UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Dipartimento di fisica

Corso di Laurea Magistrale in Fisica

Parton level generative models for fast simulation of LHC events using Generative Adversarial Networks

Codici PACS: 12.38.-t, 12.39.-x, 84.35.+i

Relatore: **Prof. Stefano Carrazza**

Correlatore: **Dott. Marco Zaro**

Tesi di Laurea di:
Luigi Favaro
Matr. 939891

Anno Accademico 2019-2020

Abstract

In this thesis I will address the problem of fast simulations for the generation of events at LHC using techniques based on algorithms known as "Machine Learning". The purpose of these algorithms is to obtain generative models able to reproduce correctly the phase space of different physical processes starting from kinematical variables of the process itself like the so-called Mandelstam variables. To this end, an innovative training process has been utilized known as "Generative Adversarial Networks" in which two neural networks compete with each other in a zero-sum game. To carry out this training simulations of the different processes have been used, obtained with Monte Carlo methods through the software MadGraph. Finally, I evaluated the performance of the models in terms of the difference between the Monte Carlo and the generated distributions of the variables used in the training process.

Sommario

In questa tesi si affronterà il problema di ottenere delle veloci simulazioni per la generazione di eventi ad LHC usando tecniche basate su algoritmi noti come di "Machine Learning". Lo scopo di questi algoritmi è quello di ottenere modelli generativi in grado di riprodurre correttamente lo spazio delle fasi di diversi processi fisici a partire da variabili cinematiche del processo stesso come, ad esempio, le variabili di Mandelstam. A tal fine, è stato utilizzato un innovativo processo di allenamento conosciuto come "Reti Generative Avversarie" nel quale due reti neurali competono tra di loro in un gioco a somma zero. Per effettuare questo allenamento sono state utilizzate simulazioni di diversi processi ottenuti con metodi Monte Carlo tramite il software MadGraph. Infine, si è valutata la performance dei modelli in termini della differenza tra le distribuzioni delle variabili usate nel processo di allenamento ottenute dal Monte Carlo e dalla rete neurale.

Contents

Introduction

Throughout the era of LHC simulations have been proved to be fundamental for precision experiments towards a better comprehension of the Standard Model. Simulations are applied in mostly all stages of a particle physics experiment such as simulation of hard scattering processes, hadronization, detector response and signal transformation. Generally, these methods are based on a Monte Carlo approach which provides a general workflow for different problems. Nevertheless, a Monte Carlo simulation has its drawbacks mainly related to the exponential increase of time and CPU usage in correspondence of the increasing complexity of the problem. A possible way to reduce these effects is the introduction of Machine Learning methods already widely utilized at LHC.

The simulation problem studied in this thesis is the generation of LHC events at parton level. The tool used to generate events is MadGraph5_aMC@NLO[4], later referred as MadGraph5. The Monte Carlo-based procedure consists in finding all the matrix elements contributions to the chosen channel and calculating the full squared matrix element. Finally, a Monte Carlo integration algorithm is used to sample unweighted events from the phase space distribution. The time and CPU usage of this procedure scales steeply with increasing number of diagrams and particles and is a bottleneck to precise events simulation at LHC. This thesis propose an innovative generative Machine Learning method, based on the simultaneous training of two neural networks a generator and a discriminator, called Generative Adversarial Networks (GAN) in order to entirely bypass the full squared matrix evaluation and the phase space integration.

To this end, the training of such generative model is done by extracting the kinematical variables needed to identify the event and defining a minimax game where the generator tries to reproduce the distributions of the kinematic variables and the discriminator tries to distinguish a true event from a generated one. The implementation of these networks has been done using Keras[9] with TensorFlow 2.3[1] backend. This thesis is focused on the reproduction of three $2 \to 2$ scattering processes: $pp \to t\bar{t}$, $gg \to ZZ$, and $gg \to HH$. The kinematical variables used to identify the events are the Mandelstam invariants s and t, and the rapidity in the reference frame of the parton scattering y.

The performance of these models has been tested measuring the similarity between the true and the generated distribution of the input features. In particular, the Kullback-Leibler divergence and the Jensen-Shannon divergence are used to perform an optimization of the parameters of the model, also called hyperparameters. Moreover, the histograms are compared using the bin-wise ratio between the two distributions. The results obtained show that the defined GAN is able to correctly reproduce the distributions of the input features and their correlations in the regime of a low statistics sample of 10k events and a high statistics sample of 1M events for the three processes. Moreover, the ability of data augmentation of these models has been tested. Various samples have been generated bigger than the sample used for training and for all the channels a data augmentation of factor 10 can be found.

This thesis is organized as follows. Chapter 1 introduces the theory behind a hard scattering process and the collisions studied at LHC with a focus on the role that simulations have nowadays and in the future. Chapter 2 describes how a Monte Carlo integration works and how these algorithms are applied in MadGrpah5. Then, the focus is centered on the events generation procedure of MadGraph5 emphasizing the drawbacks os this method. The GAN algorithm is presented in Chapter 3 preceded by a description of how a neural network is trained. In this chapter the architecture used for the used models is also presented. Finally, in Chapter 4, the results of the hyperparameters optimization are presented followed by the histograms and the correlation plots of all trained models. [1]

[1] The code used for the results presented in this thesis can be found at: `https://github.com/luigifvr/gan-lhc-events`

1 | Physics at LHC

The Large Hadron Collider (LHC) [10] is a hadron and heavy ion accelerator installed near Geneva in a 27 km long tunnel at the depth of about 100m constructed between 1984 and 1989 for the LEP machine[14].

The project was approved by CERN (European Organization for Nuclear Research) Council in December 1994 and designed to overcome in terms of center of mass energy and luminosity the existing accelerators such as the Tevatron at Fermilab and LEP (Large Electron Positron) at CERN.

The first beams were accelerated, in the center of mass reference frame, to 7 TeV in 2010/2011 and 8 TeV in 2012. In 2015 LHC reached the nominal energy of 13 TeV and in 2018 an instantaneous luminosity of $2 \times 10^{34} \mathrm{cm}^{-2}\mathrm{s}^{-1}$ well above the designed value.

In 2024 the machine and the detectors will undergo a long list of upgrades with the final goal to increase the total amount of integrated luminosity by a factor of 10 by 2035.

This chapter briefly introduce Quantum Chromodynamics (QCD), which is the theory used to describe a hadron scattering, its lagrangian and the associated Feynman rules followed by a summary of the LHC apparatus. Finally, a description of the role of computer simulations in today's environment and its drawbacks are discussed in Section 1.3.

1.1 Introduction to QCD

Quantum Chromodynamics is the theory that describes the strong interactions. It is a non-Abelian Gauge theory with Gauge group SU(3), a brief description of its lagrangian can be found in Appendix A. This theory predicts the phenomenon of asymptotic freedom which is a feature that cause the interaction between particles to become smaller as energy increases. Instead, at low energy, the interaction becomes stronger and quarks and gluons, constituents of matter, confine themselves in particles called hadrons.

This effect enables the possibility of calculation of cross sections in high energy physics through perturbation theory, but even in this case contributions from lower

energy scale are important and need to be considered suggesting a potential hole in the predictive power of the theory.

However, thanks to the factorization theorem, which is discussed in the following subsection, it is possible to separate the perturbative contribution at short distance from the non-perturbative contributions at long distance and hence restore the possibility of making predictions with both contributions.

Factorization theorem

One of the major successes of QCD theory is that it can describe asymptotic freedom. Naively, the interaction between particles becomes asymptotically zero at increasing energy thus, even if low energy contributions cannot be neglected, it is possible to obtain a perturbative expansion at the order of the QCD coupling constant α_s for a high energy hadron scattering. The hard scattering cross section is calculated by evaluating the full matrix element of the scattering process and integrating over the phase space while more efforts is needed to calculate a physically measurable hadronic cross section. In this process, the factorization theorem has the remarkable role of factorizing the short-distance (hard) contribution and the long-distance (soft) contribution which is not describable through pQCD but universal. The final hadronic cross section is then obtained as a convolution in the momentum fraction of the parton between the hard term cross section and the soft term contribution.

For the interest of LHC, consider a high energy scattering between two hadrons, namely two protons $p(P_a)$ and $p(P_b)$, where P_i is the 4-momentum of the proton i. Suppose that the hard scattering involves two partons a and b with longitudinal energy fraction x_a and x_b since the momentum fractions of the hadron constituents cannot be calculated with perturbative QCD, a function it is associated with the probability of finding the constituents a or b with longitudinal momentum fraction x_a or x_b. These are the universal function $f_i(x_i)$ for a given hadron and are called parton distribution functions.

Applying the factorization theorem and pQCD to a deep inelastic scattering the cross section $\sigma(p(P_1) + p(P_2) \to Y + X)$ at Leading Order is given by:

$$\sigma(p(P_a)+p(P_b) \to Y+X) = \int_0^1 dx_a \int_0^1 dx_b \sum_{a,b} f_a(x_a) f_b(x_b) \cdot \sigma_{ab\to Y}(x_a, x_b), \quad (1.1)$$

where Y and $\sigma_{ab\to Y}$ are, respectively, the final state and the cross section of the hard scattering process, X is any possible hadronic final state and the sum is carried over all species of quarks and antiquarks.

Due to the universality of the PDFs, the typical workflow is to estimate the PDFs from an easy to study physical process and use the fitted PDFs to make predictions in more complicated high energy scattering processes.

1.2 pp collisions at LHC

At the LHC the collision of beams doesn't involve single particles, indeed the beam is composed of about 10^{11} protons squeezed in bunches of size $\sim$17 μm in the transverse direction and $\sim$8 cm in the beam direction. This means that at the interaction point different types of collisions could happen:

- Soft collisions consisting of initial and final state radiation and soft parton interactions with typically low transferred momentum. The method used to study these collisions is with non-perturbative QCD and the inclusion of these effects is fundamental for a realistic simulation of real events.
- Hard collisions are infrequent events in which the fundamental constituents of protons come to light and are the interesting ones in experiments. They are described by perturbative QCD and characterized by short distance collision and high transferred momentum enabling them to produce new particles.

As described in Section 1.1 an hard inelastic collision could be interpreted as a scattering between two partons (quarks or gluons) each one carrying a fraction x_i of the total momentum of the particles. Therefore the center of mass energy $\sqrt{\hat{s}}$ during the collision of two partons a, b is a fraction of the total centre-of-mass energy $\sqrt{s}$:

$$\sqrt{\hat{s}} = \sqrt{x_a x_b s}, \tag{1.2}$$

where x_a, x_b are the fractions of the total momentum carried by the partons.

The cross-section for a hard scattering collision is given by Eq. 1.1 where a proper treatment of the parton distribution functions $f_i(x_i, Q^2)$, $f_j(x_j, Q^2)$ and of the hard scattering cross section $\sigma_{ab\to Y}(x_a, x_b, Q^2)$ at higher order introduce the dependence on the energy scale Q^2.

Figure 1.3 shows the PDFs for $Q^2 = 10^2\text{GeV}^2$ and $Q^2 = 10^4\text{GeV}^2$.

LHC structure

To reach the final energy of 6.5 TeV the beams are gradually speeded up by different accelerators as shown in Figure 1.2.

Initially, an electric field separate protons from electrons and then injected into the accelerators chain:

- a linear accelerator (LINAC) speeds up protons up to 50 MeV;
- the beam is then injected into the Proton Synchrotron Booster (PSB), which accelerates the protons up to 1.4 GeV;
- then the Proton Synchrotron (PS) pushes the beam up to 25 GeV;

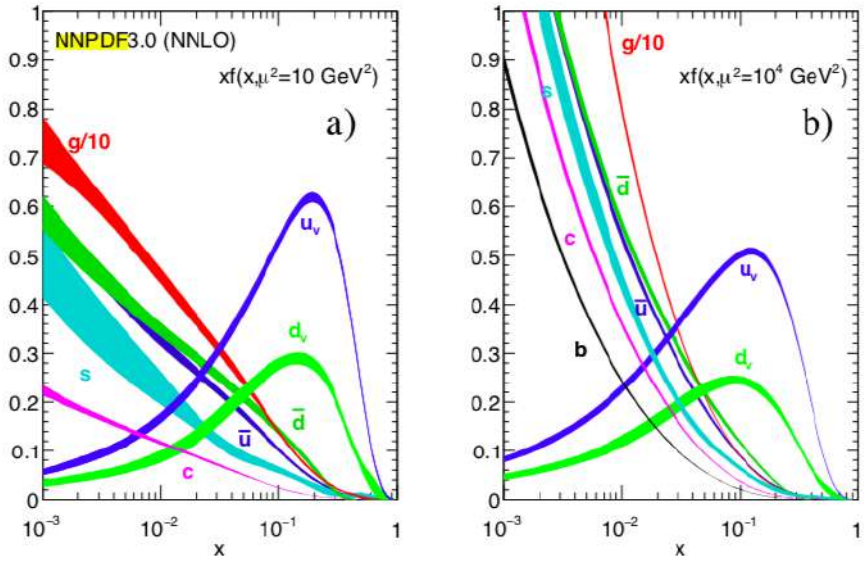


Figure 1.1: Parton Distribution Fucntions (PDFs) computed for $\mu^2 = 10^2$ GeV^2 and $\mu^2 = 10^4$ GeV^2 from NNPDF 2015 PDFs[5].

– protons are then injected in the Super Proton Synchrotron (SPS) where they are accelerated up to 450 GeV.

Finally, the protons are injected in LHC using two beam pipes in which the two beams circulate clockwise and anti-clockwise directions and then accelerated to the nominal $\frac{\sqrt{s}}{2}$ energy. It takes 4 minutes and 20 seconds to inject a beam into LHC and about 20 minutes for protons to reach their maximum energy of 6.5 TeV.

The LHC relies on more than 1000 superconducting dipole magnets to bend the beam in the ring. This magnets system is cooled to a temperature below 2 K using superfluid helium and produces a field of 8.4 T. Moreover, to ensure beam stability, quadrupole, sextupoles, octupoles, and decupoles magnets are installed at different points of the beam pipe.

1.3 The role of simulations at LHC

Simulations are a key component for studies at LHC, indeed they are the junction point between the theoretic model and the real experiment.

The full simulation of events can be summarized in four steps:

– Generation: an event generator produce the particles of the scattering process to study at parton level, e.g. proton-proton parton scattering into two top quarks;

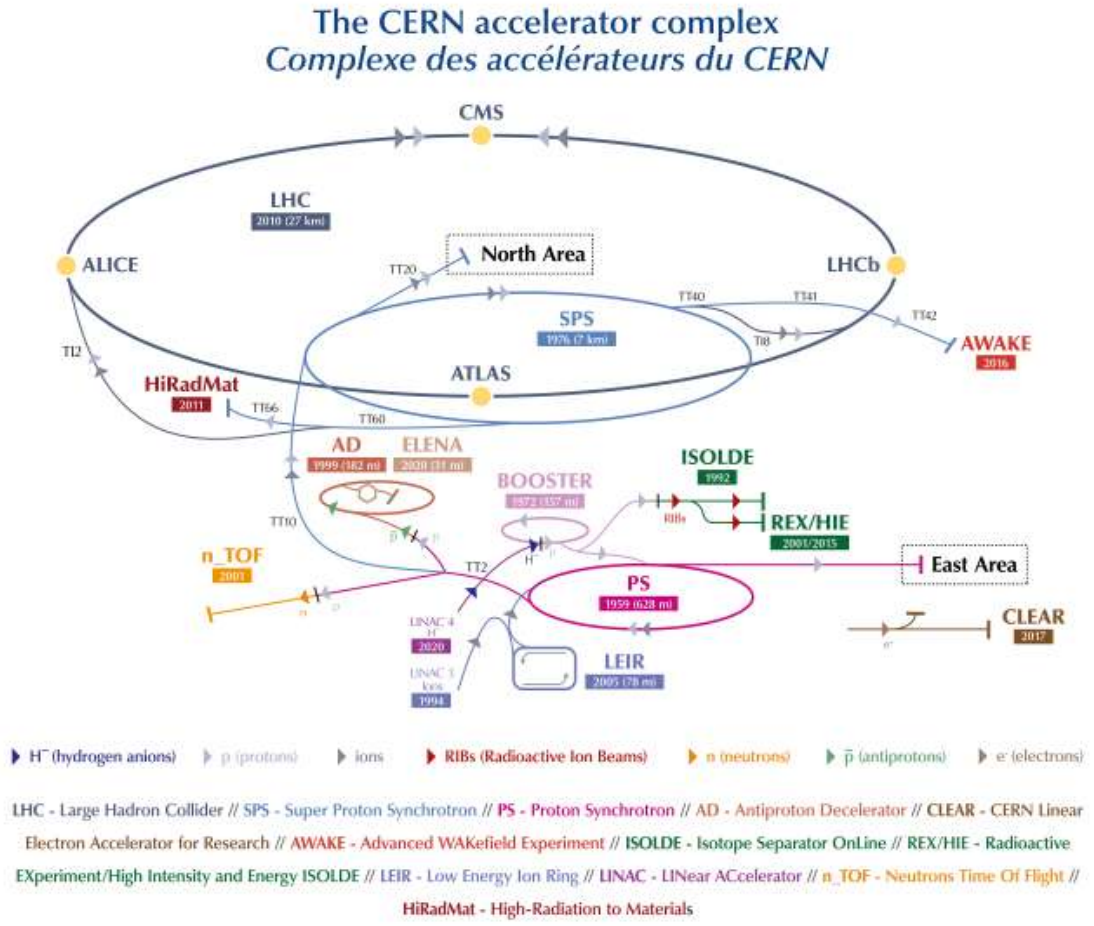


Figure 1.2: Accelerator complex at CERN[16].

- Parton shower: this step simulate the emission of soft terms in a parton shower which leads to the hadronized final state observed particles.
- Detector interaction: this step simulates the interaction between the produced particles and the detector material;
- Digitization: finally, a simulation is needed to reproduce the measured signal and to obtain data in the same form of real events. Moreover, this step accounts for the instrumental effects like noises and thresholds.

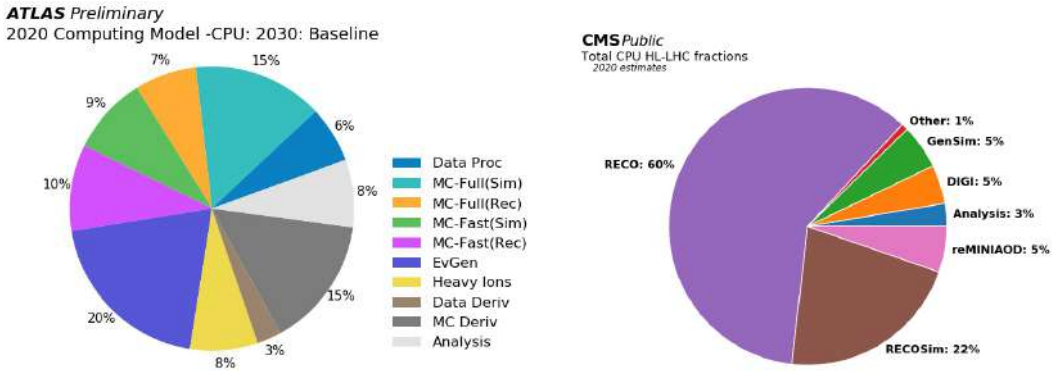
The results obtained from a full simulation provide answers to a range of different questions from the discovery of new physics, like the number of events needed to distinguish a particular channel from the background, to the deterioration state of detector components. In this environment, Machine Learning techniques already are ubiquitous in a vast range of applications from discrimination from signal and background to objects reconstruction, and with the upcoming of more performant algorithms and hardware, the expectations are that their usage will be extended.

One of the major issues for LHC in the next years will be the CPU and storage capabilities, especially during the HL-LHC era. As Figure 1.3 shows, the baseline prediction of the majority of the CPU usage for ATLAS and CMS is associated with simulations using Monte Carlo techniques. This approach can be really slow and CPU expensive and looking forward into the next years the typical increase in resource available will not be enough to account for the statistic required by LHC.

The ATLAS and CMS collaborations forecast that during the HL-LHC era, where the average number of collisions per event will be $\sim$200, the CPU consumption will be $\sim$5 times greater than the available resources. However, the LHCb collaboration reports that already in the next years that limit will be exceeded, both forecasts are shown in Figure 1.4.

To overcome this problem different projects regarding fast simulations are in development nowadays, indeed one of the key tools could be the implementation of new Machine Learning based generative techniques.

The aim of this thesis is to reduce the impact of Monte Carlo simulations in the parton level generation process described above. This step, especially for calculation beyond the LO, can be really CPU expensive and time consuming due to the large number of matrix elements to calculate. The next chapter introduce the procedures of a Monte Carlo integration and the issue of the scaling complexity of the matrix elements evaluation.



(a) (b)

Figure 1.3: Snapshot of projected CPU resources required by ATLAS (a) and CMS (b) in 2030, by computational task. [8]

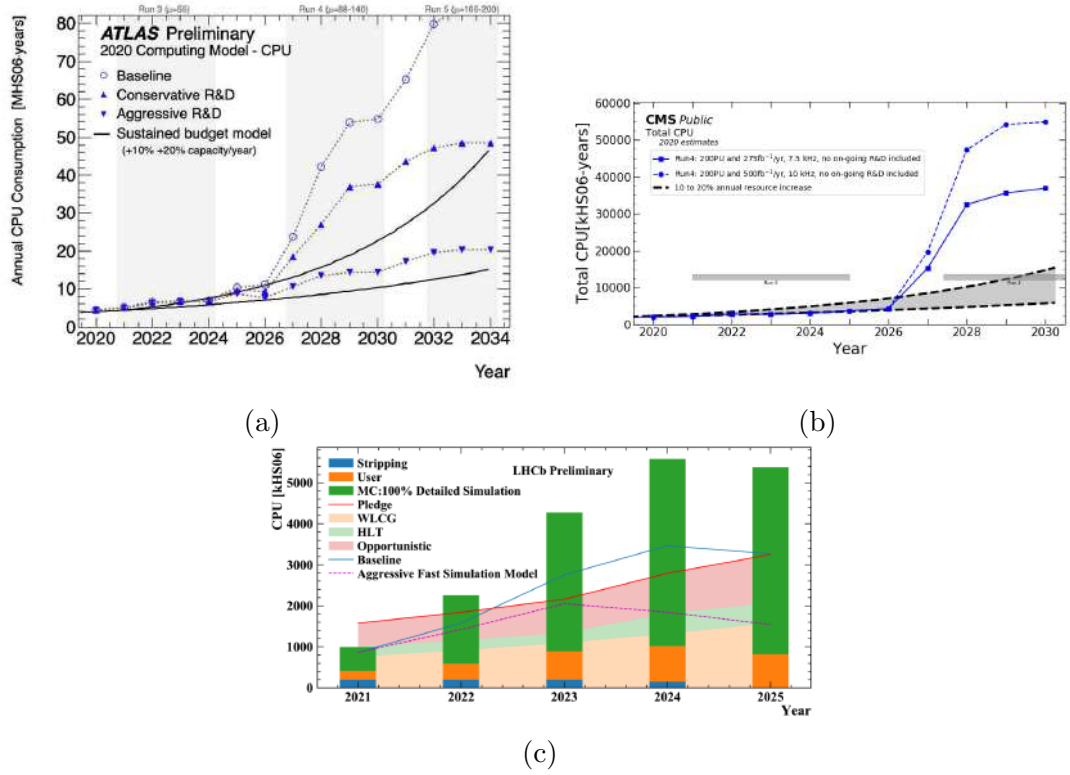


Figure 1.4: (a) Projected CPU requirements (in kilo-HEPSpec06 years) of ATLAS between 2020 and 2034 based on 2020 assessment. Three scenarios are shown, corresponding to an ambitious ("aggressive"), modest ("conservative") and minimal ("baseline") development program. The black lines indicate annual improvements of 10% and 20% in the computational capacity of new hardware for a given cost, assuming a sustained level of annual investment. The blue dots with the brown lines represent the 3 ATLAS scenarios following the present LHC schedule.[8] (b) CPU time requirements (in kilo-HEPSpec06 years) estimated to be required annually for CMS processing and analysis needs. The scenarios considered are 275 fb^{-1} per year of 7.5 kHz of data collected (solid line) and 500 fb^{-1} per year of 10 kHz of data collected (dashed line) both without R&D predictions. (c) Estimated CPU usage for the LHCb experiment from 2021 through 2025 (in kilo-HEPSpec06 years). The bars give the total CPU usage expected, composed of stripping jobs (blue), all estimated user jobs (orange) and simulation production (green). The green bars here assume only fully detailed simulation to be run.[15]

2 | Monte Carlo simulations

The Monte Carlo methods use a statistical approach to study phenomena. The idea is to solve problems that could be deterministic in nature through stochastic numerical methods. To do so, different methods of random sampling of statistical variables have been developed which go by the name of Monte Carlo.

Even if none of these methods is bound to computer simulations, the growth of computer power in the last years allowed the promotion of random sampling as an effective tool to study incredibly complex phenomena through numerical simulations in various natural fields.

In particular, in particle physics Monte Carlo methods are used throughout the entire simulation of a scattering process. The program used for event generations from matrix element is MadGraph5_aMC@NLO[4][12] which accounts for the events generations for a selected channel from a given model, e.g. Standard Model. This is accompanied by tools that simulate the hadronization and the parton showering like Pythia[20]. Also other tools, likeSherpa[7], offer the possibility of simulate event generation and parton showering. On the other hand, to fully simulate a real event it is necessary to simulate the interaction between particles and the detector materials, this is done mainly through Geant4[2] and Delphes[17].

The first section of this chapter will describe how to use Monte Carlo methods to perform multi-dimensional integration, followed by a summary of the tool used to generate events in this thesis: MadGraph5.

2.1 Monte Carlo integration

The problem of performing multi-dimensional integrals in particle physics is tied up to the calculation of cross sections. Indeed, selected a process with n final state particles and a matrix element $|M|^2$ the cross section is given by:

$$\sigma = \frac{1}{2s} \int |M|^2 d\Phi(n), \tag{2.1}$$

where s is the center of mass energy and $\Phi(n)$ is the $3n$-dimensional phase space. In order to perform integrals of such kind consider a d-dimension integral over the

space Ω of a function $f(x)$:

$$I = \int_\Omega d^d x f(\bar{x}) p(\bar{x}), \tag{2.2}$$

where:

$$\int_\Omega p(\bar{x}) d^d x = 1, \qquad p(\bar{x}) \geq 0, \quad \forall \bar{x} \in \Omega. \tag{2.3}$$

An estimate of this integral can be obtained sampling N random d-dimensional vectors $\bar{x}_i$ from the probability distribution function $p(\bar{x})$ and then evaluate the function $f(\bar{x}_i)$.

The approximated formula for I is given by:

$$I_{MC} = \frac{1}{N} \sum_{i=1}^{N} f(\bar{x}_i) \quad \xrightarrow[N\to\infty]{} \quad I, \tag{2.4}$$

which tends to I as $N \to \infty$.

The quantity I_{MC} must be accompanied by its error that can be shown to be for a Monte Carlo process with large N:

$$\sigma = \frac{\sigma_I}{\sqrt{N}}, \qquad \sigma_I = \int_\Omega f^2(\bar{x}) p(\bar{x}) d^d x - I^2, \tag{2.5}$$

where σ_I is the variance of the integral.

If this error is compared with the ones obtained from numerical methods is clear that Monte Carlo integration becomes better with high-dimensional spaces. Using a numerical method the error depends linearly on the number of points used, in case of d dimension the number of points will be $N = n^d$ where n is the fixed spacing along the axis. For the Simpson and the trapezoidal methods the error is proportional to:

$$\epsilon_{Simp} \propto \frac{1}{n^4} \propto \frac{1}{N^{4/d}}, \tag{2.6}$$

$$\epsilon_{trap} \propto \frac{1}{n^2} \propto \frac{1}{N^{2/d}}, \tag{2.7}$$

while with the MC integration it is always $\epsilon_{MC} \propto \frac{1}{\sqrt{N}}$ implying that already with 6-8 dimensions the Monte Carlo method is more efficient.

Importance sampling

Besides the effectiveness of the Monte Carlo integration, various techniques allow to reduce the variance of the integral. One of the most used is called "Importance Sampling"[13].

Given an integral as in Eq. 2.2, $p(\bar{x})$ could not be the best probability distribution to minimize the variance. In order to find it, introduce a new function $g(\bar{x})$ where $g(\bar{x}) \geq 0$ and $\int_\Omega g(\bar{x}) d^d x = 1$ like:

$$I = \int_\Omega \left[\frac{f(\bar{x})p(\bar{x})}{g(\bar{x})} \right] g(\bar{x}) d^d x. \tag{2.8}$$

To find the best $g(\bar{x})$ minimize the Lagrangian L which contains the constraint on $g(\bar{x})$ and the variance of the integral with respect to $g(\bar{x})$:

$$L\{g\} = \int_\Omega \left[\frac{f(\bar{x})p(\bar{x})}{g(\bar{x})} \right]^2 g(\bar{x}) d^d x + \lambda \left(\int_\Omega d^d x g(\bar{x}) - 1 \right). \tag{2.9}$$

This gives as best probability function $g(\bar{x}) = \lambda |f(\bar{x})p(\bar{x})|$, where λ is a Lagrange multiplier and can be found imposing $\int_\Omega g(\bar{x}) d^d x = 1$.

If $f(\bar{x}) \geq 0$, $\lambda = 1/I$ and an estimator for the integral is:

$$I_{RS} = \frac{1}{N} \sum_{i=1}^{N} \frac{f(\bar{x}_i)p(\bar{x}_i)}{g(\bar{x}_i)} = \frac{1}{N} \sum_{i=1}^{N} I \frac{f(\bar{x}_i)p(\bar{x}_i)}{f(\bar{x}_i)p(\bar{x}_i)} = I, \tag{2.10}$$

where the $\bar{x}_i$ are drawn from the probability distribution function $g(\bar{x})$ and it has zero variance if the integral value is known. Considering that it is the unknown variable instead, the variance can be reduced choosing a function close to the initial integrating function such as its Taylor expansion.

An example of the effects of the Importance Sampling technique on an exponential function is shown in Fig. 2.1.

Rejection method

The rejection method is a technique used to sample random variables from any probability distribution function. Given the target probability distribution function $f(x)$ the method works as follow:

- Find a second probability distribution function $g(x)$ integrable which is known how to sample from, and such that $g(x) \geq f(x)$ for all x in the interested range;
- Generate a number x_i distributed according to $g(x)$;
- Accept or reject the proposed number with probability $p_{acc} = \frac{f(x_i)}{g(x_i)}$.

To check that the sampling occur from the correct distribution, define the probability to generate a number between x and $x+dx$ such that $p_{gen} dx = g(x)dx$. The sampled distribution is the product between p_{gen} and p_{acc}:

$$p_{samp} dx = p_{gen} p_{acc} dx = g(x) \frac{f(x}{g(x)} dx = f(x)dx, \tag{2.11}$$

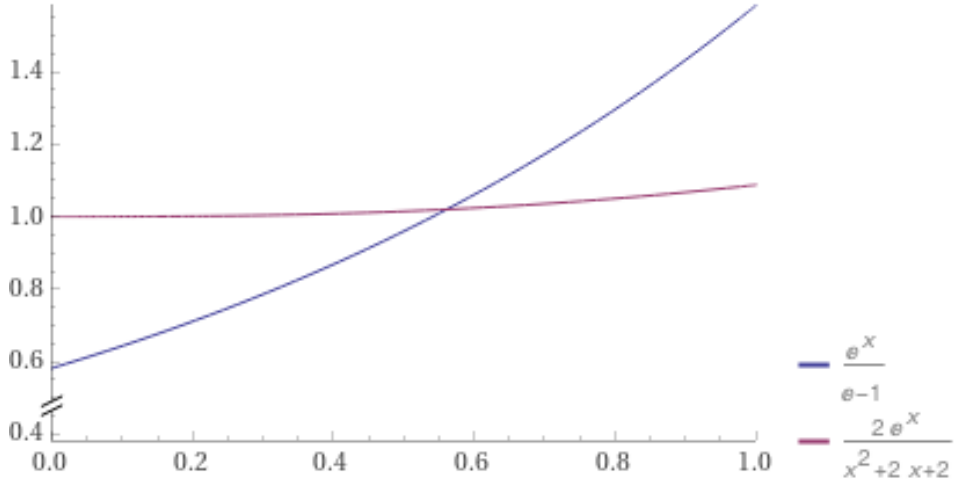


Figure 2.1: Effects of the importance sampling on $f(x) = \frac{e^x}{e-1}$ in the range $[0, 1]$. The initial integrating function (blue) has been divided by its Taylor expansion to the second order. The final integrating function (red) is much closer to a uniform distribution in the given range.

the sampled function is correctly $f(x)$. In practice, this is done by sampling a second random number y from a uniform distribution in the range $[0, 1]$ and accepting the initial variable if $y < \frac{f(x_i)}{g(x_i)}$. An example of the usage of this method is showed in Fig. 2.2.

From this outline of the methods, its advantages are clear:

- It is not needed the knowledge of the normalization of the probability distribution function nor its cumulative distribution function;
- The better $g(x)$ approximates to $f(x)$ the more efficient the method is because less proposals are rejected.

On the other hand, the more important drawback is that it can be inefficient because two random number must be generated and there is the possibility to completely lose them by rejecting the proposal.

2.2 MadGraph5_aMC@NLO

MadGraph5_aMC@NLO[4], later referred as `Madgraph5`, is a tool for automatically generating matrix elements for High Energy Physics processes. It is an open source project written in Python able to generate matrix elements at tree level for any Lagrangian model implementable with Feynrules and perform computations in the Standard Model at Next to Leading Order.

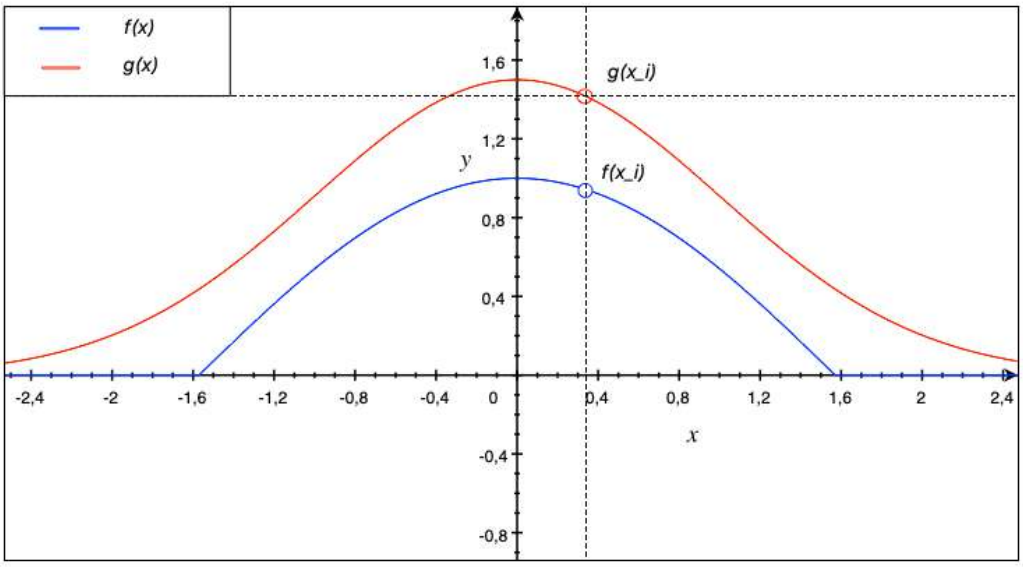


Figure 2.2: Rejection technique for the function $f(x)$ (blue) sampling random variables from the function $g(x)$ (red). A better approximation of $f(x)$ means that, after generating a number x_i, the ratio $\frac{f(x_i)}{g(x_i)} \approx 1$. Therefore, the probability to sample a number y from $[0, 1]$ uniformly such that $y < \frac{f(x_i)}{g(x_i)}$ is reduced, thus leading to a more efficient random sampling.

