AutoML for Fast Simulation *

Poliana Nascimento Ferreira
poliana.nferreira@gmail.com

Dalila Salamani
dalila.salamani@cern.ch

CERN Summer Programme Report 2021

Abstract

The extensive program of high energy physics (HEP) experiments relies on Monte Carlo (MC) simulation. MC consists of a basis for testing hypothesis of the underlying distribution of the data. The need for fast and large scale simulated samples for HEP experiments motivates the development of new simulation techniques. Some of those techniques, belonging to a so-called Fast Simulation, are based on Machine Learning (ML) algorithms. They use a set of training variables known as "hyperparameters". The choice of these parameters can significantly affect the model's performance. Automated machine learning (AutoML) allows to lift the burden of a manual optimization. The goal of this project is to understand how to use AutoML for fast simulation, specifically for shower simulation in the calorimeter. We implemented AutoML to tune the hyperparameters of a ML model for shower simulation. After, a metric was defined to evaluate the model's performance, combining physics and ML validations. This metric was then used to test and compare the existing AutoML techniques.

Keywords: *Automated Machine Learning; Fast Simulation; Generative Networks; Particle Showers; Calorimeters.*

1. Introduction

The physics program of LHC experiments relies on Monte Carlo simulation. In the ATLAS experiment for example, in 2019, 38% of the CPU time was dedicated to the Monte Carlo simulation, [3]; in CMS in the same year, the simulations were roughly half the CPU time consumption, and that need tends to grow in the next years with the High Luminosity LHC (HL-LHC) [6].

*Slides for the presentation of this project can be found in: https://github.com/pnferreira/automl_vae_shower_simulation/blob/main/AutoML%20for%20fast%20simulation%20-%20SFT%20Group%20Meeting.pdf

In this project, we focus on shower simulation in the calorimeter. During the particle collision process, an incoming particle hits the calorimeter and generates secondary particles. Those secondary particles form a shower, which is a cascade of energy deposition along the calorimeter layers. For a simulation of this process, one shower in a layer can be seen as an image by the computer. And the energy value deposited in each cell, which would represent the value for the pixels. The main method currently used for simulating showers is based on the Geant4 simulation toolkit. This method is computationally expensive in terms of CPU resources.

Some approaches were developed considering the need for fast and large scale simulated events, called Fast Simulation. One of those approaches is based on Machine Learning (ML) algorithms [2]. These ML algorithms work by learning to improve the performance based on the data provided. The algorithm optimizes its parameters with training and producing a fitted model.

A generative model (GM), a type of Machine Learning model, learns to approximate the true data distribution of the training data [4], and that can be used to generate new data. The most famous GM approaches are Variational Autoencoder (VAE) and Generative Adversarial Networks (GANs). In this project, we use a VAE for shower simulation, as inspired from previous work, such as in [2].

Generative model for simulation, or any kind of machine learning model, relies on variables to control the learning process. Those variables are called hyperparameters (hp), and they can heavily influence on the performance of the model. The tuning - picking the best values for the hp - is usually done in a manual process. Automated machine learning techniques (AutoML) optimize the hp tuning automatically using a metric. In the context of shower simulation, this project is a first attempt to use AutoML to tune the parameters of a generative model.

In this project, we investigate: (i) how to implement a VAE and AutoML tuner for the shower simulation task; (ii) how to better define a metric to choose the best model in terms of the physics and ML properties of the simulation; (iii) what are the techniques to do the tuning process in AutoML and how they compare between the different AutoML approaches to each other.

2. VAE for Shower Simulation

2.1. Definition

Deep Learning (DL) is a subfield of ML that uses neural network architectures. These networks are brain-inspired, and contain layers of neurons or nodes connected with each others. The learning process in neural networks refers to updating parameters of the model using the data and by optimizing an objective function. The term deep in DL refers to using multiple layers in the network, which progressively extract features from the input data.

