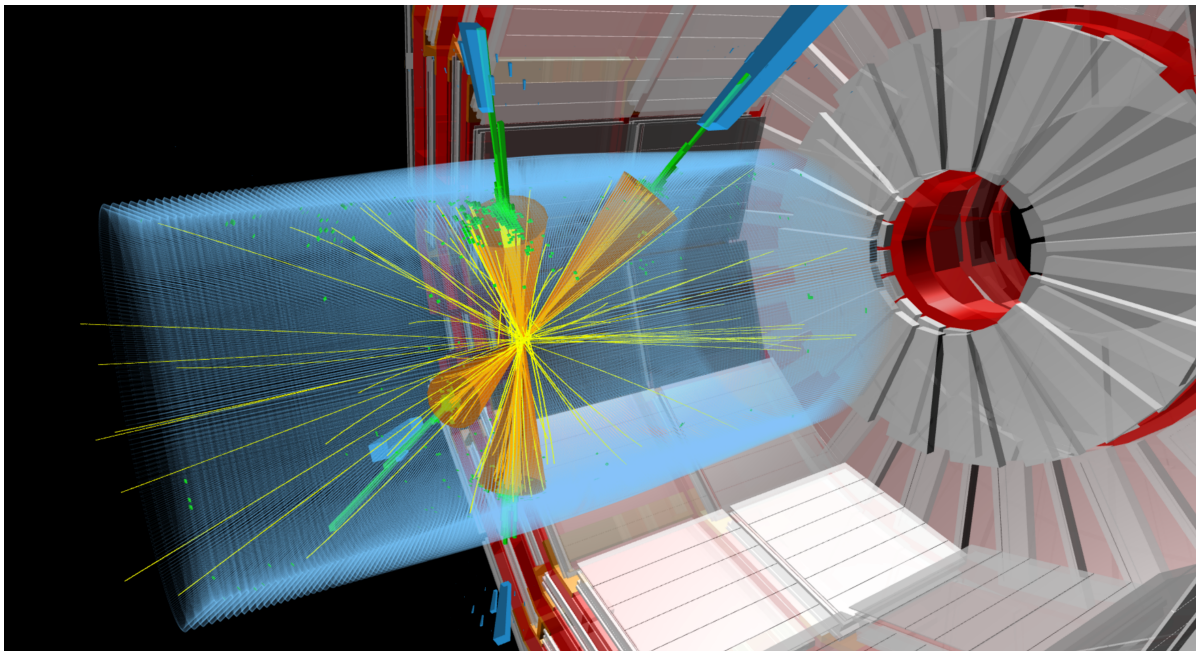
Hyperparameter Optimization for Jet Tagging



Didier Merk

Layout: typeset by the author using LaTeX.
Cover illustration: CMS detector (Mc Cauley, 2019)

Hyperparameter Optimization for Jet Tagging

Optimization studies on the Particle Transformer architecture used for Jet Tagging at CMS

Didier Merk
11037172

Bachelor thesis
Credits: 18 EC

Bachelor *Bèta-Gamma, major Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Dr. S. van Splunter

Institute for Interdisciplinary Studies
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

Semester 1, 2022

Abstract

The LHC particle accelerator collides protons at high energies, which produce heavy unstable particles that decay into lighter particle sprays known as jets. Determining which fundamental particle a jet originated from is known as jet tagging. Recent machine learning advances have allowed multiple new approaches for jet tagging, of which the current benchmark at the CMS detector is the Particle Transformer model. In this research an attempt is made to increase the accuracy of this model by performing a hyperparameter optimization study using a novel centralized machine learning platform created at CERN. First the original model is trained on the Kubeflow platform, and subsequently a hyperparameter optimization is executed. Due to memory- and execution time constraints the validation accuracy of the model is somewhat lower when trained on Kubeflow. However, the hyperparameter optimization results in six configurations of which two have higher accuracies than the previously trained model on Kubeflow. The results are promising and it is assumed more optimization is possible and further research is encouraged.

Acknowledgements

I would like to thank my supervisors at the CERN IT department, Ricardo Rocha and Dejan Golubovic, for their endless support and willingness to help. It has been a pleasure to work with both of you and I have learned more than I could have imagined. Another thank you to Jan van Eldik for giving me this very special opportunity, and setting up this internship.

Another thank you to Clemens Lange, who helped define the project of this thesis. Many thanks to Huilin Qu, one of the contributors of the Particle Transformer algorithm, for helping me with any questions I had regarding the code and physics aspects of the architecture.

In addition I would like to thank Daniel Holmberg for being my stand-in supervisor. Thank you for teaching me all about the physics aspects of the research and taking me under your wing during my time at CERN, and outside of it as well. My office mates, Robert and Spyros, thank you for making my time working with you so enjoyable. It was a pleasure. Özcan, Barnabas, Jay, Marina, Diogo, and Domingo thanks for all the inspiring conversations during lunch, the great experiences in OB and the nice climbing sessions.

Lastly, I would like to thank my family for reading through my thesis and listening to my ideas.

Contents

1 Introduction

The Large Hadron Collider (LHC) located on the border of Switzerland and France is the largest particle accelerator in the world. It is used by Particle Physicists to study the quantum fields of matter and forces of the universe. In the LHC accelerator, with a circumference of almost 27 kilometres, two beams of protons are accelerated to ultra relativistic speeds, before they are made to collide. Analysing these particle collisions helped to form, what is now known as: the Standard Model of particle physics. It is the model that summarizes our understanding of the forces and particles in the universe at a fundamental level.

When two protons produce a high-energy collision, new unstable particles can be created from the collision energy. These particles, in turn, decay and produce sprays of outgoing particles (Qu, Li, & Qian, 2022a). Detector systems such as ATLAS and CMS measure the trajectories and energies of these particle sprays and reconstruct a so-called event for each collision. The goal is then to identify the events that contain interesting physics processes, which for example lead to the discovery of the Higgs boson in 2012 (Aad et al., 2012). This thesis is written based on event data and models from the CMS-collaboration at CERN.

The sprays of outgoing particles that form an event are called *jets*, and a crucial aspect of the analysis process is identifying which fundamental particle the jet originated from. The classification of these jets is known as *jet tagging*. Based on a novel dataset consisting of over 100 million jets, researchers at CERN have recently proposed a new architecture to classify jets resulting from particle collisions, called the Particle Transformer (Qu et al., 2022a).

In this research an effort will be made to increase the accuracy of the Particle Transformer jet tagging architecture, by performing a hyperparameter optimization study. This will be done using a centralized machine learning platform launched by CERN in 2021 (Golubovic & Rocha, 2021). This is a service that was designed due to rising number of machine learning and deep learning applications in physics research.

In chapter 2 the theoretical background necessary to perform the hyperparameter optimization on the Particle Transformer algorithm will be described. It will explain concepts

such as the standard model, jet tagging and hyperparameter optimization. In chapter 3 the resulting research question will be defined, and the methodology of the research will be explained. Subsequently, in chapter 4 the results of the research will be presented and analysed. Finally chapter 5 and 6 include the discussion and conclusion.

2 Theoretical background

2.1 Particle Physics at CERN

Discovering what the fundamental building blocks of the universe are is a fundamental branch of physics called High-Energy Physics (HEP). For approximately a century physicists have been building machines known as particle accelerators, which force collisions between particles at high energy levels and help us get a deeper understanding of the world at the smallest scales.

2.1.1 The Large Hadron Collider

The largest and most recent of these high energy particle colliders is the *Large Hadron Collider* (LHC) on the border of Switzerland and France (Brüning, Burkhardt, & Myers, 2012), constructed by physicists working for the international research institute CERN. It consists of two particle beams that go through an underground tunnel of 27 kilometers in circumference, in which protons collide with each other in four different interaction zones. The tunnel was originally constructed for the Large Electron Positron Collider (LEP), an electron-positron particle accelerator that was designed for precision experiments.

The goal of the LHC is different and according to Brüning, Burkhardt and Myers (2012) can be described as a discovery machine. It uses the collisions of hadronic particles, proton-proton collisions to be precise, to explore new and heavier particles such as the Higgs boson.

2.1.2 The Standard Model

Our current understanding of the fundamental building blocks of the universe can be summarised in a theory known as 'The Standard Model' (figure 1). It has been developed by physicists over the last century and has proven successful in predicting and describing particles and their interactions (van der Poel, 2012). Robert Oerter in his book sketches the Standard Model as 'The Theory of Almost Everything', as it gives a quantum description of three of the four fundamental forces of nature: the electromagnetic force, the strong nuclear force and the weak nuclear force. The only fundamental force that remains unexplained at a quantum level in the Standard Model is gravity (Oerter, 2006).

