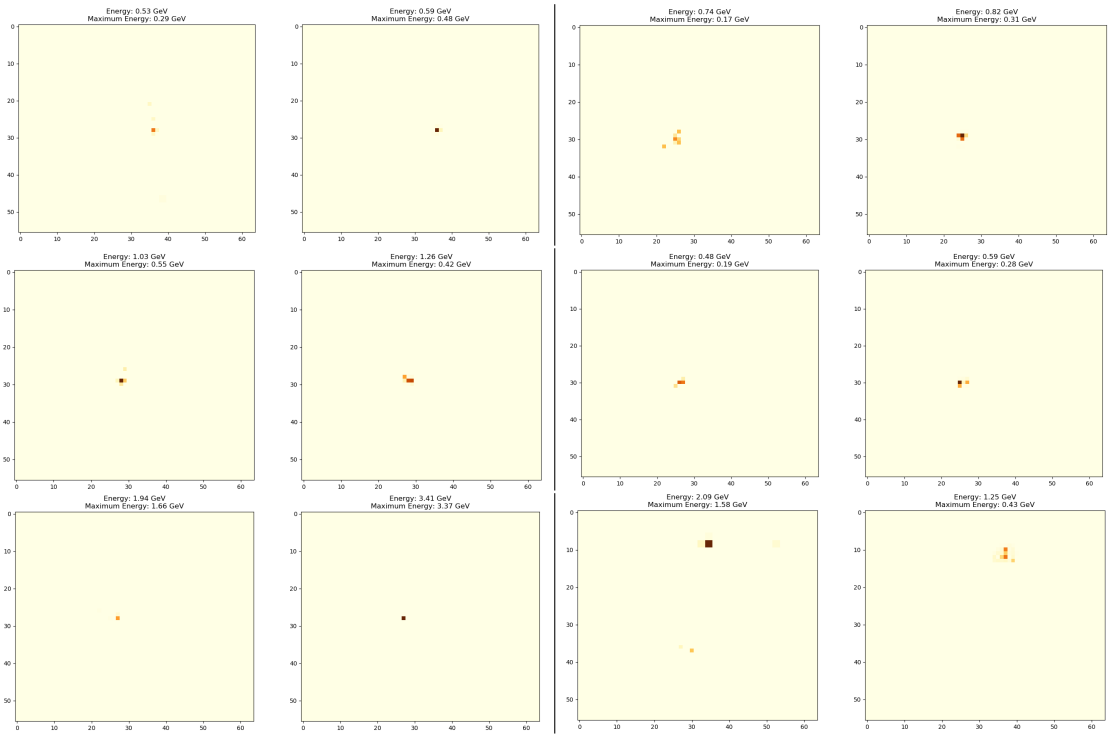


Figure 5.2.2: Exemplary results for the single track propagation .

algorithm. However, some more evidence is needed to further prove the validity of this approach by looking at the metrics of multiple generated images over a broader energy range. In figure 5.2.3 the metrics explained at the beginning of this chapter are evaluated for 20'000 test images not used during training of the GAN. Again, the difference between the Geant4 and cGAN images are not that large. Especially the center of energy and maximum energy seem to be reproduced very nicely. There are some discrepancies between the distribution of the energy and number of activated cells. The former seems to be skewed further to zero for the cGAN images compared to the Geant4 results. The latter, on the other hand, seem to be skewed to the right.
Another important property of the generator is the image variety reproduced by the cGAN. One single input triplet (x, y, E_T) should not result in one deterministic image as this would not

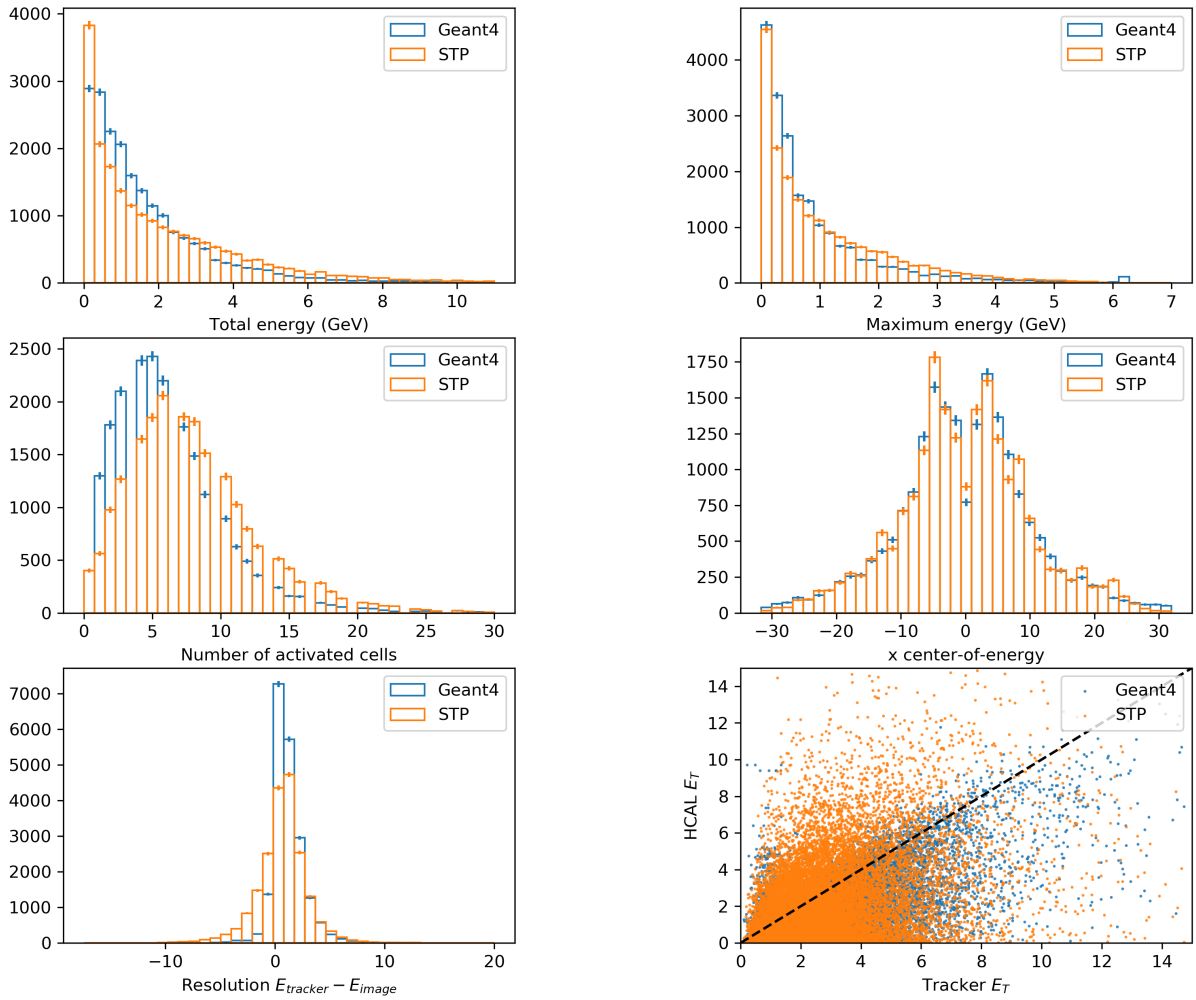


Figure 5.2.3: Results when passing single π^+ into the conditional generative adversarial network.

reflect the underlying stochastic nature of the process. Our cGAN, however, is able to generate multiple different images very efficiently from one input by sampling a random noise vector $z_i \sim N(0,1)$, concatenating it with the conditional input and passing it through the generator. These images are compared to the Geant4 procedure. Ideally a data set where one specific event was processed multiple times by Geant4 to create different calorimeter images for a single input would be available. However, this is not the case but this problem can be circumvented by utilizing the angular symmetry of the underlying physics. Neglecting asymmetry in detector efficiencies simulated by Geant4 pions of approximately the same energy within the same interval of the polar angle $\theta \in [\theta_0 - \delta_\theta, \theta_0 + \delta_\theta]$ should share the same characteristics. In figure 5.2.4 the result for a single event is shown. 1'000 images were generated with the cGAN and filtered for pions with a polar angle $\theta \in [\theta_0 - 0.01, \theta_0 + 0.01]$ and transverse energy $E_T \in [0.9 \cdot E_{T,0}, 1.1 \cdot E_{T,0}]$. These reference events are shown overlapped in the leftmost image of the central row in figure 5.2.4. The distributions match to a reasonable degree except for the energy. However, it is not clear if the images as generated by the cGAN even should match those of Geant4 for this step.