To generate the events, which can be defined as a possible collision occurring in a detector, three steps are necessary:

- Determine all the production channels, finding the Feynman diagrams with the given initial and final particles;
- Calculate the full matrix element $|M|^2 = MM^*$ where M is the sum of the Feynman diagrams;
- Integrate over the phase space to obtain the cross section as shown in Eq. 2.1

The algorithm used by MadGraph5 checks for the topologies only allowed by the model optimizing the generation procedure instead of checking for all possible topologies. To this end, two dictionaries are defined: the first called `Vertices` which maps all combination of n particles to the possible n-point interactions and the second called `Currents` which maps, for all n-point interactions, $n-1$ particles to all combinations of resulting particles for the interactions.

Given these two maps, the flag `from_group` (FG) indicates if, according to the vertices, a particle results from the merging of particles. With these tools, the algorithm will follow the following step to generate the matrix elements:

- Consider all the particles as outgoing and set the flag FG `True` for external particles;
- If a vertex can combine all the external particles create the diagram if at least two particles have FG sets as `True`;
- Group the particles with at least one FG = `True` in accordance to the currents dictionary;
- Set FG = `True` for the combined particles and repeat from the first step with these new external particles;
- Finally, stop the algorithm when at most two particles remain.

An example of the algorithm for the process $q\bar{q} \to t\bar{t}g$ is presented in Tab. 2.1.

Once all the diagrams have been generated, the full squared matrix element is evaluated. To perform the final phase space integration, the aspect of the integrating function needs to be considered. Generally, the phase space presents sharp peaks in different regions and to do an efficient integration the positions of these peaks must be found and properly mapped.

The approach used by MadGraph5 is called "Single-Diagram-Enhanced multi channel integration". The idea behind this method is to decompose the initial

Initial particles	Groups	Replacements	Result
$q, \bar{q}, t, \bar{t}, g$	$(q, \bar{q}), t, \bar{t}, g$	$(g), t, \bar{t}, g$	Fail, Only one FG = `True`
	$q, \bar{q}, (t, \bar{t}), g$	$q, \bar{q}, (g), g$	Fail, Only one FG = `True`
	$q, \bar{q}, t, (\bar{t}, g)$	$q, \bar{q}, (\bar{t}), g$	Fail, Only one FG = `True`
	$(q, g)\bar{q}, t, \bar{t}$	$(q), \bar{q}, t, \bar{t}$	Fail, Only one FG = `True`
	$(q, \bar{q}), (t, \bar{t}), g$	$(g), (g), g$	Diagram 1, two FG = `True`
	$(q, \bar{q}), t, (\bar{t}, g)$	$(g), t, (\bar{t})$	Diagram 2, two FG = `True`
	$(q, \bar{q}), (t, g), \bar{t}$	$(g), (t), \bar{t}$	Diagram 3, two FG = `True`
	$(q, g), (t, \bar{t}), \bar{q}$	$(q), (g), \bar{q}$	Diagram 4, two FG = `True`
	$q, (\bar{q}, g), (t, \bar{t})$	$q, (\bar{q}), (g)$	Diagram 5, two FG = `True`

Table 2.1: Summary of the algorithm used to generate matrix element for the process $q\bar{q} \to t\bar{t}g$, where q denotes the possible quarks in a proton-proton scattering; referring to the Standard Model the possible interaction vertices are $(q, \bar{q}, g)$, $(t, \bar{t}, g)$ and (g, g, g), as presented in Appendix A.

integrating function f in a sum of components such that:

$$f = \sum_{i=1}^{n} f_i \qquad \text{with} \qquad f_i \geq 0 \forall i. \tag{2.12}$$

Each of these components, referred as channels, should contains an easy to map peak function to get a much simpler integral for the single term.

Considering the particular problem the searched decomposition is the one that contains the single diagram squared matrix element:

$$f_i = \frac{|M_i|^2}{\sum_j |M_j|^2} |M_{tot}|^2, \tag{2.13}$$

where $|M_i|^2$ is the squared matrix element of a single diagram and $M_{tot} = \sum_i M_i$ is the full matrix element. Doing so, the initial complex integral is reduced in n independent integrations where the structure is known from the propagators:

$$\int |M_{tot}|^2 = \int \frac{\sum_i |M_i|^2}{\sum_j |M_j|^2|} |M_{tot}|^2 = \sum_i \int \frac{|M_i|^2}{\sum_j |M_j|^2} |M_{tot}|^2. \tag{2.14}$$

To further improve the results of the Monte Carlo integration for each of these channel is used Importance Sampling. This imply, as shown in Sec. 2.1, the definition of n new functions g_i each one similar to the f_i. Then, the final phase space integration is:

$$I = \int d\Phi f(\Phi) = \sum_{i=1}^{n} \int d\Phi g_i(\Phi) \frac{f_i(\Phi)}{g_i(\Phi)} = \sum_{i=1}^{n} I_i. \tag{2.15}$$

Given the basis of Eq. 2.13, the structure of the g_i is easily derived from the propagator structure of the corresponding diagram. The advantages of the splitting into n integration channels can be summarized in these points:

- More efficient single numerical integration;
- Errors are added in quadrature, so the procedure does not worsen the final result;
- The weight functions correspond to the single squared matrix diagram, so they can be calculated during the evaluation of $|M_{tot}|^2$;
- The procedure is parallel in nature, each channel can be calculated on a different node of a cluster and then combined to get the final result.

The events generated by MadGraph are *unweighted* events, this means that each event has the same weight and the number of events is proportional to a selected density of the phase space. The obtained unweighted distribution is the one that is expected to have in nature from a real experiment. On the other hand, a *weighted* sample has the same number of events for different probabilities in the phase space, resulting in an inevitable different weight for each event.

Notice that, even if the final sample produced by MadGraph5 is unweighted an un weighting procedure is needed. Indeed, the introduction of the adaptive functions g_i, as described above, cause the generation of points in the phase space to not be distributed accordingly the exact probability distribution function, so the events have different Monte Carlo weights. In addition, the channel splitting introduces a second weight related to the probability of occurrence of the single channel. Given these two weights, the unweighting procedures consist of using the rejection method to generate events for the different channels and keeping only the ones that have the same final weight that is calculated as the product of the initial ones.

MadGraph5 speed benchmarks

The concerns about the MC procedure described above are bound to the steep scaling of the CPU and time used for an increasing number of matrix elements and particles. Indeed, considering a process with N diagrams, an analytical computation leads to the calculation of N^2 matrix terms. Additionally, the number of diagrams scales factorially with the number of particles, so given n particles the number of diagrams scales like $n!^2$. MadGraph5 uses various techniques to optimize this process, firstly the squared matrix element is calculated in terms of helicity wave functions. From the starting wave functions of the external legs, they are combined to obtain new wave functions corresponding to the propagators in the diagram by successive helicity wave function calls.

Furthermore, wave functions that lead to the same sub-diagram can be reutilized between diagrams, effectively recycling all the already seen sub-diagrams and reducing the scaling complexity of the problem.

With these implementations the number of terms to calculate in the squared matrix element is reduced to N and the scaling of the number of terms with the number of particles approximately to $(n-1)!2^{n/2}$.

Even with these optimizations, it's vary likely to encounter processes with tens of diagrams that slow the events production dramatically. In Table 2.2 are presented the speed benchmarks for three processes: $pp \to t\bar{t}$, $gg \to ZZ$ and $gg \to HH$. The generation is done at LO and the number of events is 10k, notice that the last two processes are loop induced and thus even more complex to generate via MC

	# of independent diagrams	CPU time
$pp \to t\bar{t}$	4	52 s
$gg \to HH$	16	11m 9s
$gg \to ZZ$	28	2h 39m 5s

Table 2.2: Speed benchmark for the generations of 10k events for the processes $pp \to t\bar{t}$, $gg \to HH$, and $gg \to ZZ$. The number of diagrams evaluated is also reported.

simulations[1]. As can be seen, already for a simple 2 to 2 scattering the increase in time is noticeable.

The solution proposed in this thesis is to entirely bypass the matrix element calculations and train a generative model to parametrize the phase space distribution with an innovative Machine Learning technique, presented in the next Chapter, called Generative Adversarial Networks.

[1]The events generation has been performed on laptop

3 | Generative Adversarial Networks

Machine Learning is a field that aims to learn from data and make predictions. Machine Learning can be defined as a subfield of Artificial Intelligence with the goal of developing algorithms capable of learning from data automatically. The basic procedure for a Machine Learning method is to use input data and statistical analysis to produce an output. Meanwhile, the parameters of the model are updated to obtain better outputs evaluated in terms of a cost function.

The last decades have seen an increase in the ability to produce and analyze big datasets thanks to an unprecedented rise in computational capabilities. This development sets Machine Learning as an exploitable tool in categorical problems and generative models. In particular, Physics plays a unique role because it contributes to Machine Learning, indeed many techniques came from ideas in statistics, neuroscience, and biophysics, and it benefits from it as well considering that physicists are also at the forefront of using "big data" in various branches like particle physics and statistical mechanics.

In this thesis, an innovative training process has been utilized called Generative Adversarial Networks (GAN). It consists of training two separate neural networks which are Machine Learning models based on nature's concept of the nervous system. The neural network emerged as the most powerful model among the Machine Learning environment for data representation.

In this chapter, the standard setup for a Machine Learning problem is described in Sec. 1. Then the components of a neural network and its training process are presented in Secs. 2-3. Finally, these building blocks are combined to define a GAN and the particular architecture used in this thesis in Secs. 3-4.

3.1 Introduction to Machine Learning

Machine Learning can be divided into three categories:

- Supervised learning;
- Unsupervised learning;

- Reinforcement learning.

Notice that there is no distinct separation between these categories and many applications combine them in different ways.

In supervised learning, each input data is associated with the expected output value. The goal is to construct a mapping function that is able to give the correct output for new unseen data. This category is commonly used for categorical classification, where the labels are the possible classes for the problem (e.g. the ten classes for a handwritten single-digit integer), or regression, where the labels are the values of the function that we wish to approximate.

On the other hand, unsupervised learning uses unlabeled data and tries to model the underlying structure or distribution of the input data. The most used applications are clustering, finding the inherent groupings of the data, and generative modeling, generate a sample with the same distribution of the input data. GANs lies in this category, and one of its most outstanding application is the generation of high-quality images of non-existent people from a sample of close ups.

Finally, reinforcement learning consists of an algorithm or agent, that learns by interacting with its environment. Given a task, the agent will receive rewards by performing correctly and penalties for incorrect moves. The result of the model is an agent that learns without intervention from a human by maximizing its rewards and minimizing its penalties.

The workflow for a Machine Learning problem always combines three steps:

- Choosing how to represent the model. For example, neural networks, decision trees, or graphical models;

- Evaluate the model using the appropriate cost function. Subsequently, the parameters of the model are updated accordingly to the minimization of this function. Some examples include: accuracy, Mean Squared Error (MSE), Mean Absolute Error (MAE), likelihood, entropy, and Kullback-Leibler divergence;

- The search process to update the model or optimization. The most used are derivative-based algorithms like Stochastic Gradient Descend (SGD) and its more effective implementations like `Adam`, `Adadelta`, or `Adagrad`.

Even if these steps seems a straightforward procedure, correctly training and optimizing a model is a difficult task. To obtain better results and to ease the convergence of the model is often necessary to apply a transformation, referred as preprocessing, on the data before using them as input, the choice of the correct preprocess depends on the representation used. Furthermore, all the parameters

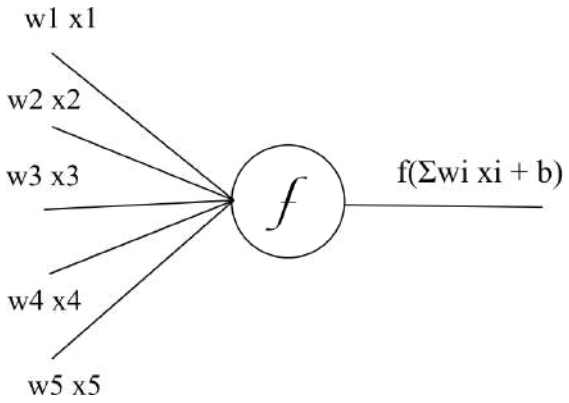


Figure 3.1: Artificial neuron with five inputs $x_1, \ldots, x_5$. Each inputs has its weight $w_1, \ldots, w_5$, after adding the bias b, the output value is obtained with the mapping function f as $O = f(\sum_i w_i x_i + b)$.

related to the model architecture and the training procedure, also called hyper-parameters, should be optimized because they can heavily influence the model output.

3.2 Neural networks

Nature's concept that neural networks are trying to imitate is the nervous system. The building block of the nervous system is a single neuron, which receives signals from other neurons and sends signals, through the axion, to another neuron.

Just like a neuron, an artificial neuron receives several inputs and produces a single output. In order to compute the output, to each input x_i is associated a weight w_i which express its importance. The output value O is not simply the sum of the weighted inputs, generally a bias weight b is added and mapped by a chosen function f called "activation function" $O = f(\sum_i w_i x_i + b)$. An example of an artificial neuron with five inputs is depicted in Fig. 3.1.

All the possible activation functions can be distinct in two groups: non-saturating functions with non-finite codomain like ReLU, Leaky ReLU, and linear, and saturating functions with possible saturation problems, caused by limited codomains, like Sigmoid, Tanh, and binary. The definitions of these functions are presented in Appendix B.

The combination of artificial neurons in different layers defines neural networks. The first and the last layers of the NN are respectively called input and output layers, while the layers in the middle are called hidden layers. Typically the input and output dimensions are fixed by the particular problem, for example, if the networks try to determine the digit in a handwritten image of $a \times b$ pixels, the

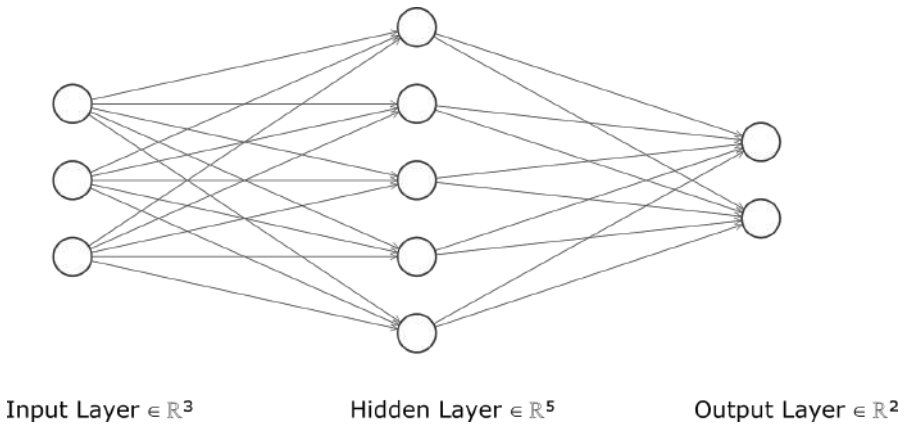


Figure 3.2: Example of the connections in a Feedforward Neural Network with one hidden layer.

input layer will have $a \times b$ neurons, one for each pixel, while the output layer 10 neurons, one for each digit.

The hidden layers that connect the network can be defined to obtain different architectures, here two of them are presented: a Feedforward Neural Network and a Convolutional Neural Network.

Feedforward Neural network

A FFNN is the simplest neural network architecture in which the data flows in one direction, from the input neurons to the output ones. Each node is connected to all nodes of the next layer creating a densely connected network. The importance of this kind of network concerns the universal approximation theorems which states that a FFNN with a single hidden layer can approximate any continuous function with arbitrary accuracy but with an exponentially large width. On the other hand, the representational power of the network increases also adding more hidden layers in a more computationally efficient way, this leads to the Deep Feedforward neural networks. An example of a FFNN is shown in Fig. 3.2.

Convolutional Neural Network

Architectures like a FFNN typically fail to exploits structures like locality and translational invariance which may be possessed by many datasets in Physics. To solve this problem, a particular class of neural networks has been developed initially for feature recognition in images called Convolutional Neural Networks (CNN). A 2D convolutional layer is characterized by the dimension of the spatial space (w, h) and by the number of channels c. The convolution consists of running several filters of fixed dimension over all locations in the spatial plane. Moreover, the stride s sets

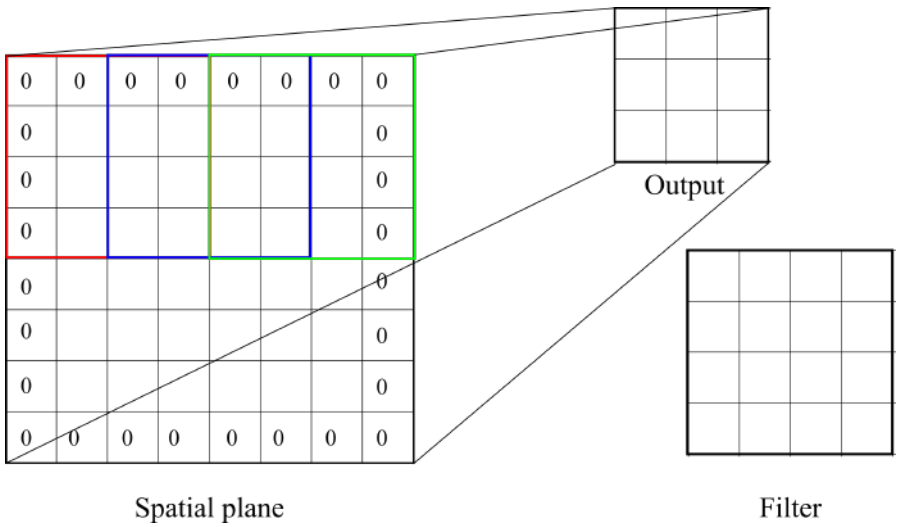


Figure 3.3: Example of convolution. The size of the filter is $(4, 4)$, the convolution is performed with strides 2 and 1 zero padding. The grids of different colors represent the filter moving in the spatial plane mapping the initial space to a single output. The final output is a grid of dimensions $(3, 3)$.

how many neurons of the convolutional layer the filter translates when performing the convolution. Finally, it is possible to pad the input with p zeros in order to get the desired output dimension. Fig. 3.3 shows an example convolution on one spatial plane.

Given these informations, the number of output neurons in the layer is given by:

$$dim = \frac{w - f + 2p}{s} + 1 \tag{3.1}$$

while the number of filters is fixed by the user. An example of a CNN with a layer of 8 filters of dimension $(32, 32)$ coarse grained in 16 filters of dimension $(16, 16)$ is shown in Fig. 3.4. The filter is of size $(4, 4)$, with stride 2 and 1 zero padding.

The neural networks used in this thesis are Deep Convolutional Neural Networks and their detailed structure is described in Sec. 3.5.

3.3 Training a neural network

Once the architecture has been defined the neural network must be trained to optimize the weights for the specific problem. The training of a neural network requires steps that are common to the vast possible architectures:

- Construct a cost/loss function to minimize;

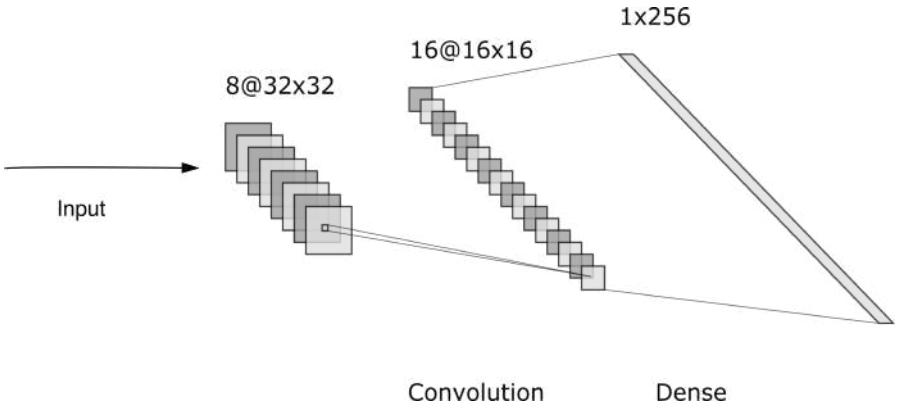


Figure 3.4: Example of a Convolutional Neural Network with two convolutional layers with 8 and 16 filters, see text for details, and a final densely connected layer with 256 nodes.

- calculate the gradients of the cost function with respect to all parameters via the "backpropagation algorithm";
- use a gradient descent-based procedure to optimize all the weights and biases.

The loss function is mainly fixed by the output layer. Given a batch size N of expected values $\bar{y}_i$, a neural network with parameters $\bar{w}$ and predicted values $f(\bar{y}_i; \bar{w})$, for continuous data the commonly used functions are the Mean Squared Error (MSE) and the Mean Absolute Error (MAE):

$$L_{MSE}(\bar{w}) = \frac{1}{N} \sum_{i=1}^{N} |\bar{y}_i - f(\bar{y}_i; \bar{w})|^2 \tag{3.2}$$

$$L_{MAE}(\bar{w}) = \frac{1}{N} \sum_{i=1}^{N} |\bar{y}_i - f(\bar{y}_i; \bar{w})| \tag{3.3}$$

Instead, for categorical data, when the output layer is a classifier with two (binary) or more labels (softmax), the most used loss function is the Cross-Entropy (CE). For binary labels, where $y_i \in \{0, 1\}$, the CE is defined by:

$$L_{CE}(\bar{w}) = -\sum_{i=1}^{N} y_i \log f(y_i; \bar{w}) + (1 - y_i) \log \left[1 - f(y_i; \bar{w})\right] \tag{3.4}$$

Fixed the loss function, a specialized algorithm called backpropagation is used to determine the gradients of the loss function. To see this algorithm in actions, consider a FFNN with L layers indexed by l. Define the weight, which connect the

k-th neuron in layer l-1 to the j-th neuron in layer l, w^l_{jk} and the bias b^l_j. The next steps is to relate the output of the j-th neuron of the l-th layer a^l_j to the outputs of the previous layer:

$$a^l_j = f(\sum_k w^l_{jk} a^{l-1}_k + b^l_j = f(z^l_j) \tag{3.5}$$

where f is the activation function and z^l_j is defined by the linear weighted sum. Moreover define the variation of the j-th neuron in the l-th layer Δ^l_j as the variation in the cost function L:

$$\Delta^l_j = \frac{\partial L}{\partial z^l_j} \tag{3.6}$$

where the derivative of L and f must be known.

It is possible to find four equations that relate the activations a^l_j, the weighted inputs z^l_j and the variations Δ^l_j of all the neurons. The first is the Eq. 3.6, the second relates the cost function and the biases:

$$\Delta^l_j = \frac{\partial L}{\partial z^l_j} = \frac{\partial L}{\partial b^l_j}\frac{\partial b^l_j}{\partial z^l_j} = \frac{\partial L}{\partial b^l_j} \tag{3.7}$$

The third and the fourth equation are defined via chain rule of Δ^l_j and $\frac{\partial L}{\partial w^l_{jk}}$:

$$\Delta^l_j = \sum_k \frac{\partial L}{\partial z^{l+1}_k}\frac{\partial z^{l+1}_k}{\partial z^l_j} = \sum_k \Delta^{l+1}_k \frac{\partial z^{l+1}_k}{\partial z^l_j} = (\sum_k \Delta^{l+1}_k w^{l+1}_{jk}) f'(z^l_j) \tag{3.8}$$

$$\frac{\partial L}{\partial w^l_{jk}} = \frac{\partial L}{\partial z^l_j}\frac{\partial z^l_j}{\partial w^l_{jk}} = \Delta^l_j a^{l-1}_k \tag{3.9}$$

The equations Eqs. 3.6, 3.7, 3.8, 3.9 are combined to get a backpropagation algorithm to calculate the gradients of the loss function:

- Calculate the activations a^1_j of all the neurons in the input layer;
- compute z^l and a^l for each subsequent layer;
- calculate the error of the top layer using Eq. 3.6
- use Eq. 3.8 to propagate the variation backwards and calculate Δ^l_j for all layers;
- use Eqs. 3.7, 3.9 to calculate $\frac{\partial L}{\partial w^l_{jk}}$ and $\frac{\partial L}{\partial b^l_j}$.

The power of this algorithm allows to calculate all the derivatives of the loss function with only a "forward" and a "backward" pass of the neural network.

Finally, the parameters are updated by a gradient descent-based algorithm. The simplest method is the Stochastic Gradient Descent (SGD), the basic idea is to take, at each iteration, steps proportional to the negative gradient of the loss function because this direction defines the fastest way to decrease L. Given the parameters $\bar{w}$ and the loss function $L(\bar{w})$, the update is done according to the equation:

$$\bar{w}_{n+1} = \bar{w}_n - \eta\bar{\nabla}L(\bar{w}_n) \tag{3.10}$$

where n is the training iteration and η is a hyperparameter fixed by the user called "learning rate". The choice of η is fundamental because a small learning rate causes slow convergence, while a high learning rate may cause jumps around the minimum of the loss function. The stochastic nature of the algorithm is tied to the usage of a subsample, defined by the batch size, of the input data to evaluate the gradients.

3.4 Generative Adversarial Networks

A Generative Adversarial Network (GAN) [11] is composed of two neural networks called Generator and Discriminator. The two networks compete against each other in a zero-sum game where an improvement for one agent implies the decay of the other. The task of the generator is to reproduce the distribution of the input data $p_{data}(\bar{x})$, to do so an input random noise $p_z(\bar{z})$ is mapped by the generator to the data space $G(\bar{z};\bar{w}_g)$ where $\bar{w}_g$ are the parameters of the network, resulting in the distribution $p_{gen}(\bar{x})$.

Instead, the discriminator outputs a single scalar $D(\bar{x};\bar{w}_d)$ that represents the probability that $\bar{x}$ belong to the input data or the generated sample. The training simultaneously optimize D, maximizing the accuracy of the assignment of the labels, and G, minimizing the quantity $\log(1 - D(G(\bar{z})))$ which is the probability to correctly recognize generated samples by the discriminator.

This algorithm defines a two-player minimax game with loss function:

$$\min_G \max_D L(D,G) = E_{\bar{x}\sim p_{data}(\bar{x})}\left[\log D(\bar{x})\right] + E_{\bar{z}\sim p_z(\bar{z})}\left[\log(1 - D(G(\bar{z})))\right] \tag{3.11}$$

This defined game has a global minimum for $p_{gen} = p_{data}$. To show that, let's prove that for G fixed the best D is the quantity:

$$D^*(\bar{x}) = \frac{p_{data}(\bar{x})}{p_{data}(\bar{x}) + p_{gen}(\bar{x})} \tag{3.12}$$

by definition of the minimax game in Eq. 3.11 the procedure for D is to maximize $L(D)$ which can be written as:

$$L(D,G) = L(G) = \int_{\bar{x}} p_{data}(\bar{x})\log(D(\bar{x})) + p_{gen}(\bar{x})\log(1 - D(\bar{x}))dx \tag{3.13}$$

the maximum in $[0,1]$ of the integrating function corresponds to $\frac{p_{data}(\bar{x})}{p_{data}(\bar{x})+p_{gen}(\bar{x})}$.

For $p_{gen} = p_{data}$, $D^*(\bar{x}) = 1/2$ and $L(D^*, G) = L(G) = -\log 4$. To prove that this is the global minimum of the loss function add and subtract to L the quantity:

$$E_{x\sim p_{data}}\left[-\log 2\right] + E_{x\sim p_{gen}}\left[-\log 2\right] = -\log 4 \tag{3.14}$$

With this manipulation the loss function can be written as:

$$\begin{aligned} L(G) &= -\log 4 + E_{x\sim p_{data}}\left[\log(2D^*(\bar{x}))\right] + E_{x\sim p_{gen}}\left[\log(2 - 2D^*(\bar{x}))\right] \\ &= -\log 4 + E_{x\sim p_{data}}\left[\log(2\frac{p_{data}(\bar{x})}{p_{data}(\bar{x}) + p_{gen}(\bar{x})})\right] + E_{x\sim p_{gen}}\left[\log(2\frac{p_{gen}(\bar{x})}{p_{data}(\bar{x}) + p_{gen}(\bar{x})})\right] \\ &= -\log 4 + KL(p_{data}||\frac{p_{data} + p_{gen}}{2}) + KL(p_{gen}||\frac{p_{data} + p_{gen}}{2}) \\ &= -\log 4 + 2 \cdot JS(p_{data}||p_{gen}) \end{aligned} \tag{3.15}$$

where the Kullback-Leibler (KL) and the Jensen-Shannon (JS) divergences have been defined such that:

$$KL(p||q) = \int_x p(x)\log\left(\frac{p(x)}{q(x)}\right) \tag{3.16}$$

$$JS(p||q) = \frac{1}{2}\left(KL(p||\frac{p+q}{2}) + KL(p||\frac{p+q}{2})\right) \tag{3.17}$$

where $p(x)$ and $q(x)$ are probability distribution functions. The KL and the JS divergences are measures of how much two distributions are different. In particular, notice that the JS divergence is always positive and zero when the two distributions are identical. Thus, the global minimum of $L(D^*, G)$ is $-\log(4)$ when the true and generated distributions match $p_{gen} = p_{data}$.

The workflow for training a GAN can be summarized in these steps:

- Some random noise is generated from a chosen distribution (e.g. uniform or Gaussian);
- the generator produces a sample from the random noise;
- the generated and the true samples are feed into the discriminator who outputs 0 for the fake sample and 1 for the true one;

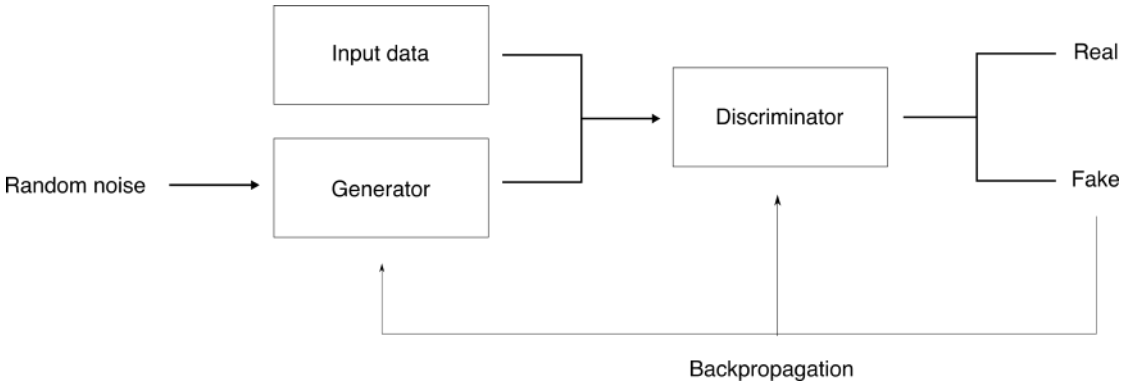


Figure 3.5: Workflow of a GAN. The generator produces samples from random noise trying to imitate the input data, while the discriminator discerns the true samples from the fake ones. The parameters of the networks are updated via backpropagation from the discriminator to the generator.

– finally, the parameters of the neural networks are updated via backpropagation.

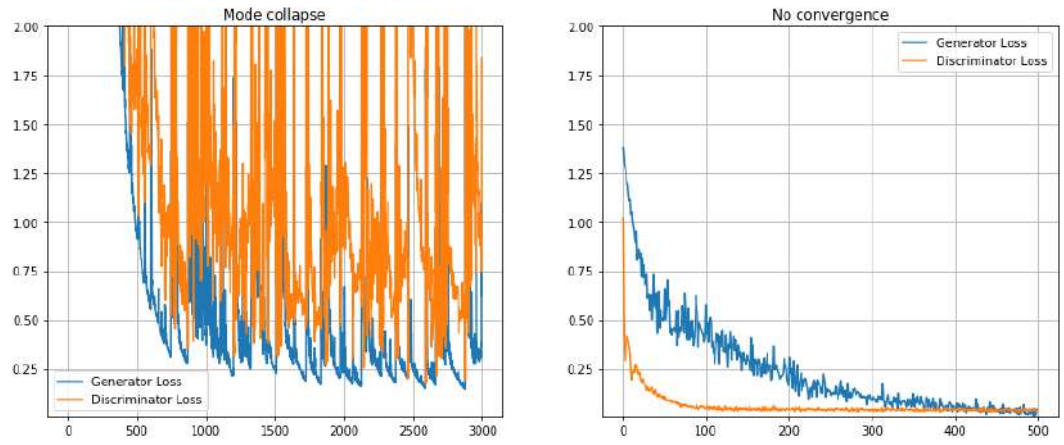
A chart of the workflow of a GAN is shown in Fig. 3.5. However, even if the defined algorithm used to train a GAN is simple, GANs have failure modes that make the achievement of good training a hard process. The most common failures are:

- Mode collapse, when the generator learns to produce a small set of outputs. In response, the discriminator learns only to reject these particular outputs;
- convergence failure is referred to as the inability of the GAN to converge at an equilibrium point.

None of these problems have been fully solved and only attempts can be done to resolve these issues depending on the particular training. A way to identify these failures is to watch the quantities that each network optimizes. In a mode collapse, both losses present fluctuations whose minimums correspond to the specialization of a single set output. Instead, with convergence failure one loss function, typically the discriminator, goes to zero while the other one rise meaning that one neural network is too good respect the other, or both go to zero. In Fig. 3.6 are shown the loss function of the failure modes encountered in this thesis.

3.5 GAN architecture

This section will describe the architecture utilized in this thesis which emerged as the one that allowed to resolve the training issues of the previous section. The



(a) (b)

Figure 3.6: Failure modes encountered. (a) Mode collapse: the loss functions keep fluctuating without stabilize. (b) Convergence failure: the GAN fail to converge.

chosen architecture is a Deep Convolutional GAN (DCGAN) where the two neural networks are deep convolutional neural networks.

Generator
The generator has 125835 trainable parameters and is formed by the following layers:

- Two **Dense** layers, the first used to add the required neurons to reshape the input in a 2D 10×10 spatial plane and the last used to obtain the desired number of outputs from the generator.
- Four **Transposed convolution** layers using several filters decreasing from 128 to 16, they can be seen as convolutional layers where the filters are applied to the output to get the spatial plane;
- Each layer is followed by a **Batch Normalization** layer, they normalize the inputs to a zero-mean and unit variance distribution. The task of these layers is to stabilize the training and prevent possible failures;
- **Leaky ReLU** layers with $\alpha = 0.2$ used to apply this activation function to the outputs of batch normalization;
- A **Flatten** layer is used to transform the 2D spatial plane back to an array of neurons.

The generator training is done by initially sampling an array of size 100 from a uniform distribution. Then, the noise is used as input in the network formed by the union of generator and discriminator where only the parameters of the generator are trainable. Finally, the binary cross-entropy between the outputs and an array of ones, corresponding to the probability of predicting the generated sample as true, is calculated and used for the weights update. The output layers is activated, depending on the used preprocessing, by a linear function or a hyperbolic tangent. Notice that the calculated BCE correspond to the maximization of the quantity $E_{\bar{z}\sim p_z(\bar{z})}\left[\log(D(G(\bar{z})))\right]$ instead of what reported in Eq. 3.11. This leads to the same fixed point of the dynamics of G and D but the early gradients calculated are bigger reducing the problem of vanishing gradients.

Discriminator
The discriminator has 101513 trainable variables and is formed by the following layers:

- Two **Dense** layers, the single scalar output, activated by a sigmoid function, and the initial layer to reshape in a spatial plane of the same dimension of the generator;
- Four **Convolutional** layers with a decreasing number of filters from 128 to 16;
- **LeakyReLU** layers to apply the activation function;
- One **Dropout** layers with rate= 0.2 used to prevent overfitting. It sets to 0 the input neurons with probability given by the rate.