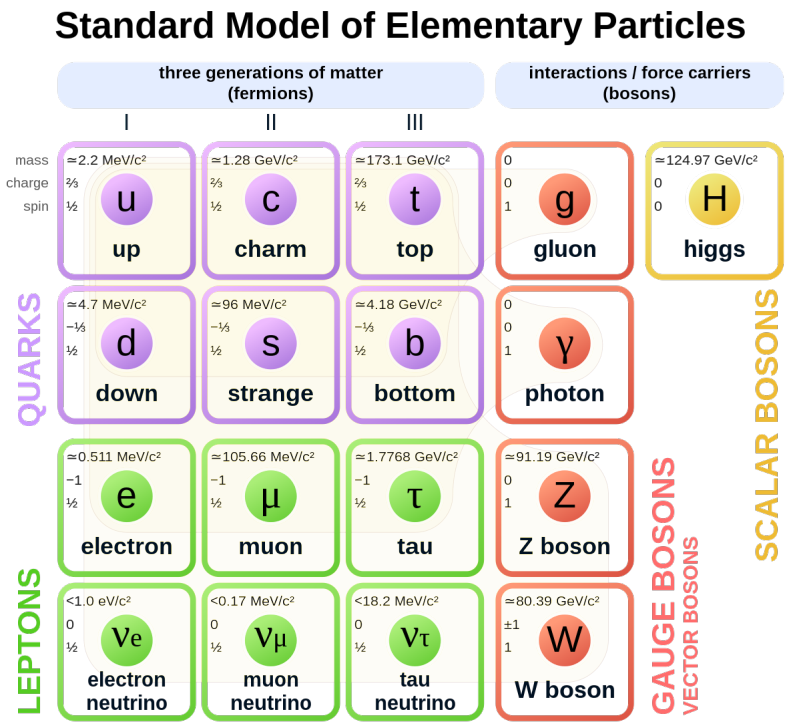


Figure 1: A schematic overview of the current Standard Model. Included are the fundamental particles of the universe: the twelve fermions, consisting of quarks and leptons, and five bosons. The brown loops describe which of the bosons pair up with which of the fermions (Workman, n.d.).

The fundamental building blocks of matter are called the *fermions*, and are grouped under quarks (purple in figure 1) or leptons (green in figure 1). A proton for example is composed of two 'up quarks' with a charge of $\frac{2}{3}$ and one 'down quark' with a charge of $-\frac{1}{3}$.

These fermion particles can interact when they are coupled with one of the carriers of the fundamental forces. These carriers, or *bosons*, can be divided in two groups: the gauge bosons (red in figure 1) or the scalar bosons (yellow in figure 1). The photon is the boson that carries the electromagnetic force, the gluon corresponds to the strong force and the Z- and W boson to the weak force (van der Poel, 2012). The brown loops in figure 1 demonstrate which elementary particles can pair up with which of the bosons.

As opposed to the LEP, which worked at lower energy levels, the main goal of the LHC is to explore the Standard Model in the TeV range, in the search of new physical processes

or fundamental particles. The way this is done is by producing very heavy particles (Top Quarks and W-, Z- and Higgs-bosons) in the particle accelerator and studying the decay rates of these unstable particles. The observed decays are compared to the predictions of the Standard Model to test whether the known forces describe the data. Physicists hope to find a deviation from the Standard Model expectation, which would be a discovery of a new particle or force.

2.2 Detectors and collisions

The Large Hadron Collider at CERN consists of two beams where protons will be accelerated and made to collide at an energy of 7TeV per beam, resulting in a 14TeV proton-proton collision. The beams cross at four major interaction regions, where the particles are made to collide and the products of these collisions are detected by four detectors: Atlas, Alice, CMS and LHCb. Experiments such as Atlas and CMS have similar research goals, however they use different technological approaches (Chatrchyan, 2008). This thesis will use data generated from the CMS detector, therefore a deeper background of this experiment will be given.

2.2.1 The CMS detector

The Compact Muon Solenoid (CMS) detector is one of the 4 detectors of the LHC at CERN. As a general-purpose detector, its goal is to find any new physics phenomena coming from the LHC.

The name comes from the fact that, despite the large amount of detector material it contains, at 15 metres high, 21 metres long and a weight of 14,000 tonnes it is still relatively *compact*. In addition, it is specifically designed to accurately detect the lepton particles known as *muons* (see figure 1). Finally it consists of a solenoid magnet able to generate a magnetic field of 4 Tesla, which is about 100,000 times the strength of the magnetic field of the earth.

When two protons collide, new unstable particles are created that decay and produce sprays of outgoing particles. To measure and identify these sprays of outgoing particles, the detector consists of multi-layered subdetector systems each responsible for the tracking of different particles, demonstrated in figure 2. Together the tracking system consists of

over 135 million electronic sensors covering the area of a tennis court (Chatrchyan, 2008).

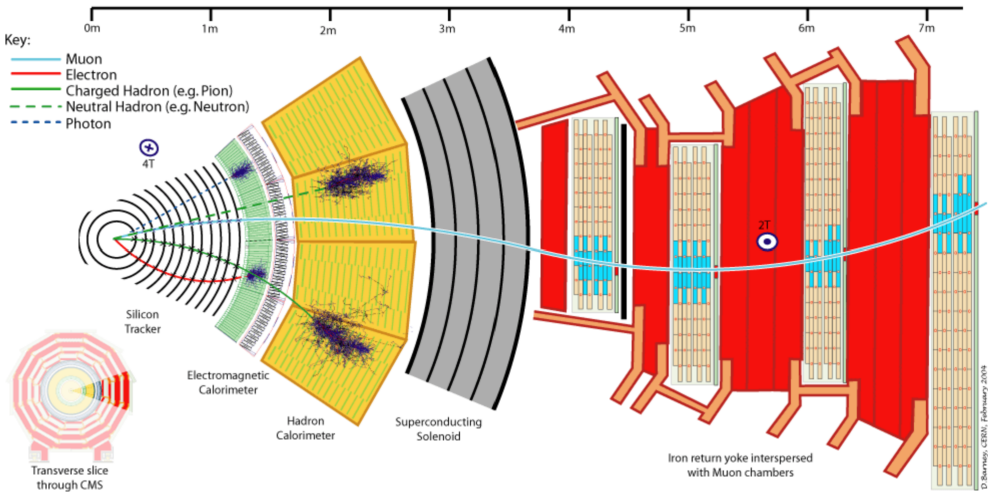


Figure 2: A slice showing the cross-section of the CMS-detector and its multiple layers. Demonstrated are where muons, electrons, charged- and uncharged hadrons and photons are being detected (Davis, 2016)

.

The most inner layer is the silicon tracker, which tracks the path of charged particles. The following layer is the electromagnetic calorimeter or the ECAL-detector, which uses scintillation light to detect electrons and photons and their energies. The hadron calorimeter or the HCAL is the next layer and, as the name suggests, detects both charged and uncharged hadrons. The largest layer of the detector is the part dedicated to detecting muons. This is due to the fact that muons are highly penetrative particles (Lipari & Stanev, 1991) and can pass meters through iron walls, however leave a trail of ions as they pass through the muon chambers.

2.3 Jet tagging

With the help of all the tracking systems, the detector measures the positions, trajectories and energies of these particle sprays and reconstructs the *event* for each collision. It basically functions as a camera taking 40 million 3D photographs per second, capturing the result of a particle collision. The goal is then to identify the events which include an interesting physics process, for example the production and decay of Higgs bosons in 2012

(Aad et al., 2012).

The spray of outgoing particles that form an event are called *jets*, and a crucial aspect of the analysis process is identifying which fundamental particle the jet originated from. The classification of these jets is known as *jet tagging*. Traditionally, jet tagging was done by applying cut-based selection criteria, inspired by the laws of quantum chromodynamics (QCD) (Larkoski, Moult, & Nachman, 2020). The recent rise of machine learning, however, has allowed multiple new approaches of jet tagging.

2.3.1 Jet representations

Jets can be described in different ways, such as images, sequences, trees, graphs, or a set of constituent particles. The image-based representation is based on information obtained from calorimeters, which measure the energy deposition of a jet. In this method, each pixel of the calorimeter is coloured according to its energy deposition, which creates an image for the event (figure 3). This representation has been extensively studied, and using a Convolutional Neural Network, a significant improvement in jet tagging performance was found over the traditional cut-based approach (Cogan, Kagan, Strauss, & Schwartzman, 2015). However, it is hard to store additional information about particles in the image and the approach is computationally highly inefficient, since by far the most pixels of the images will be blank (Qu & Gouskos, 2020).

Another way to describe jets is as a collection of particles, which is more natural to its physical representation (figure 3). A collection of particles, however, is a general term, not immediately applicable in a machine learning dataset. A better fitting data structure could be a list or a binary tree. These require a manually imposed order to the particles, which in practice can turn out to be inaccurate (Qu & Gouskos, 2020).