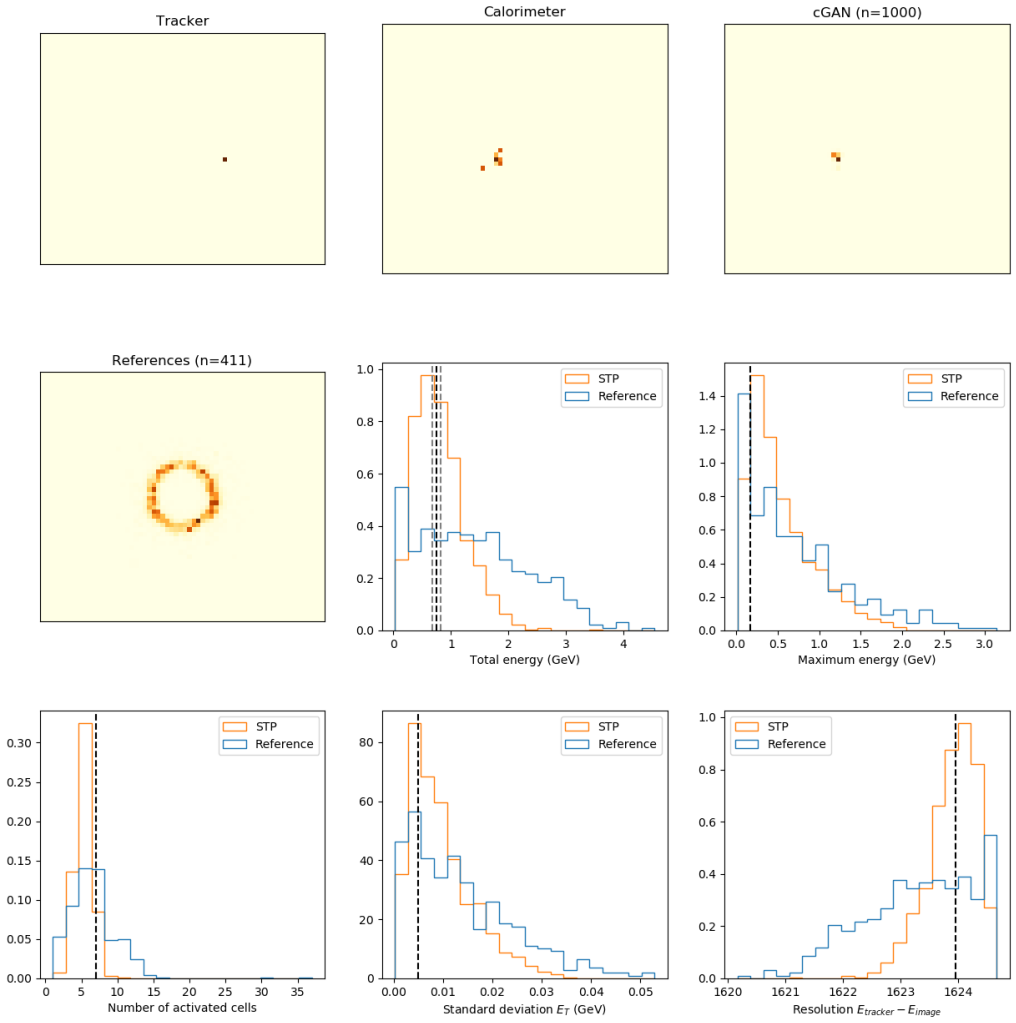


Figure 5.2.4: Passing one vector into the calorimeter image refinement GAN 1000 times to check for event variety. I expect a spread of a fairly great variety due to the stochastic nature of the process. The reference images where chosen due to symmetry reasons.

It might be that the cGAN learned a better approximation to the true image variety by looking at all images. It is a well known problem that reweighing via control channels is necessary when using Geant4 for the full track reconstruction, so the reconstruction process is known to be inherently imprecise.

5.3 Calorimeter image refinement

In this section I will explore the results of the calorimeter image refinement algorithm. It should reproduce the calorimeter image of a full event as well as possible. Similarly to the cGAN algorithm explained in the last section a lot of different hyper-parameters need to be tested and tuned. A table containing the hyper-parameters is shown in appendix B. Again it would be futile to test for all possible combinations of hyper-parameters because this would result in millions of different models to fit. Instead a random sub-sample of the parameter space was used and refined over multiple iterative steps.
Again well over a dozen networks resulted in reasonable approximations of the Geant4 simulation and the decision to choose one of the networks is highly subjective. After close inspection of all relevant metrics, generated images and the image variety it was decided to choose the network shown in figure 5.3.1. Details are given in appendix C.2. The best images were created by the Bicycle-GAN architecture consisting of a generator, a discriminator and an encoder. The encoder should help preventing mode collapse by reproducing the latent space vector z from the generated image and penalizing absolute differences between the true and reconstructed latent space vector. Additionally, a L_1 norm is applied to the pixel-wise difference between the Geant4 and calorimeter image refinement GAN calorimeter responses. The number of generator training steps was equal to those of the discriminator because the cross-entropy loss was chosen and fitting the discriminator to saturation will stagnate the training process of the generator. The weights for the different terms displayed in equation 4.3.1 were chosen to be $\lambda_x = 0.5, \lambda_z = 1.0$ and $\lambda_{VAE} = 0.1$. A one-sided label smoothing of 0.9 was used to prevent the stagnation of the Jensen-Shannon divergence. The architecture of the discriminator is based on the VGG architecture published in [78] but it utilized the PatchGAN output with a 2x2 window. The generator borrowed aspects of the U-Net structure by concatenating the output of previous layers with later ones. The network was trained with the Adam optimizer with a learning rate of 0.00001 for the generator and 0.000001 for the discriminator.
It is note-worthy that the Wasserstein loss did not work well for the calorimeter image refinement task, but the Kullback-Leibler loss for generative adversarial networks worked nearly as well as the binary cross-entropy.

Some exemplary images are shown in figure 5.3.2. On the left the images as projected by the tracker are shown. The next image displays the image after using the single track propagation cGAN by applying it to single tracks of the full event and overlapping the resulting images to obtain one calorimeter response, visually described in figure 2.2.1. The next image shows the true simulation generated by Geant4 our approach tries to imitate. It should directly be compared to the following image in column four. Here the output of the Bicycle-GAN using the cross-entropy loss is shown. The last column displays the output of the secondary approach of tracker image propagation by using the first column directly as input to reproduce the third column.

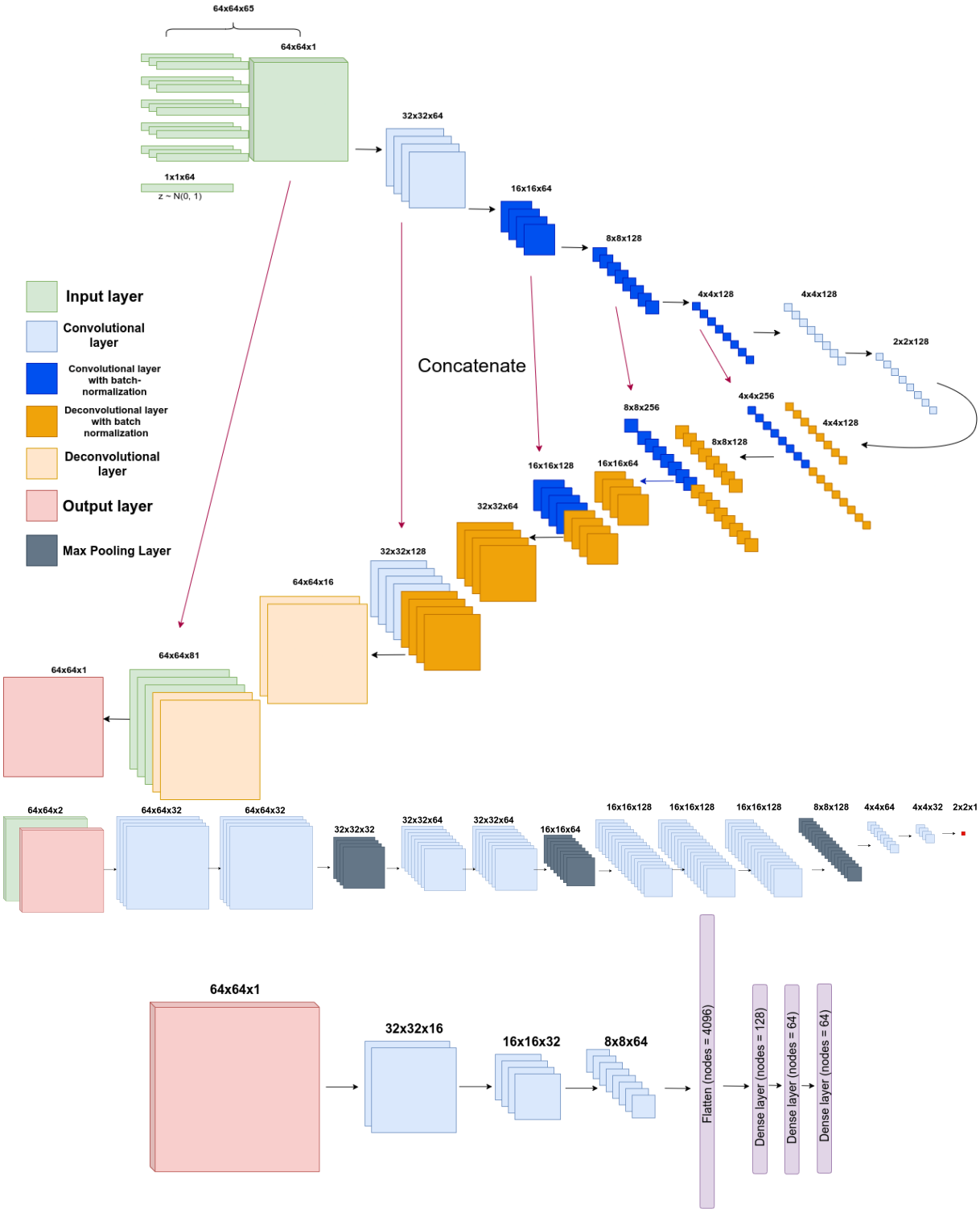


Figure 5.3.1: Image-to-image generative adversarial network and its set of hyper-parameters. The generator is shown in the upper two thirds of the figure, the critic is displayed in lower third and the encoder is shown at the bottom.