The discriminator training is done by calculating the binary cross-entropy of a generated sample with an array of zeros and the cross-entropy of a true sample with an array of ones.

This GAN has been implemented and trained using the software library `Keras` [9] with `TensorFlow 2.3`[1] backend. A summary of both networks alongside a visual representation is shown in Figs. 3.7, 3.8, 3.9.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Dense_Input (Dense)          (None, 200)               20200
_________________________________________________________________
BN_1 (BatchNormalization)    (None, 200)               800
_________________________________________________________________
Leaky_ReLU_1 (LeakyReLU)     (None, 200)               0
_________________________________________________________________
reshape (Reshape)            (None, 10, 10, 2)         0
_________________________________________________________________
TConv_1 (Conv2DTranspose)    (None, 10, 10, 128)       2304
_________________________________________________________________
BN_2 (BatchNormalization)    (None, 10, 10, 128)       512
_________________________________________________________________
Leaky_ReLU_2 (LeakyReLU)     (None, 10, 10, 128)       0
_________________________________________________________________
TConv_2 (Conv2DTranspose)    (None, 10, 10, 64)        73728
_________________________________________________________________
BN_3 (BatchNormalization)    (None, 10, 10, 64)        256
_________________________________________________________________
Leaky_ReLU_3 (LeakyReLU)     (None, 10, 10, 64)        0
_________________________________________________________________
TConv_3 (Conv2DTranspose)    (None, 10, 10, 32)        18432
_________________________________________________________________
BN_4 (BatchNormalization)    (None, 10, 10, 32)        128
_________________________________________________________________
Leaky_ReLU_4 (LeakyReLU)     (None, 10, 10, 32)        0
_________________________________________________________________
TConv_4 (Conv2DTranspose)    (None, 10, 10, 16)        4608
_________________________________________________________________
BN_5 (BatchNormalization)    (None, 10, 10, 16)        64
_________________________________________________________________
Leaky_ReLU_5 (LeakyReLU)     (None, 10, 10, 16)        0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
Dense_Output (Dense)         (None, 3)                 4803
=================================================================
Total params: 125,835
Trainable params: 124,955
Non-trainable params: 880
_________________________________________________________________
```

Figure 3.7: Generator structure.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Dense_Input (Dense)          (None, 200)               600
_________________________________________________________________
reshape_1 (Reshape)          (None, 10, 10, 2)         0
_________________________________________________________________
Conv_1 (Conv2D)              (None, 10, 10, 128)       2432
_________________________________________________________________
Leaky_ReLU_1 (LeakyReLU)     (None, 10, 10, 128)       0
_________________________________________________________________
Conv_2 (Conv2D)              (None, 10, 10, 64)        73792
_________________________________________________________________
Leaky_ReLU_2 (LeakyReLU)     (None, 10, 10, 64)        0
_________________________________________________________________
Conv_3 (Conv2D)              (None, 10, 10, 32)        18464
_________________________________________________________________
Leaky_ReLU_3 (LeakyReLU)     (None, 10, 10, 32)        0
_________________________________________________________________
Conv_4 (Conv2D)              (None, 10, 10, 16)        4624
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
Leaky_ReLU_4 (LeakyReLU)     (None, 1600)              0
_________________________________________________________________
Dropout (Dropout)            (None, 1600)              0
_________________________________________________________________
Dense_Output (Dense)         (None, 1)                 1601
=================================================================
Total params: 101,513
Trainable params: 101,513
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.8: Discriminator structure.

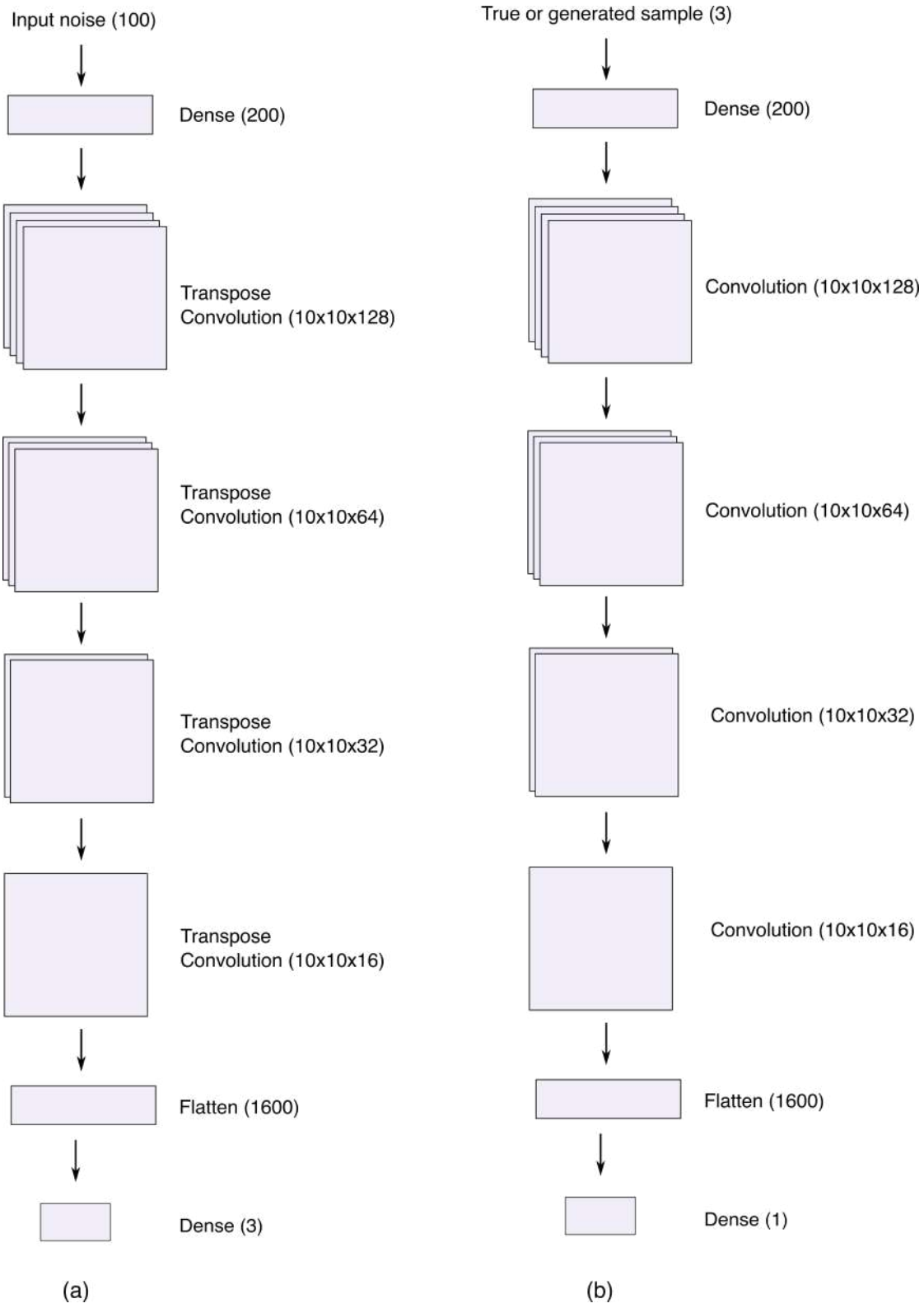


Figure 3.9: Visual representation of the GAN with layers dimension for the (a) Generator and (b) Discriminator.

4 | Results

This chapter presents the results obtained from the GAN in three channels: $pp \to t\bar{t}$, $gg \to ZZ$ and $gg \to HH$. Within each channel, the Mandelstam variables s and t and the rapidity y are used to identify the event and to train the GAN with an increasing number of events from 10k to 1M. The performance of the generator is evaluated in terms of the accordance between the true and the generated distribution evaluated with the KL divergence, JS divergence, and the ratio of these distributions. To improve the results, an optimization on a set of hyperparameters is done using the tool `Hyperopt` [6].

This chapter is organized as follows. The first section presents the pre- and post-processing done on the training sample. Then, Sec. 2 contains the results of the hyperparameters optimization and the first test of the GAN architecture. Finally, in Secs. 3-4 the results of the trainings in the three processes are presented. Firstly comparing the distributions of a low and a high statistics sample and then evaluating the data augmentation capability of these models.

4.1 Sample processing

The events within each channel are unweighted events produced using `MadGraph5` and written in a Les Houches Events (LHE) [3] file. The selected channels are three different $2 \to 2$ hadron scattering where the beam energy is $E_{beam} = 6500$ GeV:

- $pp \to t\bar{t}$, a proton-proton scattering into a top quark pair at LO;
- two loop-induced processes at LO, $gg \to ZZ$ and $gg \to HH$.

From a given LHE file the events are translated in an array of dimension $[3, N]$ where N is the number of events in the file. The rows of this matrix are filled with three kinematical variables that define the event. The chosen variables are the Mandelstam variables s, t and the rapidity y in the parton scattering reference

frame. They are defined by the equations:

$$s = (p_1 + p_2)^2 = (p_3 + p_4)^2 \tag{4.1}$$
$$t = (p_1 - p_3)^2 = (p_2 - p_4)^2 \tag{4.2}$$
$$y = \frac{1}{2}\log\left(\frac{x_1}{x_2}\right) \tag{4.3}$$

where p_i is the momentum of the i-th particle and $x_i = E_i/E_{beam}$ is the energy fraction of the beam energy carried by the scattering parton.

Before feeding the GAN with the obtained events, a preprocessing is applied using the `Scikit-Learn`[18] package. Two possible preprocessing are considered: a standardization or a Yeo-Johnson power transform followed by a `minmax` scaling in the range $[-1, 1]$ of the input features. An example of the effects of these transformations on a sample of 10k $pp \to t\bar{t}$ events is shown in Fig. 4.1. In the first case, the distributions are not limited therefore the output layer of the generator is a linear function. Instead, in the second case the output is contained in the range $[-1, 1]$, therefore the hyperbolic tangent activation function is more suitable in the output layer. Then, the generated events are post-processed using the inverse transformation of the used scaler.