An autoencoder is a model based on a DL architecture that learns to reconstruct its input [4]. This learning is achieved by first reducing the dimensionality of the input into a latent representation

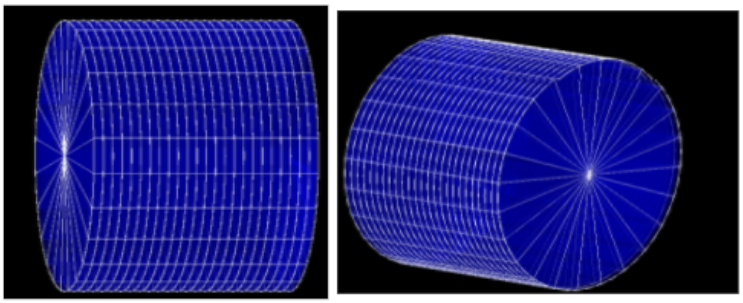

Figure 1: Calorimeter used in the project

and then reconstructing it back. The autoencoder is composed of two stacked neural networks: (i) an encoder: the network that encodes the input into the latent representation; (ii) a decoder: the network that reconstructs the input from the latent representation. The autoencoder is trained to minimize the error of the reconstruction. It is commonly used in applications for denoising data and dimensionality reduction, for example.

The model explored during this project is a Variational Autoencoder. It is an autoencoder which latent space distribution is regularized during training to guarantee that its latent space has specific properties to allow the generation of new data. In the VAE the distribution learnt in the latent space is constrained to be gaussian. Once it is trained, the decoder alone can be used to generate new events.

2.2. Implementation

The project started with a baseline implementation of a VAE model for shower simulation. The implementation was in Python and Tensorflow 2.0, using Keras as backend.

The model is tasked to learn to simulate the energy deposited in the cells of a calorimeter, which can be seen in Figure 1. In this project, we consider a lead tungstate (PbWO4) calorimeter with 24x24x24 cells segmented into radial (r), azimuthal angle (ϕ) and layers (z). To train the model, 10.000 full simulated events (or showers) with Geant4 are used. The incident particles creating the showers have an energy of 60 GeV and a direction perpendicular to the calorimeter. The full simulated events are stored in a HDF5 file. The energy depositions of the cells are scaled to be in the range [0,1] and reshaped into a 1D vector of size 13824.

In the baseline model, the encoder and decoder networks are composed of 2 dense layers with 500 and 200 nodes, both with batch normalization. The encoder learns the mean and variance of the input, which represents the parameters of a Gaussian distribution in 10 dimensions. After, it generates a vector of 10 dimensions from that distribution to represent the input, which is the latent vector. This latent vector is then passed to the decoder layers to reconstruct the input. The validation of the model is implemented in two terms: calculating the KL Divergence between the encoding distribution and the gaussian distribution and the cross-entropy loss function between the

input and output.

Figure 2 represents the VAE model with both the encoder and the decoder networks. The input is the data from the showers and the noise, it goes to the encoder, and then to the decoder, constituting one network for training.

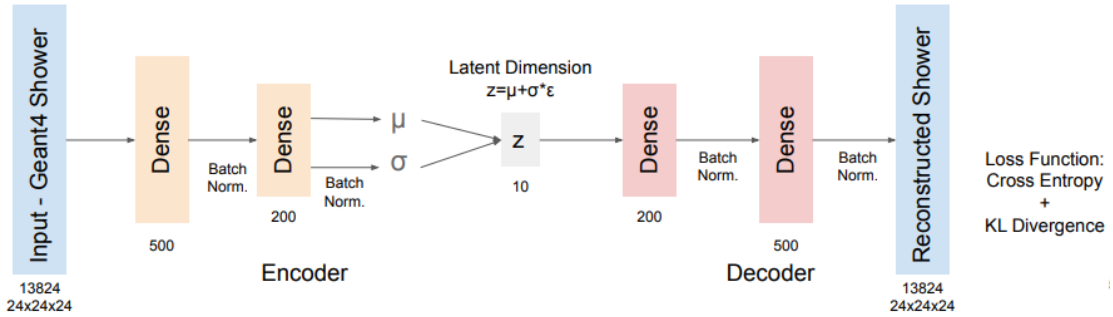

Figure 2: Schematic representation of the baseline VAE model

The model is fit/trained, using the validation error, to update the weights/parameters of the layers of the model accordingly. The trained model is saved for later use and comparison.

3. Hand tuning of the hyperparameters

Hyperparameters (hp) are parameters of the model that determine how the learning process is going to be held. The selection of these hp can have an impact on the performance of the model.