The examples shown look qualitatively rather good for the calorimeter image refinement and tracker image propagation GAN. However, it is even more difficult to assess the image quality for single images in this case than the previous single pion simulation task. Therefore I need to examine figure 5.3.3 to get a clearer picture of the quality of generated images. Displayed are again the metrics used previously for single track simulation. These metrics are displayed for the simulation created by Geant4 (blue) as well as the calorimeter responses generated by the calorimeter image refinement (green) and single track propagation (orange). Note that all metrics are significantly improved by using the refinement of the calorimeter image refinement GAN and single track propagation is not sufficient to capture all statistical properties of the full event. Especially the metrics for the total energy, the maximum energy per image and the number of activated cells are better than by using a simple track-wise generation process.

These results are compared to the tracker image propagation GAN which takes the image as generated with the 2D tracker information and directly translates it into the calorimeter response. The architecture of this direct GAN was comparable to the calorimeter image refinement GAN, see appendix C.3. Four additional plots are displayed in the bottom rows of this figure evaluating the HTOS and HTIS rates across the total energy range. For most of the evaluated metrics the direct approach seems to do slightly better. Especially the distribution of the number of cells is shifted in the correct direction, but overfits the mean. The HTIS rate is approximated better with the calorimeter image refinement while on the contrary the HTOS rate is reproduced better by using the tracker image propagation approach. Note that less time was spend on perfecting the generation of images with the direct approach so a better set of hyper parameters might very possibly exist.

Lastly the image variety of the Bicycle-GAN is investigated. In the exemplary event shown in figure 5.3.5 the results for both the tracker image propagation GAN and the two step process are shown.

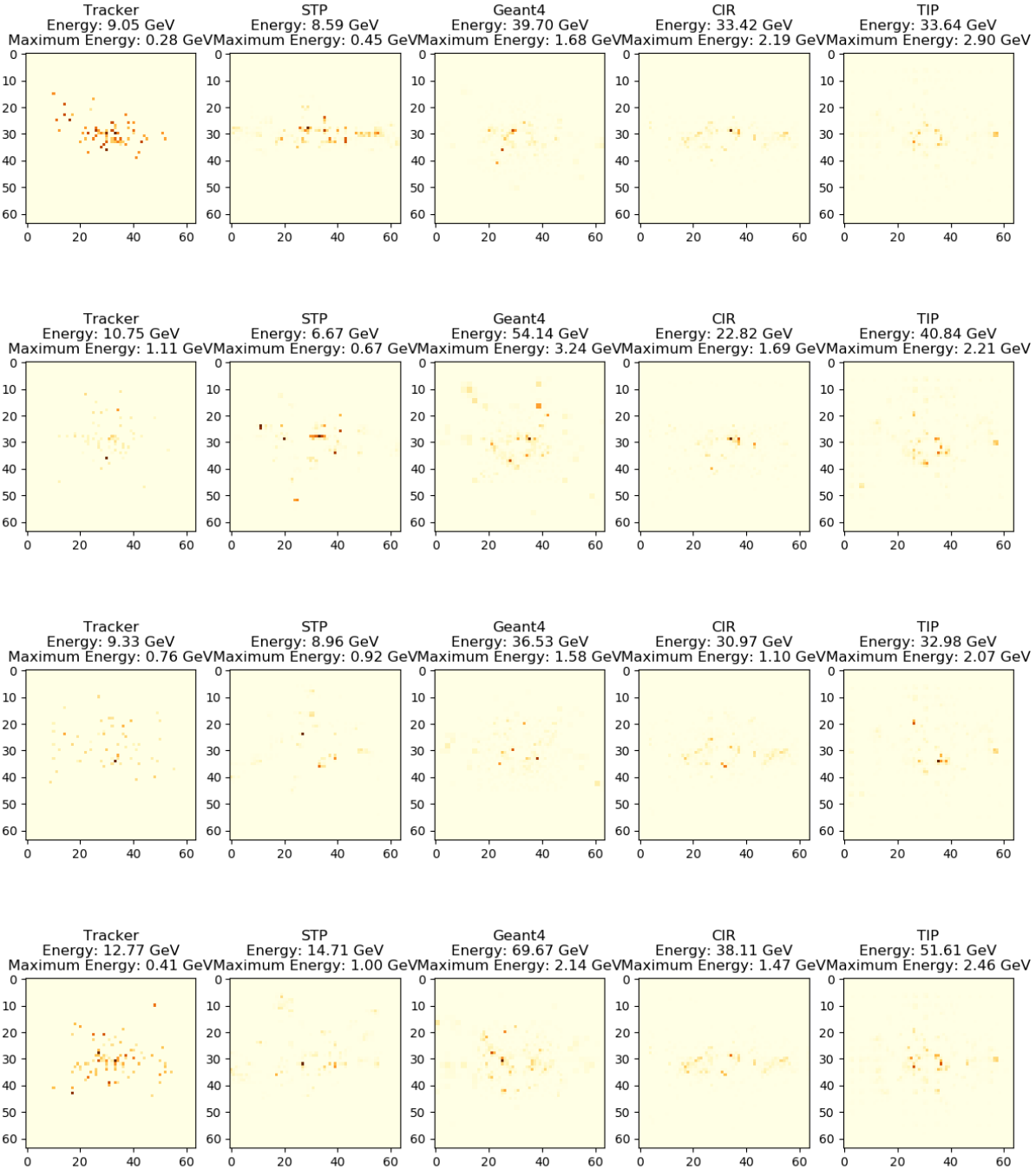


Figure 5.3.2: Exemplary results for the calorimeter image refinement . From left to right: Tracker, single track propagation GAN, Geant4, calorimeter image refinement GAN, tracker image propagation GAN.

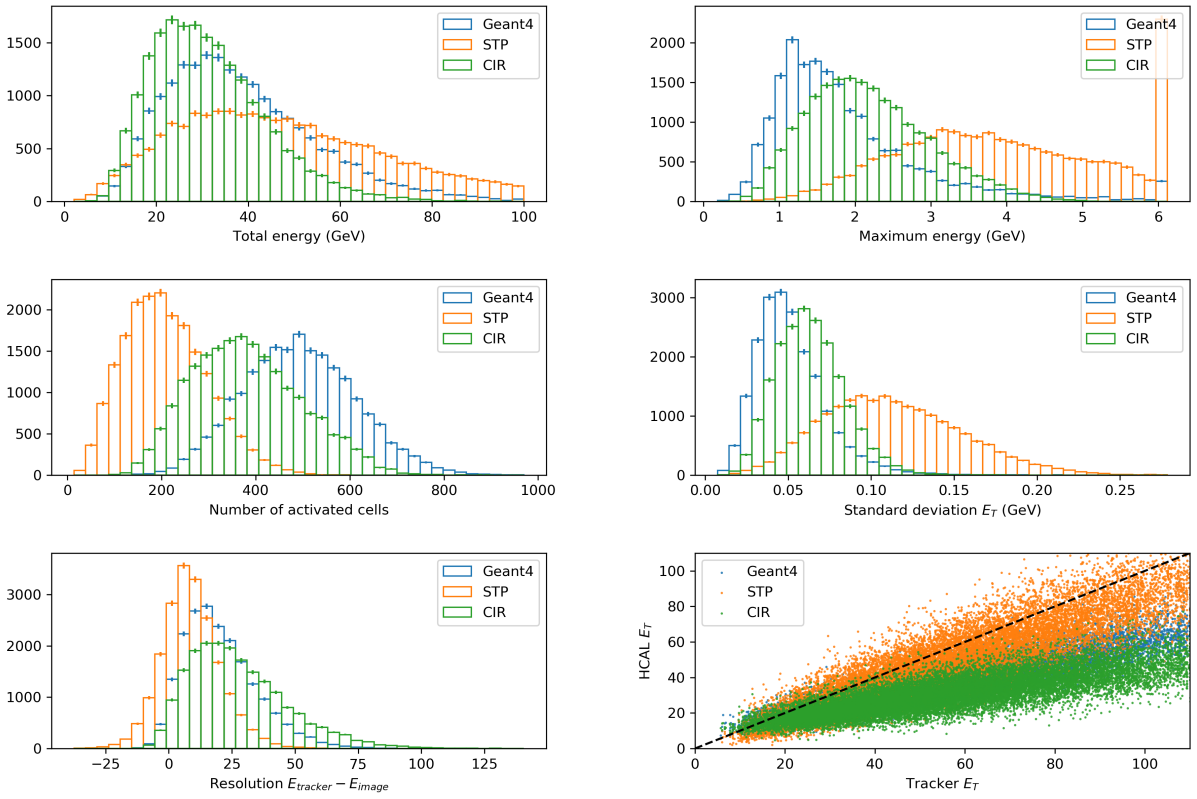


Figure 5.3.3: Results when passing the full event into the conditional generative adversarial network (orange) and after using the calorimeter image refinement network (green). In blue are the metrics when analyzing the Geant4 output.