2.3.2 ParticleNet

In recent research a new deep-learning approach for jet tagging is proposed by introducing a novel way to represent jets. As opposed to looking at jets as an ordered structure, such as a list or a binary tree, they can be viewed as a ‘particle cloud’ (figure 3). A structure that resembles the 3D ‘point cloud’ representation often seen in computer vision, where all points have coordinates and are themselves unordered. Using this novel representation

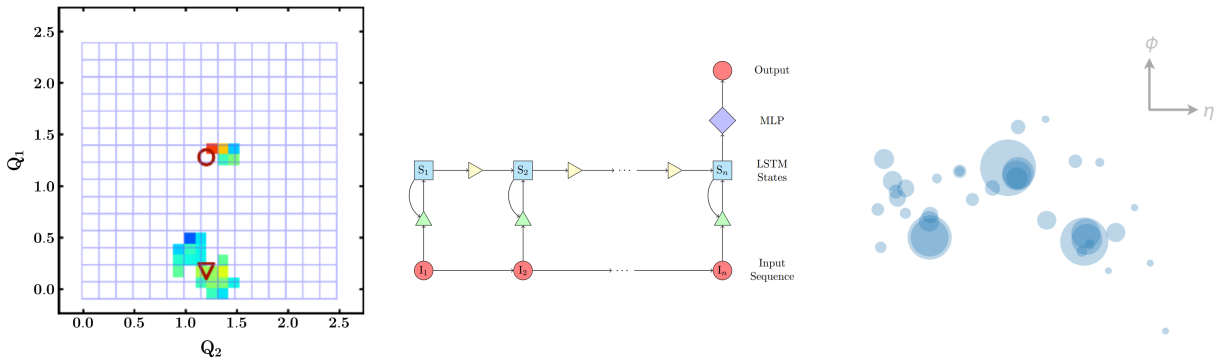


Figure 3: Three different jet representations. Demonstrated on the left is an image based representation of a jet (Cogan et al., 2015). In the middle a jet representation seen as a collection of particles with an ordered structure (Qu, 2019). On the right the most recent approach of viewing jets as particle clouds (Qu, 2019).

of jets, a customized neural network called 'ParticleNet' was designed. This ParticleNet algorithm was found to be significantly more accurate than all previous jet tagging methods (Qu & Gouskos, 2020).

ParticleNet is based on the concept of a Dynamic Graph Convolutional Neural Network, which is a form of a fast emerging deep-learning approach called Graph Neural Networks (GNN) (Wu et al., 2021). Deep learning has revolutionized many machine-learning tasks such as image classification or natural language understanding. However, in recent years there has been an increasing number of applications in which data is more complex, and it is represented in a graph with underlying relations and connections between so-called nodes. Consider for example the network of social media connections or a network of train stations. These Graph Neural Networks have formed the basis of performing more accurate machine learning tasks on data consisting of these graph representations, such as jets in the particle cloud representation.

2.4 Particle Transformer

Deep learning and multiple novel jet representations have improved jet tagging significantly in recent years, however, researchers in the CERN CMS collaboration found that a lack of a large public dataset has limited further improvement of jet tagging algorithms. Through

the use of *Monte-Carlo simulations*, mimicking real events, physicists at CERN have been able to find a solution to this problem. The result was a dataset called *Jetclass*, consisting of over 125 million jets for training, validating and testing of machine learning architectures (Qu et al., 2022a).

2.4.1 Monte-Carlo simulations

Monte-Carlo simulations have longer been used in high-energy physics to solve similar problems. The idea behind the simulations is to use randomness in a decisive manner and mimic operations of complex physical processes. A good example is throwing a coin thousands of times: when counting the number of heads and tails you can estimate the chance of each side fairly accurately.

The JetClass dataset is generated using the principles of Monte-Carlo simulations and consists of ten classes of jets, shown in figure 4. The production and decay of the Higgs-, W- and Z boson and the top quarks was simulated using a framework called MadGraph. The evolution of the produced particles (so-called hadronization) was simulated using Pythia. To make the experiment as realistic as possible, effects of the detector were also simulated using Delphes (Qu et al., 2022a).

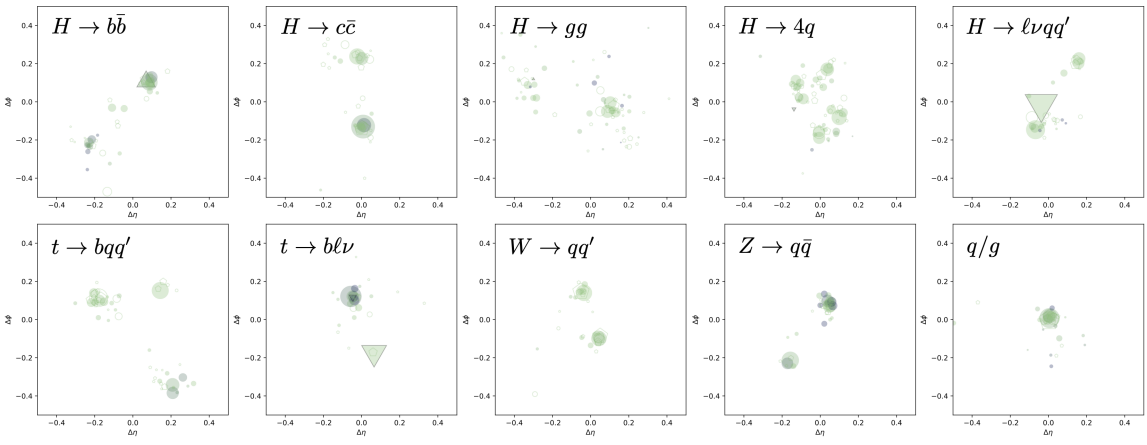


Figure 4: The different classes of jets simulated in the JetClass dataset, displayed as particle clouds. There are five different Higgs boson jets, as a Higgs boson can decay in multiple different ways. In the bottom left are the two different top quark jets and the remaining three are W- and Z boson jets and quark/gluon jets (Qu et al., 2022a).

2.4.2 Transformer Networks

In recent years *transformer* models have witnessed large successes in both Natural Language Processing and Computer Vision research. It was first proposed in 2017 by researchers at Google Brain as a model used for translation tasks (Vaswani et al., 2017). However, in recent years the design has slowly grown to be a universal architecture that has also proven to be powerful for fundamental scientific problems such as the predictions of protein structures (Tunyasuvunakool et al., 2021).

Similar to recurrent neural networks (RNNs), transformers are designed to process sequential data such as text or time series. The most accurate neural sequence models follow an 'encoder-decoder' structure. Here, an encoder maps a sequential input $x = (x_1, ..., x_n)$ to a sequence $z = (z_1, ..., z_n)$. The decoder then outputs a final sequence $y = (y_1, ..., y_n)$. These models are regressive and use the previously generated output as additional input when calculating the next. Unlike recurrent neural networks, however, transformers are able to pass the input sequence in parallel. In natural language processing for example, this means a transformer does not have to process the data one word at a time.

Transformers follow the same encoder-decoder architecture, which is illustrated in figure 5. Input data first passes through input embedding, where it is mapped to a point in space understandable by a computer, a so-called embedding space. In natural language processing, words may have different meanings depending on their place in a sentence. The positional encoder in the original architecture was a vector that gave context to a word based on its position.

Using the input embedding and positional encoding a vector can be generated that contains contextual information about the data and is understandable for a computer. This vector is passed into the encoder block (on the left in figure 5), in which it passes through a Multi-Head Attention and feed forward layer. This Multi-Head Attention layer is at the core of a transformer and allows it to process data in parallel as opposed to the sequential processing of RNNs. The concept of attention in machine learning can be seen as a way of giving context to input data, and it is a way of estimating which part of the input should be focused on.

The decoder block (on the right in figure 5) is similar to the encoder block. It has an

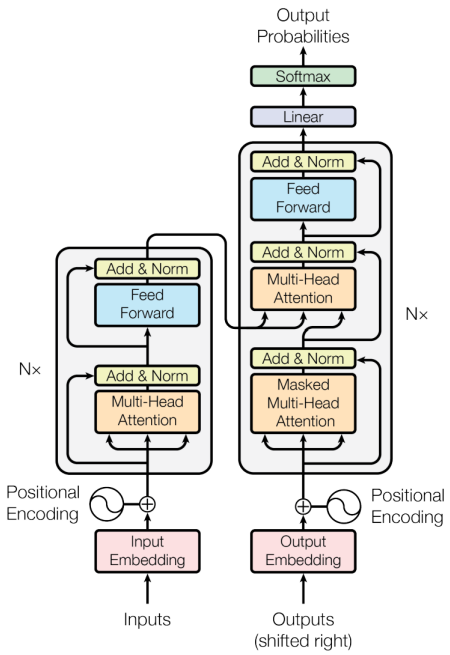


Figure 5: The architecture of the original transformer, designed by researchers at Google Brain in 2017. The left half demonstrates the design of the encoder block, and the right half shows its connections with the decoder block (Vaswani et al., 2017).

attention block that calculates the attention vector for the embedded output. These vectors are combined with the vector output from the encoder and passed into a second Multi-Head Attention block, referred to as the encoder-decoder attention block. This part of the architecture estimates how related each of the input data points are to the output data points. The following layers then transform the vectors into final probabilities.

2.4.3 Particle Transformer

Based on the notion of jets represented as particle clouds, the novel large JetClass database and transformer architectures, researchers at CMS have proposed a new jet tagging model called the 'Particle Transformer' (ParT), shown in figure 6. This architecture is found to be significantly more accurate than the previous ParticleNet and it is the current baseline for jet tagging (Qu et al., 2022a).