Some hyperparameters are:

- **Number of epochs:** number of times that the entire training set is passed through the network.
- **Batch Size:** number of training samples that are passed through the model before the trainable parameters are updated.
- **Activation Function:** it defines the output of a node given an input or set of inputs.
- **Optimizer:** the way the model try to adjust the parameters to converge into a best or close to best model.
- **Learning rate (lr):** determines the step size at to optimize the loss function.
- **Kernel initializer:** the way the weights of the layers are initialized.
- **Bias initializer:** the way the value for the bias is initialized.
- **Number of layers:** the amount layers there are in the encoder and decoder.
- **Number of nodes:** the number of units/dimensions/nodes in each layers.

The baseline hyperparameters used for this hand tuned implementation are: batch size (100), dimension of the dense layers (200, 500), dimension of the latent dimension (10), number of epochs (50), activation function for the dense layers (relu), activation function for the output layer (sigmoid), weight for the KL divergence (0.5), optimizer (Adam), learning rate (0.001), kernel initializer (Random Normal) and bias initializer (Zeros), and the number of dense layers (2).

The goal of the hand tuning phase is to analyze how each hyperparameter is impacting the performance of the model, and what is the range of the best values. For that, we train models changing the value of one at a time, starting from the baseline hyperparameters. The models trained with these different sets of hyperparameters are then compared using the Structural Similarity Index Metric (SSIM), which calculates how similar two events are giving a value from 0 to 1, with 1 being 100% similarity and zero, no similarity. This is done to analyse the performance of the VAE model on reconstructing unseen data (test set).

The hyperparameter that had the most impact on the performance of the model, is the number of epochs, because if the number is too small, it cannot be properly trained, if it too large, it can cause overfitting. The batch size, optimizer, learning rate and number of layers also had a significant impact.

4. Automated Machine Learning

Automated Machine Learning (AutoML) is an approach to automatically search for the best set of hyperparameters according to a certain metric. Moreover, it has the advantage of changing and comparing more than one hyperparameter at the same time.

AutoML uses the concept of trials, where at each trial, a set of hyperparameters is picked from a range. The model is then trained using these hyperparameters, and a score is computed based on the specified metric. After running a number of trials picked by the user, the scores for the models generated in each one are compared and ranked, from which we can have the model with the best score and set of hyperparameters.

In AutoML, different techniques are used to select the hyperparameters for each trial: random search, bayesian optimization and the hyperband. The output of each technique is the trials, from which the best model can be extracted.

4.1. Random Search

The Random Search receives as an input: the model, the number of trials we want to run, the range of each hyperparameter to pick from, and the metric used for scoring the trials at the end. As the name indicates, it randomly picks a new set of hp at each trial, trains the model using them, calculate a score using the input metric and compare the score to previous trials.

4.2. Bayesian Optimization

The bayesian optimization [1] is a technique to improve the tuning process by taking into consideration scores from previous trials to select the best hp for future trials. It works by approximating a function to calculate the probability of getting a specific score given a set of hyperparameters, called the objective function.

It has the same input as the random search, and the tuning process works as explained in Algorithm 1:

Algorithm 1 Bayesian Optimization

1: At first, pick a random set of hyperparameters and calculate the score;
2: Approximate the objective function with a gaussian process from the values of the scores and set of hyperparameters calculated from previous trials;
3: Predict the score of N random sets of hp with the approximated objective function;
4: Get the set with the best score from this prediction;
5: Build a trial using this set and train a model with it;
6: Use this trained trial to better estimate the objective function;
7: Repeat from step 2 until the maximum number of trials is reached.

The set that have the best predicted score using the approximated objective function represents the set with the most potential to generate a model with a good score during the trial. At each trial, then, the objective function is approximated better, and the algorithm tends to find best values for the hyperparameters.

4.3. Hyperband

This technique focuses on speeding up the random search through adaptive CPU resources allocation and early-stopping of the training [5]. It improves the random search by exploring a bigger space in less time. This is done by running on few epochs the first time and keeping only, based on the score, the best trials to train for a longer number of epochs.

The input is the same as the random search, but instead of the maximum number of trials, we input the maximum number of epochs to train (m) and a factor (n) with which the epochs are increased. The hyperband tuning process works with trials, rounds and brackets, as seen in the schema of Figure 3.

The round is a step at which all trials run for the same number of epochs. Each round, the best trials are selected from the previous round and trained for a number of epochs n times bigger than the previous.