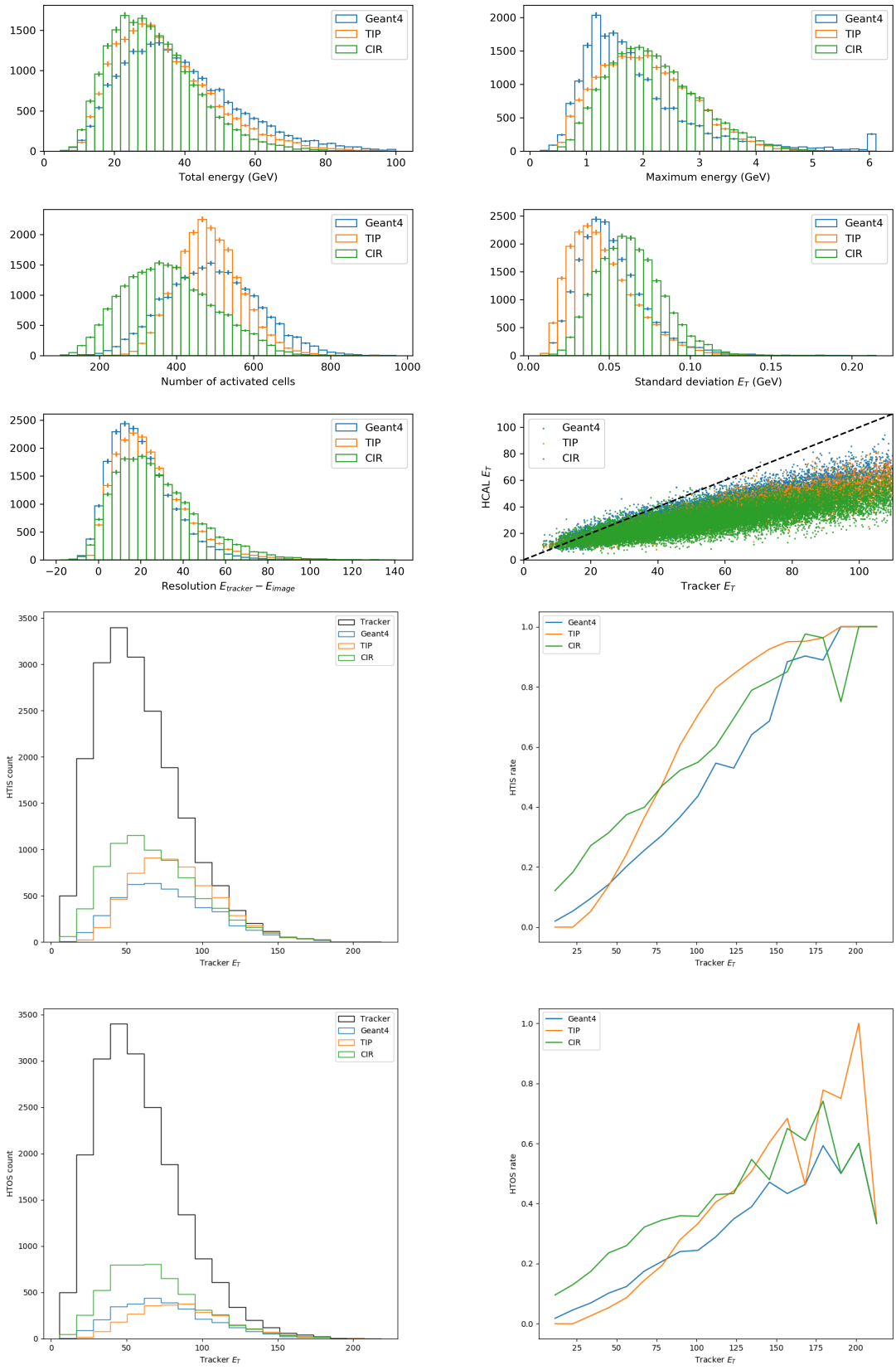


Figure 5.3.4: Results when passing the full event into the direct calorimeter image refinement GAN (orange) and after using the calorimeter image refinement network (green).

The two images on the right display the mean over 1'000 simulations when using the two step approach, and the tracker image propagation approach respectively. The latter seems to capture the distribution of a single event better than the calorimeter image refinement GAN for this particular example.

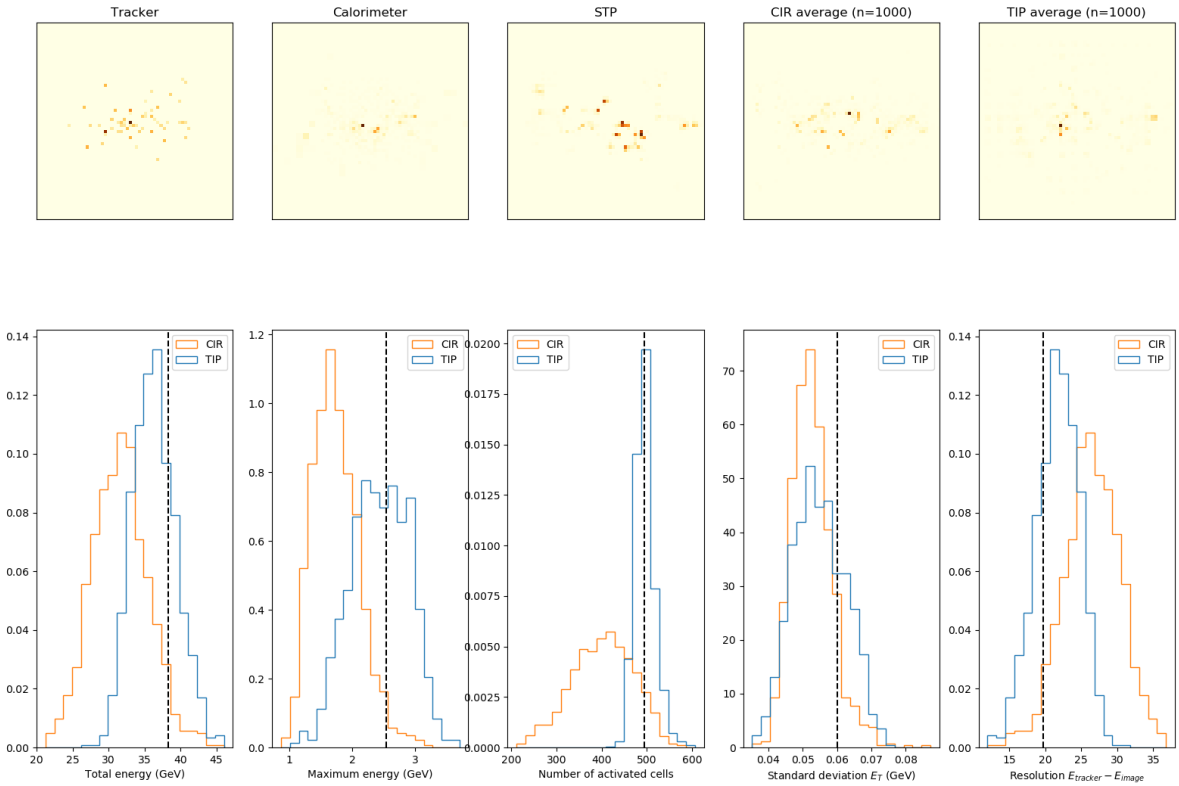


Figure 5.3.5: Passing one image into the calorimeter image refinement GAN and DirectGAN 1000 times to check for event variety. I expect a spread of a fairly great variety due to the stochastic nature of the process.

5.4 Timing

Now that the quality of the generated images is established the time necessary for creating 50'000 images by using either the single track propagation GAN followed by the calorimeter image refinement GAN and the direct tracker image propagation GAN is compared. The tests were performed in two different scenarios to investigate the scalability of these approaches. Firstly by using one Nvidia GTX 950M GPU[79] on the authors PC, secondly by using one of the two Nvidia Tesla P100 PCIe[80] available on the cluster of the physics department of the University of Zurich. 60'000 images were generated with each generative adversarial network using different