The ParT has two different sets of inputs: the particles, an $N \times C$ array consisting of C features, such as their energies and position, for a number of N particles in the jet;

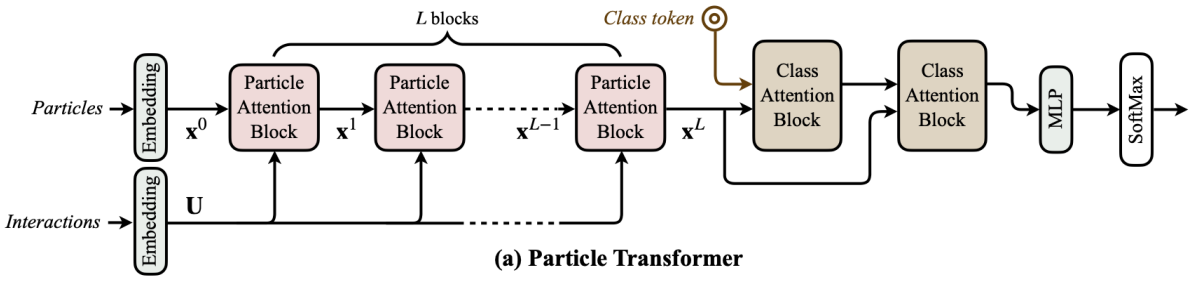


Figure 6: The architecture of the Particle Transformer, designed by researchers at Google Brain in 2017. The left half demonstrates the design of the encoder block, and the right half shows its connections with the decoder block (Qu et al., 2022a)

as well as the interaction features between pairs of particles, which is a matrix of shape $N \times N \times C'$. In every jet tagging task a number of kinematic variables, such as the energy-momentum 4-vector, $p = (E, p_x, p_y, p_z)$, are available for all the particles. This vector is used to calculate the four interaction features C':

$$\begin{aligned} \Delta &= \sqrt{(y_a - y_b)^2 + (\phi_a - \phi_b)^2}, \\ k_T &= \min(p_{T,a}, p_{T,b})\Delta, \\ z &= \min(p_{T,a}, p_{T,b})/(p_{T,a} + p_{T,b}), \\ m^2 &= (E_a + E_b)^2 - ||\mathbf{p}_a + \mathbf{p}_b||^2 \end{aligned}$$

In these equations y_i is the rapidity, a variable representing the force and momentum of a particle, and ϕ_i is the azimuthal angle, which is the angle between the particle and a reference plane. The transverse momentum of a particle pair is calculated as $p_{T,i} = (p_{x,i}^2 + p_{y,i}^2)^{\frac{1}{2}}$, using the momentum 3-vector $\mathbf{p}_i = (p_{x,i}, p_{y,i}, p_{z,i})$. To calculate the final four particle interaction features the logarithm is taken of Δ, k_T, z and m^2 (Qu et al., 2022a), to avoid long tail distributions.

Both the particle and interaction inputs are parsed through a multi-layer perceptron, to project them into a usable embedding space. Opposed to the standard transformer as described earlier, the ParT does not add any positional encoding, because its input data is not dependent on the position it is in. This might seem counter intuitive, but the spatial information, for example the flying direction, is directly included in the particle input.

The particle embedding is fed into a stack of 'L' Particle Attention Blocks (figure 6) similar

to the Multi-Head attention blocks explained earlier, and the interaction matrix is added to every Particle Attention Block as attention weights. The last particle embedding $\mathbf{x}^L$ is fed into two Class Attention Blocks, together with a class token that is used to extract final information from the particle embeddings. The class token is then passed into a final multi-layer perceptron and SoftMax function to produce the final jet tagging scores.

2.5 Machine Learning at CERN

In recent years machine learning has started to be used in the field of high energy physics, using it as an alternative to cut-based selections, in particular to detect anomalies in the search for unknown physics (Pol, Berger, Cerminara, Germain, & Pierini, 2020). The rise of machine learning also implied the development of libraries providing practical ways of implementing machine learning on datasets. Well known examples of these libraries are PyTorch, TensorFlow and scikit-learn. However, a standard procedure for high energy physicists to perform machine learning tasks on their data was still missing. Many data scientists, even in the same institution, use different ways to go from raw data to their machine learning algorithm, leading to underproductivity and suboptimal quality results.

2.5.1 Kubeflow

In 2021 researchers at the CERN IT-department developed and released a centralized machine learning platform for physicists at CERN to use, with the goal of improving the overall process of machine learning (Golubovic & Rocha, 2021). The service is based on Kubeflow, a free open-source machine learning platform offering components from the whole machine learning pipeline. It can load and pre-process data, perform distributed model training, serve machine learning models and perform automated *hyperparameter optimization*.

2.5.2 Hyperparameter Optimization

Each machine learning model has hyperparameters, a set of parameters that are not learned by the algorithm. Consider for example the number of neighbours in a K-nearest neighbour model, or the number of layers in a neural network. The process that is optimizing the value of these specific parameters is known as *hyperparameter tuning* or *hyperparameter optimization*. The optimization of these parameters can be important in various ways.

Most importantly, it improves the performance of a machine learning algorithm, by fine tuning it to a specific problem (Olson, Bartley, Urbanowicz, & Moore, 2016). In addition, it reduces the amount of human labour put into performing the machine learning and improves the reproducibility of a study.

In the case of the Particle Transformer used for jet tagging, hyperparameters are the variables such as the optimizer and inner optimizer used for the algorithm. Others are concepts such as the batch size and the learning rate used when training. However, no optimization has been performed for the hyperparameters used while training the ParT architecture, which will be the goal of this research.

3 Research and methodology

The focus in this chapter shifts from the more general theoretical background to the specific research that is addressed in this thesis. First, the research question is defined, after which the methodology and analysis method are presented.

3.1 Research question

From the theoretical background it can be concluded that there are possibilities to increase the performance of the jet tagging architecture currently proposed as the baseline for jet tagging in the CMS experiment. In this research a hyperparameter optimization study will be performed, with the goal of increasing the accuracy of the current Particle Transformer architecture and setting a new benchmark for jet tagging at CERN. The research question is as follows:

"What are the optimal values for the hyperparameters used in the Particle Transformer architecture used for jet tagging?"

To answer this, three sub-questions are formulated, which together will provide an answer to the above question.

1. What is the current accuracy of the Particle Transformer architecture when it is trained on a CERN machine learning platform?
2. What are the results of a hyperparameter studies on the Particle Transformer using Kubeflow's Katib?
3. What is the accuracy of the Particle Transformer, when it is trained using new optimized hyperparameter values?

The next sections include a detailed explanation of how the experiments answering these questions have been conducted.

3.2 Method

One of the goals of setting up the centralized machine learning platform at CERN was to improve the quality of machine learning research amongst physicists (Golubovic & Rocha,

2021). In this section a description of the experiment performed using this machine learning platform is given. The aim is to increase the reproducibility of the study and encourage researchers to explore and improve machine learning in particle physics even further.

The study is divided along the three sub questions: first, the implementation of the training of the particle transformer on CERN's centralized machine learning platform (Kubeflow); second, the use of the Katib tool available on this platform, to perform a hyperparameter optimization; finally, the comparison of the network with its optimized hyperparameters to the benchmark from the first part of the studies.

3.2.1 JetClass dataset

The first part of the study, reproducing the original training of the Particle Transformer on Kubeflow, will be described in the current and following two sections. The first step that needs to be taken is to transfer the dataset into a place where it can be used by the platform. Kubeflow has an integrated compatibility with EOS, a disk only storage technology created by the CERN IT department for LHC use cases (EOS, 2010).

The Particle Transformer repository on Github (Qu, Li, & Qian, 2022b) includes multiple datasets, among them also the large JetClass dataset containing over 125 million jets. The repository also contains a python file called `get_datasets.py`. Upon execution, this file downloads the chosen dataset from the CERN server to be stored on the user's home directory on EOS.

3.2.2 Memory

The novelty of the JetClass dataset partly comes from its large size, consisting over 125 million jets. The large number of jets in this dataset, combined with the new transformer architecture, result in a significant increase in jet tagging accuracy. However, the 175 GigaBytes of data also bring some memory issues when attempting to train the model on the Kubeflow platform, preventing the model from being able to be trained on a regular computer.

To deal with these memory constraints, a modified script is developed to be able to run the training, which can be found in the appendix section 7.1. It is based on the original script

presented in the Particle Transformer repository, however adjusted in multiple ways. An important consequence of this modification for this research, is that not all jet data will be used to train the model. The original shell script is adjusted to only load a fraction of the dataset, reducing the memory use from the original 175GB to 5GB.

The training of the Particle Transformer makes use of Weaver, a python package designed by researchers at CERN, which aims to provide a streamlined machine learning framework for High Energy Physics applications (Qu, Harju, & Li, 2020). In an attempt to deal with large datasets Weaver does not load all files into memory directly, but rather incrementally. When training a model, it is important to then properly mix different event types to improve the training performance and stability.

Weaver accomplishes this by putting all input files into `N` groups and then load them simultaneously with an `N` amount of 'worker threads'. The variable setting `N` is '`--num-workers`' and in this research will be set to `1`. Highlighted below is a fragment from the script made to run the training of the Particle Transformer on Kubeflow, which shows how the discussed `-num-workers` variable is declared.

```
dataopts="--num-workers 1 --fetch-step 0.01"
```

A different variable that is modified when training the model on Kubeflow for the first experiment is a variable called '`--batch-size`'. This hyperparameter can be described as the amount of training samples that are used in one single iteration to train the model. The `batch-size` will be reduced from `512` to `8` to save memory.

```
# PN, PFN, PCNN, ParT
model=$1
if [[ "$model" == "ParT" ]]; then
    modelopts="networks/example_ParticleTransformer.py --use-amp"
    batchopts="--batch-size 8 --start-lr 1e-3"
```

3.2.3 Convergence

The impacts on the accuracy of the jet tagging architecture, when the size of the dataset is reduced to a fraction of 0.02 of its original size, have been studied (Qu et al., 2022a). One of the observations is that the training was already converged after 100K iterations, compared to the 1M iterations required for the full dataset. Considering this, the variables `samples_per_epoch` and `samples_per_epoch_val` are changed from `1024000` and `12800` to `102400` and `1280` respectively.