The bracket is a step that encompasses the rounds that go from a minimum number of epochs to the maximum. At the start of a bracket, new random sets are chosen and, when the number of epochs reach the maximum set by the user, the bracket is over. The minimum number of epochs, in each bracket also increases by the factor n. The tuning process is over when the minimum number

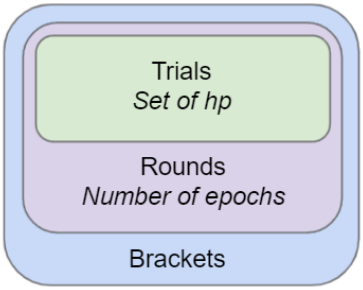

Figure 3: Hyperband Scheme

of epochs on a bracket is the same as the maximum number of epochs.

The hyperband tuning process works as in Algorithm 2:

Algorithm 2 Hyperband

Set the number of brackets as $log_m n + 1$
for each bracket j **do**
    Set the number of rounds as $j + 1$
    for each round i **do**
        If round is 0, pick n^j random sets
        Train n^j/n^i sets on $n^{i+log_m n-j}$ epochs
        Choose the best sets to be used on next round
    end for
end for

Figure 4 shows the evolution of the scores of the trials when using the hyperband. And black lines show the division of the brackets.

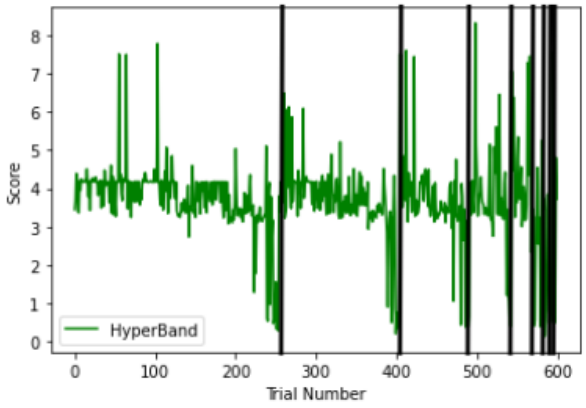

Figure 4: Score of trials of the hyperband - Visualization

5. Implementing the AutoML

The implementation of AutoML for the shower simulation [1] uses of the Keras Tuner library [2]. It consisted in defining the VAE, the tuner - which is the class that defines how to perform the tuning of the hyperparameters -, and performing the search for the better set of hyperparameter according to a chosen metric.

The first step was to define a class of a VAE model. This was based on the autoencoder baseline class available on the book AutoML in Action [7]. This autoencoder baseline code was converted into a VAE. The conversion included setting the hyperparameters we wanted to optimize, modifying the encoder and decoder functions, defining the layers for the latent space, defining the loss function within the class and defining the metric function for the tuning process. We also had to modify the function to build the model to use the values of the optimizer and learning rate selected on each trial.

The second step was to define the tuner. In Keras Tuner, there's a tuner implemented using the random search, bayesian optimization and hyperband techniques. We created a class inheriting the tuning process from one of them, and redefining the function used to run each trial. The function we created was to include the possibility of inserting the epoch and batch size as hyperparameters in the model and to personalize the metric used by the tuner to calculate the scores.

The third step was to run the tuner. The tuner chooses a set of hyperparameters at each trial and fits the model using them. Then, the score is calculated and the trial is saved. At the end, the scores for each trial are compared and we can pick the model with the best score, and see the set of hyperparameters used.

6. Validation Metrics

The metric is defined to access the perfomance of the model. In the context of this project, it is important to consider the ML metrics, which are used to guarantee a good reconstruction performance of the inputs. It is also important to guarantee a good performance of physics quantities when simulating new events.

We can combine the ML and physics properties to build a metric to choose the best performing model by the tuner. The ML and physics properties are:

Reconstruction: this measure consists in using unseen data, passing it through the encoder and decoder and generating an output. Then, comparing it to the input. In this case, we use the metric of the SSIM.