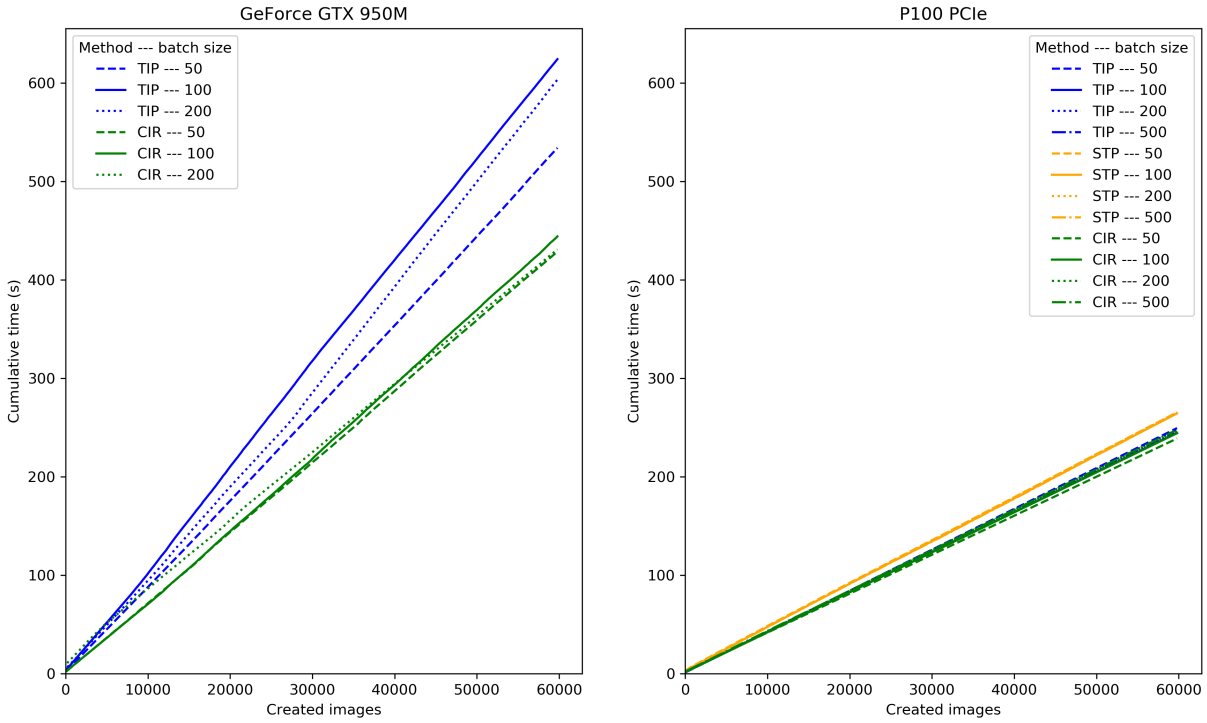


Figure 5.4.1: Timing of the final generative adversarial networks omitting the single track propagation on the single 950M GPU.

batch sizes for the generation process.
On the left side of figure 5.4.1 lower cumulative time needed to perform these simulations when using the calorimeter image refinement model compared to the more direct tracker image propagation is observed. Note that the GeForce GTX 950M was rather slow when using the conditional GAN for single track propagation to generate the full event. It did not fit very well on the linear scale used for this plot therefore it was omitted, but it was still recorded to be around one hour for 60'000 images. While the calorimeter image refinement model seems to perform slightly faster on the first GPU we need to take into account that this is only the second step of a two step approach. It is required to add up the times needed for the single track propagation GAN and the calorimeter image refinement GAN resulting in a simulation time of about one hour

and ten minutes. The same amount of images were generated with the single step tracker image propagation approach in only about 9 minutes on average. It is also necessary to take the time into account necessary for the generation of the tracker images from the tracker data. However, this time is minimal compared to the time needed to create the images with the generative adversarial networks and for 60'000 images this process only takes around 6 seconds.
However, the difference gets much smaller when using the P100 PCIe. The simulation times for all approaches reduce drastically. Notably the single track propagation GAN now takes only around 4 minutes requiring only about 30 seconds more than either the calorimeter image refinement GAN or tracker image propagation GAN, respectively. This still means that generating images with the two step approach takes about twice as long compared to the direct approach. With this approach it is still possible to produce around 7500 calorimeter responses per minute or around 125 per second on a single GPU greatly outperforming Geant4 by at least two orders of magnitude.

6 Conclusions

In this thesis I explored the recently established generative models and their applicability to particle physics detector simulations. I aimed to get an overview over most available image generation algorithms using generative adversarial networks, be it over a two step approach or a more direct one step method. These simulations should be at least of comparable quality to Geant4 generated hadronic calorimeter responses while still being applicable to data and processing events at a much higher speed. I found that the single track propagation approach followed by an calorimeter image refinement GAN delivered reasonable results whilst resulting in two separate model components with their own set of applicability. The direct method of translating 2D representations of the tracker information directly into a hadronic calorimeter image resulted in very high quality simulations as well while shortening both the training time as well as the simulation time of events due to its single network usage. In general I could prove with this work that generative adversarial networks are well suited for the generation of calorimeter responses and they might very well be a viable alternative to the current state-of-the art simulation technique via Geant4. Of course the resulting model results in much lower output verbosity and detail which might be a desirable property in some applications like usage of this algorithm directly as part of the L0 trigger.
While it is hard to put a number on the quality of generated images, especially when considering the imperfections of the Monte-Carlo based Geant4 itself, I showed that by using generative adversarial networks with the right hardware an improvement of the simulation time of the order 100 can easily be achieved, even when performing the calculations on limited hardware. A dedicated implementation of this algorithm could probably result in speed-up of order 1000, being able to compete with the increasing luminosity of the upcoming LHC upgrade and the requirements of physics beyond the Standard Model on suppressed loop level diagrams.

6.1 Limitations and further improvements

In our experiments with the hyper-parameter space I personally gained new insights which should be considered in future experiments. Like suggested in [65] the loss function is not the most

important component of any generative adversarial network. Even though I emphasized the theoretical advantages and disadvantages of the various loss functions like the Jensen-Shannon or Wasserstein divergence their practical impact is much less obvious. Even though the Wasserstein loss was well suited for the task of single track propagation it proved to be rather ineffective for the task of calorimeter image refinement . This might be due to the sparsity of the input as well as output. However, it is not clear what the practical reasons are as the theory behind this divergence seems to classify it as the clear winner.

So the authors recommendation is to choose one loss and stick with it focusing more on other hyper parameters. Firstly, the network architecture is the most important aspect of the algorithm as it controls the complexity encoded within the model. This includes the number of networks used (Generator and Discriminator vs. Generator, Encoder and Discriminator) as well as their number of layers, activation function, number of nodes or kernel size, usage of batch normalization and dropout and other regularization techniques. The most effective extensions of the basic generative adversarial network structure for this work seemed to be the usage of a PatchGAN architecture for the discriminator, a variant of the U-Net structure for the generator as well as the one-sided label-smoothing. All of these are implemented rather easily, but result in great improvement of the image quality in better approximation of the true distribution regardless of the loss function used.

Another extremely important hyper parameter is the choice of learning rate. It should not be chosen to large or training might not actually begin and the loss might not ever decrease. A learning rate too small, of course, will result in very slow training and the loss may seem to remain constant. Also the user of a generative adversarial network should consider using different learning rates for at least the generator and discriminator. It was observed reliably that a learning rate for the discriminator of one order of magnitude lower than the generator works rather well, while the learning rate of the encoder (for example in the Bicycle GAN) did not have a large impact.

Another extremely important "hyper parameter" is the ratio of complexity between the generator and discriminator. It was observed multiple times that some very promising generator architectures seemed to change their behavior rather drastically when making changes to the corresponding discriminator architecture. For example, the conditional Wasserstein GAN used in the first part has a rather peculiar structure with around 60'000 parameters for the discriminator and 1.7 million for the generator, corresponding to 30 times more parameters in the generator compared to the discriminator. However, when trying to balance the number of weights and biases by increasing the number of layers in the discriminator or the number of weights in single layers training did not deteriorate at best and completely stop at worst.

This instability is the main drawback when using generative adversarial networks, especially when dealing with sparse input and output spaces. Small changes to a working structure might result in drastic changes in the resulting image quality. This makes the problem of finding an appropriate model architecture similar to finding the proverbial needle in a haystack.

While this work focused on the two step approach of generating calorimeter responses with a

first crude single track network and a second refinement GAN it did not prove better than a single network transforming 2D representations of tracker information into calorimeter responses. Even though our approach has its own benefits by being able to train on single tracks and applying it to the full event, the direct approach seems more promising for this task. After training the second calorimeter image refinement network and obtaining the presented results much less time was spend on perfecting the direct tracker image propagation approach as it is very similar to the former network. In fact, basically the same networks were used when refining images generated by the single track propagation GAN and performing the tracker image propagation task. I assume that the approach focused on in this thesis resulted in slightly less desirable results, especially when looking at the image variety of a single event, as the calorimeter image refinement network had to deal with uncertainties and errors generated by the imperfect single track propagation GAN. So in future work in might be a more promising approach to work with tracker images directly instead of using two separate GANs requiring to tune two set of hyper parameters.

7 Questions

1. Does the energy cut-off at 6.12 GeV come from the hardware in general or was this chosen by our preprocessing?
2. For how many generated images per second are we aiming for? What is the time needed by Geant4?
3. Considering that the direct image-to-image translation task works really well, is there any benefit in the two-step approach
4. How much Geant4 full event reconstruction is spent for calorimeter?