As a result a large amount of memory and execution time is saved, that would otherwise have been spent on already converged training. For the same reason the amount of `epochs` is changed from `50` to `5`. An epoch can be seen as one full training cycle of the network, meaning the architecture will be trained on the data for 5 cycles.

```
epochs=5
samples_per_epoch=$((100 * 1024 / $NGPUS))
samples_per_epoch_val=$((100 * 128))
```

3.2.4 Katib

The second part of the research contains preparing and running a hyperparameter optimization job using the machine learning platform, which will be described in the current and following section. Kubeflow has a component called Katib that assists in hyperparameter tuning, which works by optimizing a target variable or *objective metric*. Often this objective metric is the validation accuracy, but concepts such as loss are often included.

Various hyperparameters can be specified, after which Katib will calculate the objective metric for different combinations of these hyperparameter values. The way Katib chooses the different values for the hyperparameters can be set using a search algorithm, such as grid search or random search. The way the tuning process works is straightforward: Katib runs multiple training jobs (called *trials*) in one experiment. Each trial tests a different set of hyperparameter configurations, after which Katib outputs the values for the hyperparameters and their corresponding objective metric (Kubeflow, 2021).

3.2.5 Pods and containers

The set up for the second part of the research is less straightforward than the set-up for the training of the architecture. This is because the hyperparameter tuning experiment that is executed on Katib runs as a so-called Kubernetes *pod*.

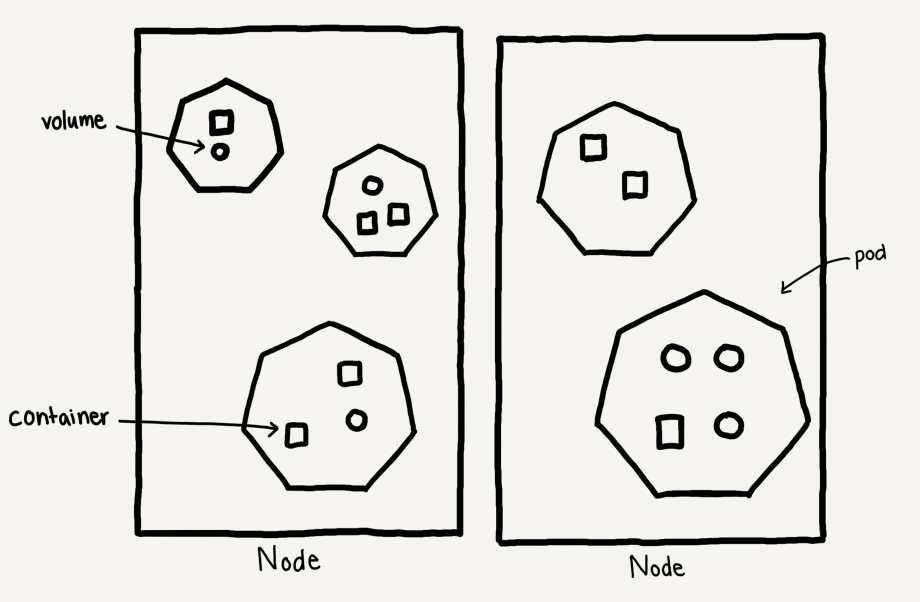


Figure 7: A schematic overview of the Kubernetes architecture. Multiple pods can run and communicate with each other in a node. The pods consists of software-packages called containers or volumes (Merk, 2022).

Kubernetes has a specific architecture (figure 7) where it consists of *nodes*, in which *pods* are running. Pods are the smallest unit of computing that can be created in Kubernetes and can communicate with other pods in the same node. These pods can contain one or more *containers*, which are ready-to-run software packages that contain all the information needed to deploy an application or, in this case, an experiment.

Pods can be created using a special type of file called 'yaml-files': a computer language specifically used for configuration and data storage or transmission. The yaml-file for this experiment can be found in the appendix in section 7.3.

A key component in the yaml-file in a yaml, is the *docker image* it uses. A docker image is a file that creates the environment of the previously mentioned container, and in principle

presents the pod with all its needs for the current experiment. For this experiment it was necessary to include a *kerberos secret* in the docker image, which makes sure that the data stored on EOS (behind a user-password) is accessible. Other packages that were required for the parameter tuning include: `weaver-core`, `pyarrow` and more standard libraries such as `numpy`, `pandas` and `scikit-learn`. The full list of libraries was read from the 'requirements.txt' file, which can be seen in the snippet of code below. The complete docker image used for this experiment can be found in section 7.2.

```
COPY requirements.txt .
RUN export PATH=/home/$NAME/.local/bin:$PATH && \
    pip install -Ur requirements.txt --no-cache-dir
```

Another key-component in the yaml-file is the declaration of the objective metric, hyper-parameters and search space. These define the metric that will be optimized and the parameters used in this optimization process. In this experiment the objective metric is the validation accuracy (`Acc`), and the goal is to maximize it. The additional metrics that are outputted are the `AvgAcc`, `Loss` and `AvgLoss`, but these are not the variables that are being optimized.

```
  objective:
    type: maximize
    objectiveMetricName: Acc
    additionalMetricNames:
        - AvgAcc
        - Loss
        - AvgLoss
```

The three main hyperparameters of this algorithm that will be optimized are the following:

1. `-start-lr`: The start learning rate variable, with a search space from `5e-4` to `1.5e-3`
2. `-batch-size`: The batch size integer variable, with the search space ranging from `1` to `64`.

3. `-optimizer`: The categorical optimizer variable, with the 3 possible options being: `adam`, `adamW` and `ranger`.

The learning rate parameter can be described as the size of the step taken in the direction of the gradient, in the process of minimizing a loss function during training. In this architecture the learning rate is constant for the first 70% of the iterations, after which it decays down to 1% of its original value. Secondly, the batch size can be described as the amount of training samples that are used in one single iteration to train the model. Finally, optimizer are the algorithms that change the attributes of a neural network, such as weights, in order to minimize the losses.

```
parameters:
  - name: --batch-size
    parameterType: int
    feasibleSpace:
      min: "1"
      max: "64"
  - name: --start-lr
    parameterType: double
    feasibleSpace:
      min: "5.0e-4"
      max: "1.5e-3"
  - name: --optimizer
    parameterType: categorical
    feasibleSpace:
      list:
        - "adam"
        - "adamW"
        - "ranger"
```

The way the different combinations of hyperparameter values are selected is defined by the search algorithm. In this research the `random` algorithm is chosen, which randomly selects values in the search space of the hyperparameters. In the timeframe of this research it is more effective to do a random search, as opposed to a grid search, in which every single

combination of hyperparameter values is trained.

```
algorithm:
  algorithmName: random
```

The last important element in the yaml-file is the request of resources, meaning: CPUs and GPUs. Multiple GPUs were made available and the amount that the experiment will use has to be declared in this configuration file.

To summarize, creating a pod with this yaml-file (section 7.3) and docker image (section 7.2) will start a hyperparameter optimization experiment that will optimize the validation accuracy. It will attempt this by trying different configurations of the hyperparameters, constrained by their respective search spaces. The set-up for the hyperparameter tuning using Katib is not fully completed in this research, such that instead a manual tool was written to perform the optimization, which can be found in section 7.4. However, this is essentially an identical procedure.

3.3 Result analysis

It is important to consider how the three sub-questions can lead to an evaluation of what the optimal values for the hyperparameters used in the Particle Transformer architecture are.

An important concept in evaluating the results is the *validation accuracy* of the model. This is calculated by dividing the correct number of predictions by the total number of predictions. This will output a number between 0 and 1, where 1 means flawless performance.

The validation accuracy of the Particle Transformer will be calculated during the training of the model and during the hyperparameter optimization. Afterwards, these accuracies will be compared to each other to analyze whether any improvement has been measured. These accuracies will also be compared to those of other models and the one in the original research as a benchmark.

4 Results

In this chapter the result of the machine learning experiment will be presented. In the first section the results of training the model on Kubeflow will be shown. The results of the hyperparameter optimization done using Katib is presented in the second section.

4.1 Particle transformer on Kubeflow

The first part of the research was dedicated to training the Particle Transformer model on the centralized machine learning platform at CERN. As explained in the previous section, due to memory constraints on the platform, some adjustments to the data and model training were made. The results of the training can be seen in table 1-3.