To rebuild the momenta of the particles from the kinematical variables, assuming massless incoming partons, in the laboratory reference frame, use Eqs. 4.1 and 4.3 to find the energy fractions of the incoming particles:

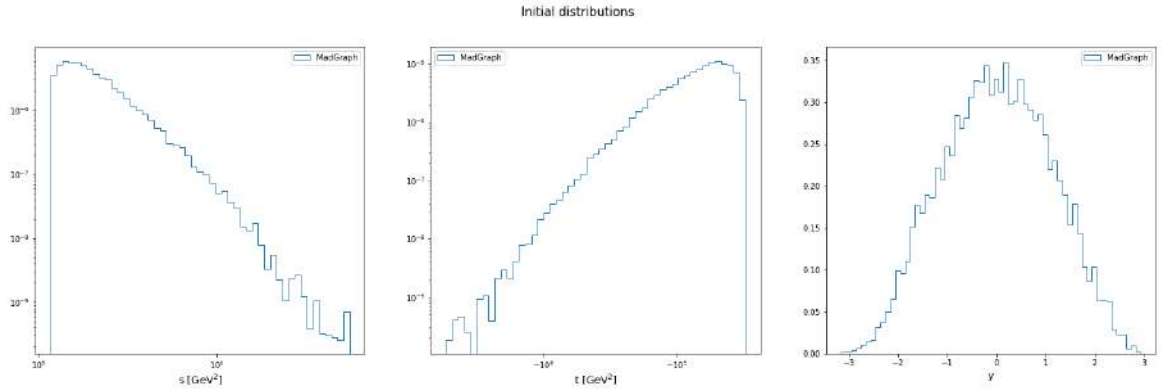
$$x_2 = \frac{\sqrt{s}}{2E_{beam}}e^{-y}, \tag{4.4}$$
$$x_1 = x_2 e^{2y}, \tag{4.5}$$

where E_{beam} is the beam energy. Thus, the incoming momenta in the lab frame are:

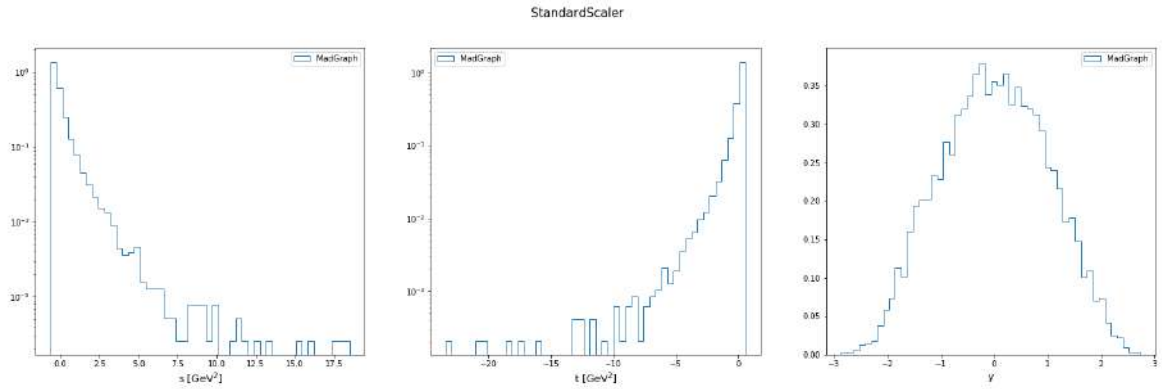
$$p_1 = [x_1 E_{beam}, 0, 0, x_1 E_{beam}]\,, \tag{4.6}$$
$$p_2 = [x_2 E_{beam}, 0, 0, -x_2 E_{beam}]\,. \tag{4.7}$$
$$\tag{4.8}$$

The other two momenta can be found using the Eqs. 4.1 and 4.2. In the parton scattering reference frame, the energy E_3^*, the scattering angle θ^* and the longitudinal momentum p_z^* of the third particle are:

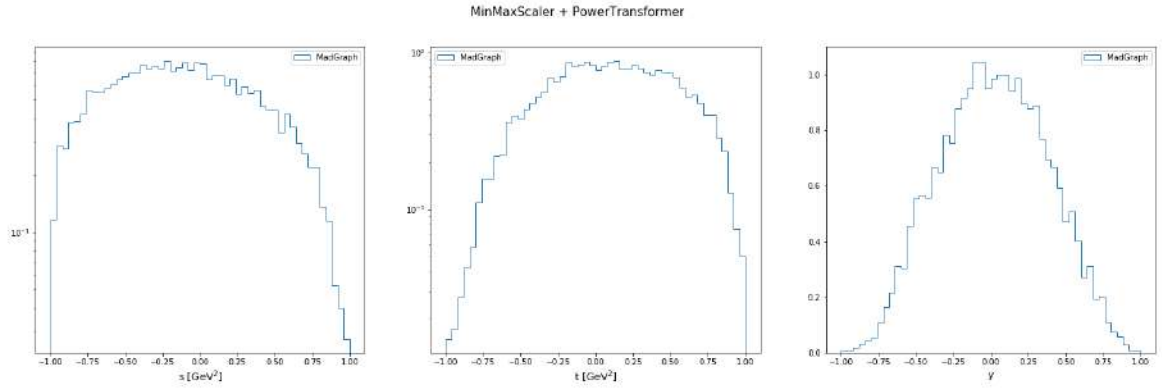
$$E_3^* = \frac{\sqrt{s}}{2}, \tag{4.9}$$
$$\theta^* = 1 + 2\frac{t}{s}, \tag{4.10}$$
$$p_z^* = E_3^* \mathrm{cos}.\theta^*. \tag{4.11}$$



(a)



(b)



(c)

Figure 4.1: Preprocessing of the input features. (a) Distribution without preprocessing. (b) Standard scaling. (c) Power Transformer followed by a `MinMax` scaling in range $[-1, 1]$.

Finally, a Lorentz boost is needed in the longitudinal direction to obtain the momentum in the laboratory reference frame:

$$E_3 = E_3^* \cosh y + p_z^* \sinh y, \tag{4.12}$$
$$p_z = E_3^* \sinh y + p_z^* \cosh y, \tag{4.13}$$
$$p_T = \sqrt{E_3^2 - (p_z^2 + m^2)}, \tag{4.14}$$

where p_T is the transverse momentum, found by energy conservation, and m is the mass of the particle. Applying the momentum conservation law, it is possible to find the related quantities of the final outgoing particle:

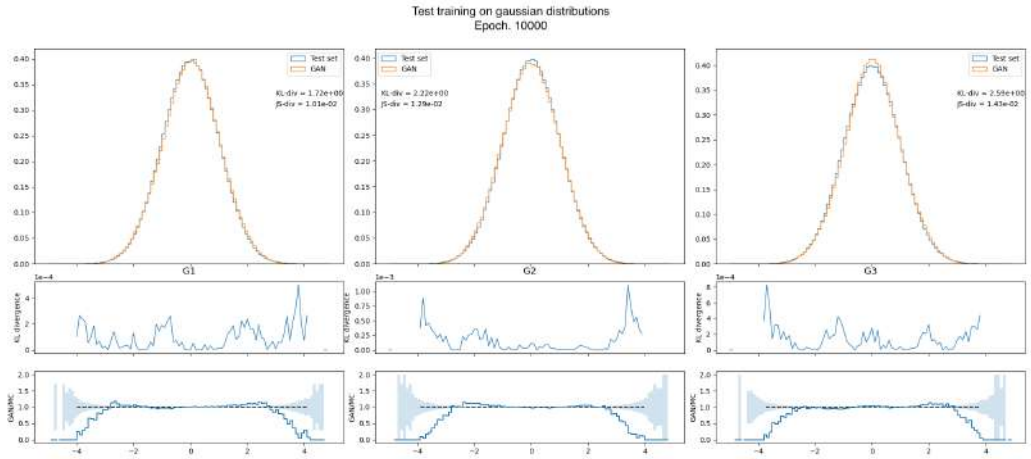
$$p_3 = [E_3, p_T, 0, p_z], \tag{4.15}$$
$$p_4 = [E_1 + E_2 - E_3, -p_T, 0, E_1 - E_2 - p_z]. \tag{4.16}$$

4.2 GAN test and hyperparameter optimization

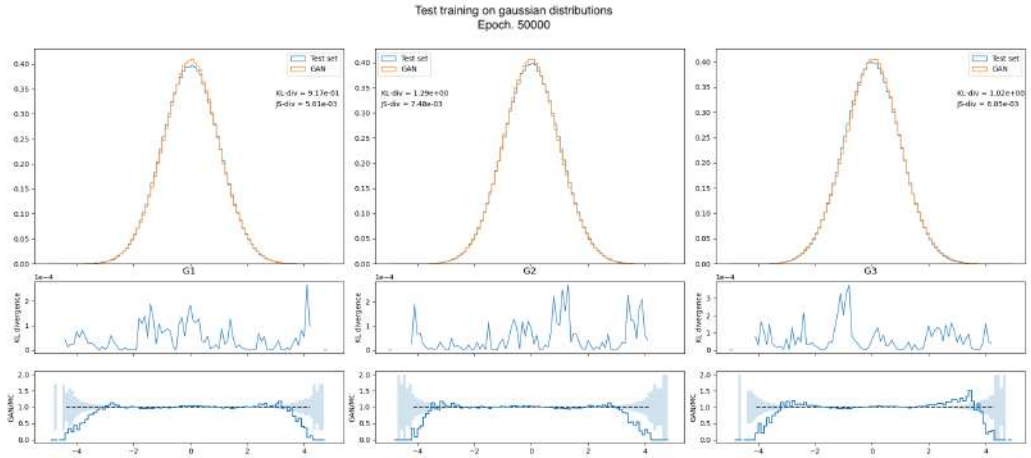
Before starting to train the models in different channels, benchmark training is done to test the effectiveness of the GAN. The test consists of training the GAN using data sampled from three Gaussians, one for each input feature, and measuring the ability of the GAN to reproduce these distributions. The input sample is firstly scaled in the range $[-1, 1]$ and then feed into the neural networks. The methods used to evaluate the performance are the KL divergence and the ratio of the GAN to the expected distribution in each bin. The optimizer used is SGD with default learning rate and the total number of epochs utilized is 100k, a snapshot of the obtained distributions at different epochs is shown if Fig. 4.2.

This test shows that the proposed GAN is working and it is able to reproduce the initial core distribution already after 10k epochs where the error in the ratio plot stabilize within few percent. A deterioration of the output can be seen in the tails of the distribution where the ratio error falls at 50% due to lack of data. However, increasing the number of training epoch improve the result also in the aforementioned region as it can be seen in Fig. 4.2 (c).

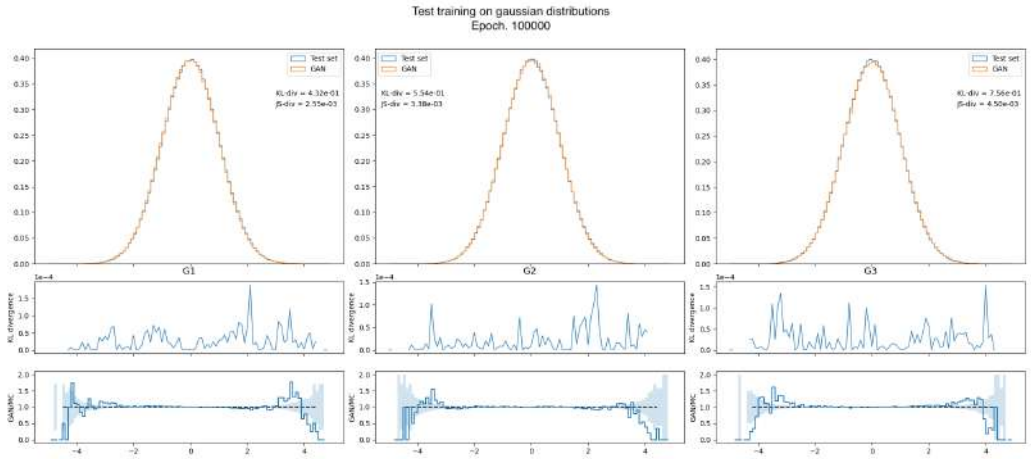
Subsequently, the optimization through `Hyperopt` has been performed. The optimization process trains different models selecting the hyperparameters with a Bayesian method called Tree-structure Parzen Estimator (TPE). At the end of the training, the mean of the KL divergence between the distributions of the three input features is calculated and used as a metric to evaluate the performance of the model. The first search space defined includes the number of epochs, the batch size, the learning rate plus the distinction in the two possible preprocessing defined in the above section.



(a)



(b)



(c)

Figure 4.2: Training on three gaussian distributions. The first subplot shows the KL divergence for each bin, while the second one shows the ratio between the generated and the true distribution. The snapshot is done at 10k (a), 50k (b) and 100k (c) epochs.

In Fig. 4.3 the loss metric for each iteration obtained using 10k $pp \to t\bar{t}$ events and the SGD optimizer is shown. The results show that the minimax preprocessing is less spread and thus preferable over the standard one. Regarding the other parameters, a higher batch size reduces the fluctuations introduced by the stochastic optimizer while, with both preprocessing, a higher learning rate gave better results. Finally, the number of epochs shows that there is no improvement in the performances over 30k epochs.

A second optimization is done focusing on the optimizer and the learning rate. The possible choices and the resulting loss metrics are shown in Fig. 4.4. From the results, `Adadelta` is the most consistent and performing optimizer and a learning rate below 0.05 deteriorates the predictions of the model. After the hyperparameters optimization, all trainings have been done with a `minmax` preprocessing, with batch size 512, using the `Adadelta` optimizer with a learning rate of 0.3. However, due to the usage of the stochastic approach, the output distributions present fluctuations around the best local minimum found. In order to select the best model, the training is prolonged to 50k epochs and the best iteration, evaluated in terms of the KL divergence, is the reference for the best model to save. These results for 10k events are rescaled for samples of higher size keeping constant the ratio between the total number of events fed into the network and the sample size.

The loss functions of all these trainings are similar and an example one is reported in Fig. 4.5. The loss functions show that the optimal loss value is reached after $\sim$1000 epochs. Nevertheless, the training continues after that point because the correlation between the input features keeps improving up to tens of thousands epochs, where the models start to be saved. The training time for such architecture goes from $\sim$2 hours to $\sim$6 hours depending on the sample size, done using an NVIDIA Tesla P100-PCIe-12GB GPU.

4.3 Generated samples

Using the training setup defined in the previous section, the first target is to reproduce the same statistics obtained with `MadGraph` using the GAN. To this purpose, two samples for each channel have been generated, a low statistics sample with 10k events and a high statistics sample with 1M events. The following plots present a comparison between the generated and the true distributions for both samples, the KL divergence, and the ratio have been calculated for each bin. The error on the ratio subplot is the statistical counting error calculated as the square root of the number of events. Furthermore, the correlation between invariants has been checked through three 2D scatter plots and the agreement between the plots has been evaluated binning the space and calculating the counting error. Finally, the distributions of the standard deviations formed by all bins are produced and

fitted to a gaussian,

$pp \to t\bar{t}$ channel
The obtained distributions in the $pp \to t\bar{t}$ channel are shown in Fig. 4.6. In order to do not neglect the tails region the s and t histograms have log-scaled bins. The first histogram shows that the ratio error, for the sample of 10k events, goes from 10% in the core region to 20% in the tail region and that the result is everywhere compatible within 3σ. As shown in Fig. 4.7, the correlations between the kinematical variables are well reproduced. In particular, the subplot (b) shows that the model uniformly learned the input distribution with few bins that have an error, in terms of standard deviations, above 3σ. These bin-wise errors are collected into the (c) histogram which shows that this distribution is well centered in zero with variance 1.5.