A

Information theory

In this chapter the foundations of information theory necessary to understand this thesis will be summarized. Claude Shannon defined the fundamental unit of information theory, called the `bit`, in his 1948 paper[81]. One bit of information corresponds to the bisection of the prior uncertainty of an event given that it will happen. This means that if an event has probability $p_1 = 0.5$ it carries $\log_2(1/0.5) = 1$ bit of information while an event with $p_2 = 0.25$ carries $\log_2(1/0.25) = 2$ bits of information. The expected value of the contained information over all events is called the entropy, in mathematical terms written as

$$H(\mathbf{p}) = -\sum_i p_i \cdot \log_2(p_i). \tag{A.0.1}$$

It tells you how unpredictable your event space is. The cross-entropy is a measure of the difference between two probability distributions, one often being a real distribution, the other a predicting distribution. It is given by:

$$H(\mathbf{p}, \mathbf{q}) = -\sum_i p_i \cdot \log_2(q_i). \tag{A.0.2}$$

If the predicted probability q_i is equal to the true probability for all p_i, this of course reduces to the entropy of the system. The Kullback Leibler divergence now measures the difference between the cross-entropy and the entropy of a system with a true probability distribution and a predicted one, which should be zero if $q_i = p_i, \forall i$:

$$\begin{aligned} \mathrm{KL}(\mathbf{p}||\mathbf{q}) &= H(\mathbf{p}, \mathbf{q}) - H(\mathbf{p}) \\ &= -\sum_i p_i \cdot \log_2(q_i) + \sum_i p_i \cdot \log_2(p_i) \\ &= \sum_i p_i \cdot \log_2\left(\frac{p_i}{q_i}\right) \end{aligned} \tag{A.0.3}$$

In many supervised machine learning applications this is used as a loss function because if the label of an observation is known, it's probability $p_i = 1$ for a certain class, so

$$\mathrm{KL}(\mathbf{p} = 1||\mathbf{q}) = -\sum_i \cdot \log_2\left(q_i\right).$$

Minimizing this function leads to $q_i \to 1$ which is the desired outcome. The motivation for Jensen-Shannon divergence is similar to the Kullback-Leibler divergence, with the additional constraints of being bounded and symmetric in P and Q. This leads to the equation

$$\mathrm{JS}(\mathbf{p}||\mathbf{q}) = \frac{1}{2}\left(\mathrm{KL}(P||M) + \mathrm{KL}(P||M)\right), \quad \text{with: } M = \frac{P+Q}{2}. \tag{A.0.4}$$

B

Hyper-parameter space

Parameter	Options	#Options	Product
Adversarial steps	[1, 5, 10]	3	3
Batch size	[1, 4, 8, 16, 32, 64]	6	18
Dim_z	[4, 8, 16, 32, 64, 128]	6	108
Feature matching	[True, False]	2	216
LR Discriminator	[0.001, 0.0005, 0.0001, 5e-05, 1e-05, 1e-06]	6	1296
LR Generator	[0.001, 0.0005, 0.0001, 5e-05, 1e-05]	5	6480
Label smoothing	[0.8, 0.9, 0.95]	3	19440
Loss	['JS', 'KL', 'Wasserstein']	3	58320
Optimizer	['RMSProp', 'AdamOptimizer']	2	116640
PatchGAN	[True, False]	2	233280
Random labels	[0, 0.05, 0.1]	3	699840
lambda VAE	[0, 0.5, 1, 5, 10]	5	3499200
lambda X	[0, 0.5, 1, 5, 10]	5	17496000
lambda Z	[0, 0.5, 1, 5, 10]	5	87480000

Not yet included is the variation of the general network structures which is a important part, if not the single most important part of the hyper-parameter choice. This includes the number of layers, the number of nodes per layer, the activation function, the number of filters, usage

of batch normalization or dropout, etc. So in total there are more than one billion viable combinations of hyper-parameters. Of course, it is unrealistic (and unreasonable) to train all of these methods. Therefore a random grid-search was performed too narrow down the hyper parameter space.

C

Final network architectures

C.1 Single track propagation

Generator (params=1688449)

Nr	Layer	Weights	Bias	Params	Activation	Output
-1	Input					(?, 35)
0	dense	[35, 28672]	[28672]	1032192	leaky_relu	(?, 28672)
1	reshape_layer	0	0	0		(?, 7, 8, 512)
2	conv2d_transpose	[2, 2, 256, 512]	[256]	524544	leaky_relu	(?, 14, 16, 256)
3	conv2d_transpose	[2, 2, 128, 256]	[128]	131200	leaky_relu	(?, 28, 32, 128)
4	conv2d_transpose	[2, 2, 1, 128]	[1]	513	sigmoid	(?, 56, 64, 1)

Critic (params=62657)

Nr	Layer	Weights	Bias	Params	Activation	Output
-1	Input					(?, 56, 64, 4)
0	conv2d	[2, 2, 4, 64]	[64]	1088	leaky_relu	(?, 28, 32, 64)
1	conv2d	[2, 2, 64, 128]	[128]	32896	leaky_relu	(?, 14, 16, 128)
2	flatten	0	0	0		(?, 28672)
3	dense	[28672, 1]	[1]	28673	identity	(?, 1)

Total parameters: 1751106

C.2 Calorimeter image refinement

Encoder (params=601344)

Nr	Layer	Weights	Bias	Params	Activation	Output
-1	Input					(?, 64, 64, 1)
0	conv2d	[5, 5, 1, 16]	[16]	416	leaky_relu	(?, 32, 32, 16)
1	conv2d	[5, 5, 16, 32]	[32]	12832	leaky_relu	(?, 16, 16, 32)
2	conv2d	[5, 5, 32, 64]	[64]	51264	leaky_relu	(?, 8, 8, 64)
3	flatten	0	0	0		(?, 4096)
4	dense	[4096, 128]	[128]	524416	leaky_relu	(?, 128)
5	dense	[128, 64]	[64]	8256	leaky_relu	(?, 64)
6	flatten	0	0	0		(?, 64)
7	dense	[64, 64]	[64]	4160	identity	(?, 64)

Generator (params=2266490)

Nr	Layer	Weights	Bias	Params	Activation	Output
-1	Input					(?, 64, 64, 65)
0	conv2d	[4, 4, 65, 64]	[64]	66624	leaky_relu	(?, 32, 32, 64)
1	conv2d	[4, 4, 64, 64]	[64]	65600	leaky_relu	(?, 16, 16, 64)
2	batch_normalization	[64]	[64]	128		(?, 16, 16, 64)
3	conv2d	[4, 4, 64, 128]	[128]	131200	leaky_relu	(?, 8, 8, 128)
4	batch_normalization	[128]	[128]	256		(?, 8, 8, 128)
5	conv2d	[4, 4, 128, 128]	[128]	262272	leaky_relu	(?, 4, 4, 128)
6	batch_normalization	[128]	[128]	256		(?, 4, 4, 128)
7	conv2d	[4, 4, 128, 128]	[128]	262272	relu	(?, 4, 4, 128)
8	conv2d	[4, 4, 128, 128]	[128]	262272	relu	(?, 2, 2, 128)

9	conv2d_transpose	[4, 4, 128, 128]	[128]	262272	relu	(?, 4, 4, 128)
10	batch_normalization	[128]	[128]	256		(?, 4, 4, 128)
11	concatenate_with 6			0		(?, 4, 4, 256)
12	conv2d_transpose	[4, 4, 128, 256]	[128]	524416	relu	(?, 8, 8, 128)
13	batch_normalization	[128]	[128]	256		(?, 8, 8, 128)
14	concatenate_with 4			0		(?, 8, 8, 256)
15	conv2d_transpose	[4, 4, 64, 256]	[64]	262208	relu	(?, 16, 16, 64)
16	batch_normalization	[64]	[64]	128		(?, 16, 16, 64)
17	concatenate_with 2			0		(?, 16, 16, 128)
18	conv2d_transpose	[4, 4, 64, 128]	[64]	131136	relu	(?, 32, 32, 64)
19	batch_normalization	[64]	[64]	128		(?, 32, 32, 64)
20	concatenate_with 0			0		(?, 32, 32, 128)
21	conv2d_transpose	[4, 4, 16, 128]	[16]	32784	leaky_relu	(?, 64, 64, 16)
22	concatenate_with -1			0		(?, 64, 64, 81)
23	conv2d	[5, 5, 81, 1]	[1]	2026	sigmoid	(?, 64, 64, 1)

Discriminator (params=1058690)

Nr	Layer	Weights	Bias	Params	Activation	Output
-1	Input					(?, 64, 64, 2)
0	conv2d	[3, 3, 2, 32]	[32]	608	leaky_relu	(?, 64, 64, 32)
1	conv2d	[3, 3, 32, 32]	[32]	395009	leaky_relu	(?, 64, 64, 32)
2	max_pooling2d	0	0	0		(?, 32, 32, 32)
3	batch_normalization	[32]	[32]	64		(?, 32, 32, 32)
4	conv2d	[3, 3, 32, 64]	[64]	18496	leaky_relu	(?, 32, 32, 64)
5	conv2d	[3, 3, 64, 64]	[64]	36928	leaky_relu	(?, 32, 32, 64)

```
| 6 | max_pooling2d          | 0                | 0        | 0        |            | (?, 16, 16, 64)   |
-------------------------------------------------------------------------------------------------------
| 7 | batch_normalization    | [64]             | [64]     | 128      |            | (?, 16, 16, 64)   |
-------------------------------------------------------------------------------------------------------
| 8 | conv2d                 | [3, 3, 64, 128]  | [128]    | 73856    | leaky_relu | (?, 16, 16, 128)  |
-------------------------------------------------------------------------------------------------------
| 9 | conv2d                 | [3, 3, 128, 128] | [128]    | 147584   | leaky_relu | (?, 16, 16, 128)  |
-------------------------------------------------------------------------------------------------------
| 10| conv2d                 | [3, 3, 128, 128] | [128]    | 147584   | leaky_relu | (?, 16, 16, 128)  |
-------------------------------------------------------------------------------------------------------
| 11| max_pooling2d          | 0                | 0        | 0        |            | (?, 8, 8, 128)    |
-------------------------------------------------------------------------------------------------------
| 12| batch_normalization    | [128]            | [128]    | 256      |            | (?, 8, 8, 128)    |
-------------------------------------------------------------------------------------------------------
| 13| conv2d                 | [5, 5, 128, 64]  | [64]     | 204864   | leaky_relu | (?, 4, 4, 64)     |
-------------------------------------------------------------------------------------------------------
| 14| conv2d                 | [4, 4, 64, 32]   | [32]     | 32800    | leaky_relu | (?, 4, 4, 32)     |
-------------------------------------------------------------------------------------------------------
| 15| conv2d                 | [4, 4, 32, 1]    | [1]      | 513      | sigmoid    | (?, 2, 2, 1)      |
-------------------------------------------------------------------------------------------------------
Total parameters: 3926524
```