	$H \to b\bar{b}$	$H \to c\bar{c}$	$H \to gg$	$H \to 4q$	$H \to \ell\nu qq'$	$Z \to q\bar{q}$	$Z \to \nu\nu$	$W \to qq'$	$t \to bqq'$	$t \to b\ell\nu$
$H \to b\bar{b}$	**0.82342**	0.06718	0.01475	0.0117	0.0013	0.03147	0.00407	0.00266	0.04077	0.00268
$H \to c\bar{c}$	0.05608	**0.67693**	0.07452	0.07563	0.00342	0.03194	0.01729	0.01687	0.04652	0.0008
$H \to gg$	0.0304	0.04439	**0.71967**	0.11523	0.00053	0.02124	0.04215	0.0104	0.01575	0.00024
$H \to 4q$	0.00932	0.03469	0.13233	**0.73954**	0.00214	0.01519	0.00953	0.01534	0.04185	0.00007
$H \to \ell\nu qq'$	0.00104	0.00465	0.00033	0.00473	**0.93764**	0.00613	0.00104	0.00571	0.00162	0.03711
$Z \to q\bar{q}$	0.0717	0.0538	0.03837	0.03806	0.0027	**0.48767**	0.05044	0.24146	0.01469	0.00111
$Z \to \nu\nu$	0.00761	0.02112	0.10167	0.03267	0.00185	0.04311	**0.69351**	0.06956	0.02796	0.00094
$W \to qq'$	0.00106	0.01635	0.01383	0.03375	0.00382	0.15997	0.06067	**0.69598**	0.01424	0.00032
$t \to bqq'$	0.02135	0.01089	0.00711	0.02482	0.00158	0.00094	0.01519	0.00282	**0.91335**	0.00195
$t \to b\ell\nu$	0.00277	0.00073	0.00005	0.00013	0.003782	0.00066	0.00165	0.00018	0.00218	**0.95383**

Table 1: The confusion matrix displaying the performance of the Particle Transformer jet tagging architecture when trained on Kubeflow. The '$Z \to \nu\nu$' class corresponds to the quark/gluon class described earlier, and it is treated as the background class. The table displays the distribution of predictions made by the model for each class. On the diagonal the accuracy for each class is displayed, summarized in table 2.

	All classes	$H \to b\bar{b}$	$H \to c\bar{c}$	$H \to gg$	$H \to 4q$	$H \to \ell\nu qq'$	$t \to bqq'$	$t \to b\ell\nu$	$W \to qq'$	$Z \to q\bar{q}$	$Z \to \nu\nu$
ParT	**0.7612**	0.8234	0.6769	0.7197	0.7395	0.9376	0.9134	0.9538	0.6960	0.4877	0.69351

Table 2: Summary of the validation accuracies of all the classes combined and for each class separately. In this table a higher number means a better performance, ranging from 0 to 1, with 1 meaning perfect performance.

Model	Accuracy
ParT (Kubeflow)	0.761
ParT (2M jets)	0.836
ParT (10M jets)	0.850
ParT (100M jets)	0.861

Table 3: The accuracy of the Particle Transformer trained on Kubeflow (in blue), compared to the accuracy of the architecture trained on different sizes of the dataset in the original research (Qu et al., 2022a).

4.2 Hyperparameter optimization

The second part of the research was dedicated to performing a hyperparameter optimization study. As detailed in the methodology, six different configurations of hyperparameters were used to train the Particle Transformer model. The resulting accuracies can be observed in table 4.

`-batch-size`	`-start-lr`	`optimizer`	Accuracy
19	0.00059	`adam`	0.755
31	0.00075	`adam`	**0.762**
40	0.00104	`adam`	0.753
24	0.00103	`adamW`	0.743
60	0.00127	`adamW`	0.745
52	0.00114	`ranger`	**0.770**
8	0.00100	`ranger`	0.761

Table 4: The results of the hyperparameter optimization studies performed on Kubeflow. The values of the three hyperparameters and their resulting validation accuracies are given. The bottom row, displayed in blue, contains the values of the hyperparameters in the original Particle Transformer architecture. The configurations that resulted in a higher accuracy than the original architecture, are displayed in green.

	$H \to b\bar{b}$	$H \to c\bar{c}$	$H \to gg$	$H \to 4q$	$H \to \ell\nu qq'$	$Z \to q\bar{q}$	$Z \to \nu\nu$	$W \to qq'$	$t \to bqq'$	$t \to b\ell\nu$
$H \to b\bar{b}$	**0.82743**	0.05859	0.01246	0.01304	0.00122	0.0434	0.00409	0.00389	0.03432	0.00156
$H \to c\bar{c}$	0.05429	**0.67665**	0.07160	0.08330	0.00421	0.03467	0.01602	0.02119	0.03745	0.00062
$H \to gg$	0.03184	0.04166	**0.71259**	0.11971	0.00057	0.02579	0.04038	0.01202	0.01525	0.00019
$H \to 4q$	0.00906	0.03605	0.12441	**0.7494**	0.00339	0.01596	0.00921	0.01895	0.03354	0.00003
$H \to \ell\nu qq'$	0.00127	0.00378	0.00029	0.00403	**0.95107**	0.00255	0.00124	0.00613	0.00146	0.02818
$Z \to q\bar{q}$	0.04548	0.03721	0.02528	0.03421	0.00459	**0.49999**	0.05184	0.28577	0.01492	0.00071
$Z \to \nu\nu$	0.00855	0.02061	0.09436	0.03195	0.0023	0.04032	**0.70008**	0.0709	0.03012	0.00081
$W \to qq'$	0.00111	0.01135	0.00886	0.02915	0.00712	0.1195	0.06583	**0.74892**	0.01541	0.0003
$t \to bqq'$	0.01989	0.01513	0.00611	0.02772	0.00185	0.00081	0.01102	0.00193	**0.91446**	0.001008
$t \to b\ell\nu$	0.00233	0.00074	0.00003	0.00017	0.0415	0.00047	0.00174	0.00017	0.00249	**0.95036**

Table 5: The confusion matrix displaying the performance of the Particle Transformer jet tagging architecture trained on Kubeflow, using the hyperparameters that had the highest accuracy in table 4. The table displays the distribution of predictions made by the model for each class, with the classes that performed better than the original architecture displayed in green.

5 Discussion

In this research the Particle Transformer architecture is trained on a novel centralized machine learning platform at CERN. The model achieved a validation accuracy of 0.761, meaning that 76.1% of all jets are classified correctly.

To train this model on Kubeflow, some adjustments are made in the set-up of the model compared to the original training configuration. The dataset is reduced in size to approximately contain 1 million jets, and the number of iterations and epochs are lowered. This is done to comply with memory constraints encountered while training on Kubeflow, and potentially explains the slightly reduced accuracy of the model trained on Kubeflow in comparison to the original model.

The accuracy found in these results, however, serves as a good benchmark for further research: the hyperparameter optimization that is performed. Three parameters are chosen for the hyperparameter tuning investigated in this experiment. Different configurations of the batch size, start learning rate and optimizer are selected using the random search accuracy, which results in six new validation accuracies.

Two of the six validation accuracies are higher than the accuracy of the model trained on the original hyperparameters, and four are lower. However, it is difficult to conclude that a global optimum is found for the hyperparameters. This is due to the relatively low number of random trials compared to the many different hyperparameter configurations that are possible. It is encouraged in further research to attempt the tuning with a higher number of trials, which in this research is limited due to execution time constraints, to increase chances of finding the global optimum hyperparameter values.

6 Conclusion

There are not many environments more suited for machine learning use cases than particle physics. In the past few decades deep learning algorithms have had a rise and have become a necessity for the physicists at CERN, since its particle accelerator and detectors generate billions of events. A centralized machine learning platform created at CERN has helped giving physicists easier access to their machine learning workflows and pipelines in order to gain productivity.

The purpose of this research is to use this novel Kubeflow based platform to perform the first hyperparameter optimization on the Particle Transformer architecture. The specific goal is to find the optimal values for three hyperparameters in an algorithm that determines the origin of particle jets, to improve the accuracy of the model. Already with a random search algorithm and a low number of trials used in this optimization study, a configuration with higher accuracy than the original model trained on Kubeflow was discovered. These results are promising, and further research is encouraged. The concept of hyperparameter optimization is promising in particle physics applications and perhaps in many other applications in general.

References

Aad, G., Abajyan, T., Abbott, B., Abdallah, J., Abdel Khalek, S., Abdelalim, A., ... Zwalinski, L. (2012). Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc. *Physics Letters B*, *716*(1), 1-29. Retrieved from https://www.sciencedirect.com/science/article/pii/S037026931200857X doi: https://doi.org/10.1016/j.physletb.2012.08.020

Brüning, O., Burkhardt, H., & Myers, S. (2012). The large hadron collider. *Progress in Particle and Nuclear Physics*, *67*(3), 705-734. Retrieved from https://www.sciencedirect.com/science/article/pii/S0146641012000695 doi: https://doi.org/10.1016/j.ppnp.2012.03.001

Chatrchyan, S. e. a. (2008). The CMS Experiment at the CERN LHC. *JINST*, *3*, S08004. doi: 10.1088/1748-0221/3/08/S08004

Cogan, J., Kagan, M., Strauss, E., & Schwartzman, A. (2015, feb). Jet-images: computer vision inspired techniques for jet tagging. *Journal of High Energy Physics*, *2015*(2). Retrieved from https://doi.org/10.1007/JHEP02(2015)118 doi: 10.1007/jhep02(2015)118

Davis, S. R. (2016, Aug). *Interactive slice of the cms detector.* Retrieved from https://cds.cern.ch/record/2205172

EOS. (2010). *Eos - open storage documentation.* Retrieved from https://eos-docs.web.cern.ch/

Golubovic, D., & Rocha, R. (2021). Training and serving ml workloads with kubeflow at cern. *EPJ Web Conf.*, *251*, 02067. Retrieved from https://doi.org/10.1051/epjconf/202125102067 doi: 10.1051/epjconf/202125102067

Kubeflow. (2021). *Introduction to katib.* Retrieved from https://www.kubeflow.org/docs/components/katib/overview/