Regarding the sample with 1M events, the ratio error is below 5% in the core region for all input features and it degrades up to 10% before the high energy region. Here and for high rapidity points, the model has difficulties in finding the proper parameters due to the lack of data. The studies on the correlations are summarized in Fig. 4.8. Here, the same conclusions of the sample with 10k events can be drawn with the exception of the correlation between s and t. The models struggle to parametrize the cut imposed by the Mandelstam invariants, resulting in the leakage of events over the boundary and its worst representation.

$gg \to ZZ$ channel
The results in this channel are in accordance with the ones presented above. Fig. 4.9 shows that the ratio between the true and generated distributions is compatible within 3σ in the majority of the bins for the 10k and 1M events samples and the KL divergence subplots show that the distribution is uniformly learned. The only region not well reproduced is the tails region where the model falls faster than the true distribution.

The correlations plots in Figs. 4.10, 4.11 show that the correct relation between kinematics is uniformly learned for both samples. Indeed, the error distributions are zero-centered with a variance of $\sim$1.5. The noticeable areas with deviations from the true correlations are in the sample with 1M events. These areas are the cut between s and t, which shows some events leakage, and the high-rapidity region, which is not symmetrical.

$gg \to HH$ channel
This last channel has a slightly different phase space where the matrix element suppresses the Higgs di-boson at the production threshold. These differences can be seen in the histogram plots in Fig. 4.12 for the samples of 10k and 1M events.

This effect impacts the s and t distributions especially in the case of 1M events. Indeed, the histograms of the sample with 10k events show a ratio error within 20% and compatible with the statistical error of the true sample. On the other hand, the sample with 1M events presents a ratio error of 5% in the core region above the statistical error. The rapidity is not affected and shows similar results to the other channel with errors below 5%. Regarding the tail region, the model struggles to reproduce the true distribution, as in the other channels, due to lack of data.

The correlations plot in Fig. 4.13 refers to the sample of 10k events. In this case, a minor leakage is present in the cut between the Mandelstam invariants, while the other plots show that the correlations are correctly uniformly reproduced. As in the other channels, the distributions of the errors are zero-centered with a variance of $\sim$1.5. On the contrary, the correlations of the high statistics sample, presented in Fig. 4.14, show better where the distributions are wrongly reproduced. The interested areas are the cut between the two Mandelstam variables, which shows a major leak of events, and the low energy region, which is, compared to the other channels, less populated.

Effect of kinematical cuts

To study the effects of cuts in the training sample, an additional training has been done with the application of two cuts in the $pp \to t\bar{t}$ channel:

- a cut on the transverse momentum: $p_T > 250$ GeV;
- a cut on the rapidity of the outgoing particles: $-2.5 < \eta < 2.5$.

Fig. 4.15 shows the histograms of the input features and the cuts reconstructed by the model. As expected, the histogram of the input distribution is well reproduced. Regarding the imposed cuts, the model identifies and tries to reproduce them. Indeed, only a few percent of events are beyond the imposed cuts.

Overall, these plots show that the defined model is able to correctly reproduce the input distribution for each channel. The difficulties found are mainly related to the correlation between the Mandelstam variables and the tails region. The first issue is related to the presence of a cut imposed by Mandelstam invariants, while the second is caused by the low number of events on which the training is done in these regions. Having in mind these drawbacks, these models are a possible way to speed up the generation process of events and decrease the CPU usage[1].

[1]The time needed for the generation of 1M events was $\sim$2 minutes on laptop

4.4 Data augmentation

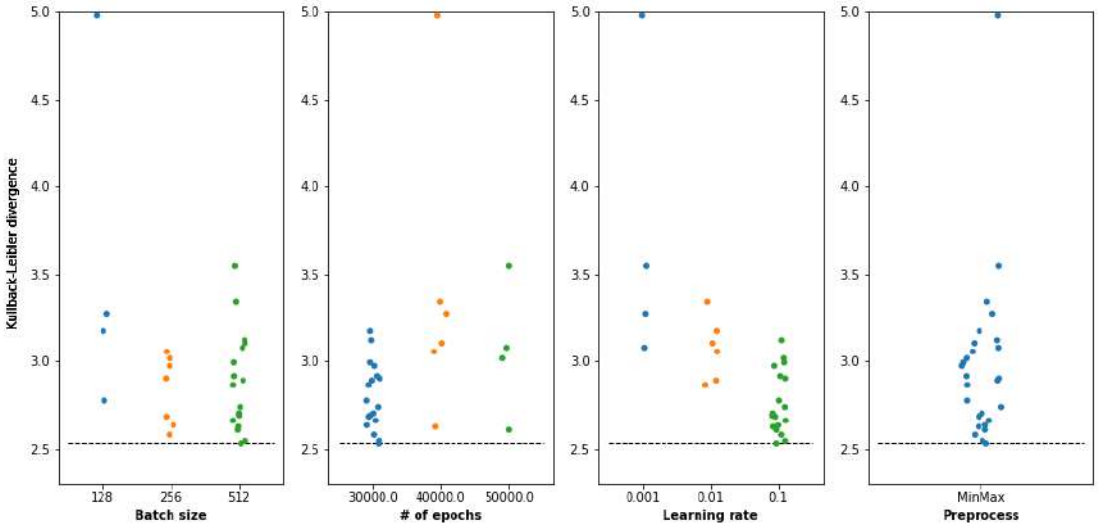
The next goal of this thesis is to study if the proposed models can generate an events sample with higher statistics than the one of the training sample. This would mean that the GAN is able to learn the underlying distribution of the kinematical variables without reproducing the statistical noise present in the training sample. To this purpose, the data augmentation capability of the models has been tested with four different events samples, on the same channel of the previous section, with increasing size: 10k events, 50k events, 100k events, and 500k events. The ceiling number of events fixed to test these models is a sample of 1M events for each channel. The procedure consists of training five models in each channel, one for each sample size defined above. Then, each model generates a sample with 10k, 50k, 100k and 500k events which is compared to a slice of the same size extracted from the sample of 1M events.

The following plots show, for each channel, the histograms and the correlations of the input features. They are structured such that a row corresponds to a single trained model whereas a column corresponds to a fixed size of the sample. In this context, the comparison between the true distribution and the generated one, obtained from the same number of events, appears on the diagonal of the 5×5 matrix plot.

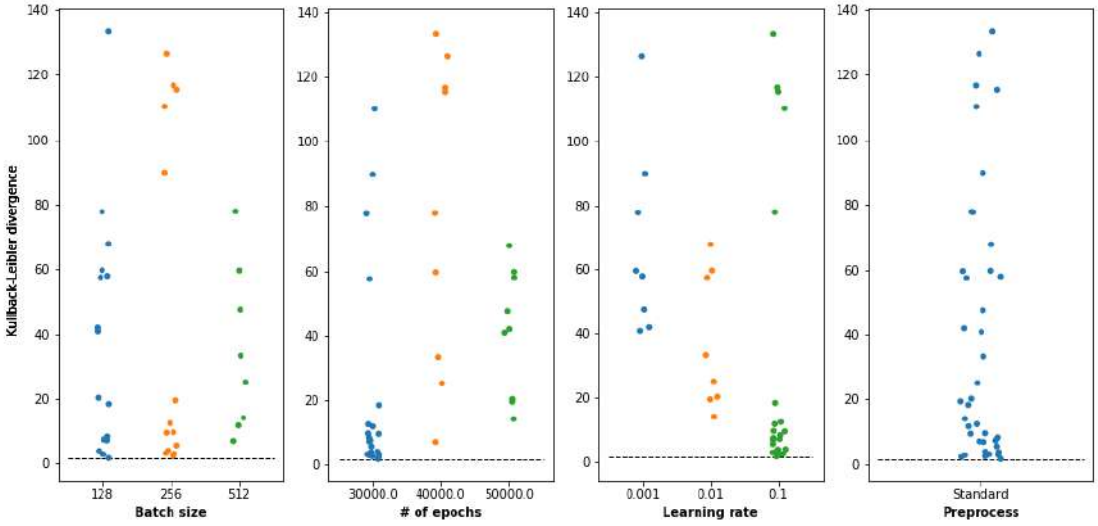
$pp \to t\bar{t}$ channel
The histograms of the input features organized as described above are presented in Figs. 4.16, 4.17 and 4.18. From these plots emerge that the ratio error between the true and the generated distributions is always within the 20%. The more noticeable areas where the discrepancy between the two distributions arise are the tails region for the Mandelstam invariants where, in case of low statistics of the training sample, the model falls off without reproducing the true high statistics tail. Moreover, the rapidity learned at 10k events shows an increasing asymmetry with respect to the true rapidity at bigger samples. The correlations plot of these histograms are shown in Figs. 4.19, 4.20, 4.21 together with the summary of the errors distribution in Fig. 4.22. These plots confirm the difficulty of the models to reproduce the cut of the Mandelstam variables and that a model trained on a low statistics sample is not able to reproduce an arbitrarily large sample. A possible estimate of data augmentation capability, considering the error distributions, is an increase of factor 10 from the starting sample size.

$gg \to ZZ$ and $gg \to HH$ channels
The histogram of the other two processes are presented in Figs. 4.23, 4.24, 4.25 for $gg \to ZZ$ channel and in Figs. 4.30, 4.31, 4.32 for $gg \to HH$ channel. For