C.3 Tracker image propagation

The encoder and discriminator of the direct approach were chosen to be the same as for the Im2Im algorithm. They are not listed here again. The generator was made slightly more complex to account for the fact that it should be harder task to translate the images directly instead of converting an approximate calorimeter response to the true one.

```
Generator (params=2815674)
--------------------------
| Nr| Layer                  | Weights          | Bias     | Params   | Activation | Output            |
-------------------------------------------------------------------------------------------------------
| -1| Input                  |                  |          |          |            | (?, 64, 64, 65)   |
-------------------------------------------------------------------------------------------------------
| 0 | conv2d                 | [4, 4, 65, 64]   | [64]     | 66624    | leaky_relu | (?, 32, 32, 64)   |
-------------------------------------------------------------------------------------------------------
| 1 | conv2d                 | [4, 4, 64, 64]   | [64]     | 65600    | leaky_relu | (?, 16, 16, 64)   |
-------------------------------------------------------------------------------------------------------
| 2 | batch_normalization    | [64]             | [64]     | 256      |            | (?, 16, 16, 64)   |
-------------------------------------------------------------------------------------------------------
| 3 | conv2d                 | [4, 4, 64, 128]  | [128]    | 131200   | leaky_relu | (?, 8, 8, 128)    |
-------------------------------------------------------------------------------------------------------
| 4 | batch_normalization    | [128]            | [128]    | 256      |            | (?, 8, 8, 128)    |
-------------------------------------------------------------------------------------------------------
| 5 | conv2d                 | [4, 4, 128, 128] | [128]    | 262272   | leaky_relu | (?, 4, 4, 128)    |
-------------------------------------------------------------------------------------------------------
| 6 | batch_normalization    | [128]            | [128]    | 256      |            | (?, 4, 4, 128)    |
```

7	conv2d	[4, 4, 128, 128]	[128]	262272	relu	(?, 4, 4, 128)
8	conv2d	[4, 4, 128, 128]	[128]	262272	relu	(?, 2, 2, 128)
9	conv2d_transpose	[4, 4, 128, 128]	[128]	262272	relu	(?, 4, 4, 128)
10	batch_normalization	[128]	[128]	256		(?, 4, 4, 128)
11	concatenate_with 6			0		(?, 4, 4, 256)
12	conv2d_transpose	[4, 4, 128, 256]	[128]	524416	relu	(?, 8, 8, 128)
13	batch_normalization	[128]	[128]	256		(?, 8, 8, 128)
14	concatenate_with 4			0		(?, 8, 8, 256)
	residual_block-skips: 1					
15	conv2d	[5, 5, 256, 32]	[32]		relu	(?, 8, 8, 32)
16	concat			204832		(?, 8, 8, 288)
17	conv2d_transpose	[4, 4, 64, 288]	[64]	294976	relu	(?, 16, 16, 64)
18	batch_normalization	[64]	[64]	128		(?, 16, 16, 64)
19	concatenate_with 2			0		(?, 16, 16, 128)
	residual_block-skips: 1					
20	conv2d	[5, 5, 128, 32]	[32]		relu	(?, 16, 16, 32)
21	concat			102432		(?, 16, 16, 160)
22	conv2d_transpose	[4, 4, 64, 160]	[64]	163904	relu	(?, 32, 32, 64)
23	batch_normalization	[64]	[64]	128		(?, 32, 32, 64)
24	concatenate_with 0			0		(?, 32, 32, 128)
	residual_block-skips: 1					
25	conv2d	[5, 5, 128, 32]	[32]		relu	(?, 32, 32, 32)
26	concat			102432		(?, 32, 32, 160)
27	conv2d_transpose	[4, 4, 16, 160]	[16]	40976	leaky_relu	(?, 64, 64, 16)
28	concatenate_with -1			0		(?, 64, 64, 81)
	residual_block-skips: 1					
29	conv2d	[5, 5, 81, 32]	[32]		relu	(?, 64, 64, 32)
30	concat			64832		(?, 64, 64, 113)

```
| 31| conv2d                | [5, 5, 113, 1]    | [1]       | 2826     | sigmoid    | (?, 64, 64, 1)    |
```

Bibliography

[1] M. Thomson. *Modern Particle Physics*. Cambridge University Press, 2013.

[2] G. Aad, T. Abajyan, B. Abbott, J. Abdallah, S. Abdel Khalek, A.A. Abdelalim, O. Abdinov, R. Aben, B. Abi, M. Abolins, and et al. Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc. *Physics Letters B*, 716(1):1–29, Sep 2012. ISSN 0370-2693. doi: 10.1016/j.physletb.2012.08.020. URL http://dx.doi.org/10.1016/j.physletb.2012.08.020.

[3] M. Peskin and D. Schroeder. *Introduction to quantum field theory*. Addison-Wesley Pub. Co, 1995.

[4] D. Hanneke, S. Fogwell, and G. Gabrielse. New measurement of the electron magnetic moment and the fine structure constant. *Physical Review Letters*, 100(12), Mar 2008. ISSN 1079-7114. doi: 10.1103/physrevlett.100.120801. URL http://dx.doi.org/10.1103/PhysRevLett.100.120801.

[5] C. Di Paolo, P. Salucci, and J. P. Fontaine. The radial acceleration relation (rar): Crucial cases of dwarf disks and low-surface-brightness galaxies. *The Astrophysical Journal*, 873(2):106, Mar 2019. ISSN 1538-4357. doi: 10.3847/1538-4357/aaffd6. URL http://dx.doi.org/10.3847/1538-4357/aaffd6.

[6] J. Diemand and B. Moore. The structure and evolution of cold dark matter halos. *Advanced Science Letters*, 4(2):297–310, Feb 2011. ISSN 1936-7317. doi: 10.1166/asl.2011.1211. URL http://dx.doi.org/10.1166/asl.2011.1211.

[7] V. Alan Kostelecký. Gravity, lorentz violation, and the standard model. *Phys. Rev. D*, 69:105009, May 2004. doi: 10.1103/PhysRevD.69.105009. URL https://link.aps.org/doi/10.1103/PhysRevD.69.105009.

[8] Theory and phenomenology in high energy physics, 2006. https://www.hep.ucl.ac.uk/theory/sm.shtml[Accessed: 12.11.2020].

[9] Jr. Alves, A.Augusto et al. The LHCb Detector at the LHC. *JINST*, 3:S08005, 2008. doi: 10.1088/1748-0221/3/08/S08005.

[10] Y. Guz and the LHCb collaboration. The LHCb hadron calorimeter. *Journal of Physics: Conference Series*, 160:012054, apr 2009. doi: 10.1088/1742-6596/160/1/012054. URL https://doi.org/10.1088%2F1742-6596%2F160%2F1%2F012054.

[11] R Aaij, A Affolder, K Akiba, M Alexander, S Ali, R B Appleby, M Artuso, A Bates, A Bay, O Behrendt, and et al. Performance of the lhcb vertex locator. *Journal of Instrumentation*, 9(09):P09007–P09007, Sep 2014.

ISSN 1748-0221. doi: 10.1088/1748-0221/9/09/p09007. URL http://dx.doi.org/10.1088/1748-0221/9/09/P09007.

[12] M Anelli, R Antunes Nobrega, G Auriemma, W Baldini, G Bencivenni, R Berutti, V Bocci, N Bondar, W Bonivento, B Botchin, and et al. Performance of the lhcb muon system with cosmic rays. *Journal of Instrumentation*, 5(10):P10003–P10003, Oct 2010. ISSN 1748-0221. doi: 10.1088/1748-0221/5/10/p10003. URL http://dx.doi.org/10.1088/1748-0221/5/10/P10003.

[13] Lhcb trigger system, 2008. https://lhcb-public.web.cern.ch/en/Data%20Collection/Triggers2-en.html[Accessed: 26.10.2020].

[14] The LhCb collaboration. Measurement of the track reconstruction efficiency at lhcb. *Journal of Instrumentation*, 10(02):P02007–P02007, Feb 2015. ISSN 1748-0221. doi: 10.1088/1748-0221/10/02/p02007. URL http://dx.doi.org/10.1088/1748-0221/10/02/P02007.

[15] G. Apollinari, L. Rossi, and O. Brüning. High luminosity lhc project description. Technical report, 2014.

[16] C. Bozzi. LHCb Computing Resource usage in 2014 (II). Technical Report LHCb-PUB-2015-004. CERN-LHCb-PUB-2015-004, CERN, Geneva, Jan 2015. URL https://cds.cern.ch/record/1984010.