Gaussianity of the latent space: this measure is the distance between a gaussian distribution and the learned latent space distribution. This computation was performed using the mean squared

[1] https://github.com/pnferreira/automl_vae_shower_simulation
[2] https://keras.io/keras_tuner/

	Random Search	**Bayesian Optimization**	**Hyperband**	**Hand Tuned**
Number of epochs	150	90	64	1000
Batch size	90	50	60	100
Activation Function	Softmax	Relu	Softplus	Sigmoid
Optimizer	Nadam	Adam	RMSprop	Adam
Learning rate	0.001	0.01	0.0001	0.001
Kernel initializer	Random Uniform	HeNormal	LecunUniform	RandomNormal
Bias initializer	Glorot Normal	HeNormal	HeUniform	Zeros
Number of layers	3	4	3	2
Number of nodes	250, 200, 750	100, 100, 50, 100	100, 250, 500	200, 500
Number of trials	250	250	610	130
Time to run all trials	10h50min	13h14min	7h34min	3-4 days
Score	0.13421	0.18003	0.13950	0.26573

Table 1: Comparison of the best model for each technique

error (MSE).

Total energy deposition: this measure consists on the value of MSE of the total energy distribution of the Geant4 data and the VAE generated distribution.

Energy per layer: this measure is the mean of the MSE of the total energy for each layer distributions of the Geant4 data and the VAE generated.

A model can satisfy one property better than the others, and it is important to build a metric that takes all of them into consideration. Then, when evaluating the model, the model with the best score will do equally well in the ML and physics properties, and will be the best one overall.

In the implemented model for the AutoML, we created a metric that combined the four properties using the same weight and order of magnitude and the score was the sum of them. The best model was the one that minimized this metric.

7. Results of the AutoML for Shower Simulation

In this section, we present and compare the best models generated with the AutoML using the three different techniques - random search, bayesian optimization and hyperband - and also with the hand tuned model. Table 1 presents the set of hyperparameters on the models, and the score using the combined metric defined in the section 6.

Figure 5 shows the SSIM metric, where the hyperband performs better, with the closest value to one, followed by the hand tuned and the bayesian. The random had a value close to 92% similarity, but it wasn't as good as the other ones.

Figure 6 shows that almost all dimensions are close to the Gaussian distribution. In dimensions 3 and 4, we can see that the random search, the hyperband and the hand tuned one had some

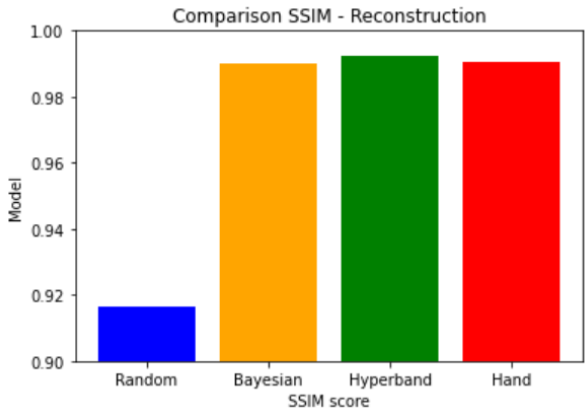

Figure 5: Reconstruction metric using the SSIM measure

disagreement with the gaussian distribution (z). This shows that the bayesian performed better using this metric.