both channels, similar points can be highlighted. As in the $pp \to t\bar{t}$ channel, the ratio error is within 20% in the majority of the bins and becomes lower in the core of the distributions for training with a higher training sample. The worst reconstruction of the true distribution of the augmented samples is also in the tail region because, during the training, these events are not fed into the GAN. On the other hand, the augmented rapidity distributions are better reproduced as can be seen by the lower asymmetry and the ratio error within 10% in the core region. The correlations plot and the connected errors distribution plot of the $gg \to ZZ$ process, shown in Figs. 4.26, 4.27, 4.28, and 4.29, highlight again that the cut is difficult to parametrize whereas the correlations between the other input features are uniformly learned. The same conclusion can be drawn from the $gg \to HH$ channel, the correlations between s and t are not well reproduced due to the different phase space as stated in the previous section. On the other hand, the correlations of the other input features uphold the data augmentation capability of factor 10. The plots are shown in Figs. 4.33, 4.34, 4.35, 4.36.



(a)



(b)

Figure 4.3: First hyperparameters optimization. The search space includes: the batch size, number of epochs, learning rate and the preprocessing used. The y-axis indicates the mean of the KL divergence in the output distributions which is the metric to minimize.

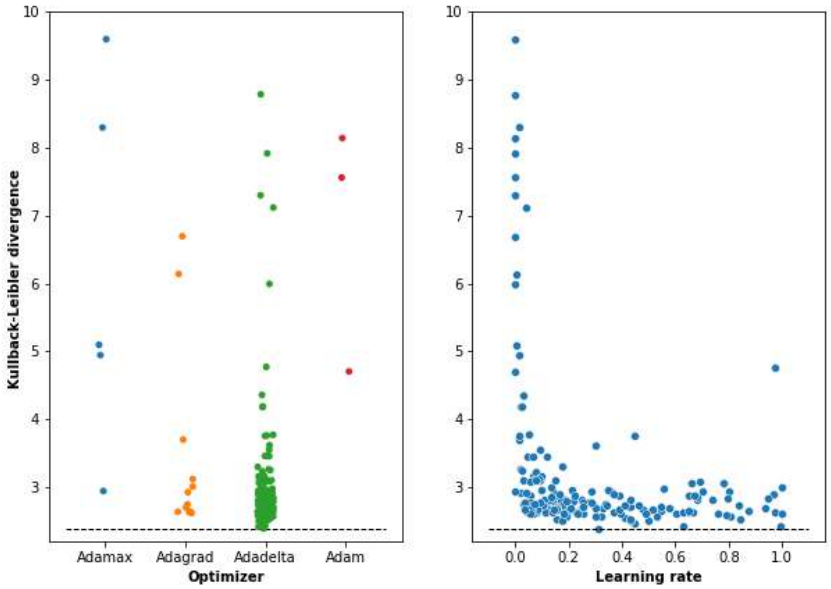


Figure 4.4: Second hyperparameters optimization. The search space includes: the optimizer and a continuous range of possible learning rates. The metric used is the same of Fig.4.3.

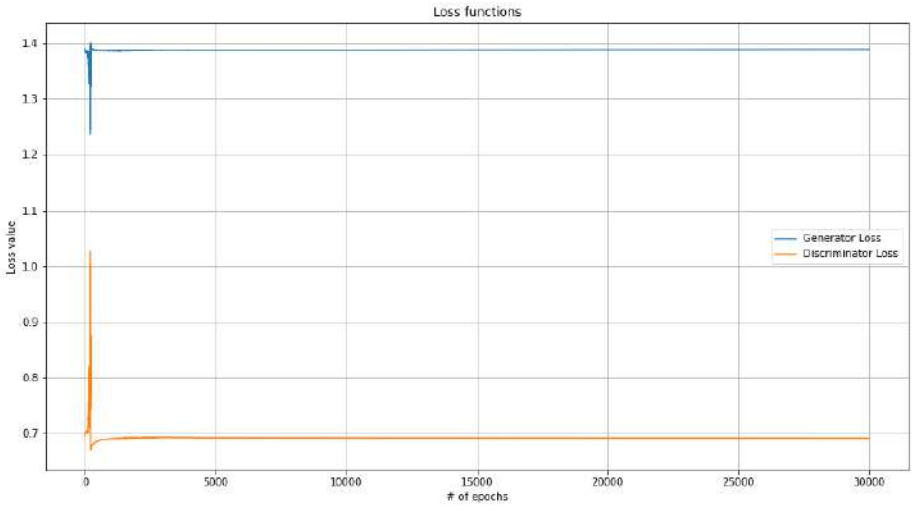
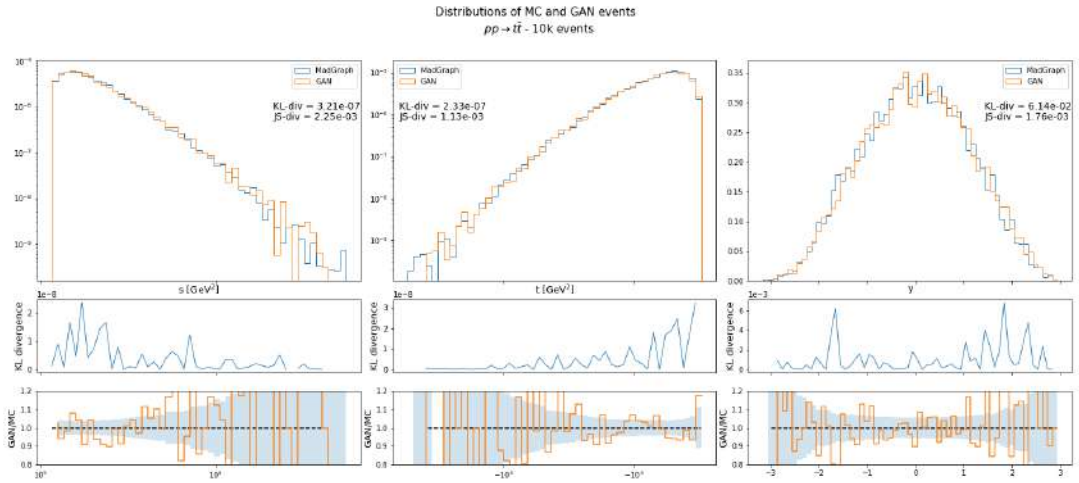
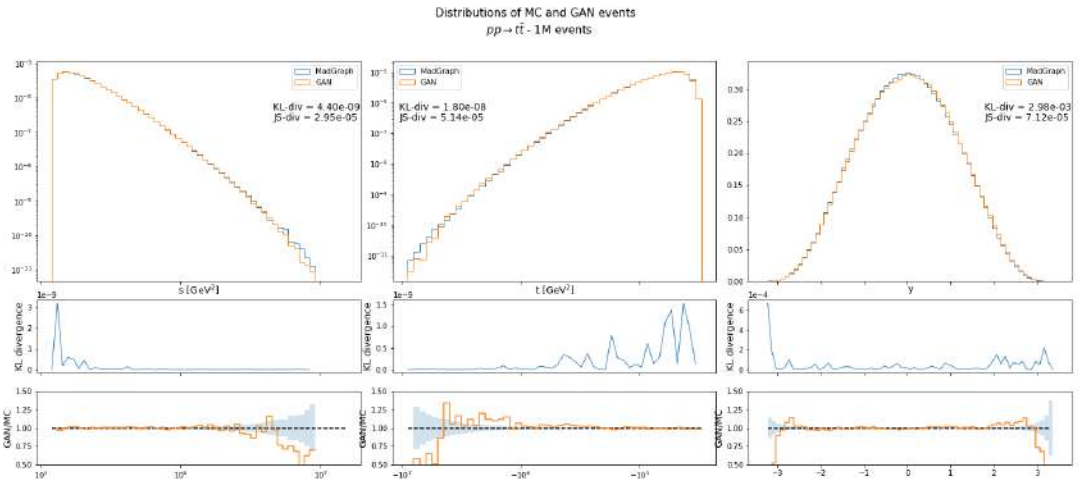


Figure 4.5: Loss functions for a training with 30k epochs.

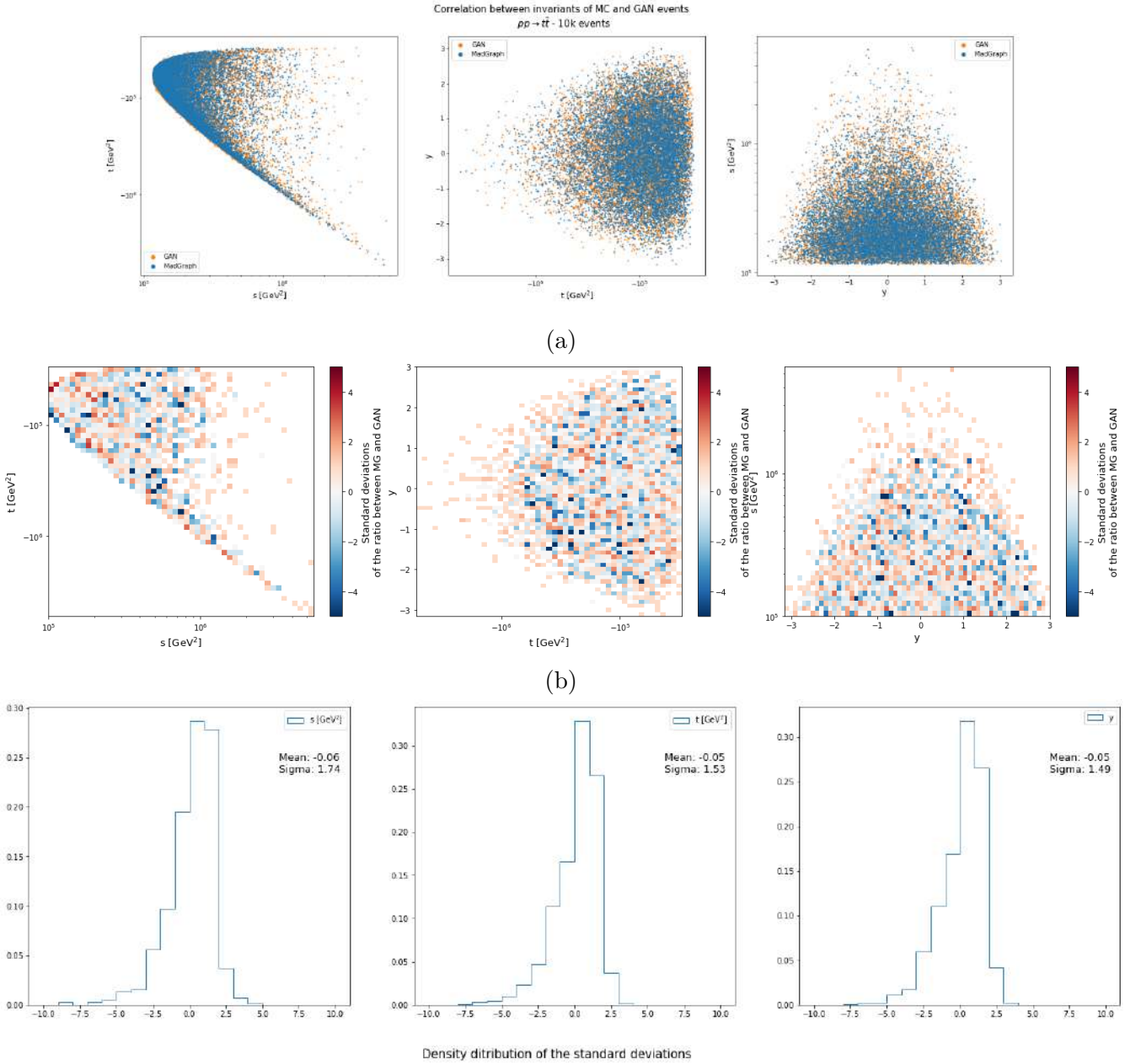


(a)



(b)

Figure 4.6: Histograms for the $pp \to t\bar{t}$ channel with (a) 10k events and (b) 1M events. the first and the second subplots show respectively the bin-wise KL divergence and the ratio of the histograms.

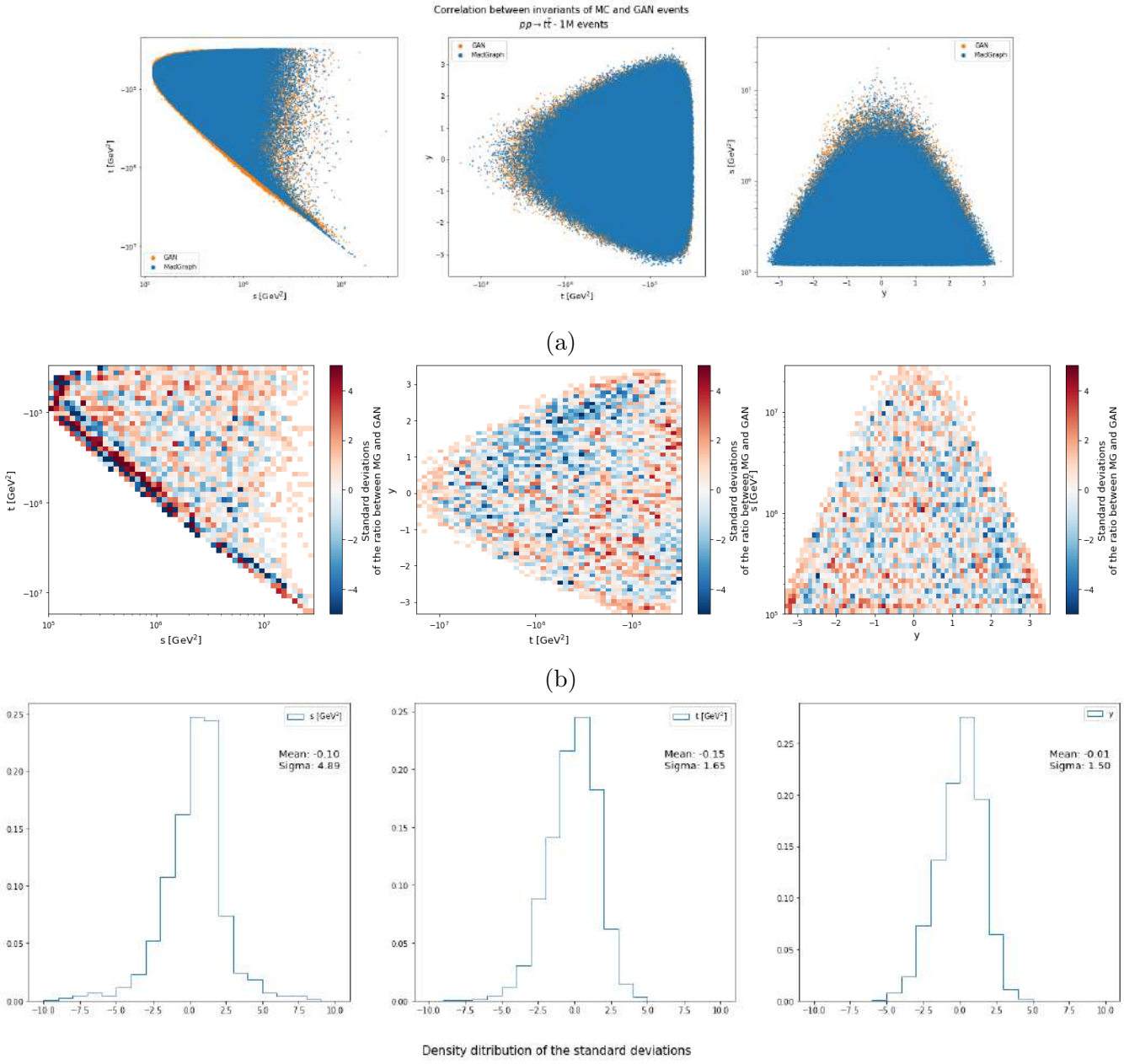


(a)

(b)

(c)

Figure 4.7: Correlations plot for the channel $pp \to t\bar{t}$ with 10k events. (a) Scatter plot of the two distributions, (b) bin-wise counting error in units of standard deviations, (c) distribution of the errors in units of standard deviations for all bins.

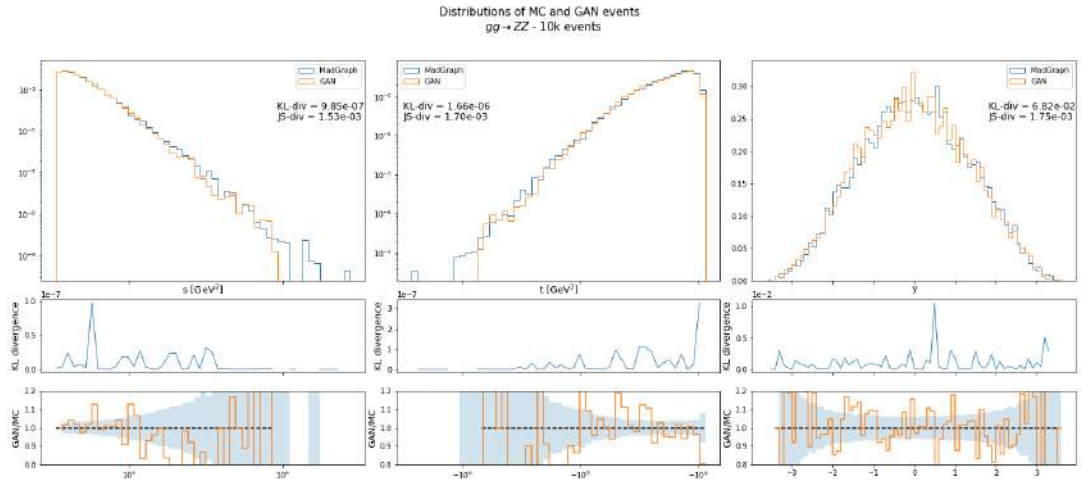


(a)

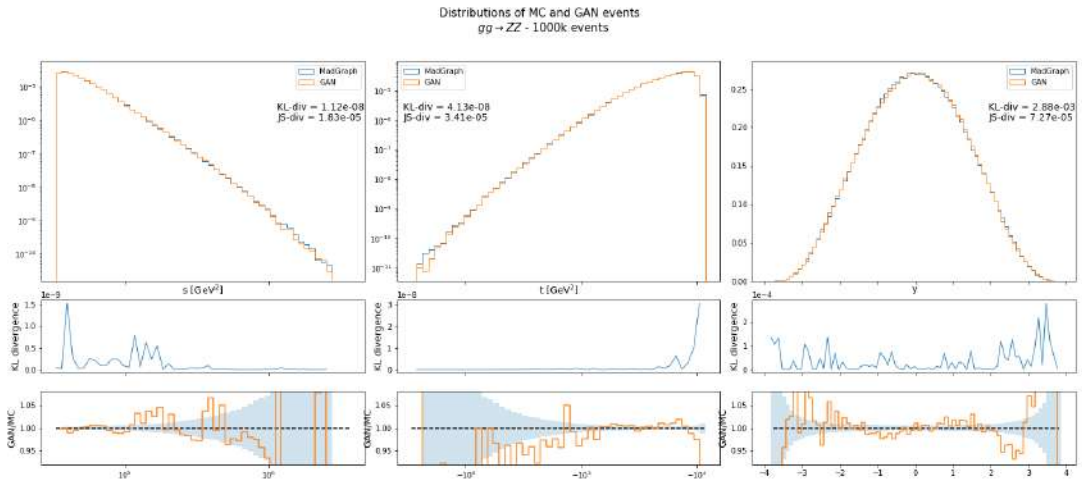
(b)

(c)

Figure 4.8: Correlations plot for the channel $pp \to t\bar{t}$ with 1M events. (a) Scatter plot of the two distributions, (b) bin-wise counting error in units of standard deviations, (c) distribution of the errors in units of standard deviations for all bins.

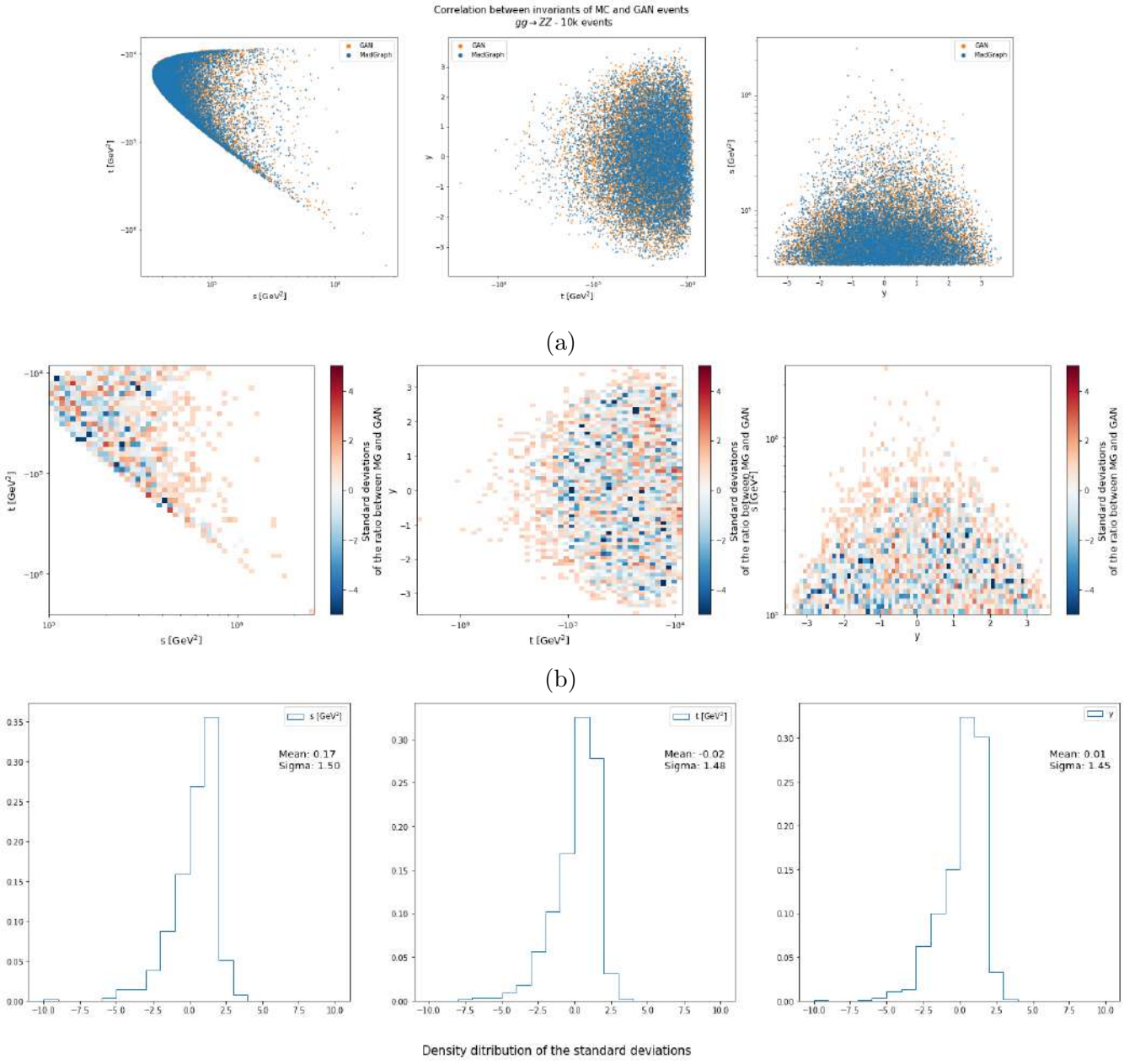


(a)



(b)

Figure 4.9: Histograms for the $gg \to ZZ$ channel with (a) 10k events and (b) 1M events. the first and the second subplots show respectively the bin-wise KL divergence and the ratio of the histograms.

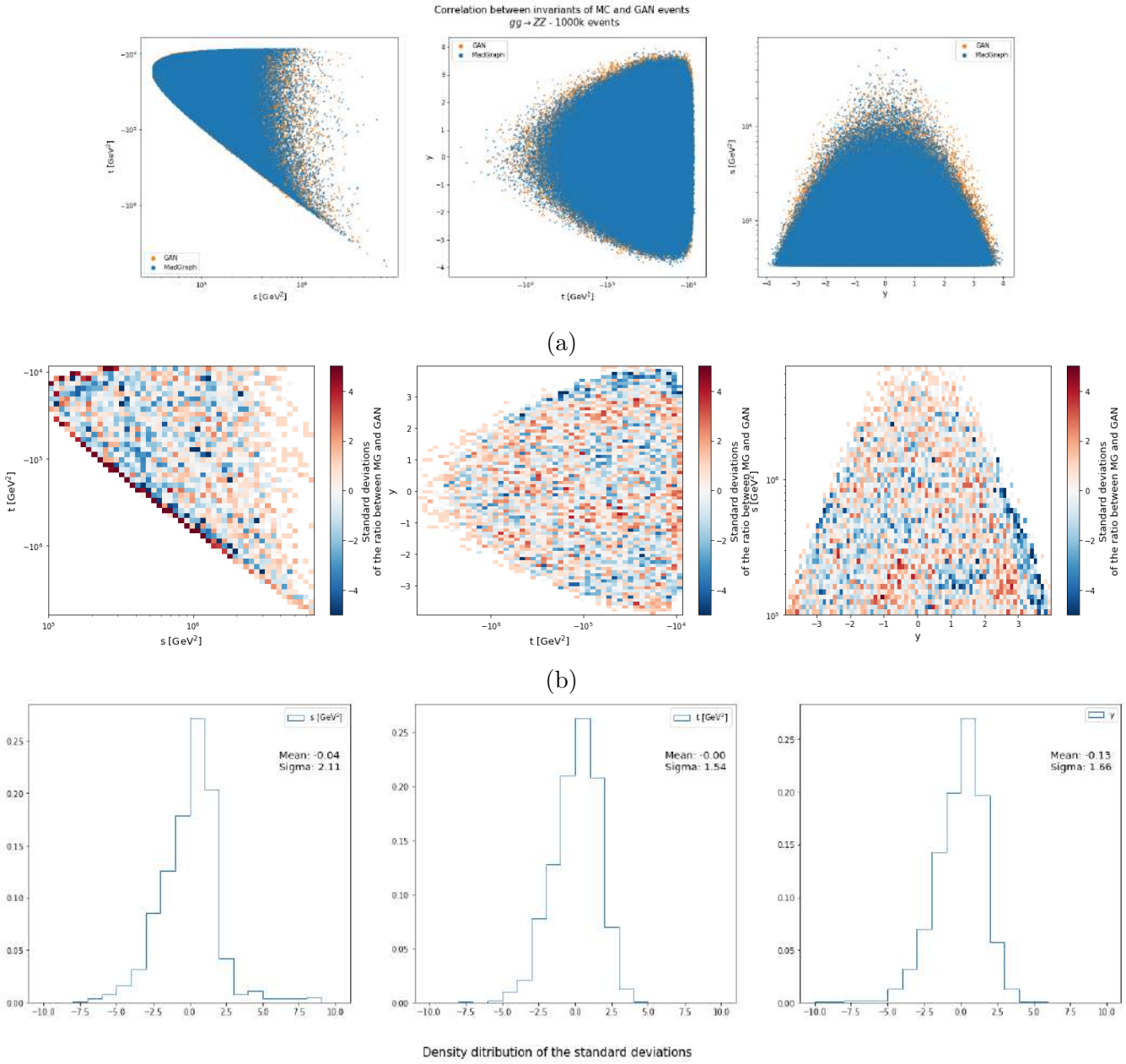


(a)

(b)

(c)

Figure 4.10: Correlations plot for the channel $gg \to ZZ$ with 10k events. (a) Scatter plot of the two distributions, (b) bin-wise counting error in units of standard deviations, (c) distribution of the errors in units of standard deviations for all bins.

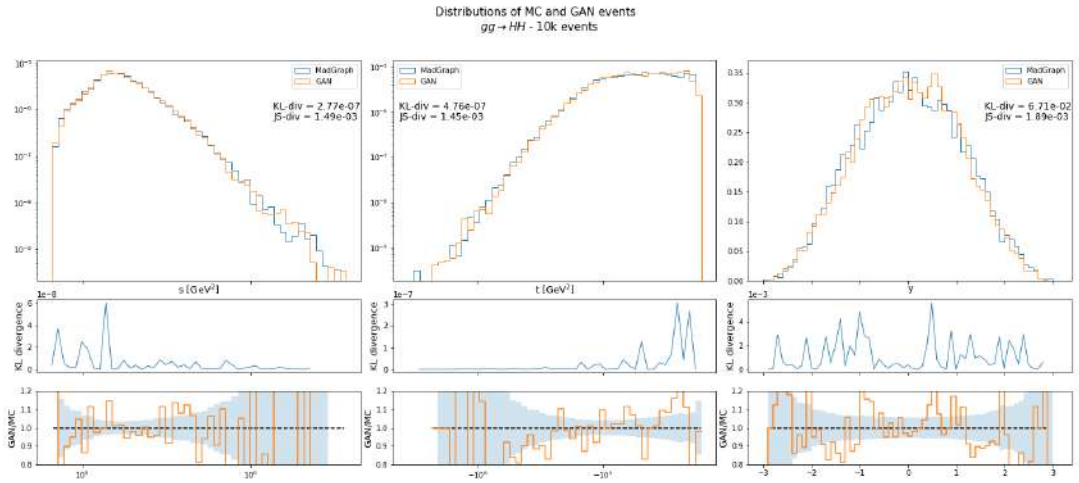


(a)

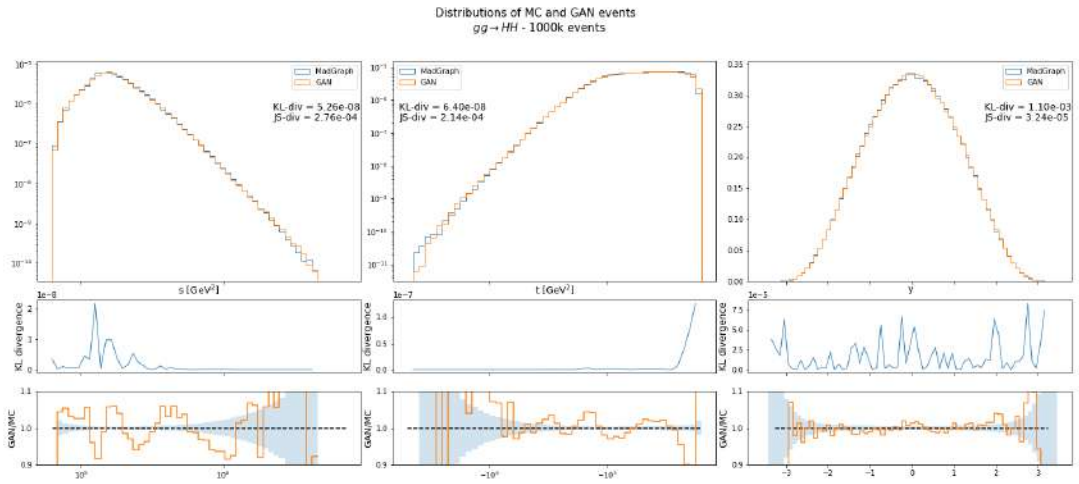
(b)

(c)

Figure 4.11: Correlations plot for the channel $gg \to ZZ$ with 1M events. (a) Scatter plot of the two distributions, (b) bin-wise counting error in units of standard deviations, (c) distribution of the errors in units of standard deviations for all bins.

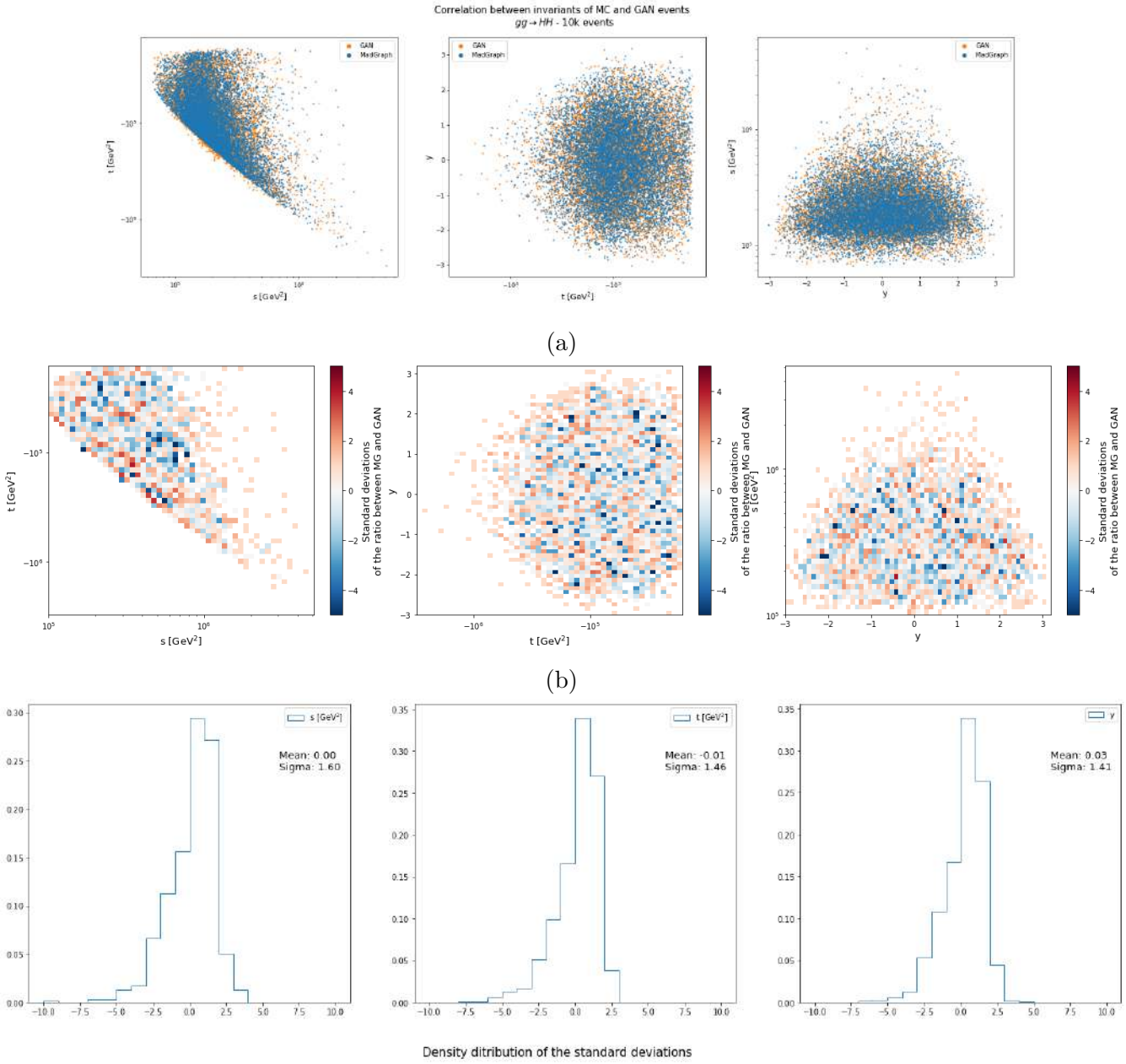


(a)



(b)

Figure 4.12: Histograms for the $gg \to HH$ channel with (a) 10k events and (b) 1M events. the first and the second subplots show respectively the bin-wise KL divergence and the ratio of the histograms.

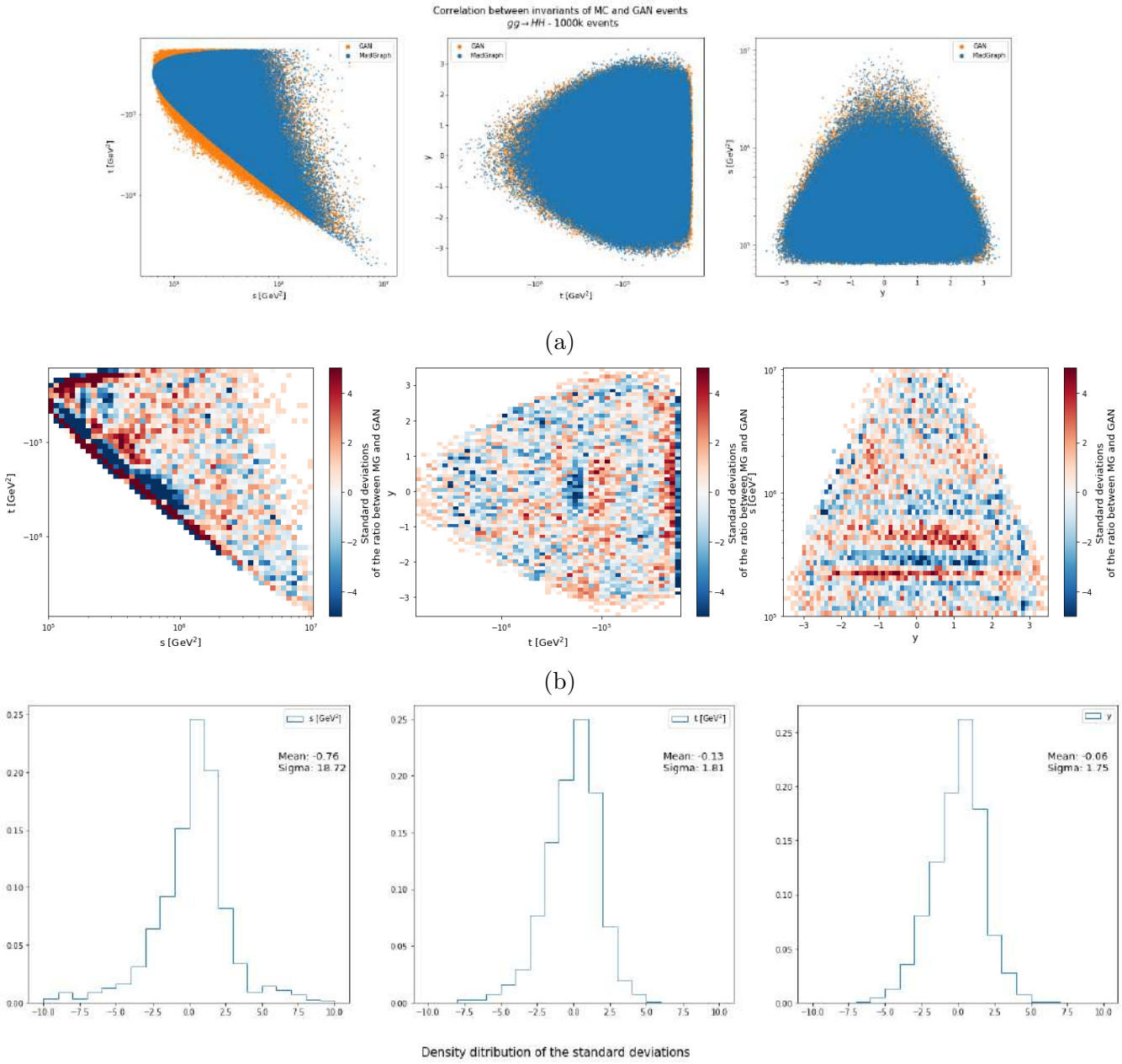


(a)

(b)

(c)

Figure 4.13: Correlations plot for the channel $gg \to HH$ with 10k events. (a) Scatter plot of the two distributions, (b) bin-wise counting error in units of standard deviations, (c) distribution of the errors in units of standard deviations for all bins.

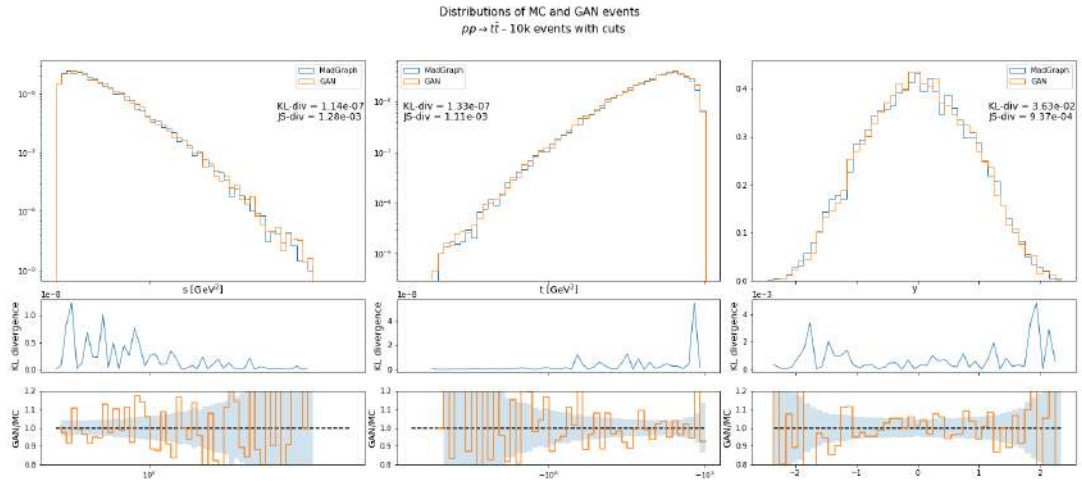


(a)

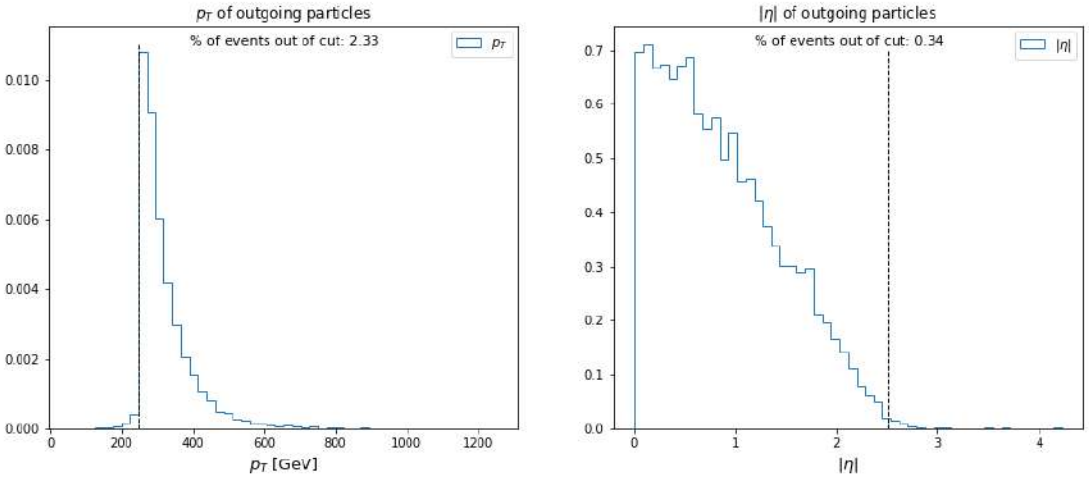
(b)

(c)

Figure 4.14: Correlations plot for the channel $gg \to HH$ with 10k events. (a) Scatter plot of the two distributions, (b) bin-wise counting error in units of standard deviations, (c) distribution of the errors in units of standard deviations for all bins.



(a)



(b)

Figure 4.15: Histograms of the distributions of the input feature with imposed cuts on external particles (a). Histograms of the imposed cuts on the external particles (b).

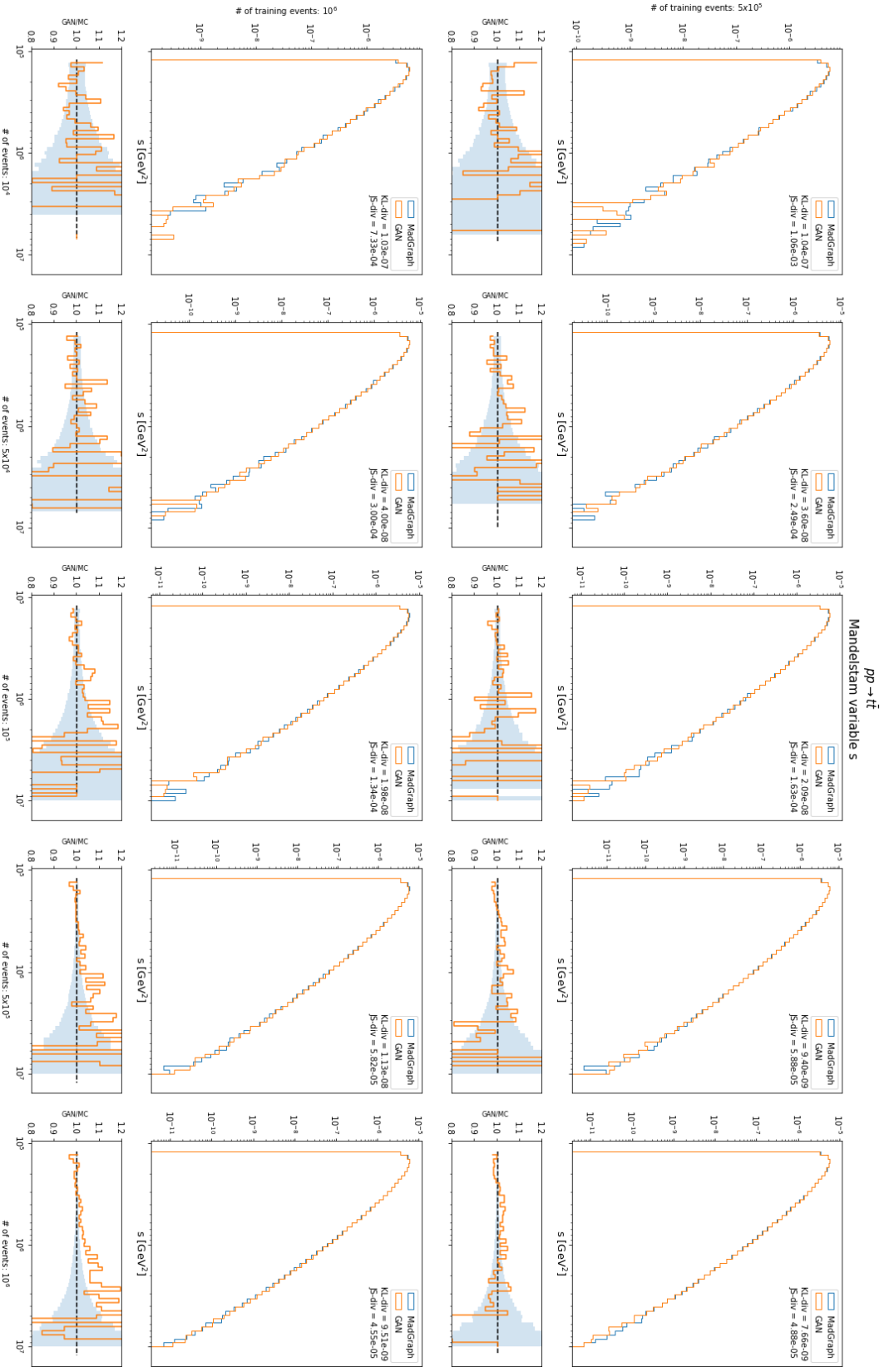


Figure 4.16 (b): Matrix of histograms of the Mandelstam variable s for the channel $pp \to t\bar{t}$. Models trained on 500k, 1M events.

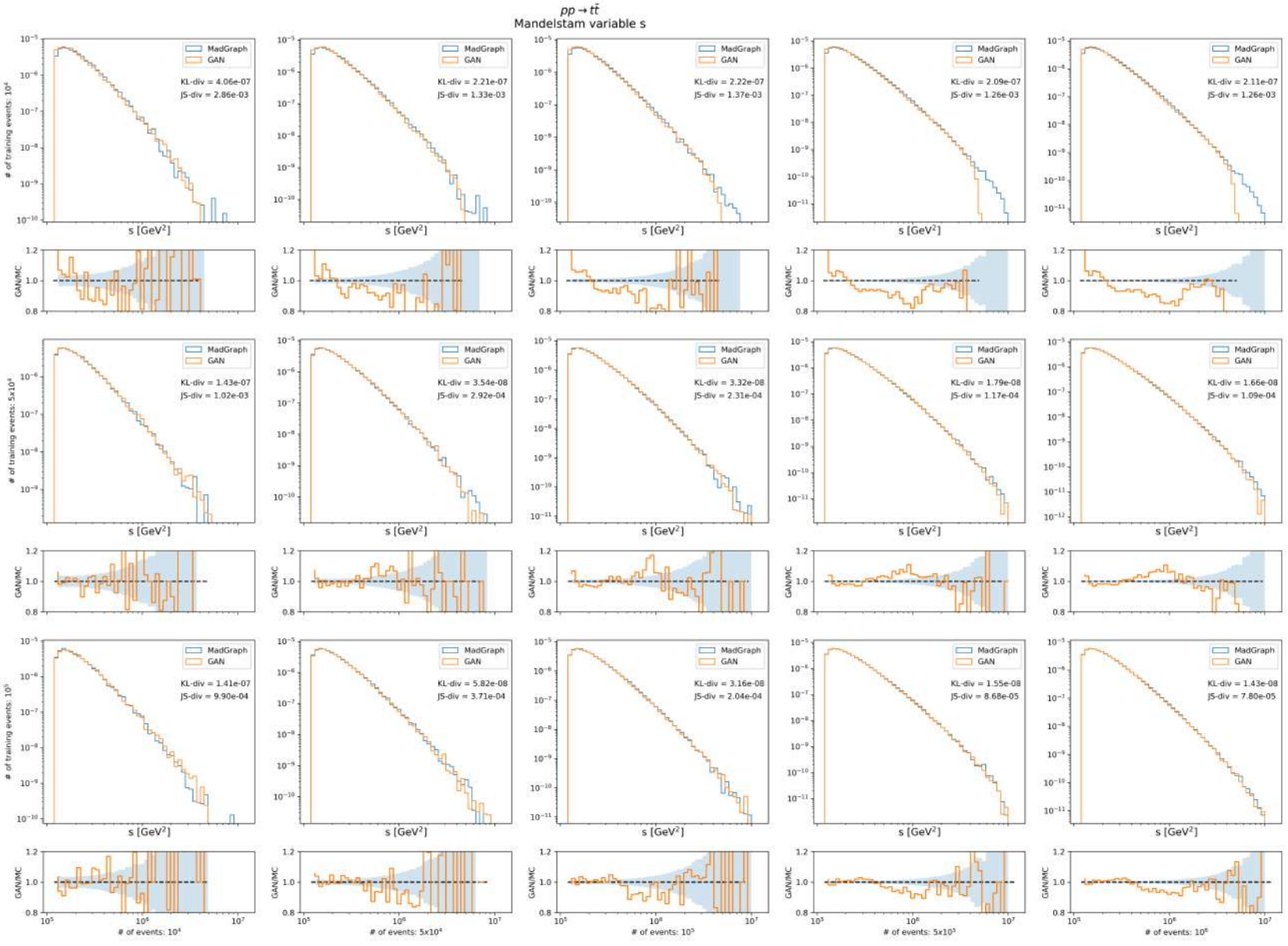


Figure 4.16 (a): Matrix of histograms of the Mandelstam variable s for the channel $pp \to t\bar{t}$. Models trained on 10k, 50k, 100k events.

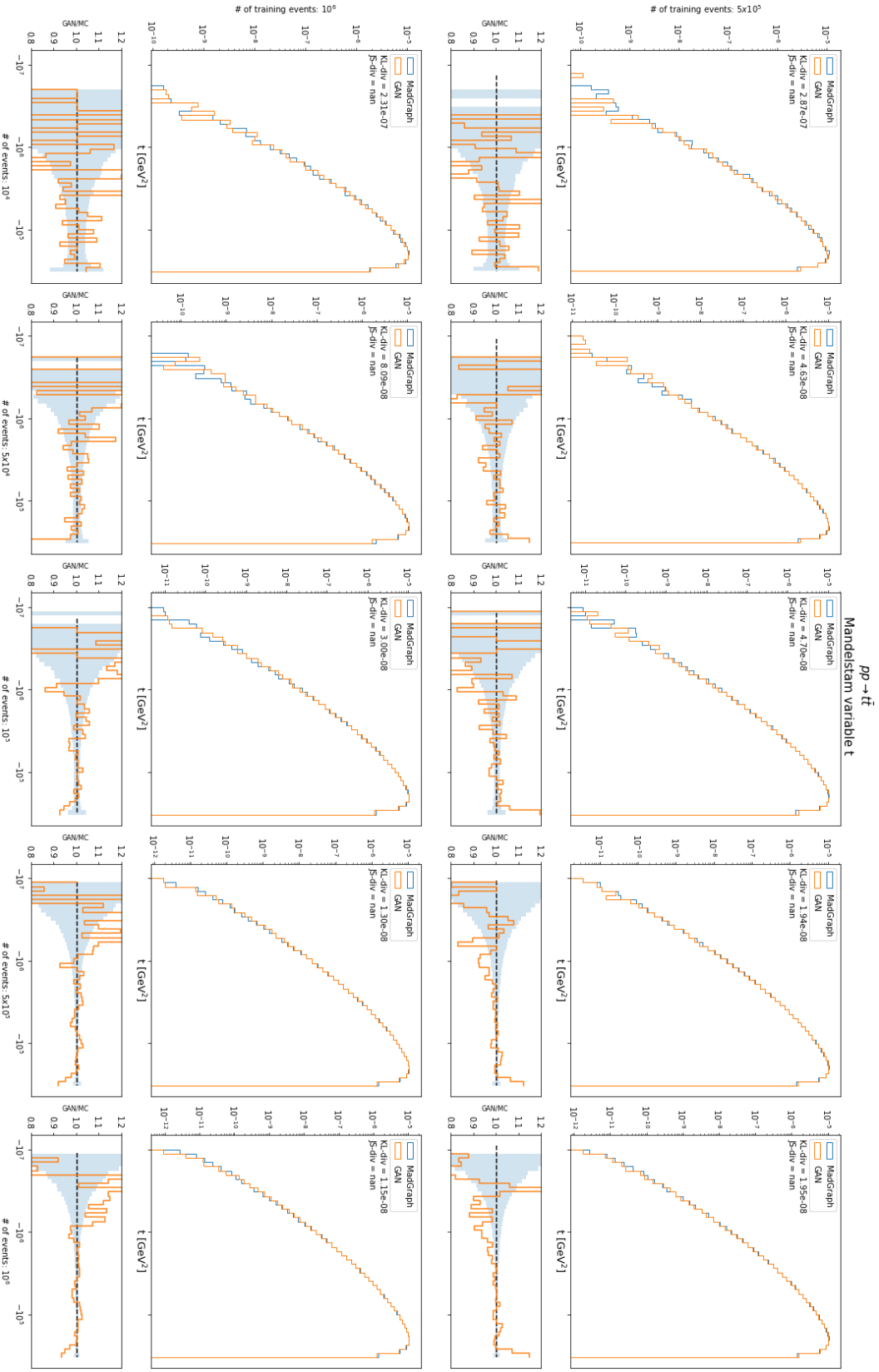


Figure 4.17 (b): Matrix of histograms of the Mandelstam variable t for the channel $pp \to t\bar{t}$. Models trained on 500k, 1M events.

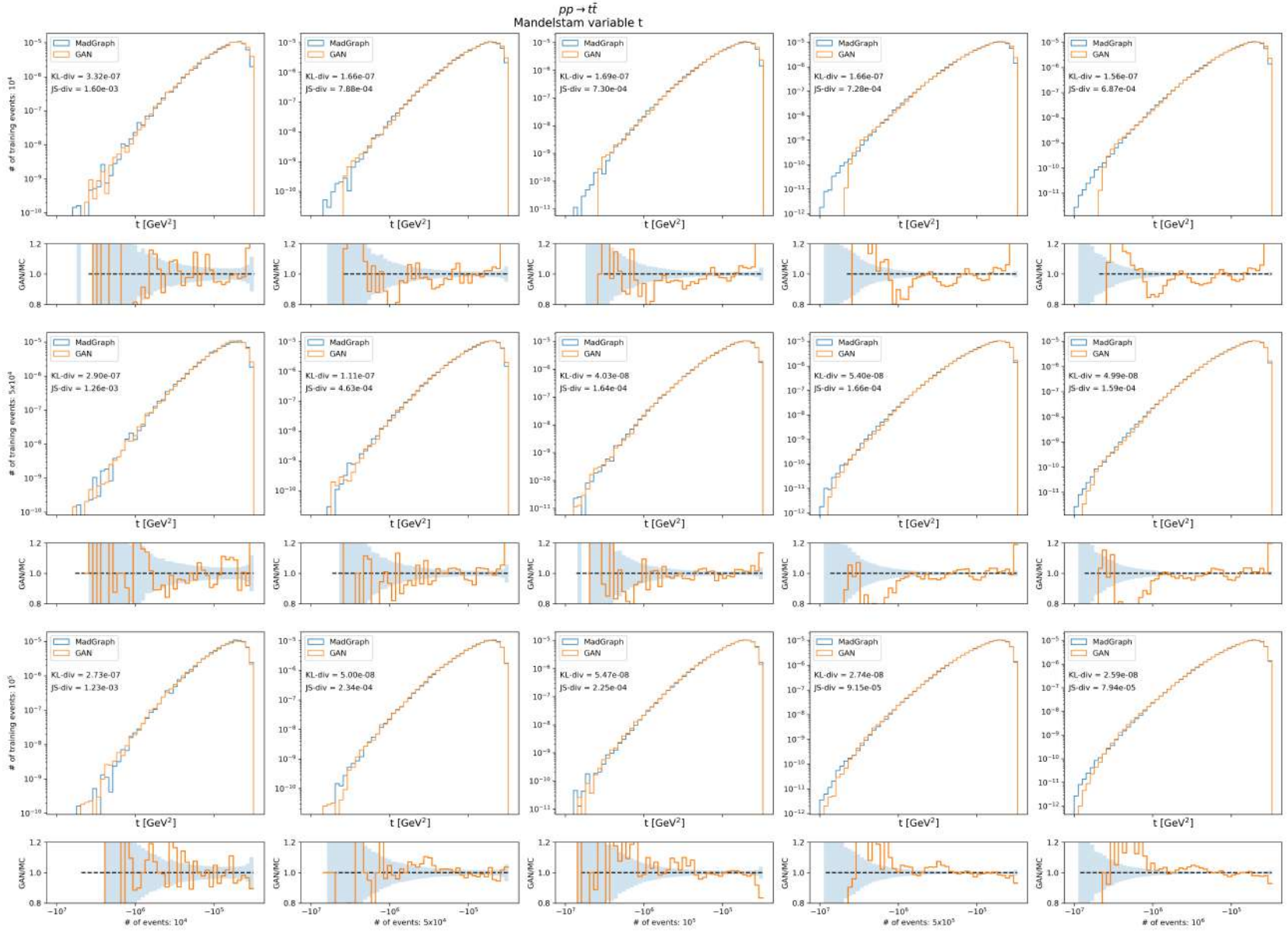


Figure 4.17 (a): Matrix of histograms of the Mandelstam variable t for the channel $pp \to t\bar{t}$. Models trained on 10k, 50k, 100k events.

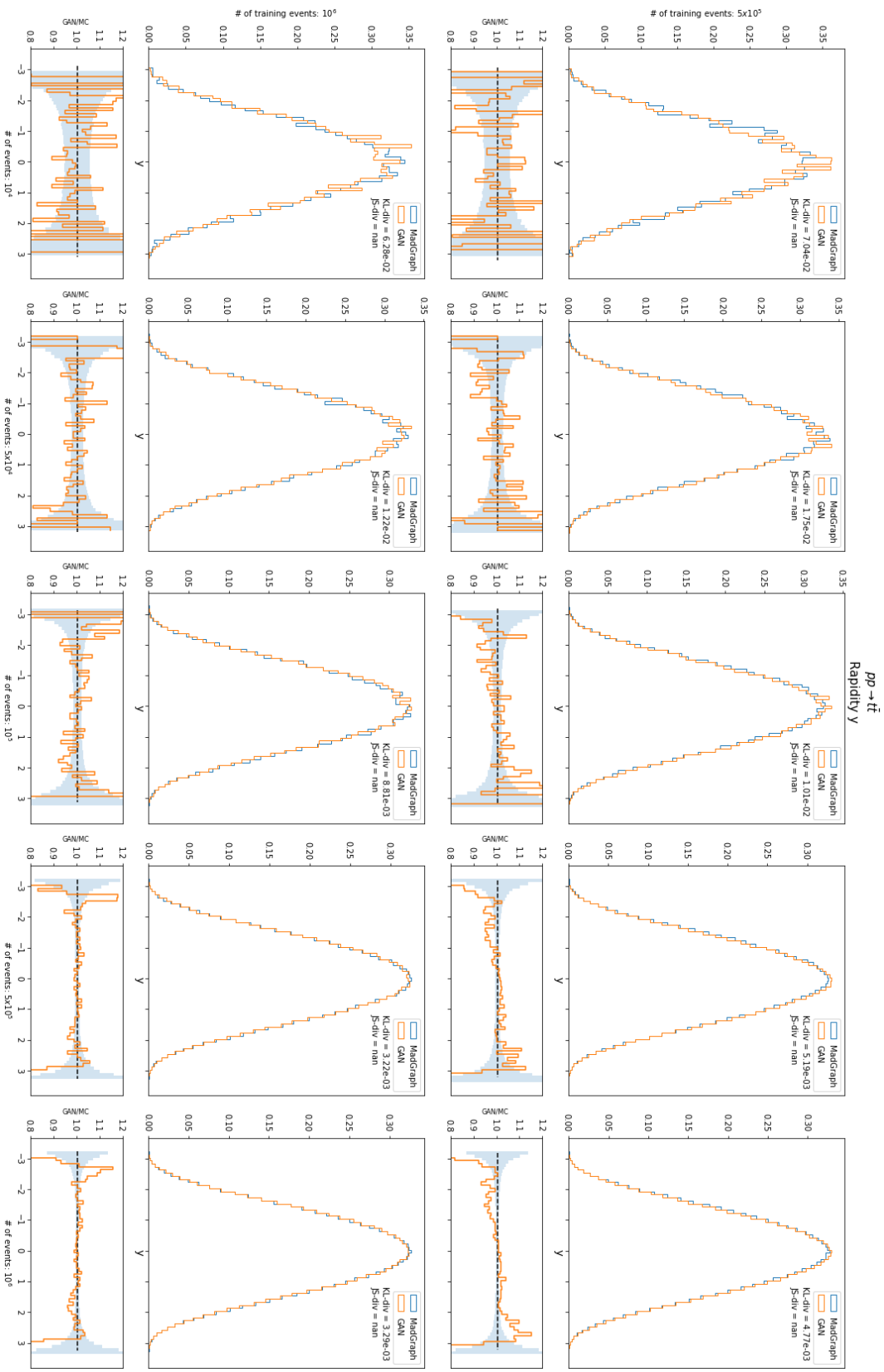


Figure 4.18 (b): Matrix of histograms of the rapidity y for the channel $pp \to t\bar{t}$. Models trained on 500k, 1M events.

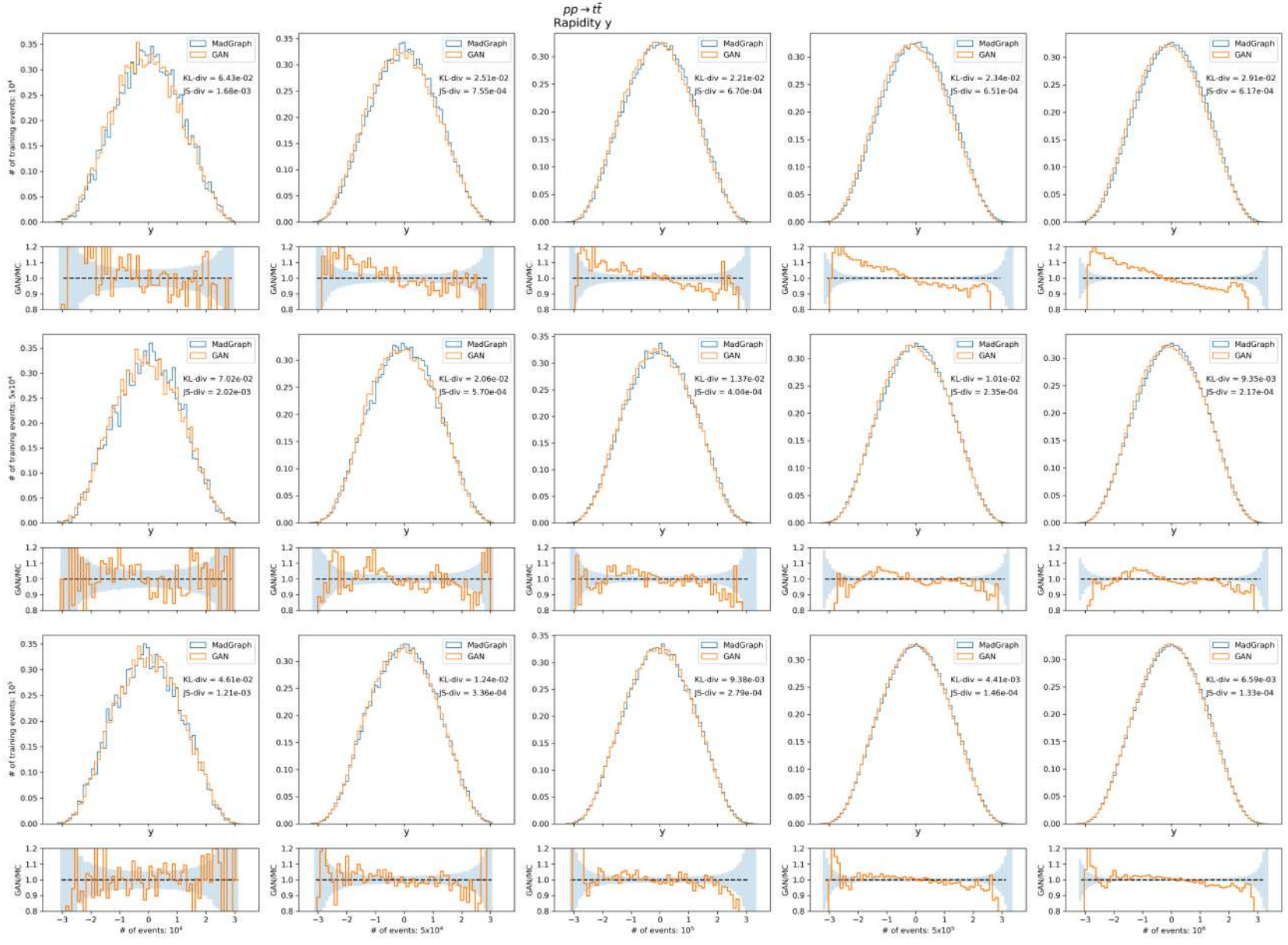


Figure 4.18 (a): Matrix of histograms of the rapidity y for the channel $pp \to t\bar{t}$. Models trained on 10k, 50k, 100k events.

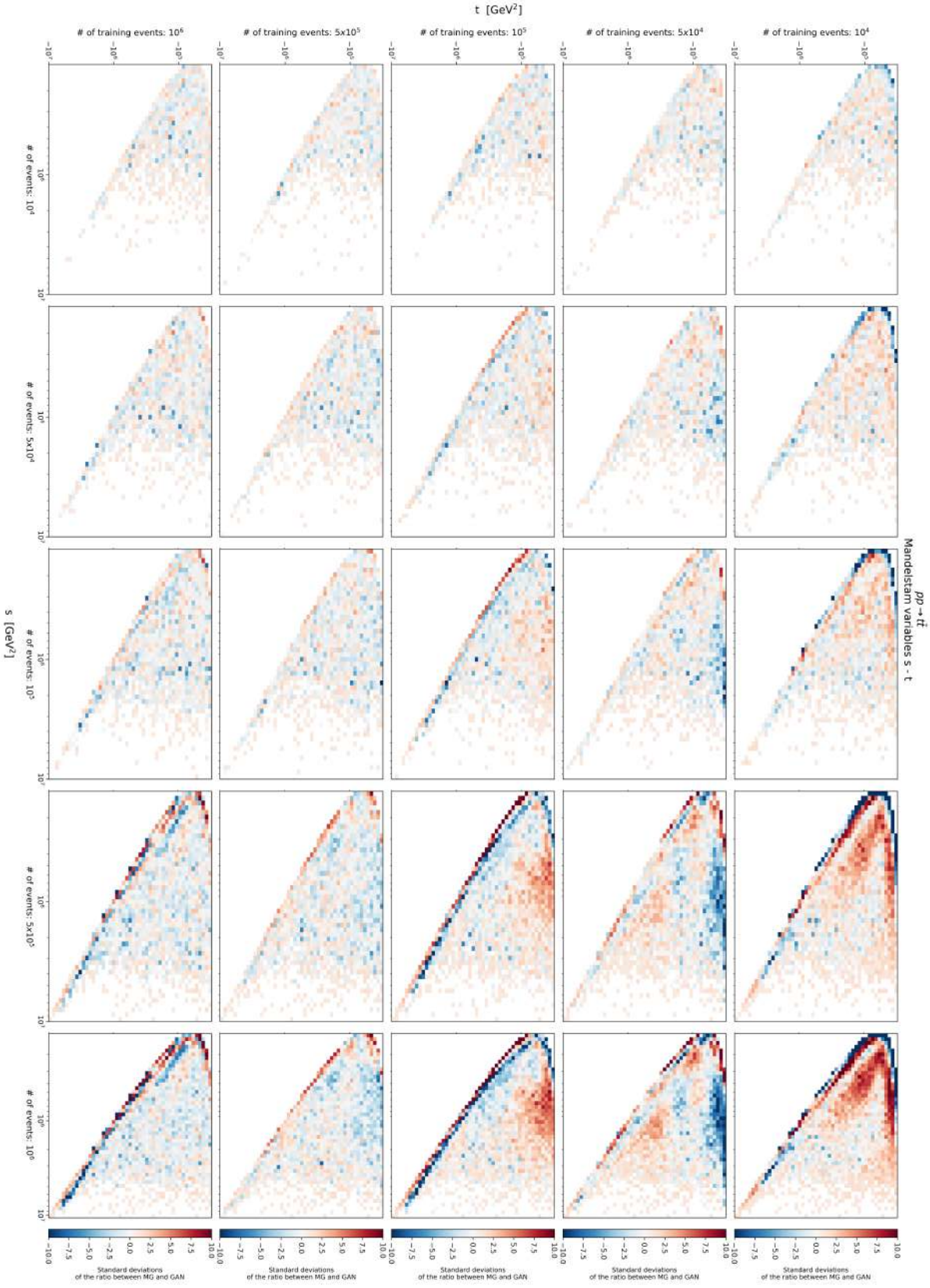


Figure 4.19: Matrix of correlations plot of the Mandelstam variable s for the channel $pp \to t\bar{t}$.

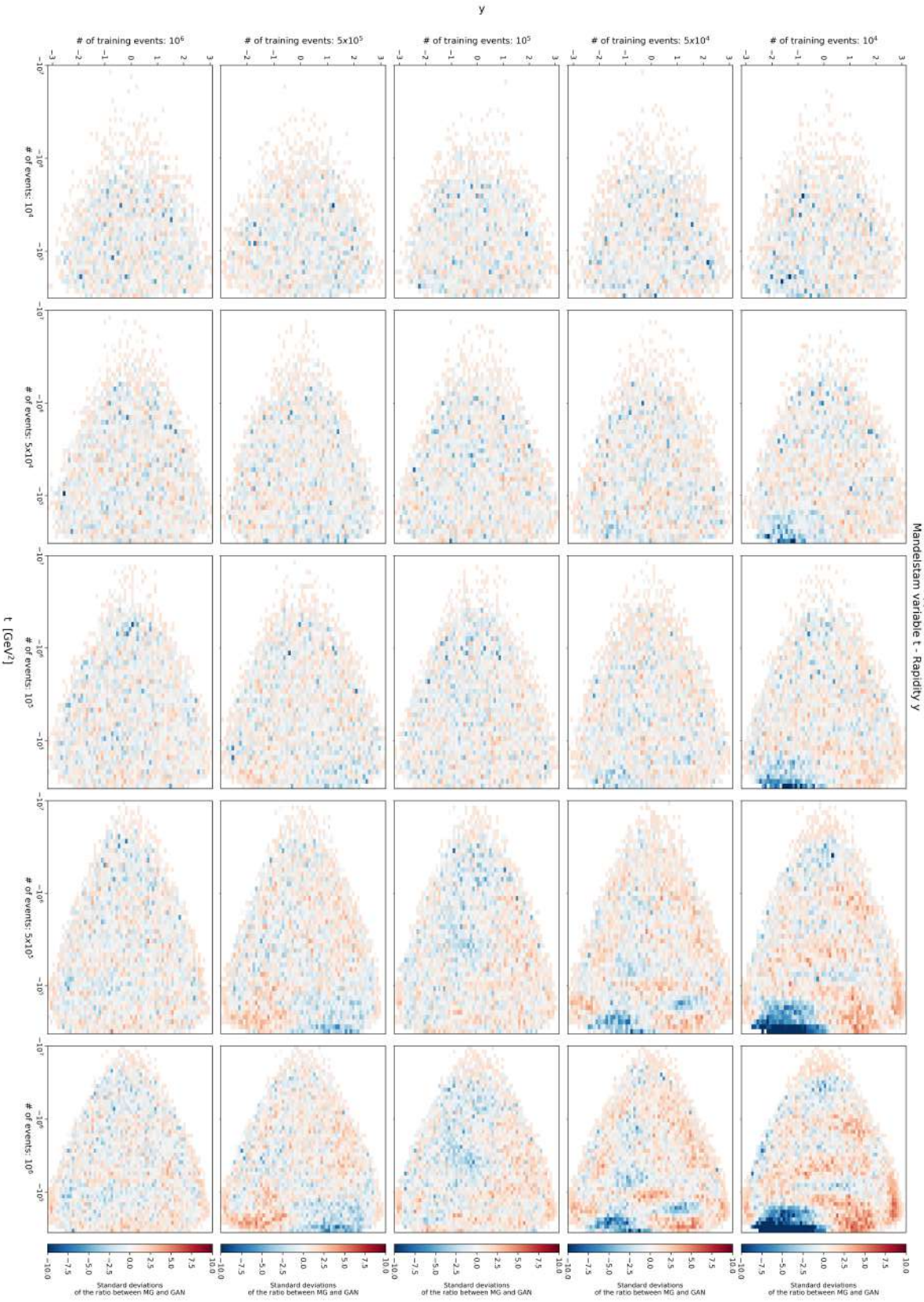


Figure 4.20: Matrix of correlations plot of the Mandelstam variable t for the channel $pp \to t\bar{t}$.

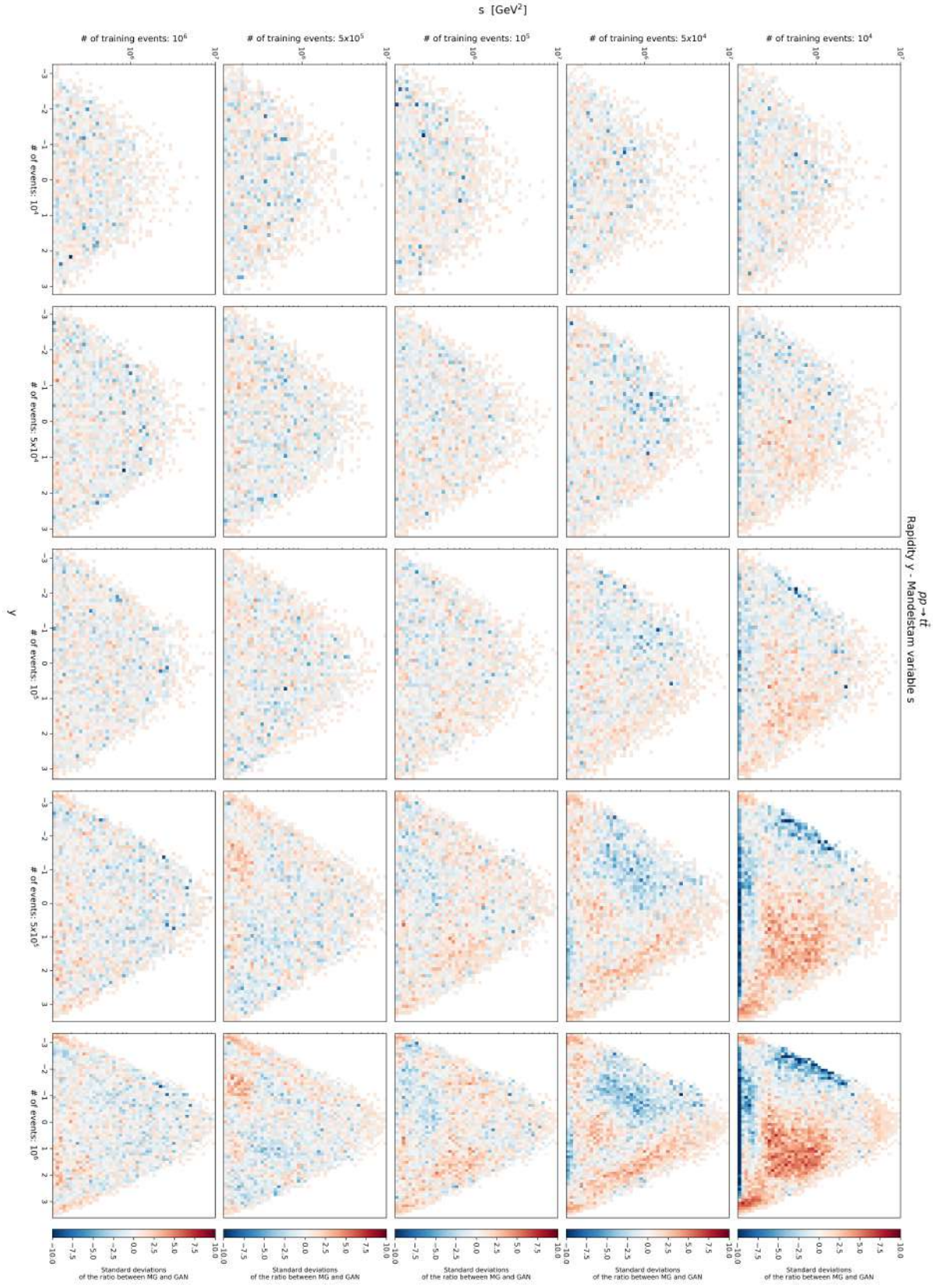
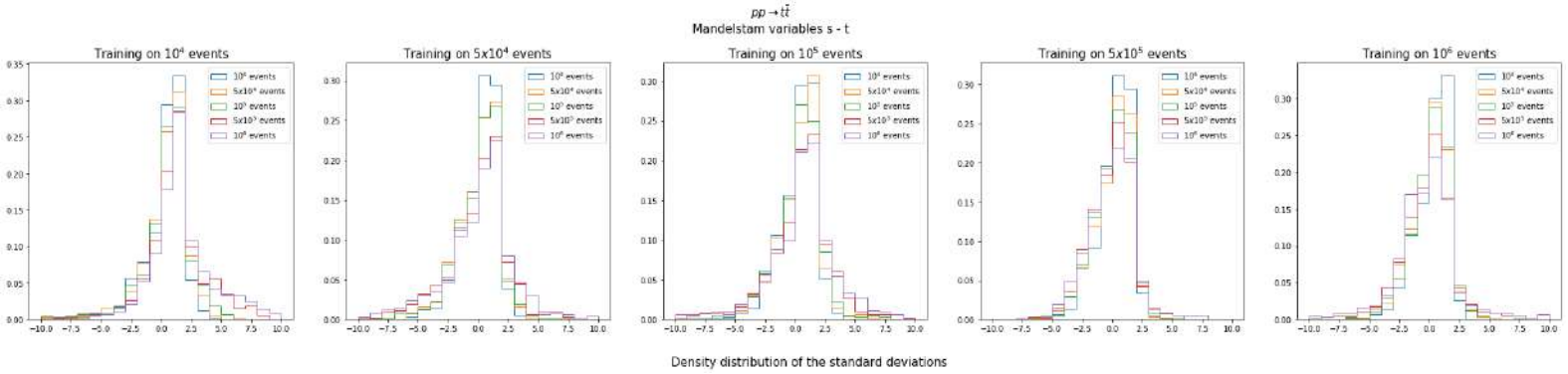
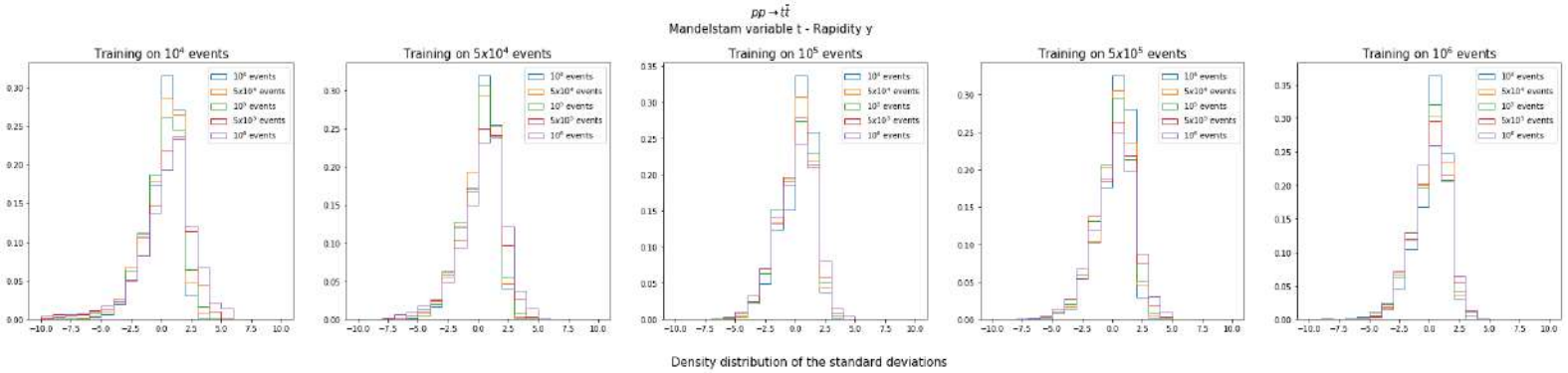


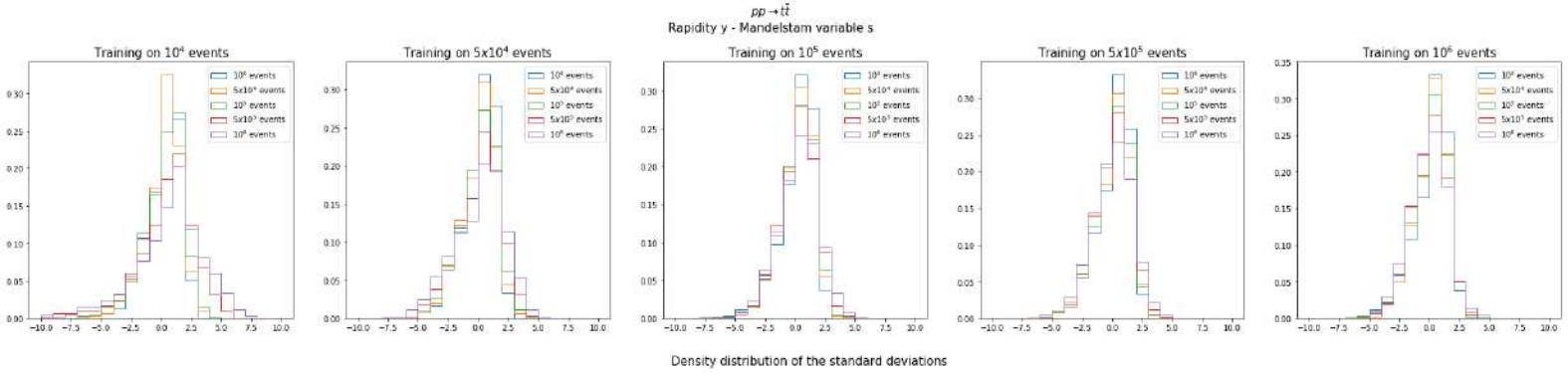
Figure 4.21: Matrix of correlations plot of the rapidity y for the channel $pp \to t\bar{t}$.



(a)



(b)



(c)

Figure 4.22: Distribution of the errors obtained from the correlations plot for the variable s (a), t (b) and y (c). A single plot correspond to a row of the correlations plot.

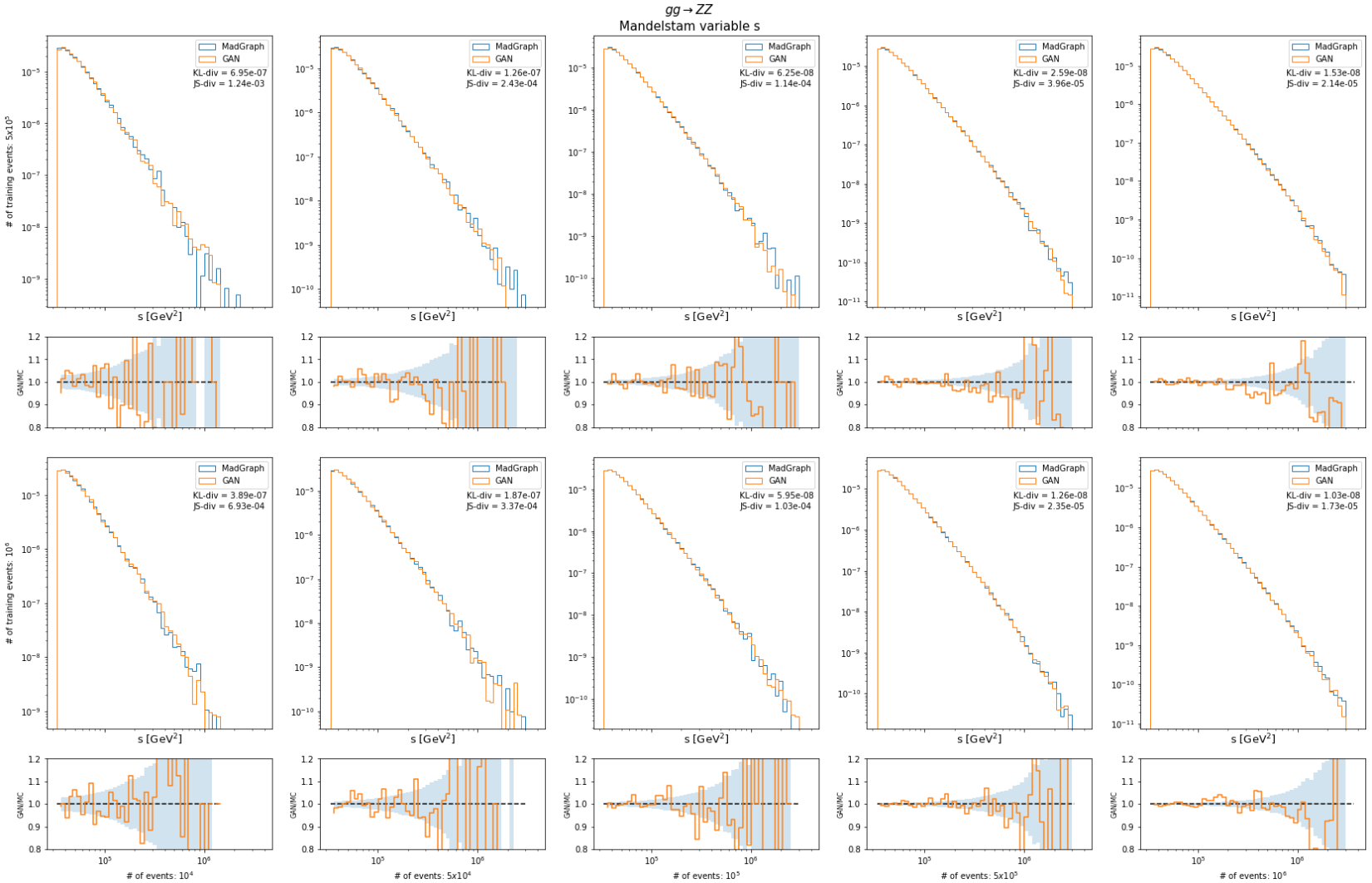


Figure 4.23 (b): Matrix of histograms of the Mandelstam variable s for the channel $gg \to ZZ$. Models trained on 500k, 1M events.

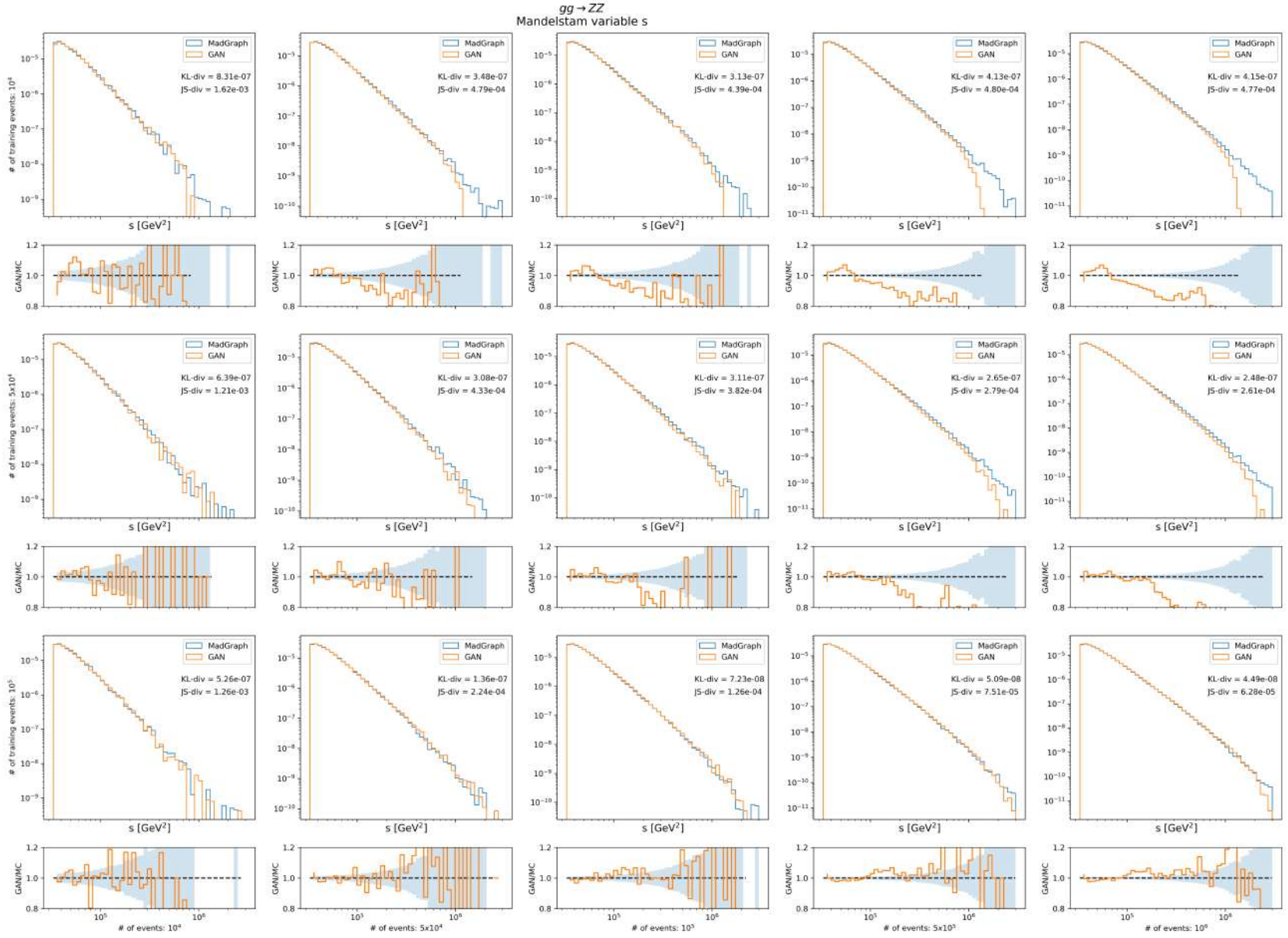


Figure 4.23 (a): Matrix of histograms of the Mandelstam variable s for the channel $gg \to ZZ$. Models trained on 10k, 50k, 100k events.

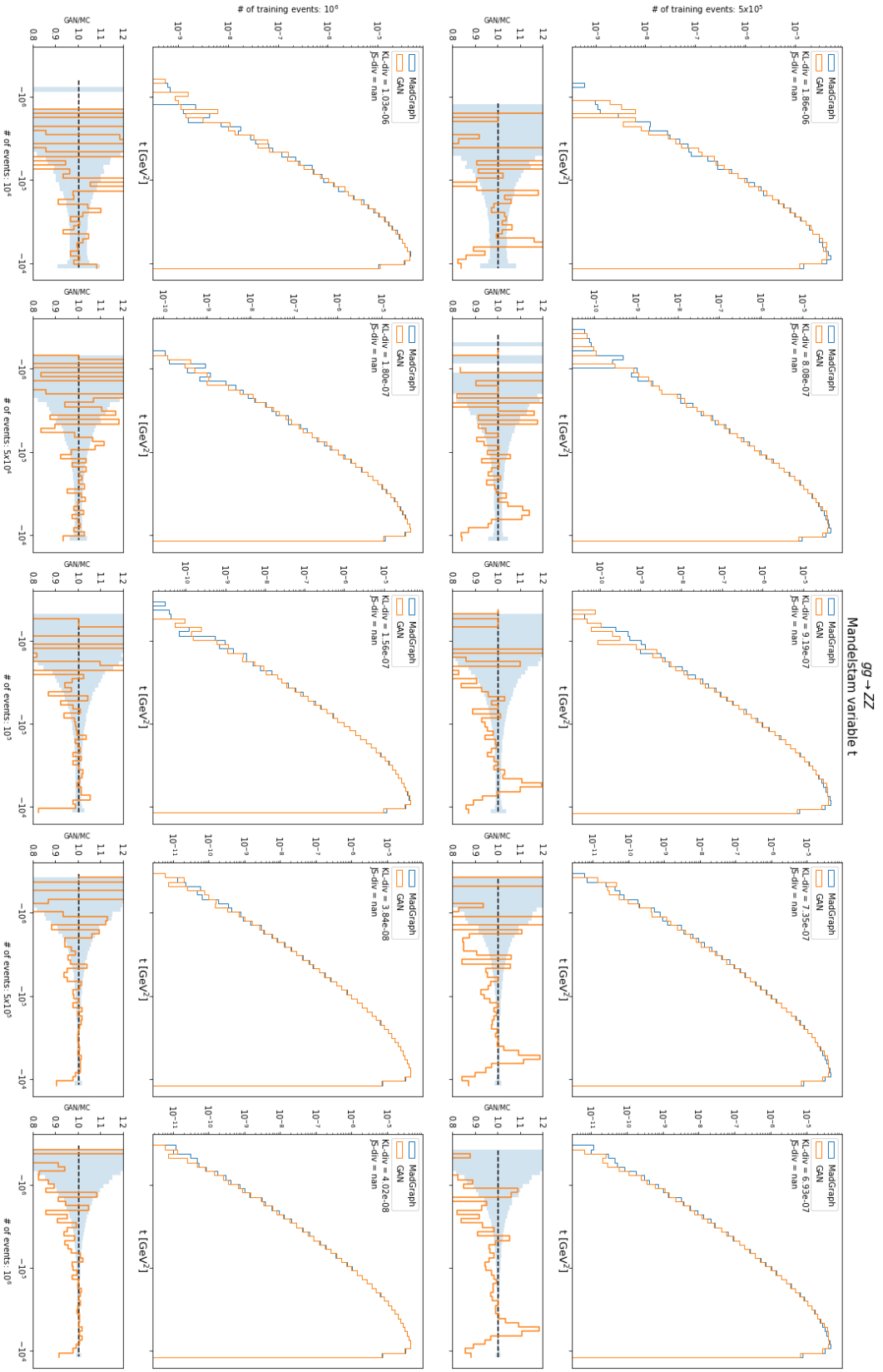


Figure 4.24 (b): Matrix of histograms of the Mandelstam variable t for the channel $gg \to ZZ$. Models trained on 500k, 1M events.

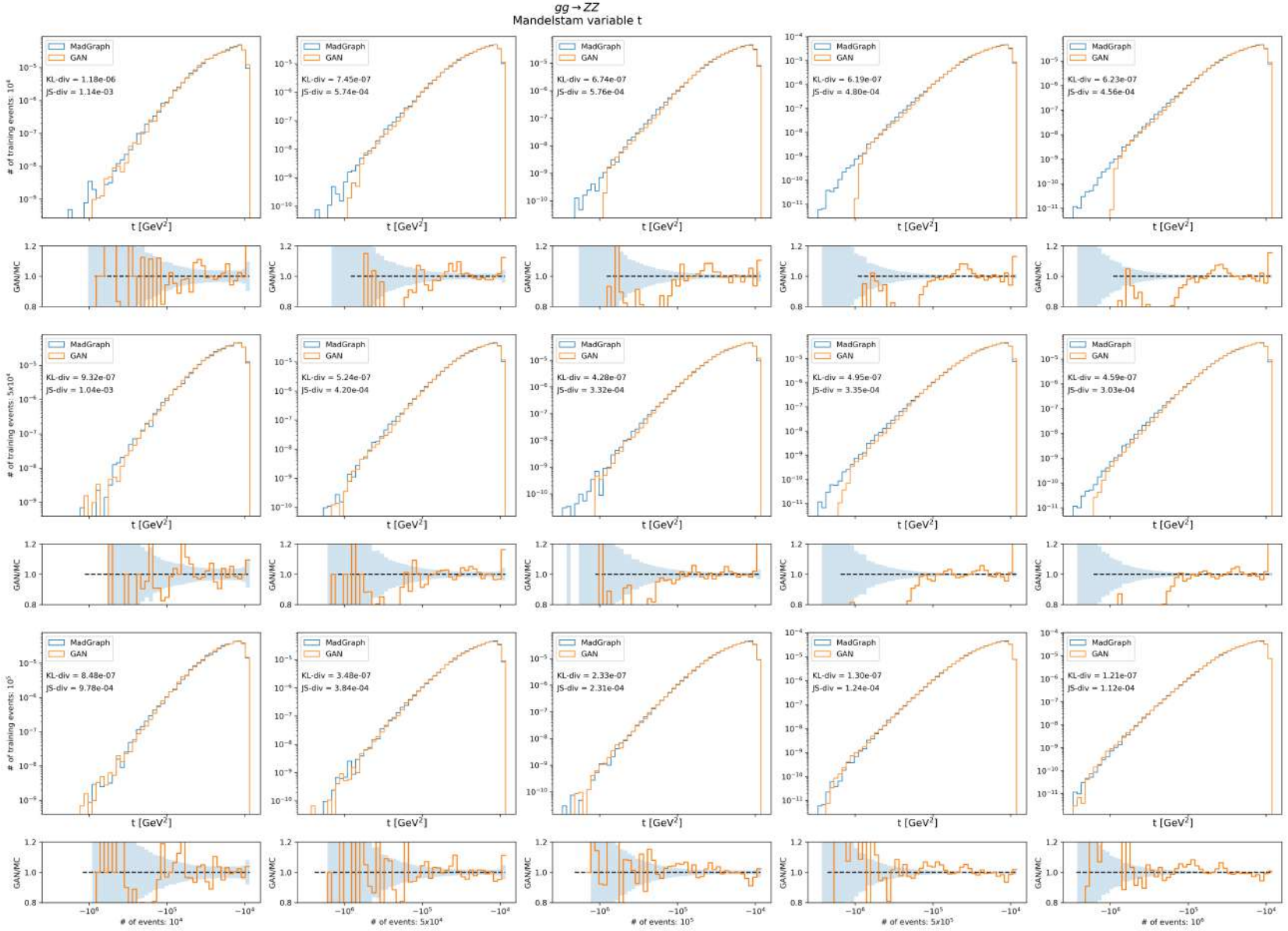


Figure 4.24 (a): Matrix of histograms of the Mandelstam variable t for the channel $gg \to ZZ$. Models trained on 10k, 50k, 100k events.

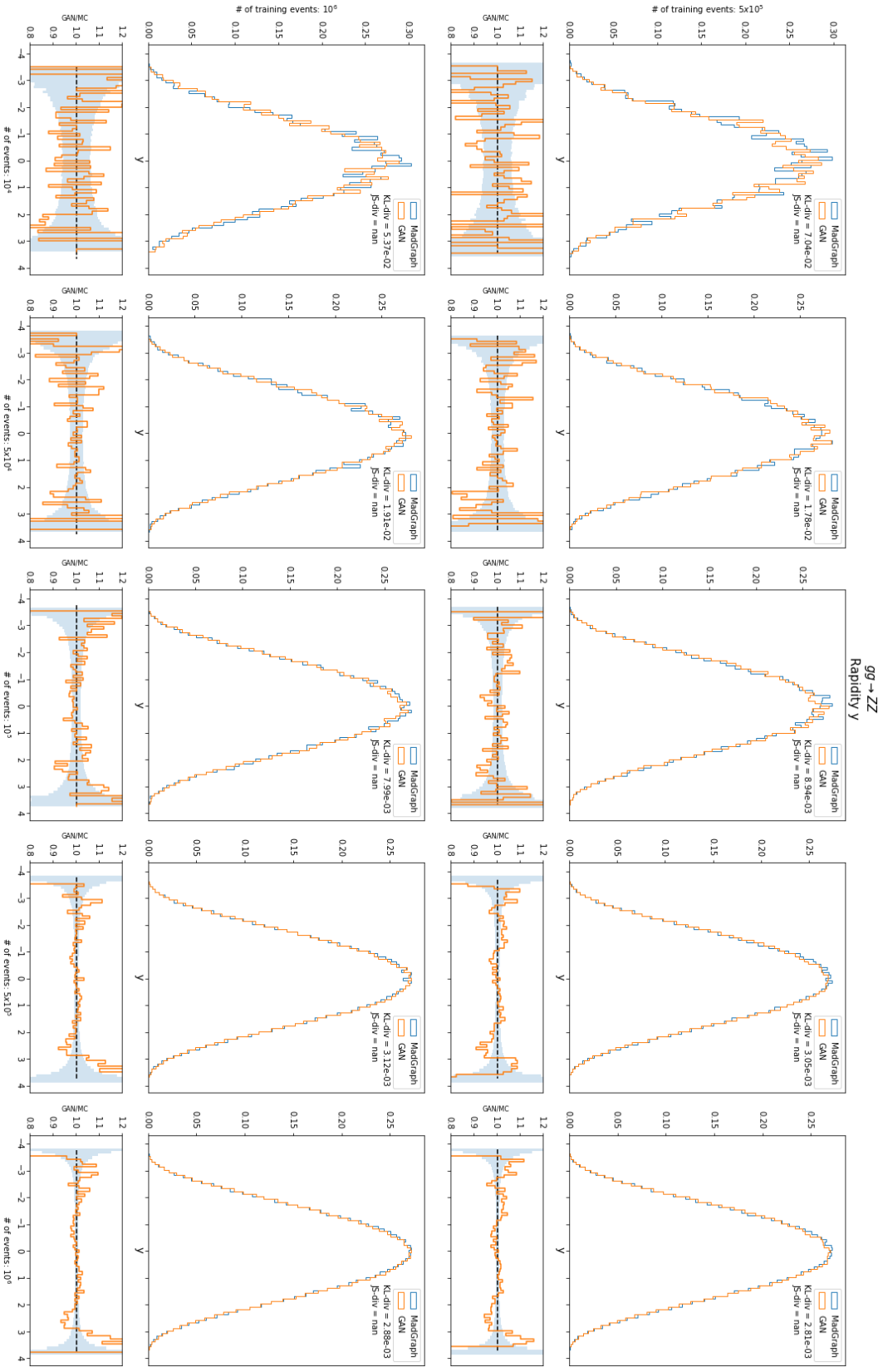


Figure 4.25 (b): Matrix of histograms of the rapidity y for the channel $gg \to ZZ$. Models trained on 500k, 1M events.