[17] G. Aad, B. Abbott, J. Abdallah, A. A. Abdelalim, A. Abdesselam, O. Abdinov, B. Abi, M. Abolins, H. Abramowicz, and et al. The atlas simulation infrastructure. *The European Physical Journal C*, 70 (3):823–874, Sep 2010. ISSN 1434-6052. doi: 10.1140/epjc/s10052-010-1429-9. URL http://dx.doi.org/10.1140/epjc/s10052-010-1429-9.

[18] Rahmat R., R. Kroeger, and A. Giammanco. The fast simulation of the CMS experiment. *Journal of Physics: Conference Series*, 396(6):062016, dec 2012. doi: 10.1088/1742-6596/396/6/062016. URL https://doi.org/10.1088%2F1742-6596%2F396%2F6%2F062016.

[19] S. Agostinelli. Geant4 — a simulation toolkit. *Nucl. Instrum. Methods Phys. A*, 506, 01 2003.

[20] M G Pia, T Basaglia, Z W Bell, and P V Dressendorfer. Geant4 in Scientific Literature. Dec 2009. URL https://cds.cern.ch/record/1226629.

[21] URL https://ep-dep.web.cern.ch/.

[22] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "generative adversarial networks", 2014.

[23] M. Paganini, L. de Oliveira, and B. Nachman. Calogan: Simulating 3d high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks. *Physical Review D*, 97(1), Jan 2018. ISSN 2470-0029. doi: 10.1103/physrevd.97.014021. URL http://dx.doi.org/10.1103/PhysRevD.97.014021.

[24] 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*, 1(1), Sep 2017. ISSN 2510-2044. doi: 10.1007/s41781-017-0004-6. URL http://dx.doi.org/10.1007/s41781-017-0004-6.

[25] V. Chekalina, E. Orlova, F. Ratnikov, D. Ulyanov, A. Ustyuzhanin, and E. Zakharov. Generative models for fast calorimeter simulation.lhcb case. 12 2018.

[26] S. Vallecorsa. Generative models for fast simulation. *Journal of Physics: Conference Series*, 1085:022005, 09 2018. doi: 10.1088/1742-6596/1085/2/022005.

[27] M. Albergo. *Toward Generative Modeling of Calorimetry Signals*. PhD thesis, 08 2018.

[28] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn, A. Rousselot, and R. Winterhalder. Invertible networks or partons to detector and back again, 06 2020.

[29] J. Arjona Martínez, T. Q. Nguyen, M. Pierini, M. Spiropulu, and J. Vlimant. Particle generative adversarial networks for full-event simulation at the lhc and their application to pileup description. *Journal of Physics: Conference Series*, 1525:012081, Apr 2020. ISSN 1742-6596. doi: 10.1088/1742-6596/1525/1/012081. URL http://dx.doi.org/10.1088/1742-6596/1525/1/012081.

[30] A. Butter, T. Plehn, and R. Winterhalder. How to gan lhc events. *SciPost Physics*, 7(6), Dec 2019. ISSN 2542-4653. doi: 10.21468/scipostphys.7.6.075. URL http://dx.doi.org/10.21468/SciPostPhys.7.6.075.

[31] B. Hashemi, N. Amin, K. Datta, D. Olivito, and M. Pierini. Lhc analysis-specific datasets with generative adversarial networks, 2019.

[32] M. Erdmann, L. Geiger, J. Glombitza, and D. Schmidt. Generating and refining particle detector simulations using the wasserstein distance in adversarial networks, 2018.

[33] J. Lin, W. Bhimji, and B. Nachman. Machine learning templates for qcd factorization in the search for physics beyond the standard model. *Journal of High Energy Physics*, 2019(5), May 2019. ISSN 1029-8479. doi: 10.1007/jhep05(2019)181. URL http://dx.doi.org/10.1007/JHEP05(2019)181.

[34] Y. Alanazi, N. Sato, T. Liu, W. Melnitchouk, M. P. Kuchera, E. Pritchard, M. Robertson, R. Strauss, L. Velasco, and Y. Li. Simulation of electron-proton scattering events by a Feature-Augmented and Transformed Generative Adversarial Network (FAT-GAN). 1 2020.

[35] Simon Badger and Joseph Bullock. Using neural networks for efficient evaluation of high multiplicity scattering amplitudes, 2020.

[36] D. Lancierini. *Study of the $B \to D\mu\nu$ differential decay width : analysis of the sensitivity of LHCb to the scalar form factor*. PhD thesis, University of Zurich, 2020.

[37] A. B. Lindbo Larsen, S.K. Sønderby, H. Larochelle, and O. Winther. Autoencoding beyond pixels using a learned similarity metric, 2015.

[38] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto. Deep convolutional autoencoder-based lossy image compression. pages 253–257, 2018. doi: 10.1109/PCS.2018.8456308.

[39] Q. Meng, D. Catchpoole, D. Skillicom, and P. J. Kennedy. Relational autoencoder for feature extraction. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 364–371, 2017. doi: 10.1109/IJCNN.2017.7965877.

[40] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[41] L. Weng. From autoencoder to beta-vae, August 2018 2018. URL https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html?from=hackcv&hmsr=hackcv.com.

[42] Y. Pu, Z. Gan, R. Henao, X. Yuan, C. Li, A. Stevens, and L. Carin. Variational autoencoder for deep learning of images, labels and captions. In *Advances in neural information processing systems*, pages 2352–2360, 2016.

[43] X. Yang. "understanding the variational lower bound", April 14 2017. URL https://xyang35.github.io/2017/04/14/variational-lower-bound/.

[44] R. Gupta. "kl divergence between 2 gaussian distributions", April 16 2020. URL https://mr-easy.github.io/2020-04-16-kl-divergence-between-2-gaussian-distributions/.

[45] C Doersch. Tutorial on variational autoencoders, 2016.

[46] M. Mendez. Variational autoencoders (vaes): A simple explanation, January 18 2019. URL https://medium.com/@miguelmendez_/vaes-i-generating-images-with-tensorflow-f81b2f1c63b0.

[47] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

[48] Microsoft. Cntk 206: Part a - basic gan with mnist data, 2017. URL https://www.cntk.ai/pythondocs/CNTK_206A_Basic_GAN.html.

[49] F. Farnia and A. Ozdaglar. Do gans always have nash equilibria? In *International Conference on Machine Learning (ICML)*, 2020.

[50] I. Goodfellow. "nips 2016 tutorial: Generative adversarial networks", 2016.

[51] F. Huszár. "an alternative update rule for generative adversarial networks", March 21 2016. URL https://www.inference.vc/an-alternative-update-rule-for-generative-adversarial-networks/.

[52] F. Huszár. "how to train your generative models? and why does adversarial training work so well?", November 17 2015. URL https://www.inference.vc/how-to-train-your-generative-models-why-generative-adversarial-networks-work-so-well-2/.

[53] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. 2017.

[54] L. Ambrogioni. "an intuitive guide to optimal transport, part ii: the wasserstein gan made easy", September 23 2018. URL http://modelai.gettysburg.edu/2020/wgan/Resources/Lesson4/IntuitiveGuideOT.htm.

[55] J. Hui. "gan — wasserstein gan and wgan-gp", 2018.

[56] L. Ambrogioni. "an intuitive guide to optimal transport, part i: formulating the problem", September 17 2018. URL http://modelai.gettysburg.edu/2020/wgan/Resources/Lesson4/IntuitiveGuideOT1.htm.

[57] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. In *Advances in neural information processing systems*, pages 6231–6239, 2017.

[58] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. "improved training of wasserstein gans", 2017.

[59] J. Hui. "gan — lsgan (how to be a good helper?)", 2018.

[60] A. Pan. "read-through: Wasserstein gan", February 26 2017. URL https://www.alexirpan.com/2017/02/22/wasserstein-gan.html.

[61] J. Hui. Gan — ways to improve gan performance, June 19 2018. URL https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b.

[62] H. Thanh-Tung and T. Tran. Catastrophic forgetting and mode collapse in gans. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10. IEEE, 2020.

[63] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks, 2017.

[64] J. Hui. "gan — why it is so hard to train generative adversarial networks!", 2018.

[65] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet. Are gans created equal? a large-scale study. In *Advances in neural information processing systems*, pages 700–709, 2018.

[66] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[67] V. Lin, J. M. Girard, M. A. Sayette, and L. Morency. Toward multimodal modeling of emotional expressiveness, 2020.

[68] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[69] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville. Adversarially learned inference. 2016.

[70] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial feature learning. 2016.

[71] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. "improved techniques for training gans", 2016.

[72] F. Huszár. "understanding minibatch discrimination in gans", Juli 15 2016. URL `https://www.inference.vc/understanding-minibatch-discrimination-in-gans/`.

[73] J. W. Park. "how to implement minibatching in tensorflow", 2017.

[74] m. Lee. "(nn methodology) patchgan discriminator", March 13 2019. URL `https://brstar96.github.io/mldlstudy/what-is-patchgan-D/`.

[75] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. 2018.

[76] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015.

[77] K. He, X. Zhang, S. Ren, and J Sun. Deep residual learning for image recognition, 2015.

[78] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[79] "nvidia geforce gtx 950m", March 13 2015. URL `https://www.techpowerup.com/gpu-specs/geforce-gtx-950m.c2642`.

[80] "nvidia tesla p100 pcie 16 gb", June 20 2016. URL `https://www.techpowerup.com/gpu-specs/tesla-p100-pcie-16-gb.c2888`.

[81] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3): 379–423, 1948.