Larkoski, A. J., Moult, I., & Nachman, B. (2020). Jet substructure at the large hadron collider: A review of recent advances in theory and machine learning. *Physics Reports*, *841*, 1-63. Retrieved from https://www.sciencedirect.com/science/article/pii/S0370157319303643 (Jet substructure at the Large Hadron Collider: A review of recent advances in

theory and machine learning) doi: https://doi.org/10.1016/j.physrep.2019.11.001

Lipari, P., & Stanev, T. (1991, Dec). Propagation of multi-tev muons. *Phys. Rev. D*, *44*, 3543–3554. Retrieved from https://link.aps.org/doi/10.1103/PhysRevD.44.3543 doi: 10.1103/PhysRevD.44.3543

Mc Cauley, T. (2019). *Display of an event in a dijet resonance search in cms.* Retrieved from https://cds.cern.ch/record/2681224

Merk, H. (2022). *Drawing of the kubernetes architecture.* Retrieved from None, drawn by author on paper

Oerter, R. (2006, 07). The theory of almost everything: The standard model, the unsung triumph of modern physics. *Physics Today - PHYS TODAY*, *59*. doi: 10.1063/1.2337829

Olson, R. S., Bartley, N., Urbanowicz, R. J., & Moore, J. H. (2016). *Evaluation of a tree-based pipeline optimization tool for automating data science.* arXiv. Retrieved from https://arxiv.org/abs/1603.06212 doi: 10.48550/ARXIV.1603.06212

Pol, A. A., Berger, V., Cerminara, G., Germain, C., & Pierini, M. (2020). *Anomaly detection with conditional variational autoencoders.* arXiv. Retrieved from https://arxiv.org/abs/2010.05531 doi: 10.48550/ARXIV.2010.05531

Qu, H. (2019). *Particlenet: Jet tagging via particle clouds.* Retrieved from https://indico.cern.ch/event/766872/

Qu, H., & Gouskos, L. (2020, Mar). Jet tagging via particle clouds. *Phys. Rev. D*, *101*, 056019. Retrieved from https://link.aps.org/doi/10.1103/PhysRevD.101.056019 doi: 10.1103/PhysRevD.101.056019

Qu, H., Harju, N., & Li, C. (2020). *Weaver.* Retrieved from https://github.com/hqucms/weaver

Qu, H., Li, C., & Qian, S. (2022a). *Particle transformer for jet tagging.* arXiv. Retrieved from https://arxiv.org/abs/2202.03772 doi: 10.48550/ARXIV.2202.03772

Qu, H., Li, C., & Qian, S. (2022b, 2). Particle Transformer for Jet Tagging. Retrieved from https://github.com/jet-universe/particle_transformer

Tunyasuvunakool, K., Adler, J., Wu, Z., Green, T., Zielinski, M., Žídek, A., ... Hassabis, D. (2021, 08). Highly accurate protein structure prediction for the human proteome. *Nature*, *596*, 1-9. doi: 10.1038/s41586-021-03828-1

van der Poel, E. F. (2012). *Muon Performance studies in ATLAS towards a search for the Standard Model Higgs boson* (Unpublished doctoral dissertation). U. Amsterdam, IHEF.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). *Attention is all you need.* arXiv. Retrieved from `https://arxiv.org/abs/1706.03762` doi: 10.48550/ARXIV.1706.03762

Workman, R. e. a. (n.d.). *Review of particle physics.* (to be published (2022))

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(1), 4-24. doi: 10.1109/TNNLS.2020.2978386

7 Appendix

7.1 Training script

In this section the full training script used for training the Particle Transformer architecture is presented. Information about specific code used can be found in section 3.2.

```
#!/bin/bash

set -x

source env.sh

echo "args: $@"

# set the dataset dir via `DATADIR_JetClass`
DATADIR=${DATADIR_JetClass}
[[ -z $DATADIR ]] && DATADIR='./datasets/JetClass'

# set a comment via `COMMENT`
suffix=${COMMENT}

# set the number of gpus for DDP training via `DDP_NGPUS`
NGPUS=${DDP_NGPUS}
[[ -z $NGPUS ]] && NGPUS=1
if ((NGPUS > 1)); then
    CMD="torchrun --standalone --nnodes=1 --nproc_per_node=$NGPUS $(which
        weaver) --backend nccl"
else
    CMD="weaver"
fi

epochs=5
samples_per_epoch=$((100 * 1024 / $NGPUS))
samples_per_epoch_val=$((100 * 128))
```

```
dataopts="--num-workers 1 --fetch-step 0.01"

# PN, PFN, PCNN, ParT
model=$1
if [[ "$model" == "ParT" ]]; then
    modelopts="networks/example_ParticleTransformer.py --use-amp"
    batchopts="--batch-size 8 --start-lr 1e-3"
elif [[ "$model" == "PN" ]]; then
    modelopts="networks/example_ParticleNet.py"
    batchopts="--batch-size 512 --start-lr 1e-2"
elif [[ "$model" == "PFN" ]]; then
    modelopts="networks/example_PFN.py"
    batchopts="--batch-size 4096 --start-lr 2e-2"
elif [[ "$model" == "PCNN" ]]; then
    modelopts="networks/example_PCNN.py"
    batchopts="--batch-size 4096 --start-lr 2e-2"
else
    echo "Invalid model $model!"
    exit 1
fi

# "kin", "kinpid", "full"
FEATURE_TYPE=$2
[[ -z ${FEATURE_TYPE} ]] && FEATURE_TYPE="full"

if ! [[ "${FEATURE_TYPE}" =~ ^(full|kin|kinpid)$ ]]; then
    echo "Invalid feature type ${FEATURE_TYPE}!"
    exit 1
fi

# currently only Pythia
SAMPLE_TYPE=Pythia

weaver \
    --data-train \
```

```
"HToBB:${DATADIR}/${SAMPLE_TYPE}/train_100M/HToBB_000.root" \
"HToCC:${DATADIR}/${SAMPLE_TYPE}/train_100M/HToCC_000.root" \
"HToGG:${DATADIR}/${SAMPLE_TYPE}/train_100M/HToGG_000.root" \
"HToWW2Q1L:${DATADIR}/${SAMPLE_TYPE}/train_100M/HToWW2Q1L_000.root" \
"HToWW4Q:${DATADIR}/${SAMPLE_TYPE}/train_100M/HToWW4Q_000.root" \
"TTBar:${DATADIR}/${SAMPLE_TYPE}/train_100M/TTBar_000.root" \
"TTBarLep:${DATADIR}/${SAMPLE_TYPE}/train_100M/TTBarLep_000.root" \
"WToQQ:${DATADIR}/${SAMPLE_TYPE}/train_100M/WToQQ_000.root" \
"ZToQQ:${DATADIR}/${SAMPLE_TYPE}/train_100M/ZToQQ_000.root" \
"ZJetsToNuNu:${DATADIR}/${SAMPLE_TYPE}/train_100M/ZJetsToNuNu_000.root" \
--data-val \
"HToBB:${DATADIR}/${SAMPLE_TYPE}/val_5M/HToBB_120.root" \
"HToCC:${DATADIR}/${SAMPLE_TYPE}/val_5M/HToCC_120.root" \
"HToGG:${DATADIR}/${SAMPLE_TYPE}/val_5M/HToGG_120.root" \
"HToWW2Q1L:${DATADIR}/${SAMPLE_TYPE}/val_5M/HToWW2Q1L_120.root" \
"HToWW4Q:${DATADIR}/${SAMPLE_TYPE}/val_5M/HToWW4Q_120.root" \
"TTBar:${DATADIR}/${SAMPLE_TYPE}/val_5M/TTBar_120.root" \
"TTBarLep:${DATADIR}/${SAMPLE_TYPE}/val_5M/TTBarLep_120.root" \
"WToQQ:${DATADIR}/${SAMPLE_TYPE}/val_5M/WToQQ_120.root" \
"ZToQQ:${DATADIR}/${SAMPLE_TYPE}/val_5M/ZToQQ_120.root" \
"ZJetsToNuNu:${DATADIR}/${SAMPLE_TYPE}/val_5M/ZJetsToNuNu_120.root" \
--data-test \
"HToBB:${DATADIR}/${SAMPLE_TYPE}/test_20M/HToBB_100.root" \
"HToCC:${DATADIR}/${SAMPLE_TYPE}/test_20M/HToCC_100.root" \
"HToGG:${DATADIR}/${SAMPLE_TYPE}/test_20M/HToGG_100.root" \
"HToWW2Q1L:${DATADIR}/${SAMPLE_TYPE}/test_20M/HToWW2Q1L_100.root" \
"HToWW4Q:${DATADIR}/${SAMPLE_TYPE}/test_20M/HToWW4Q_100.root" \
"TTBar:${DATADIR}/${SAMPLE_TYPE}/test_20M/TTBar_100.root" \
"TTBarLep:${DATADIR}/${SAMPLE_TYPE}/test_20M/TTBarLep_100.root" \
"WToQQ:${DATADIR}/${SAMPLE_TYPE}/test_20M/WToQQ_100.root" \
"ZToQQ:${DATADIR}/${SAMPLE_TYPE}/test_20M/ZToQQ_100.root" \
"ZJetsToNuNu:${DATADIR}/${SAMPLE_TYPE}/test_20M/ZJetsToNuNu_100.root" \
--data-config data/JetClass/JetClass_${FEATURE_TYPE}.yaml --network-config
    $modelopts \
--model-prefix
```

```
    training/JetClass/${SAMPLE_TYPE}/${FEATURE_TYPE}/${model}/{auto}${suffix}/net
    \
$dataopts $batchopts \
--samples-per-epoch ${samples_per_epoch} --samples-per-epoch-val
    ${samples_per_epoch_val} --num-epochs $epochs --gpus 0 \
--optimizer ranger --log
    logs/JetClass_${SAMPLE_TYPE}_${FEATURE_TYPE}_${model}_{auto}${suffix}.log
    --predict-output pred.root \
--tensorboard JetClass_${SAMPLE_TYPE}_${FEATURE_TYPE}_${model}${suffix} \
"${@:3}"
```