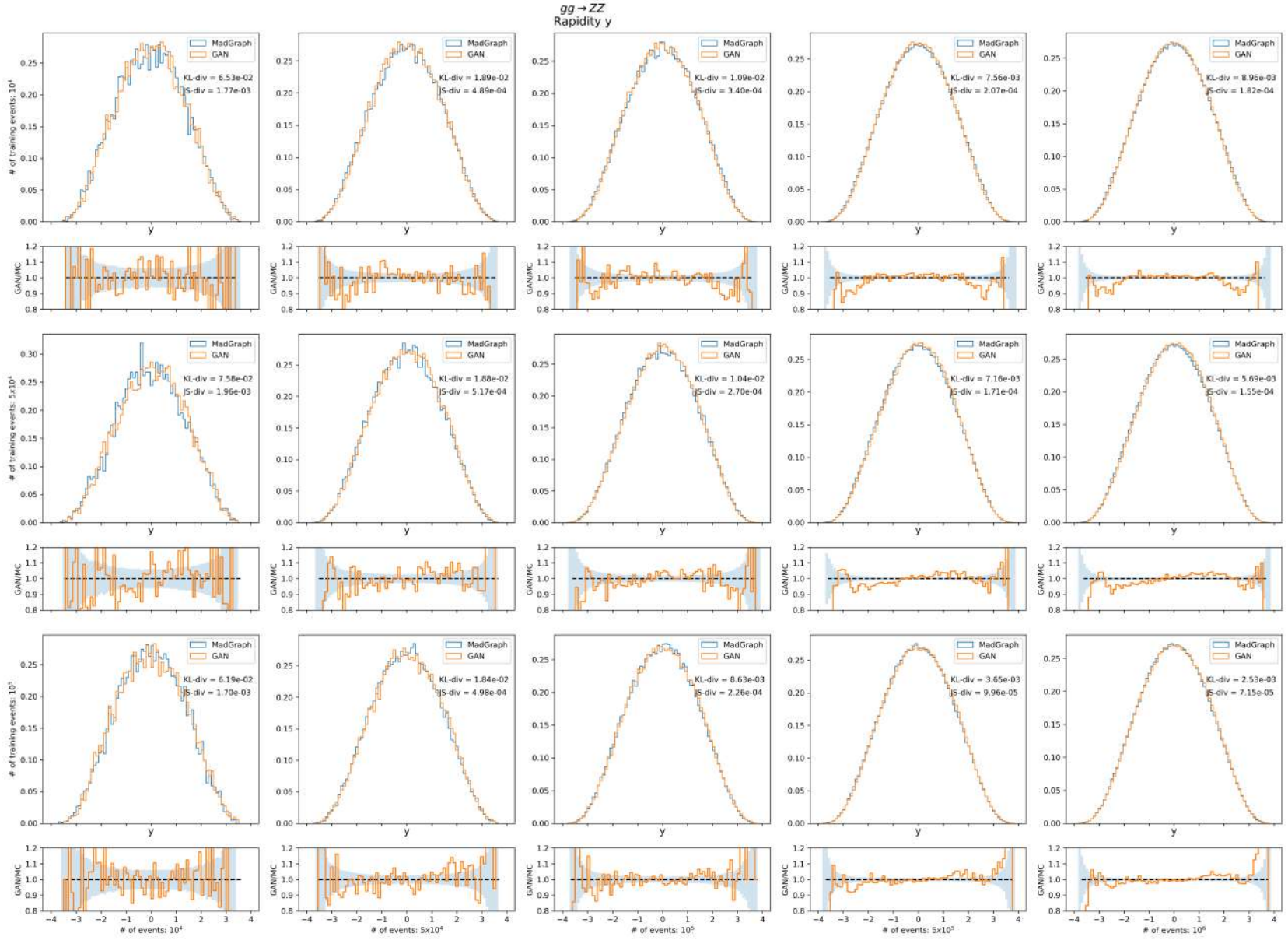


Figure 4.25 (a): Matrix of histograms of the rapidity y for the channel $gg \to ZZ$. Models trained on 10k, 50k, 100k events.

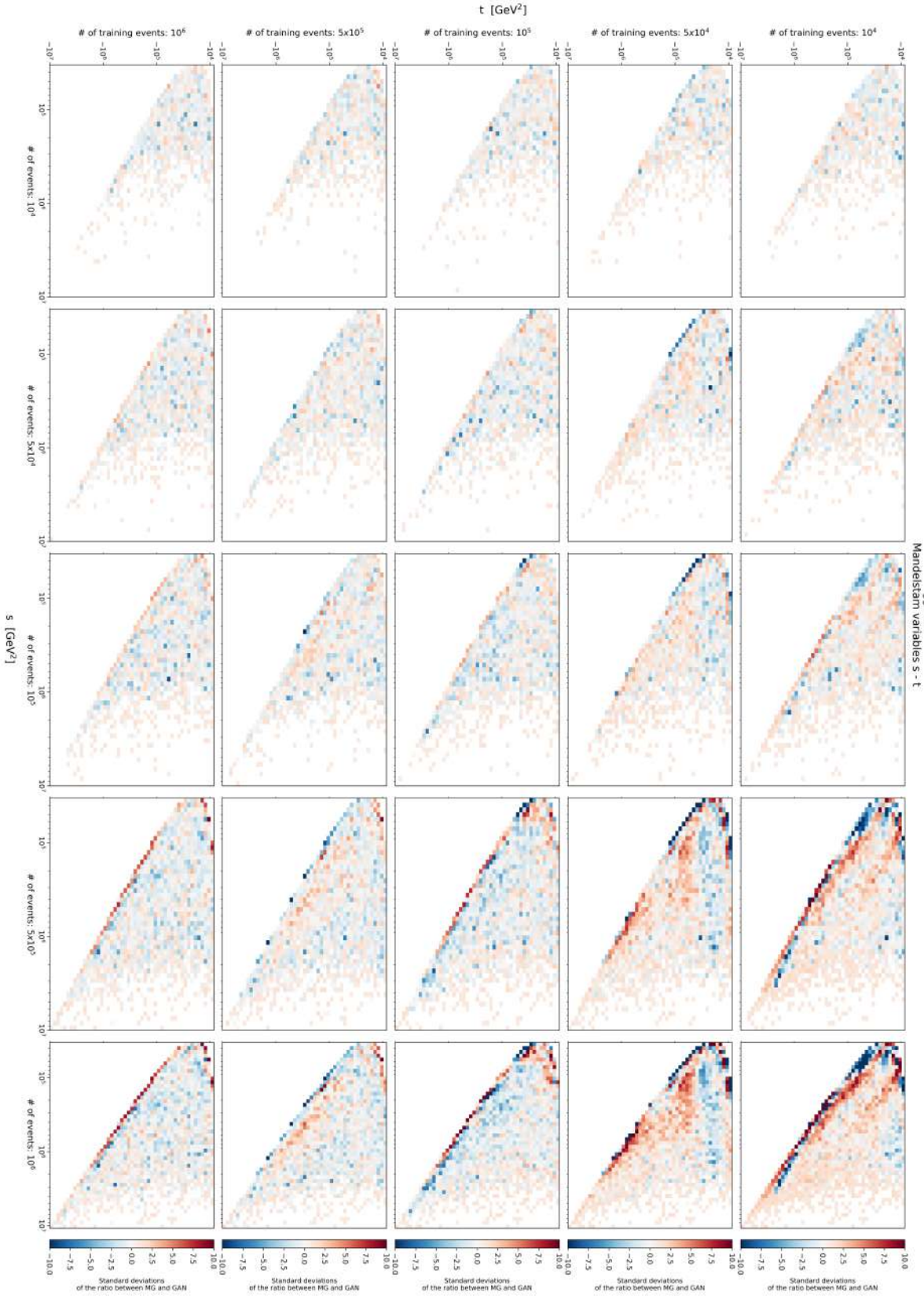


Figure 4.26: Matrix of correlations plot of the Mandelstam variable s for the channel $gg \to ZZ$.

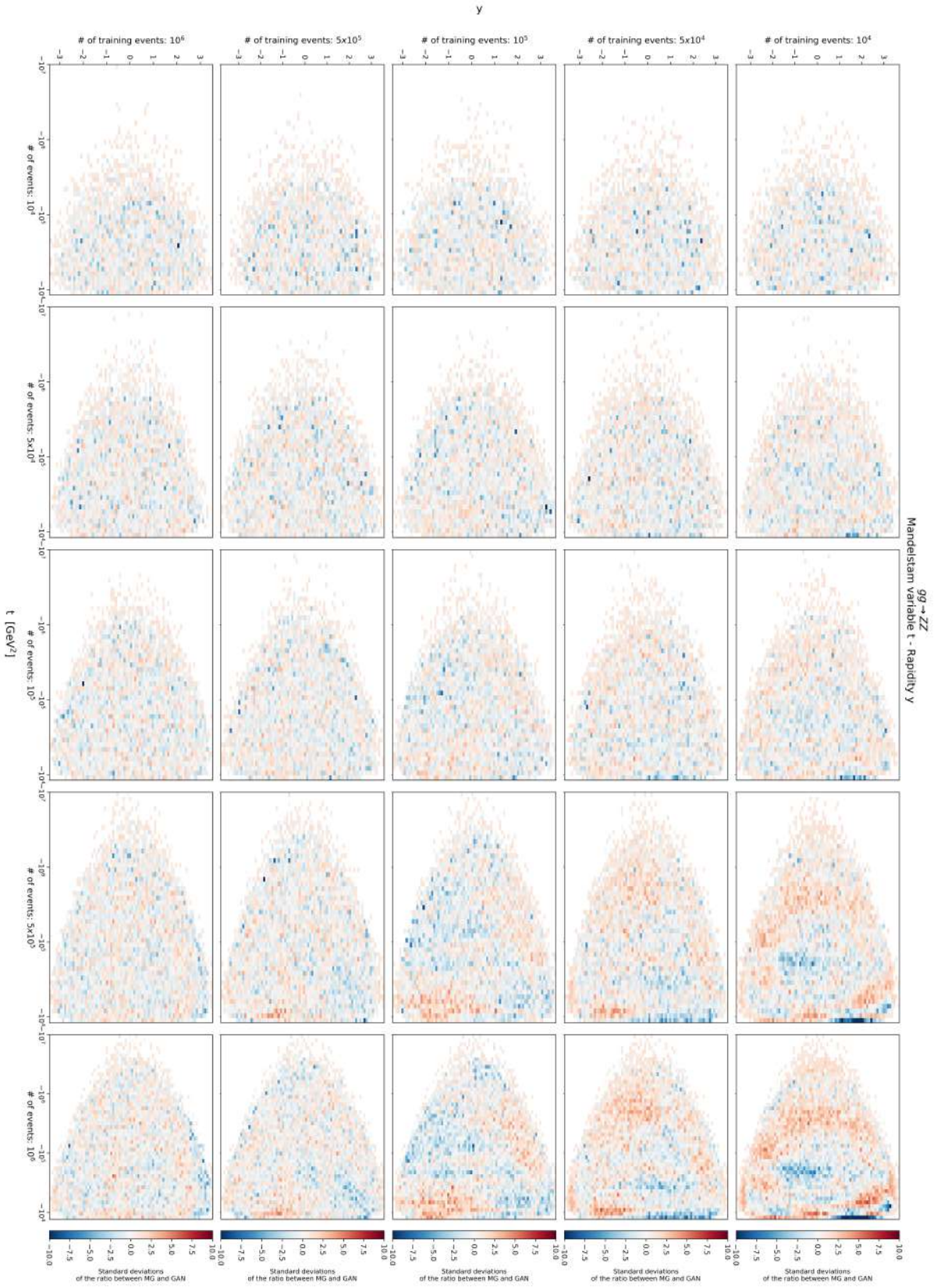


Figure 4.27: Matrix of correlations plot of the Mandelstam variable t for the channel $gg \to ZZ$.

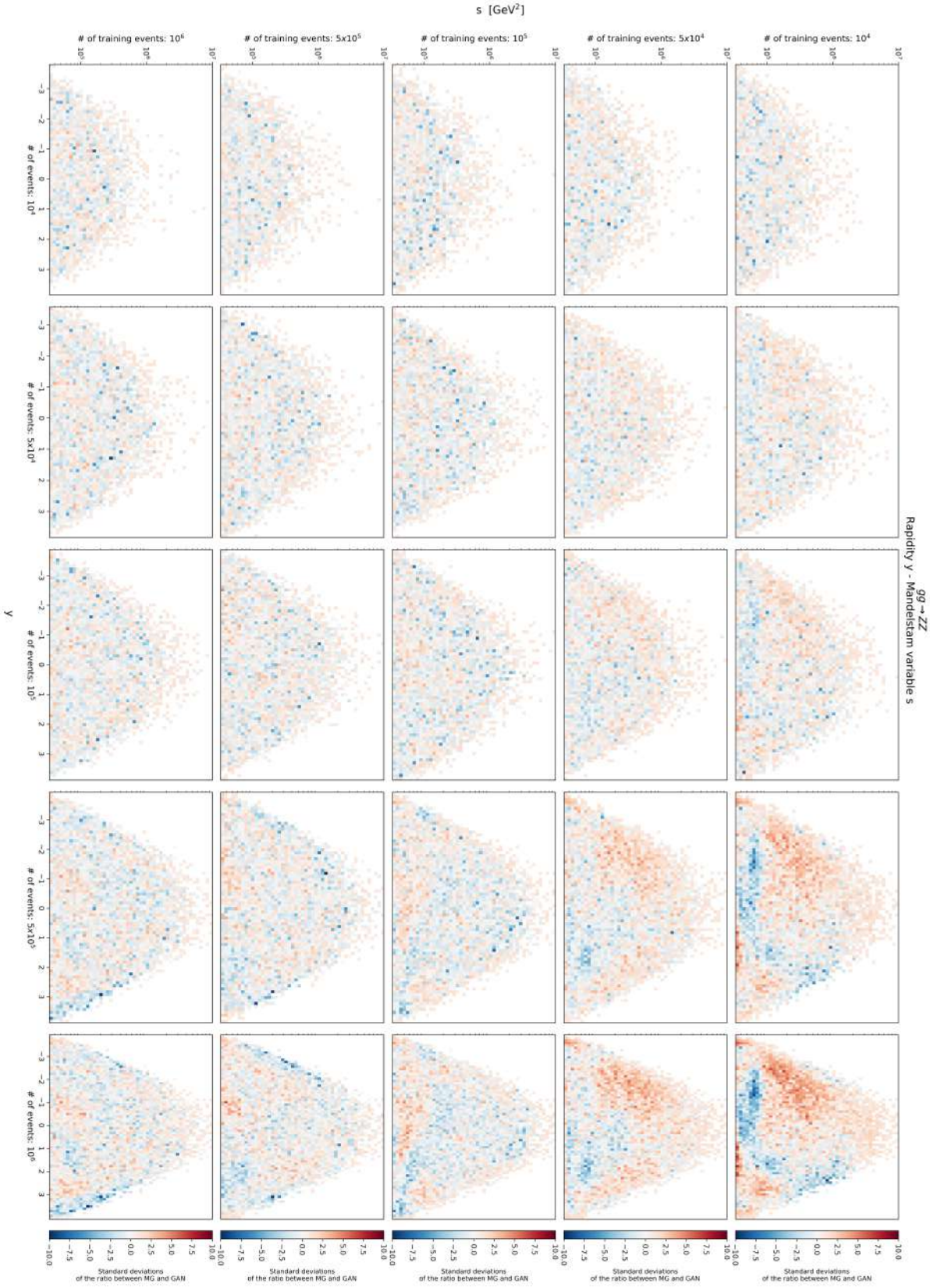
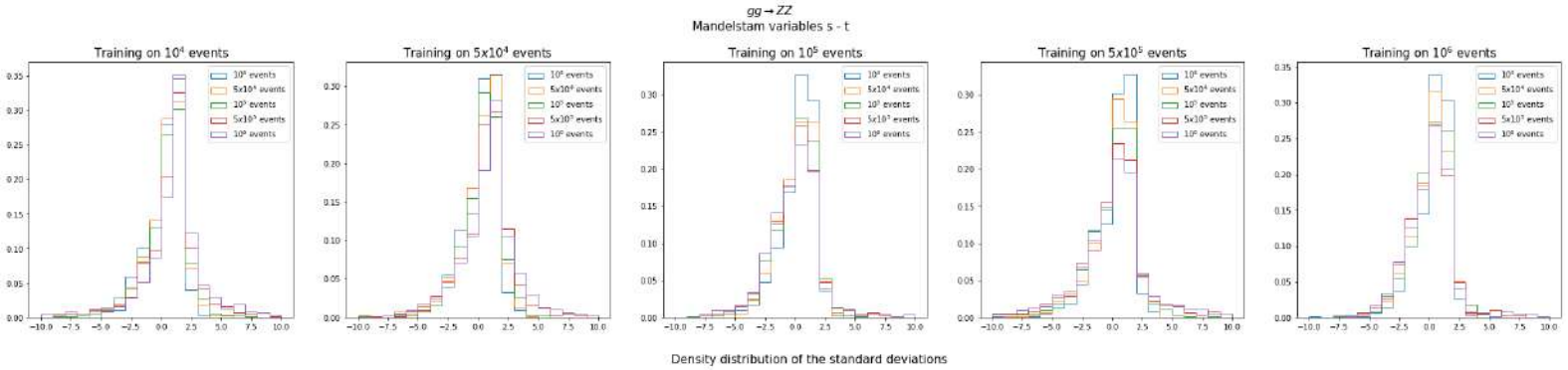
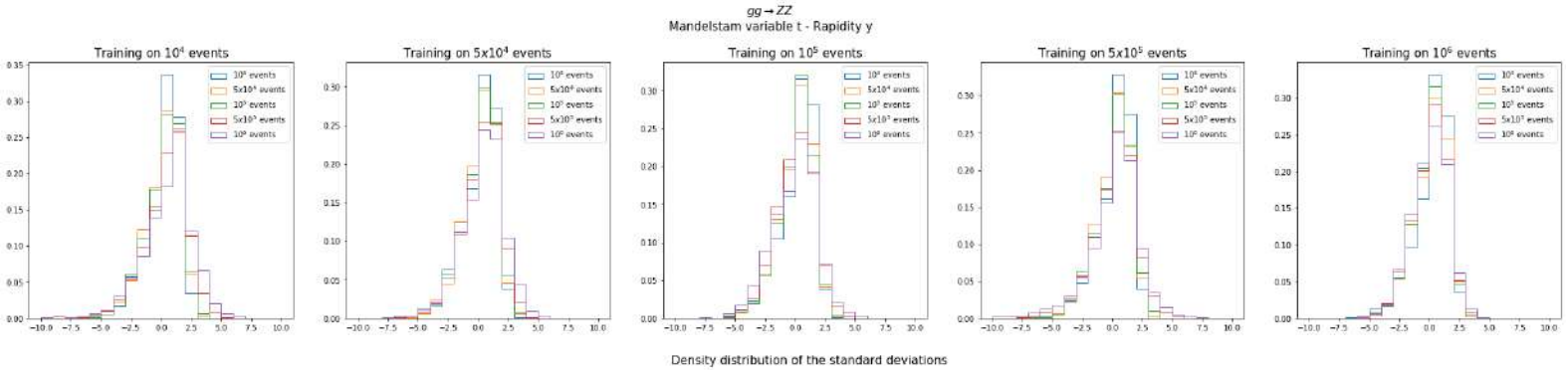


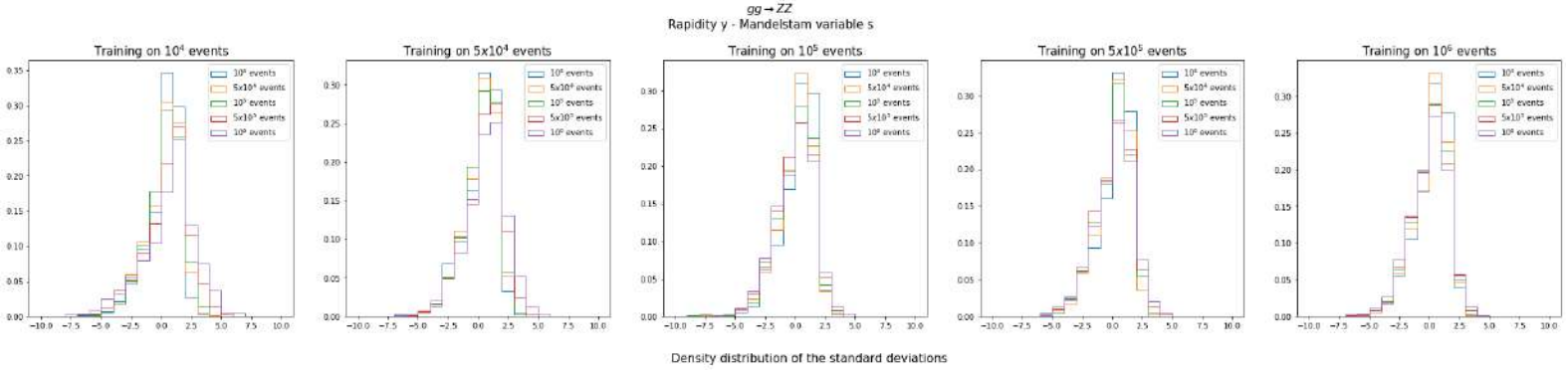
Figure 4.28: Matrix of correlations plot of the rapidity y for the channel $gg \to ZZ$.



(a)



(b)



(c)

Figure 4.29: Distribution of the errors obtained from the correlations plot for the variable s (a), t (b) and y (c). A single plot correspond to a row of the correlations plot.

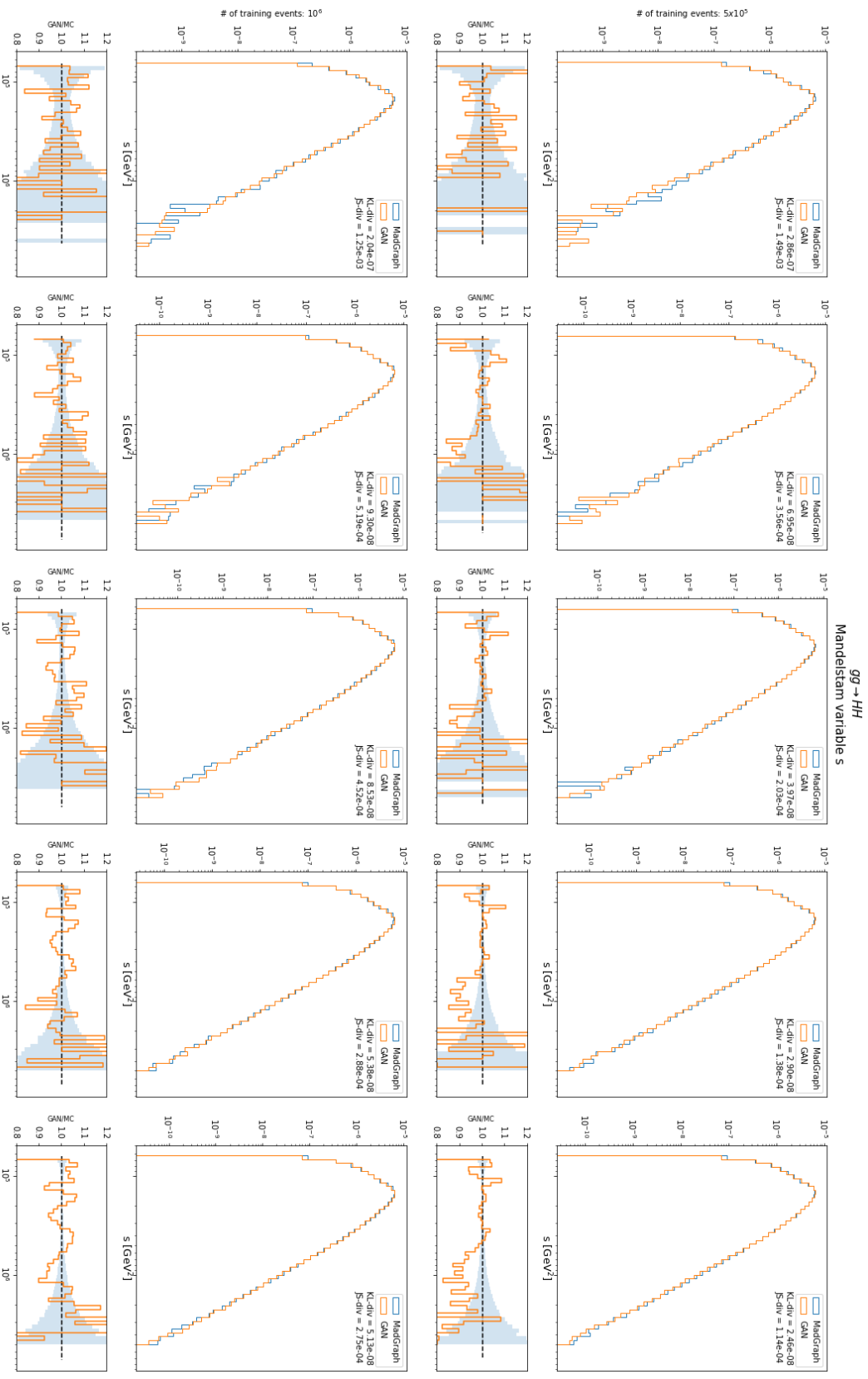


Figure 4.30 (b): Matrix of histograms of the Mandelstam variable s for the channel $gg \to HH$. Models trained on 500k, 1M events.

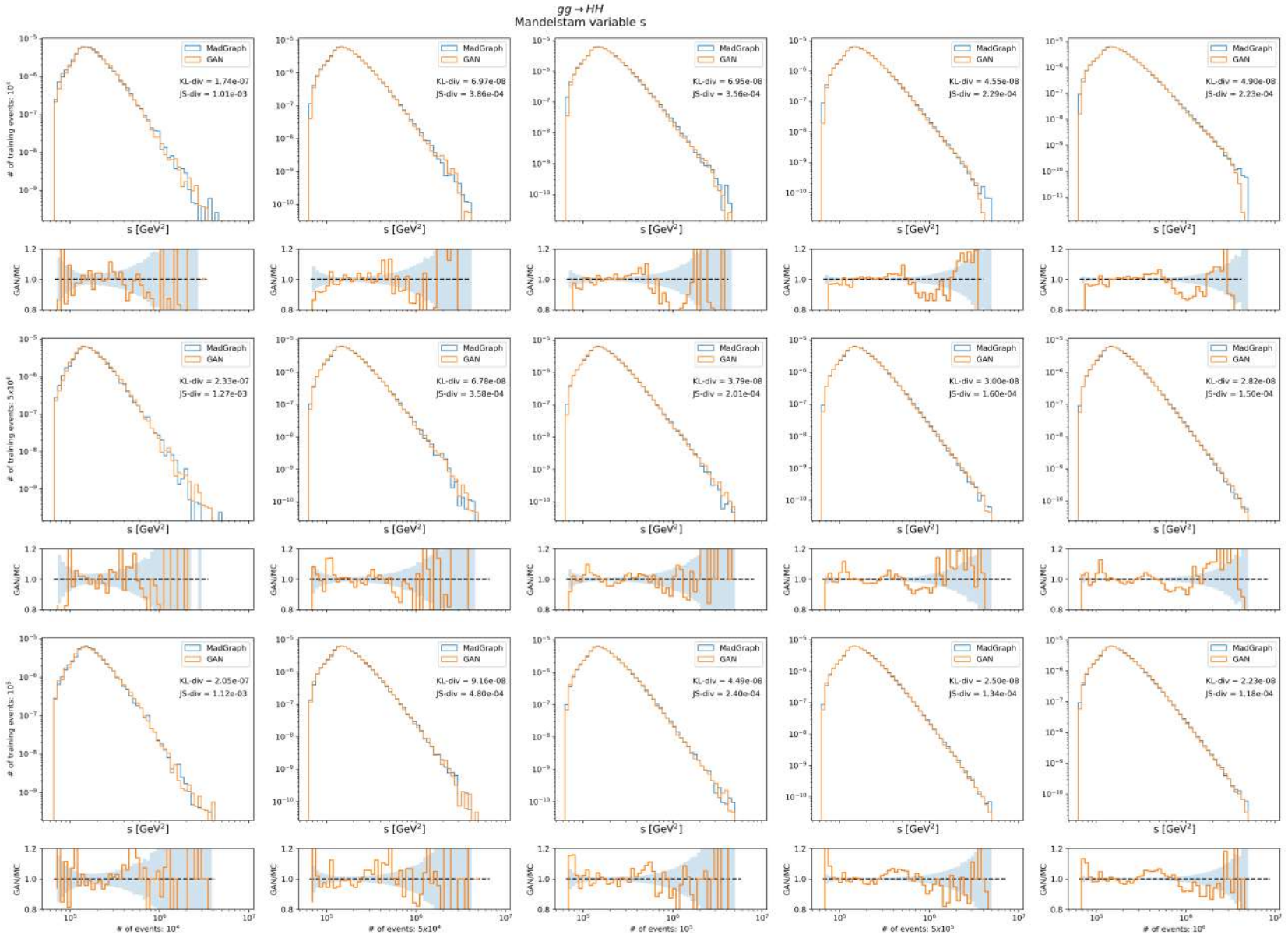


Figure 4.30 (a): Matrix of histograms of the Mandelstam variable s for the channel $gg \to HH$. Models trained on 10k, 50k, 100k events.

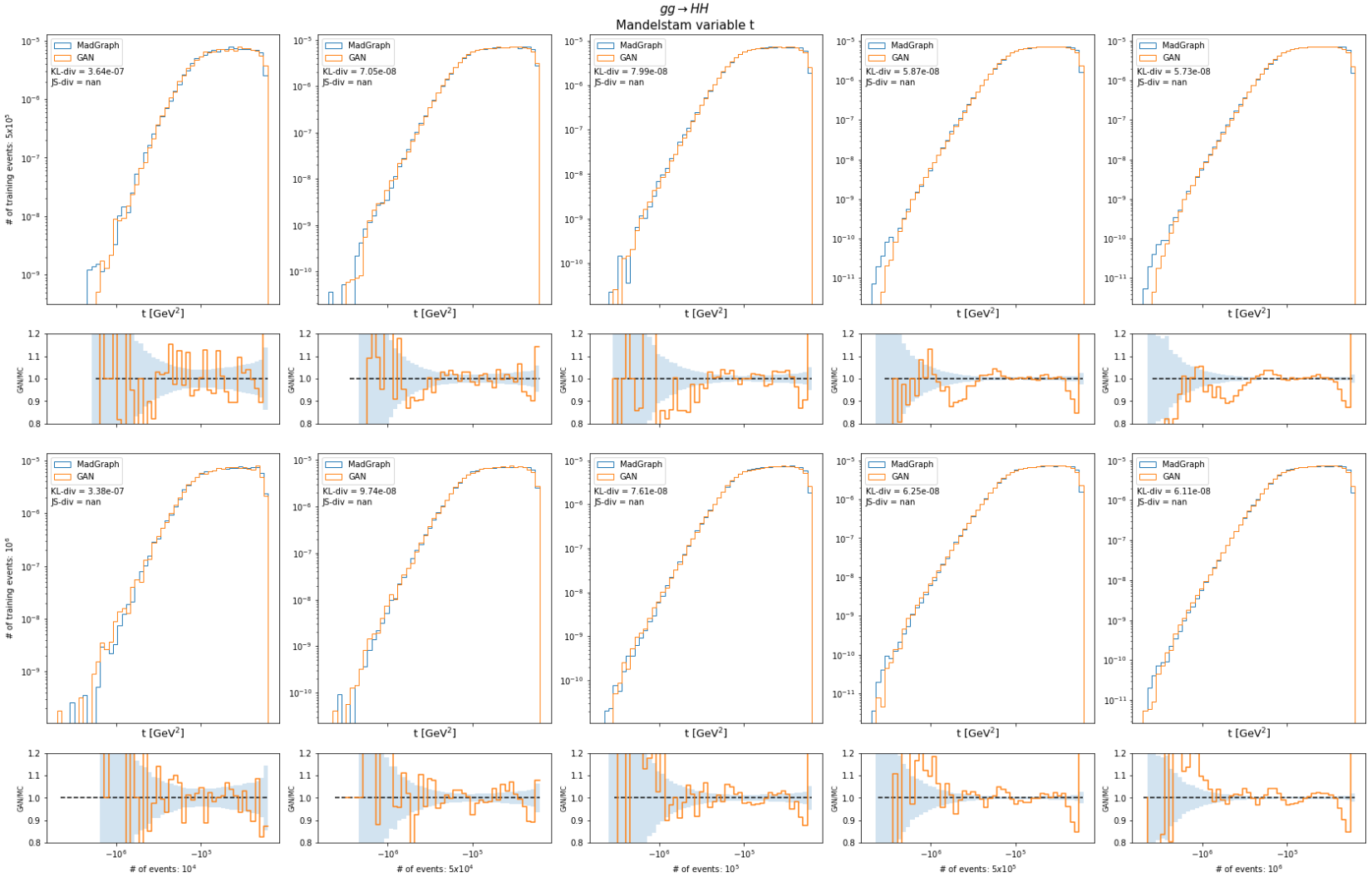


Figure 4.31 (b): Matrix of histograms of the Mandelstam variable t for the channel $gg \to HH$. Models trained on 500k, 1M events.

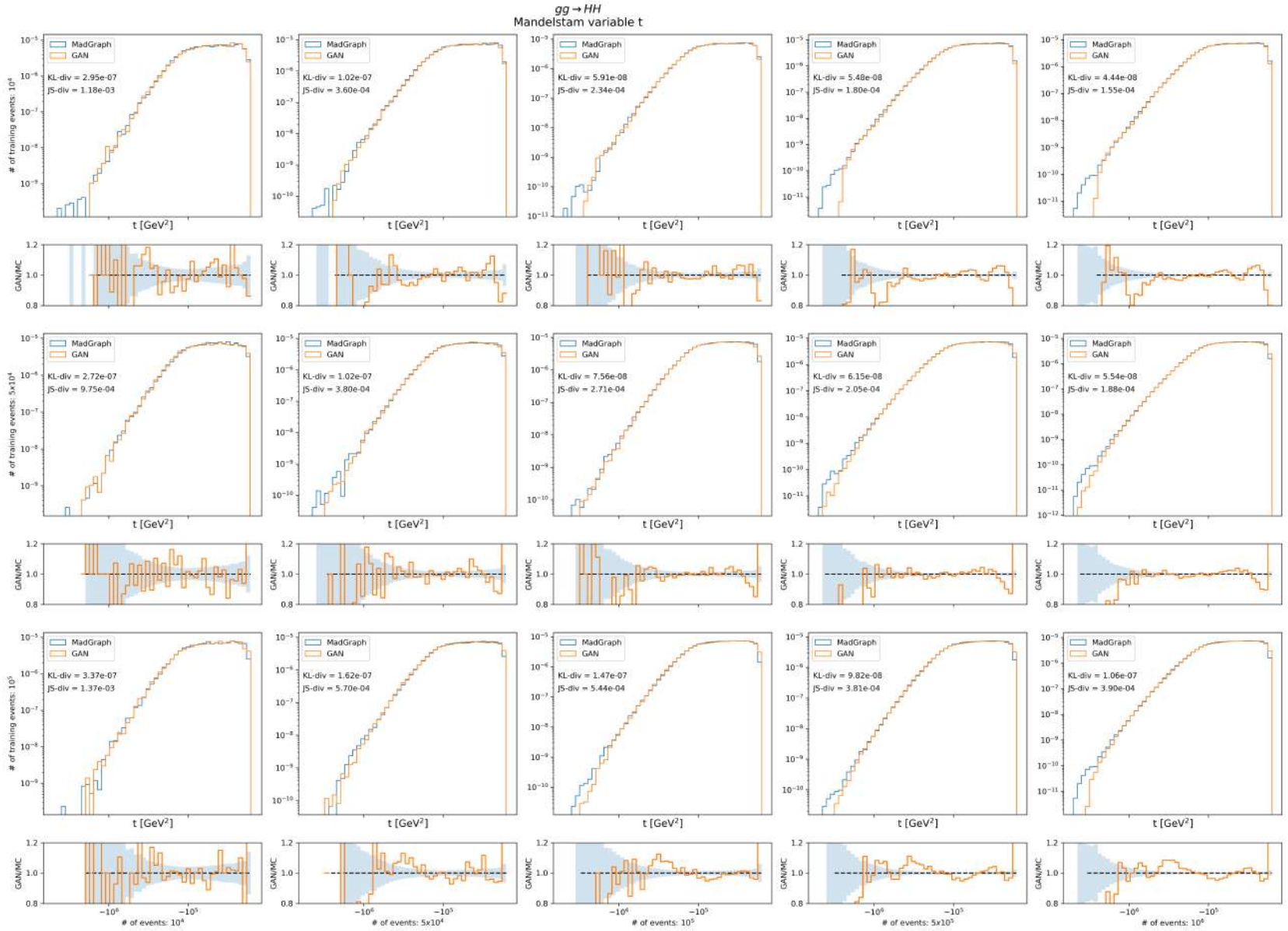


Figure 4.31 (a): Matrix of histograms of the Mandelstam variablet t for the channel $gg \to HH$. Models trained on 10k, 50k, 100k events.

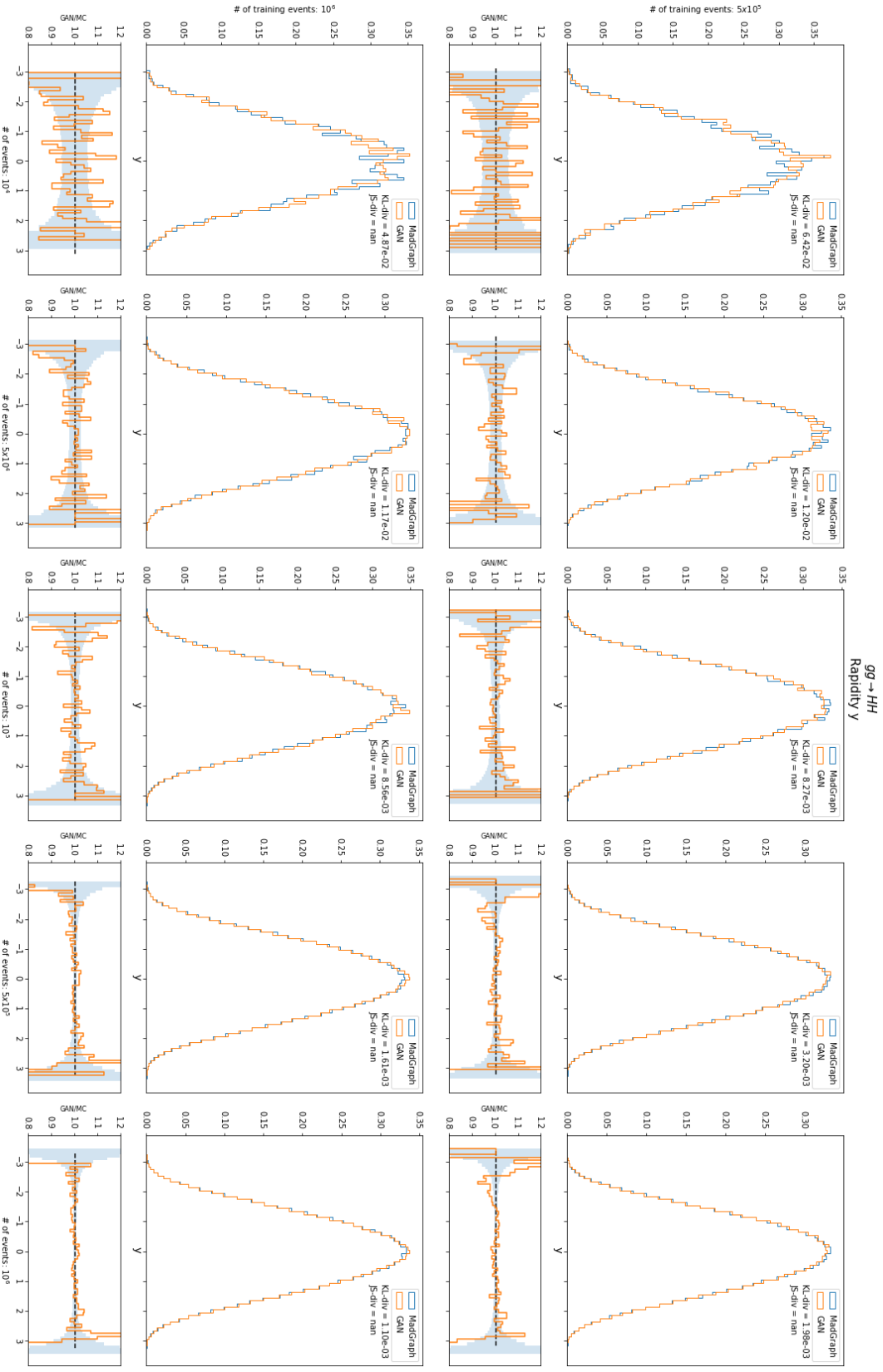


Figure 4.32 (b): Matrix of histograms of the rapidity y for the channel $gg \to HH$. Models trained on 500k, 1M events.

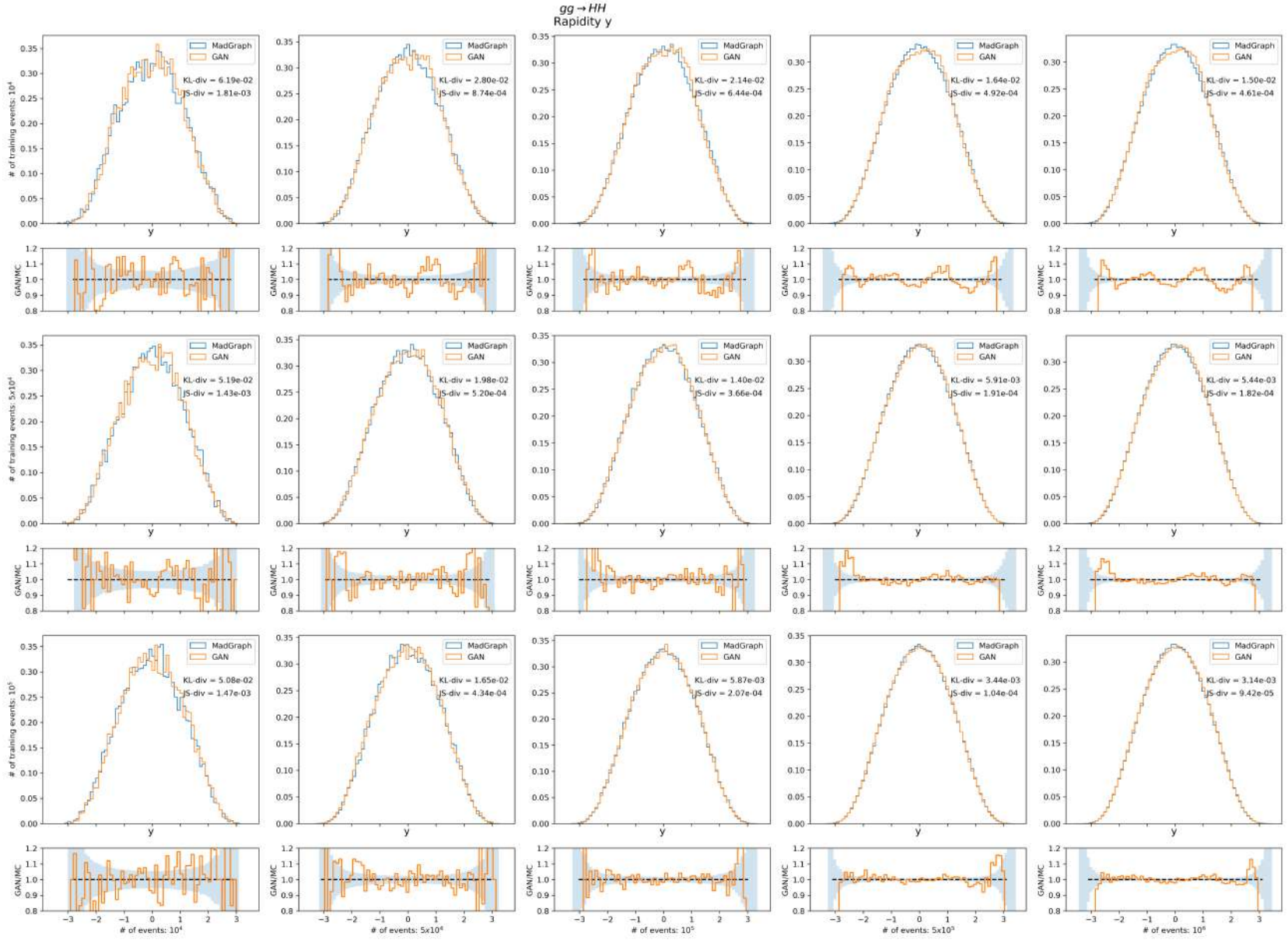


Figure 4.32 (a): Matrix of histograms of the rapidity y for the channel $gg \to HH$. Models trained on 10k, 50k, 100k events.

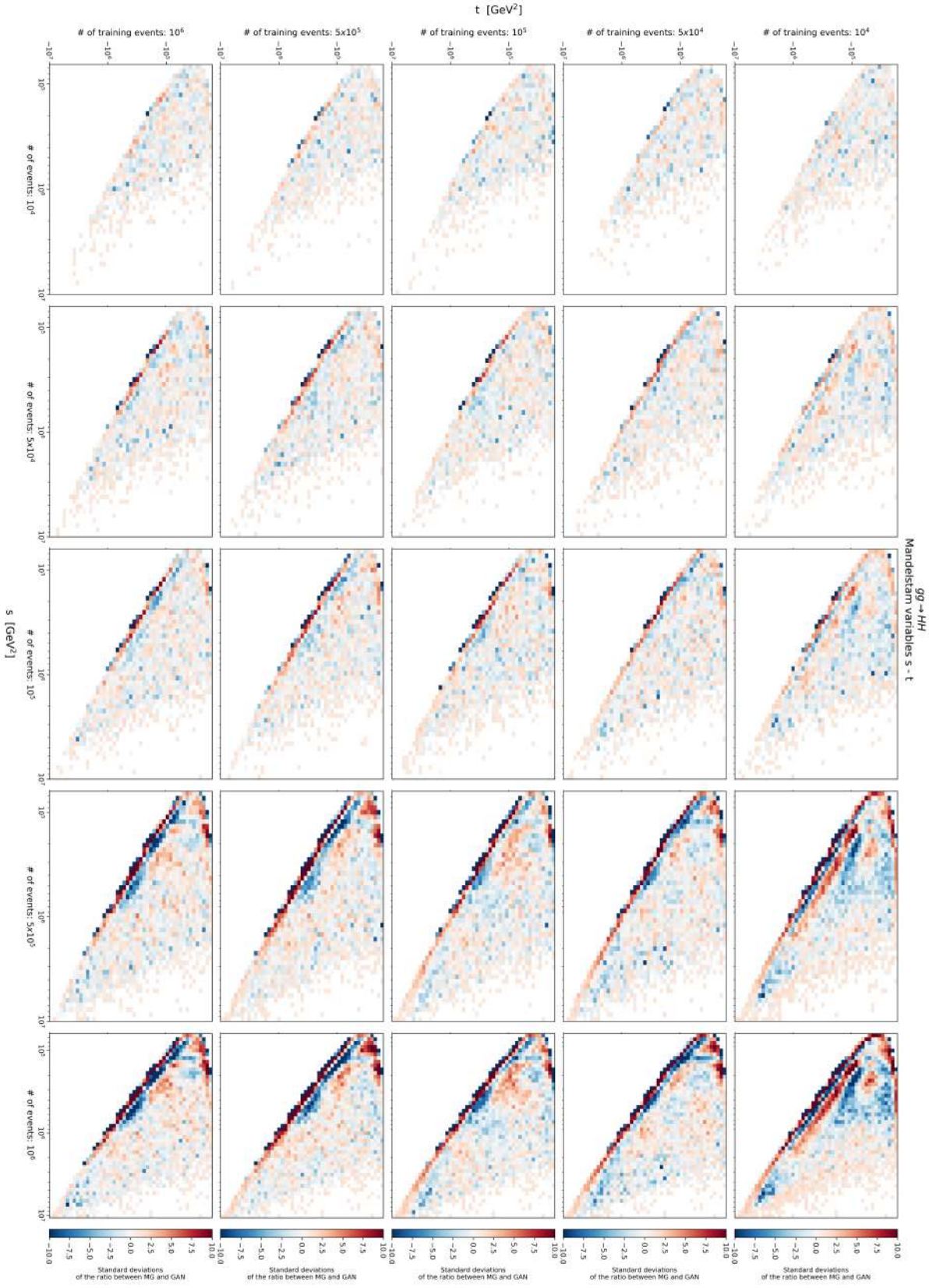


Figure 4.33: Matrix of correlations plot of the Mandelstam variable s for the channel $gg \to HH$.

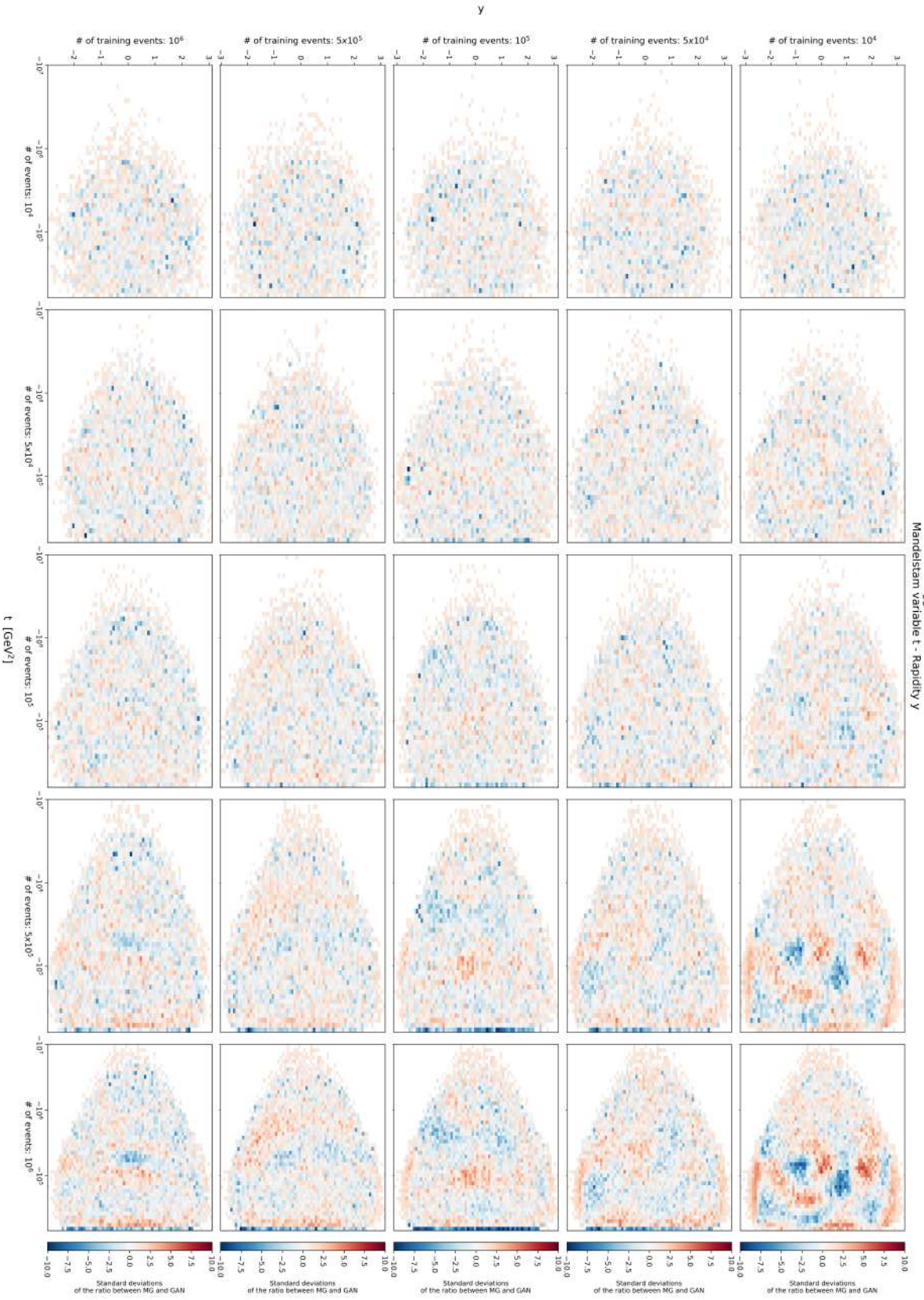
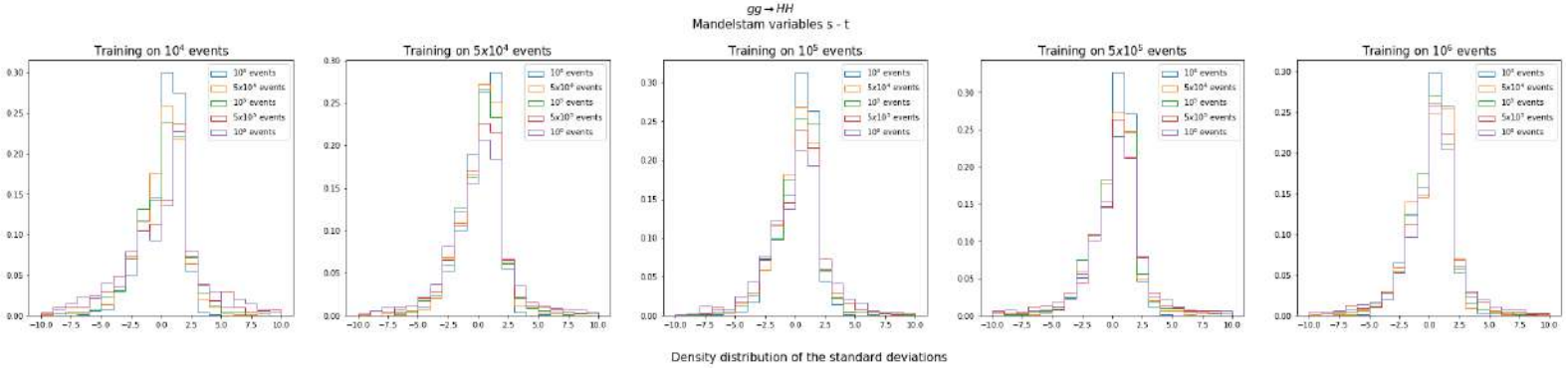
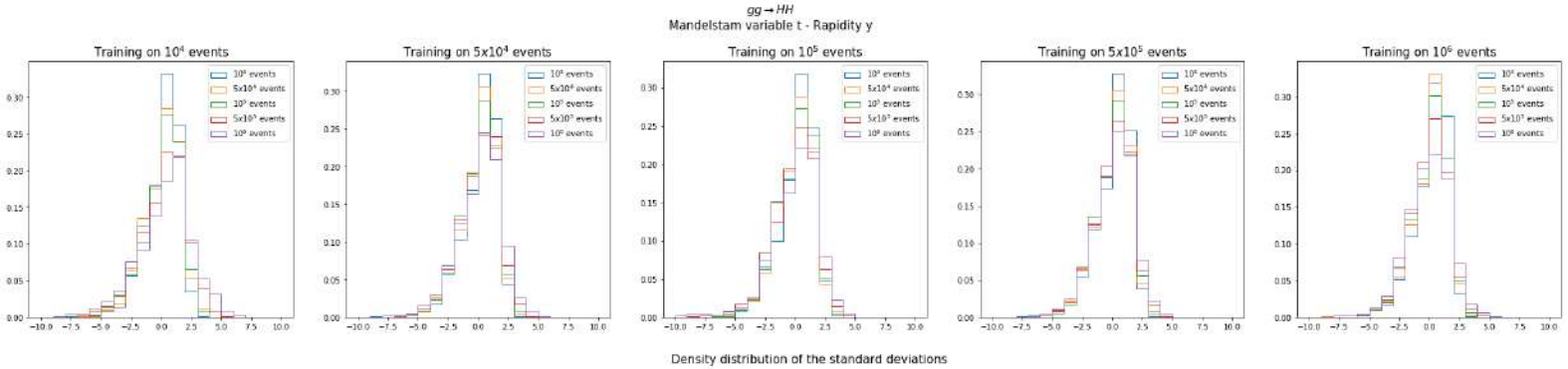


Figure 4.34: Matrix of correlations plot of the Mandelstam variable t for the channel $gg \to HH$.

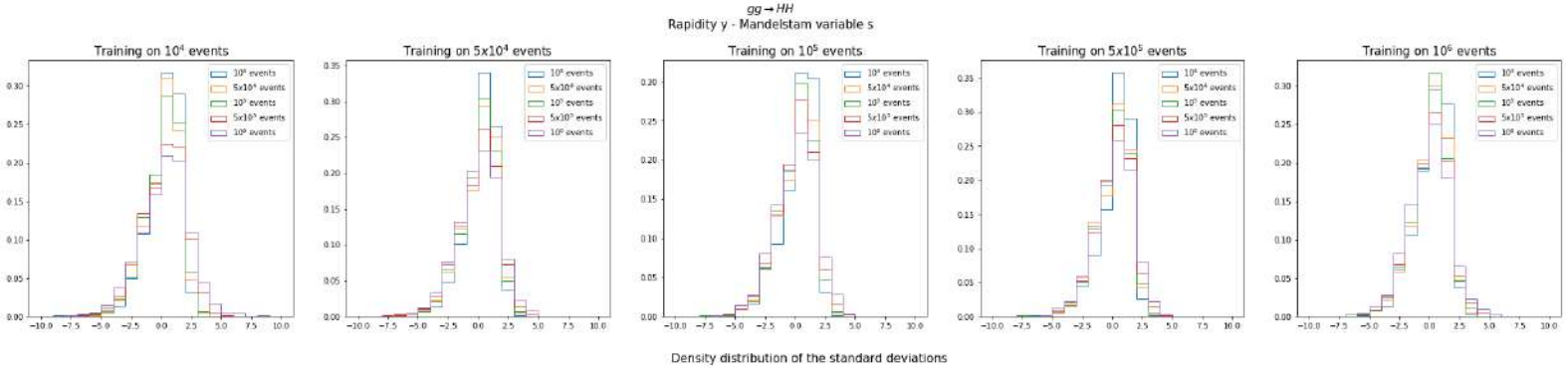
Figure 4.35: Matrix of correlations plot of the rapidity y for the channel $gg \to HH$.



(a)



(b)



(c)

Figure 4.36: Distribution of the errors obtained from the correlations plot for the variable s (a), t (b) and y (c). A single plot correspond to a row of the correlations plot.

Conclusion

In this thesis has been studied the problem of fast generation of LHC events using a Machine Learning technique called Generative Adversarial Networks (GAN). The generation process of events that is used nowadays in most simulation problems is based on Monte Carlo techniques. However, the problem of sampling from the phase space distribution with an increasing number of Feynman diagrams and number of particles results in a bottleneck that slows the full event simulation process. The solution proposed in this thesis is to use a generative model, trained using a GAN, to skip the complex events sampling from the phase space and directly reproducing the distributions of kinematical variables representing the event.

Three $2 \to 2$ scattering processes have been considered: $pp \to t\bar{t}$, $gg \to ZZ$, and $gg \to HH$. All events are generated at Leading Order using the tool `MadGraph5_aMC@NLO`. The kinematical variables used as input features of the GAN are the Mandelstam variables s and t, and the rapidity in the reference frame of the parton scattering. Before being fed into the networks, the input features are preprocessed, using the `Scikit-Learn` package, in a Yeo-Johnson transform followed by a `minmax` scaling in order to have the outputs within the range $[-1, 1]$.

The trained neural networks are implemented in `Keras` using `TensorFlow 2.3` backend and are mainly defined by four convolutional/transpose convolutional layers and a subset of their hyperparameters has been optimized using `Hyperopt`. The best hyperparametrs found for a sample of 10k events are $\sim$30k training epochs, using the `Adadelta` optimizer with a learning rate of 0.3 and a batch size of 512. The performance test of the models is based on the evaluation of the Kullback-Leibler and the Jensen-Shannon divergence and on the bin-wise ratio of the histograms between the Monte Carlo and the generated distribution.

Initially, the GAN has been trained in order to reproduce a low statistics sample with 10k events and a high statistics sample with 1M events. The results showed that the final models are able to reproduce the distribution of the input features and their correlation of all three processes. Indeed the majority of bins show a ratio error between the two distributions within three standard deviations of the Monte Carlo statistical counting error for the trainings with the sample of 10k events.

This error goes from ∼10% in the core region to ∼20% in the tail region of the distribution.. Regarding the sample of 1M events, the ratio error oscillates from ∼1% in the core region to ∼10% in the tail region of the distribution. However, the most difficult areas that the model struggles to parametrize are the cut imposed by the Mandelstam invariants and the high-energy region where shortage of data makes difficult the recognition of the right behavior of the distribution.

Finally, the ability of generation of samples with more events than the ones used for training, known as data augmentation, has been checked. Five different trainings are done with an increasing number of events for each channel from 10k events to 1M events and each model generates sample up to 1M events. The results showed that the GAN is able to learn the underlying distribution without reproducing the noise and that a factor 10 of data augmentation capability is acceptable in the smaller samples.

Appendix

A QCD lagrangian and Feynman rules

In this section is described how to write the QCD lagrangian[19]. In order to do that, let us define a fermionic quantum field with a color triplet:

$$\psi(x) = \begin{pmatrix} \psi_1(x) \\ \psi_2(x) \\ \psi_3(x) \end{pmatrix} \tag{5.1}$$

QCD is a non-Abelian Gauge theory, also called Yang-Mills theory, in which the symmetry group is SU(3). Moreover, the global symmetry of SU(3) is promoted to a local symmetry, this means that the field $\psi(x)$ transform as $\psi(x) \to U(x)\psi(x)$ where $U(x) = e^{ig\lambda_a(x)T^a/2}$ and g is a dimensionless coupling constant. The T^a's are hermitian and traceless 3 x 3 matrices and they generate the Lie algebra with commutation rules given by:

$$\left[T^a, T^b\right] = if^{abc}T_c \tag{5.2}$$

where the structure constants f^{abc} are real and antisymmetric.

Assuming the local symmetry of the group, it is necessary to define a new Gauge field and it's transformation rule in order to ensure that the derivative term of the field transform like the field itself. To do so, we define the Gauge field A^a_μ and the covariant derivative $D_\mu = (\partial_\mu - igT_aA^a_\mu)$, with the transformation rule:

$$A_\mu \to U(x)(A_\mu - \frac{i}{g}\partial_\mu)U^\dagger(x) \tag{5.3}$$

the derivative term transform as $D_\mu\psi(x) \to U(x)D_\mu\psi(x)$, where A_μ is a shorthand of $A^a_\mu T^a$. The T^a are written in the adjoint representation in which the index a