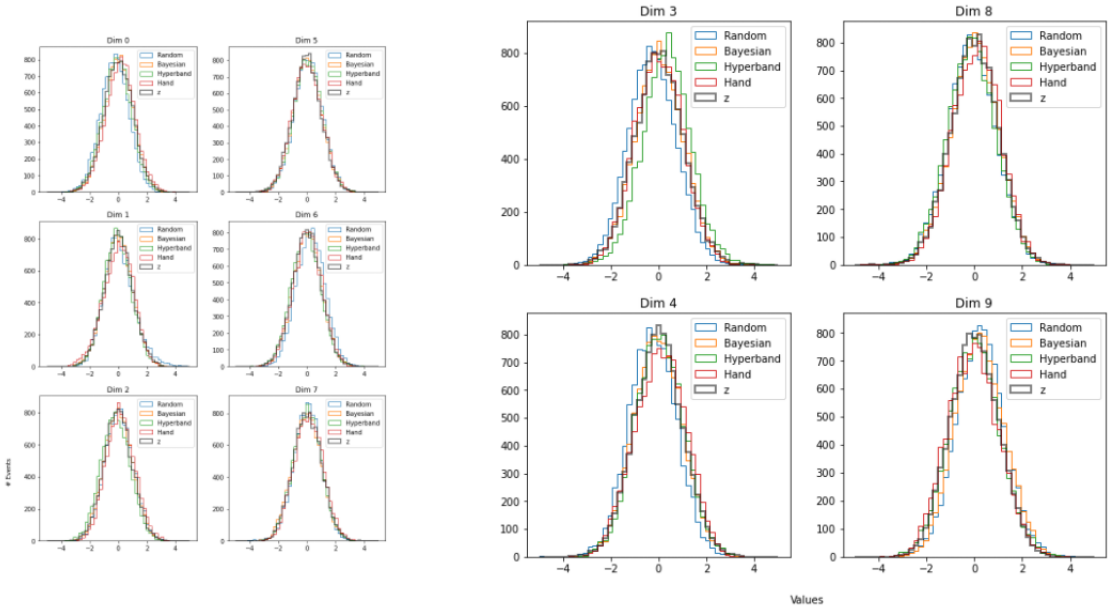

Figure 6: Gaussianity of the Latent Space

In the energy per layer metric in Figure 7, overall the hyperband performs better with a close agreement for almost all the layers, followed by the bayesian, and the hand tuned.

For the total energy in Figure 8, the random search performed better. The agreement is much closer compared to all the other approaches, and didn't generate events with an energy greater than the incident particle energy.

Overall the random search technique generated the best model. This can be seen using the score

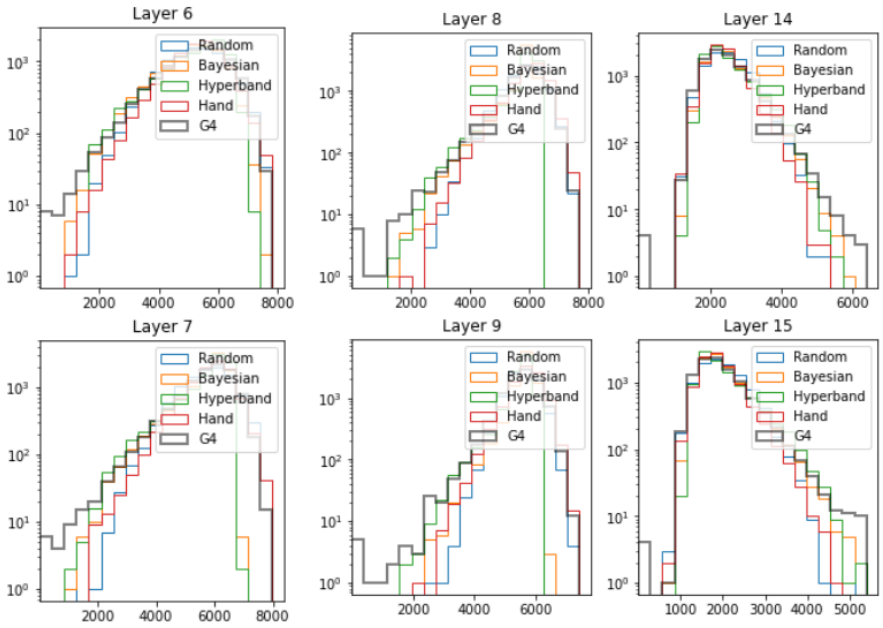

Figure 7: Energy per layer in 6 layers

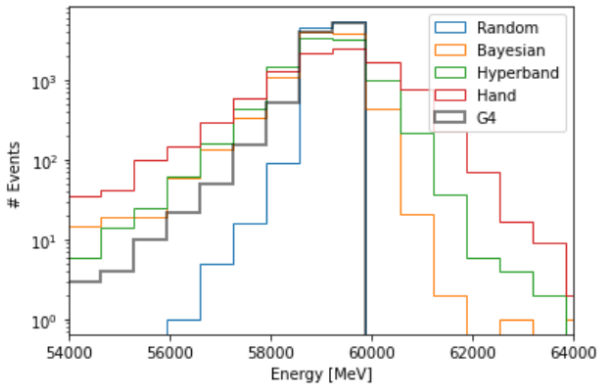

Figure 8: Total Energy

in Table 1, but also by the performance seen on the plots. And, even though not the best based the ML properties, it did well on the energy per layer and did much better on the total energy. On the other hand, there was the hyperband, which was very close to the random search considering the score, and then the bayesian. The poor performing model was the hand tuned. So, even though the model trained for longer (1000 epochs), it didn't have a better set of hyperparameters than the best models picked with the AutoML.

We tested the model training for a larger number of trials and epochs. The random search was trained with 100 to 500 epochs and 1000 trials. The best model did even better then the previous ones (score=0.04403). We also trained the hyperband for maximum of 512 epochs, instead of 128,

but no improvement was noticed. This might have been caused by the fixed learning rate of a value of 0.0001, since some good set of hyperparameters could have had a low score because they didn't improve fast enough in the first epochs.