7.2 Docker Image

In this section the docker image used to build the container for the Kubernetes pod will be presented. Information about specific code used can be found in section 3.2.

```
FROM pytorch/pytorch:1.8.1-cuda10.2-cudnn7-runtime

ENV NAME freud

RUN apt-get -qq update && \
    apt-get -yqq install git && \
    DEBIAN_FRONTEND=noninteractive apt-get -yqq install libpam-krb5 krb5-user &&
    \
    apt-get -yqq clean

RUN useradd -m $NAME
USER $NAME
WORKDIR /home/$NAME

COPY krb5.conf /etc/krb5.conf

COPY requirements.txt .
RUN export PATH=/home/$NAME/.local/bin:$PATH && \
```

```
    pip install -Ur requirements.txt --no-cache-dir

RUN git clone https://github.com/DidierMerk/weaverpart.git
WORKDIR /home/$NAME/weaver

ENTRYPOINT ["python", "train.py"]
```

7.3 YAML-file

In this section the YAML-file used to set up the Katib hyperparameter training job will be presented. Information about specific code used can be found in section 3.2.

```
apiVersion: kubeflow.org/v1beta1
kind: Experiment
metadata:
  name: katibjob

spec:
  parallelTrialCount: 2
  maxTrialCount: 6
  maxFailedTrialCount: 1
  objective:
    type: maximize
    objectiveMetricName: Acc
    additionalMetricNames:
        - AvgAcc
        - Loss
        - AvgLoss
  algorithm:
    algorithmName: random
  metricsCollectorSpec:
    collector:
      kind: StdOut
  parameters:
```

```
  - name: --batch-size
    parameterType: int
    feasibleSpace:
      min: "1"
      max: "64"
  - name: --start-lr
    parameterType: double
    feasibleSpace:
      min: "5.0e-4"
      max: "1.5e-3"
  - name: --optimizer
    parameterType: categorical
    feasibleSpace:
      list:
        - "adam"
        - "adamW"
        - "ranger"
trialTemplate:
  primaryContainerName: katibjob
  trialParameters:
    - name: learningRate
      description: Learning rate for the training model
      reference: --start-lr
    - name: batchSize
      description: Size of the batch for the training model
      reference: --batch-size
    - name: optimizer
      description: Optimizer used for the training model
      reference: --optimizer
  trialSpec:
    apiVersion: batch/v1
    kind: Job
    metadata:
      annotations:
        "custom-key": "custom-annotation"
```

```
  labels:
    "custom-key": "custom-label"
spec:
  template:
    metadata:
      labels:
        mount-kerberos-secret: "true"
        mount-eos: "true"
      annotations:
        sidecar.istio.io/inject: "false"
    spec:
        restartPolicy: Never
        volumes:
          - name: nvidia-driver
            hostPath:
              path: /opt/nvidia-driver
              type: ""
        containers:
        - name: katibjob
          resources:
            limits:
              nvidia.com/gpu: 1
              memory: 16Gi
              cpu: 2
            requests:
              cpu: 2
              memory: 8Gi
          volumeMounts:
            - name: nvidia-driver
              mountPath: /opt/nvidia-driver
          env:
            - name: PATH
              value: /opt/conda/bin:/bin:/usr/bin:/usr/local/bin:/opt/
                     nvidia-driver/bin:/home/freud/.local/bin/
          image: registry.cern.ch/dmerk-images/katibjob:latest
```

```
command: ['/bin/sh', '-c']
args:
  - ls;
    pwd;
    nvidia-smi;
    PATH=$PATH:/home/freud/.local/bin/;
    env;
    /home/freud/.local/bin/weaver
    --data-train \
    "HToBB:/katibjob-4jflmgj6-wmp52
        eos/user/d/dmerk/particle_transformer/datasets/
    JetClass/Pythia/train_100M/HToBB_000.root" \
    "HToCC:/eos/user/d/dmerk/particle_transformer/datasets/
    JetClass/Pythia/train_100M/HToCC_000.root" \
    "HToGG:/eos/user/d/dmerk/particle_transformer/datasets/
    JetClass/Pythia/train_100M/HToGG_000.root" \
    "HToWW2Q1L:/eos/user/d/dmerk/particle_transformer/datasets/
    JetClass/Pythia/train_100M/HToWW2Q1L_000.root" \
    "HToWW4Q:/eos/user/d/dmerk/particle_transformer/datasets/
    JetClass/Pythia/train_100M/HToWW4Q_000.root" \
    "TTBar:/eos/user/d/dmerk/particle_transformer/datasets/
    JetClass/Pythia/train_100M/TTBar_000.root" \
    "TTBarLep:/eos/user/d/dmerk/particle_transformer/datasets/
    JetClass/Pythia/train_100M/TTBarLep_000.root" \
    "WToQQ:/eos/user/d/dmerk/particle_transformer/datasets/
    JetClass/Pythia/train_100M/WToQQ_000.root" \
    "ZToQQ:/eos/user/d/dmerk/particle_transformer/datasets/
    JetClass/Pythia/train_100M/ZToQQ_000.root" \
    "ZJetsToNuNu:/eos/user/d/dmerk/particle_transformer/datasets/
    JetClass/Pythia/train_100M/ZJetsToNuNu_000.root"
    --data-val \
    "HToBB:/eos/user/d/dmerk/particle_transformer/datasets/
    JetClass/Pythia/HToBB_120.root" \
    "HToCC:/eos/user/d/dmerk/particle_transformer/datasets/
    JetClass/Pythia/HToCC_120.root" \
```

```
"HToGG:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/HToGG_120.root" \
"HToWW2Q1L:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/val_5M/HToWW2Q1L_120.root" \
"HToWW4Q:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/val_5M/HToWW4Q_120.root" \
"TTBar:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/val_5M/TTBar_120.root" \
"TTBarLep:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/val_5M/TTBarLep_120.root" \
"WToQQ:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/val_5M/WToQQ_120.root" \
"ZToQQ:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/val_5M/ZToQQ_120.root" \
"ZJetsToNuNu:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/val_5M/ZJetsToNuNu_120.root"
--data-test \
"HToBB:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/test_20M/HToBB_100.root" \
"HToCC:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/test_20M/HToCC_100.root" \
"HToGG:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/test_20M/HToGG_100.root" \
"HToWW2Q1L:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/test_20M/HToWW2Q1L_100.root" \
"HToWW4Q:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/test_20M/HToWW4Q_100.root" \
"TTBar:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/test_20M/TTBar_100.root" \
"TTBarLep:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/test_20M/TTBarLep_100.root" \
"WToQQ:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/test_20M/WToQQ_100.root" \
"ZToQQ:/eos/user/d/dmerk/particle_transformer/datasets/
JetClass/Pythia/test_20M/ZToQQ_100.root" \
```

```
"ZJetsToNuNu:/eos/user/d/dmerk/particle_transformer/
datasets/JetClass/Pythia/test_20M/ZJetsToNuNu_100.root"
--data-config
    /eos/user/d/dmerk/particle_transformer/data/JetClass/
JetClass_full.yaml
--network-config
    /eos/user/d/dmerk/particle_transformer/networks/
example_ParticleTransformer.py --use-amp
--model-prefix
    /eos/user/d/dmerk/particle_transformer/training/JetClass/
Pythia/full/ParT/{auto}/net
--num-workers 0 --fetch-step 0.01
--batch-size=${trialParameters.batchSize}
--start-lr=${trialParameters.learningRate}
--optimizer=${trialParameters.optimizer}
--samples-per-epoch 102400
--samples-per-epoch-val 12800
--num-epochs 5
--gpus 0
--log /eos/user/d/dmerk/particle_transformer/logs/
JetClass_Pythia_full_ParT_{auto}.log
--predict-output pred.root
--tensorboard JetClass_Pythia_Full_ParT
```

7.4 Random search script

In this section the python script used to set up the manual selection of hyperparameter configurations will be presented. The output of this script provided the configurations used in the experiment.

```
# Program to manually implement random search algorithm
# hyperparameter tuning on Kubeflow.

from random import randrange, uniform
```

```
# Maximum number of trials to perform
trial_count = 6

# Print the random parameter configurations
for i in range(trial_count):
    print("--batch-size: ", randrange(1,64), \
          "  ||  ", \
          "--start-lr: ", uniform(0.0005, 0.0015), \
          "  ||  ", \
          "optimizer: ", randrange(1, 4))

# Optimizer correspondency:
# - 1: adam
# - 2: adamW
# - 3: ranger
```