runs from 1 to 8, this allows to interpret the fields A^a_μ as eight gluons mediators of the strong force.

Summarizing, the full QCD lagrangian with the kinetic terms for $\psi(x)$ and $A_\mu(x)$ plus the mass terms can be written as:

$$\mathcal{L} = -\frac{1}{4}F^a_{\mu\nu}F^{a\mu\nu} + \sum_q \bar{\psi}^j_k(i\not{D}^k_j - m\delta^k_j)\psi_k \tag{5.4}$$

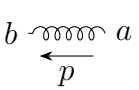
where the sum is carried over the number of quarks and $F^a_{\mu\nu} = \partial_\mu A^a_\nu - \partial_\nu A^a_\mu + gf^{abc}A^b_\mu A^c_\nu$ is the gauge field strength tensor. In order to correctly quantize the theory it's necessary to use the Fadeev-Popov method which introduce other virtual particles called 'ghosts' adding to the previous lagrangian the terms:

$$\mathcal{L}_{ghost} = \partial_\mu \bar{c}^a \partial^\mu c^a + gf^{abc}(\partial^\mu \bar{c}^a)A^b_\mu c^c \tag{5.5}$$

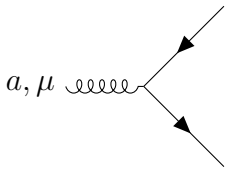
Writing explicitly the gauge strength tensor and the covariant derivative it is possible to derive the Feynman rules of QCD reported below.

Fermion propagator:

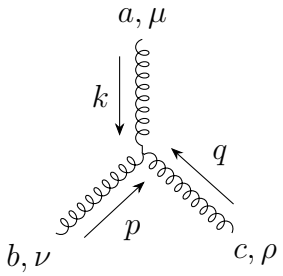
$= \frac{i(\not{p}+m)}{p^2-m^2+i\epsilon}\delta^{a,b}$

Gluon propagator: 

$= \frac{-ig^{\mu\nu}}{p^2+i\epsilon}\delta^{a,b}$

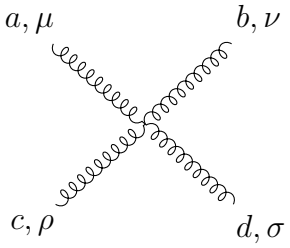
Fermion vertex: 

$= ig\gamma^\mu T^a$

3-boson vertex: 

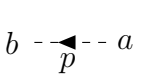
$= gf^{abc}[g^{\mu\nu}(k-p)^\rho + g^{\nu\rho}(p-q)^\mu + g^{\rho\mu}(q-k)^\nu]$

4-boson vertex:



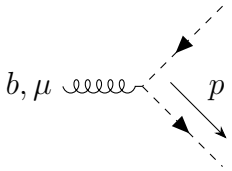
$$= \begin{aligned} -ig^2[&f^{abc}f^{cde}(g^{\mu\rho}g^{\nu\sigma} - g^{\mu\sigma}g^{\nu\rho}) \\ &+f^{ace}f^{bde}(g^{\mu\nu}g^{\rho\sigma} - g^{\mu\sigma}g^{\nu\rho}) \\ &+f^{ade}f^{bce}(g^{\mu\nu}g^{\rho\sigma} - g^{\mu\rho}g^{\nu\sigma})] \end{aligned}$$

Ghost propagator:



$$= \frac{-i\delta^{ab}}{p^2+i\epsilon}$$

Ghost vertex:

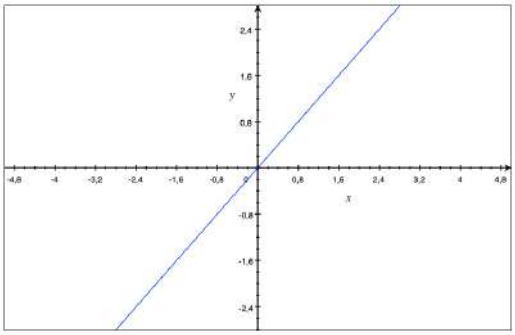


$$= -gf^{abc}p^{\mu}$$

B Activation functions

Here are reported the most commonly used activation functions for an artificial neuron in a neural network.

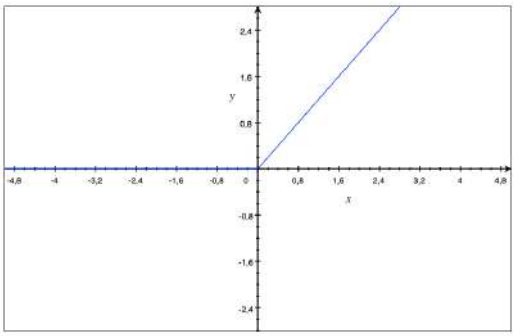
Linear function
Simple non-saturating linear function:



$$f(x) = x \tag{5.6}$$

Figure 5.1: Linear activation function

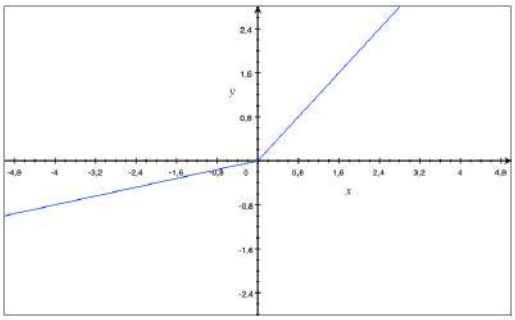
ReLU
Rectified Linear Unit (ReLU) introduces non linearities in the output. It is a non-saturating function with the disadvantage of having dead neurons, stopping the gradient flow, for negative inputs.



$$f(x) = max(0, x) \tag{5.7}$$

Figure 5.2: ReLU activation function

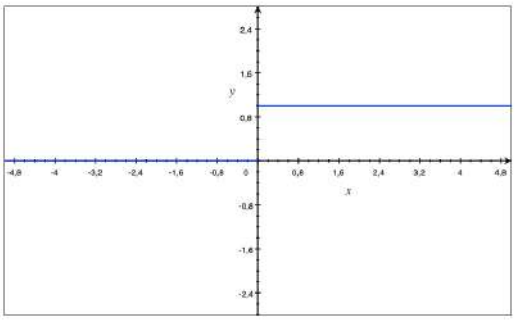
Leaky ReLU
Resolve the problem of dead neurons adding a small slope for negative inputs which ensure a gradient flow in this range.



$$f(x) = \begin{cases} x & x > 0 \\ \alpha x & x < 0 \end{cases} \tag{5.8}$$

Figure 5.3: Leaky ReLU activation function

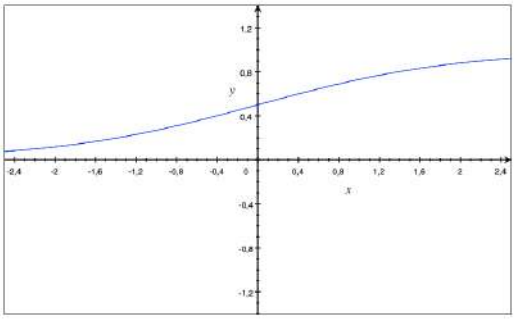
Binary
Standard limited step function where the output of the neuron is 0 or 1. In this case a small change in the weights can cause the output of the neuron to completely flip. That flip may then cause the behaviour of the rest of the network to change in an unpredicted way. Therefore, it is preferable to use a smooth activation function which imply small variations of the output for small variations of the weights.



$$f(x) = \Theta(x) \tag{5.9}$$

Figure 5.4: Step activation function

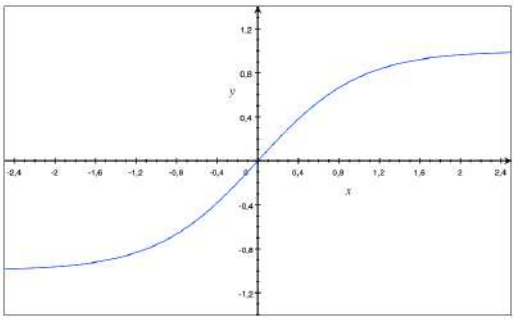
Sigmoid
Smooth limited non-linear function which maps in the range $[0, 1]$. The drawbacks of this function are that the gradients near the tails are almost zero causing the problem of vanishing gradients in these regions and that the outputs are not zero-centered.



$$f(x) = \frac{1}{1 + e^{-x}} \tag{5.10}$$

Figure 5.5: Sigmoid activation function

Tanh
Smooth non-linear limited function which maps in the range $[-1, 1]$. Like the Sigmoid function vanishing gradients are possible in the tails but in this case the outputs are zero-centered.



$$f(x) = tanh(x) \tag{5.11}$$

Figure 5.6: Tanh activation function

Softmax
The Softmax activation functions is used in multinomial logistic regression in the last layer. Each output lies in the range $[0, 1]$ and it's normalized such that it sums to 1. This allow to interpret the output of a Softmax layer as a probability distribution.

$$\begin{pmatrix} 3 \\ 1 \\ 0.2 \end{pmatrix} \xrightarrow{\bar{f}(\bar{x})} \begin{pmatrix} 0.84 \\ 0.11 \\ 0.05 \end{pmatrix} \qquad f(\bar{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{5.12}$$

Figure 5.6: Softmax activation function

C Feynman diagrams

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] S. Agostinelli et al. "Geant4—a simulation toolkit". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 506.3 (2003). ISSN: 0168-9002. DOI: https://doi.org/10.1016/S0168-9002(03)01368-8.

[3] J. et al. Alwall. "A standard format for Les Houches Event Files". In: *Computer Physics Communications* 176.4 (2007). ISSN: 0010-4655. DOI: 10.1016/j.cpc.2006.11.010. URL: http://dx.doi.org/10.1016/j.cpc.2006.11.010.

[4] Johan Alwall et al. "MadGraph 5: going beyond". In: *Journal of High Energy Physics* 2011.6 (2011). ISSN: 1029-8479. DOI: 10.1007/jhep06(2011)128. URL: http://dx.doi.org/10.1007/JHEP06(2011)128.

[5] Richard D. et al. Ball. "Parton distributions for the LHC run II". In: *Journal of High Energy Physics* 2015.4 (2015). ISSN: 1029-8479. DOI: 10.1007/jhep04(2015)040. URL: http://dx.doi.org/10.1007/JHEP04(2015)040.

[6] J. Bergstra, D. Yamins, and D. D. Cox. *Making a Science of Model Search*. 2012. arXiv: 1209.5111 [cs.CV].

[7] Enrico et al. Bothmann. "Event generation with Sherpa 2.2". In: *SciPost Physics* 3 (2019). DOI: 10.21468/scipostphys.7.3.034. URL: http://dx.doi.org/10.21468/SciPostPhys.7.3.034.

[8] P Calafiura et al. *ATLAS HL-LHC Computing Conceptual Design Report*. Tech. rep. CERN-LHCC-2020-015. LHCC-G-178. Geneva: CERN, 2020. URL: https://cds.cern.ch/record/2729668.

[9] François Chollet. *Keras*. https://github.com/fchollet/keras. 2015.

[10] Lyndon Evans and Philip Bryant. "LHC Machine". In: *Journal of Instrumentation* 3.08 (2008), S08001–S08001. DOI: 10.1088/1748-0221/3/08/s08001. URL: https://doi.org/10.1088/1748-0221/3/08/s08001.

[11] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[12] Valentin Hirschi and Olivier Mattelaer. *Automated event generation for loop-induced processes*. 2015. arXiv: 1507.00020 [hep-ph].

[13] Malvin H Kalos and Paula A Whitlock. *Monte Carlo methods*. New York, NY: Wiley, 1986.

[14] *LEP design report*. Geneva: CERN, 1984. URL: https://cds.cern.ch/record/102083.

[15] "LHCb CPU Usage Forecast". In: (2019). URL: https://cds.cern.ch/record/2696552.

[16] Esma Mobs. "The CERN accelerator complex - 2019. Complexe des accelerateurs du CERN - 2019". In: (2019). General Photo. URL: https://cds.cern.ch/record/2684277.

[17] S. Ovyn, X. Rouby, and V. Lemaitre. *Delphes, a framework for fast simulation of a generic collider experiment*. 2010. arXiv: 0903.2225 [hep-ph].

[18] F. et al. Pedregosa. "Scikit-learn: Machine Learning in Python". In: 12 (2011), pp. 2825–2830.

[19] Michael E. Peskin and Daniel V. Schroeder. *An Introduction to quantum field theory*. Reading, USA: Addison-Wesley, 1995. ISBN: 978-0-201-50397-5.

[20] Torbjörn et al. Sjöstrand. "An introduction to PYTHIA 8.2". In: *Computer Physics Communications* 191 (2015). ISSN: 0010-4655. DOI: 10.1016/j.cpc.2015.01.024. URL: http://dx.doi.org/10.1016/j.cpc.2015.01.024.

Acknowledgments