Moreover, the hyperband in the first round trains the selected sets or trials for one epoch, and this can be a limitation and discarding potential trials very fast. In the Keras Tuner, that was not a customizable parameter. So we changed the library code to make it an input parameter to the tuner [3]. Moreover, we made the number of sets the first round picks an input parameter, meaning, how much of the space it would explore with the minimum number of epochs).

8. Conclusion

Monte Carlo simulated events are a large part of CPU consumption for the LHC experiments, and an alternative to that is to use fast simulation with Machine Learning. To build a good ML model, the hyperparameters need a tuning procedure. AutoML can lift the burden of a hand tuning process. This optimization can be done using various techniques, such as RandomSearch, Bayesian Optimization and Hyperband. Moreover, a metric is used to score the performance of the model with the different hyperparameter sets, and a combined metric (ML and physics) allows the tuner to choose the best considering all aspects of the problem.

Figure 9 shows the phases of this project, which consisted in: (i) developing a VAE model for shower simulation. (ii) Understanding how the hyperparameters impacted the model results, and hand tuning of the model. (iii) Implementing a model and tuner for different AutoML techniques with a VAE model (never tested before). (iv) Creating a metric to pick the best set of hyperparameters considered the ML and physics properties. (v) Comparing the results of the AutoML techniques.

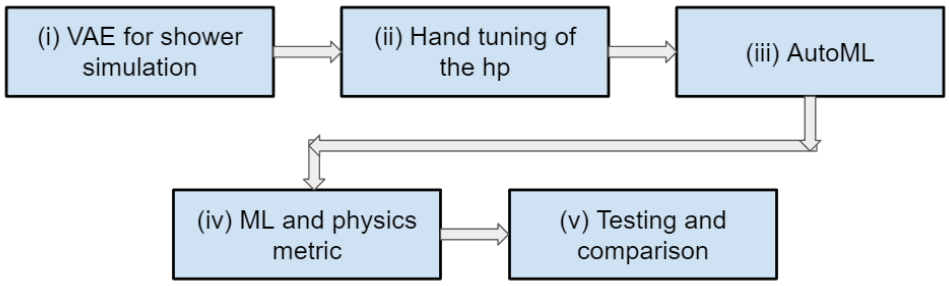

Figure 9: Phases of the project.

As a result for the specific data that we worked on, the three best models from the three tuning approaches did better than the hand tuned best one. The three different techniques to tune can be compared in terms of advantages and disadvantages in the Table 2.

[3] https://github.com/pnferreira/modified_hyperband_ktuner

	Random Search	**Bayesian Optimization**	**Hyperband**
+	Simple	Tracking of the tuning process Approximate objective function for fast scoring	Fast Explore larger space Tracking of the tuning process
-	No control of the tuning process	Too long to approximate a good enough function	Discard some trials too fast

Table 2: Comparison of the tuning techniques

The algorithm developed and analyzed in the project can be expanded to more complex problems in HEP. Besides that, other models can also benefit from AutoML, and also other areas that uses Machine Learning models.

References

[1] Jason Brownlee. How to Implement Bayesian Optimization from Scratch in Python. https://machinelearningmastery.com/what-is-bayesian-optimization/, 2019. [Online; accessed 23-Sep-2021].

[2] Kyle Cranmer, Stefan Gadatsch, Aishik Gosh, Tobias Golling, Gilles Louppe, David Rousseau, Dalila Salamani, and Graeme Stewart. Deep generative models for fast shower simulation in atlas. 2018.

[3] Davide Constanzo James Catmore. ATLAS Computing update - WLCG referees meeting. https://indico.cern.ch/event/754731/contributions/3127500/subcontributions/262952/attachments/1802165/2939783/LHCC_20100225_Costanzo.pdf, 2019. [Online; accessed 23-Sep-2021].

[4] Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

[5] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[6] Elizabeth Sexton-Kennedy. Hep software development in the next decade; the views of the hsf community. In *Journal of Physics: Conference Series*, volume 1085, page 022006. IOP Publishing, 2018.

[7] Qingquan Song, Haifeng Jin, and Xia Hu. *Automated Machine Learning in Action*. Manning, 2021.