UNIVERSITÀ DELLA CALABRIA DIPARTIMENTO DI FISICA

Unraveling JetNet: Insights into Particle Collision Data

Comparative Performance of Neural Network Architectures

Advanced Computer Science & Machine Learning

Instructors: Luigi Delle Rose, Fabrizio Lo Scudo,

Authors:

Michele Arcuri, Luca Coscarelli, Nelson Manuel Mora Fernández

Date of Submission:

January 25, 2025

Department of Physics

University of Calabria

Mary's mother had four children: April, May and June.

What is the name of the fourth child?

Contents

1 Introduction

Jets are one of the most ubiquitous events at the Large Hadron Collider (LHC). The need to understand the early stage of the universe has led scientists to attempt to recreate the microscopic early universe by accelerating particles to extremely high energies and studying their collisions. The key to understanding particle collisions is by measuring the properties of the particles emanating from them. In particular, jets are the result of the fragmentation of Standard Model particles like quarks and gluons [12].

One of the main complications in studying these collisions is the classification task of distinguishing between different types of jets. Therefore, developing new tools and classification techniques has become a mandatory pursuit not just for particle physicists, but for the entire scientific community. Recently, highly innovative machine learning (ML) algorithms have opened the doors for the study and classification of these jets by using neural networks and computer vision demonstrating the promising performance of jet image techniques [3]. This project focuses on studying JetNet, a particle cloud dataset aimed at improving accessibility for jet-based machine learning models [9]. The primary goal is to develop a multi-class classification model from scratch, capable of distinguishing between different types of jets based on input data describing their features.

2 Dataset

JetNet [8, 11, 9] is a dataset which describes high-energy proton-proton collisions, which produce particles that decay into other types over time. Eventually, the hadronization process occurs, leading to the formation of stable final products. We define the jet as the collimated set of hadrons produced in the collision [10].

We work in Cartesian coordinates and set the z-axis parallel to the jet axis, the x-axis and the y-axis pointing as shown in the following Fig. (1). Then, we define ϕ as the azimuthal angle, θ as the polar angle, and the pseudo-rapidity $\eta = -\log(\tan(\theta/2))$. The transverse momentum p_T is the projection of the particle momentum on the (x, y) plane. Finally, we rewrite the Cartesian moment of the particle in longitudinal-boost-invariant pseudo-angular coordinates (p_T, η, ϕ) , which are the coordinates used in the dataset. In Fig. (1), we show the full coordinate system and a visual description of some types of jets.

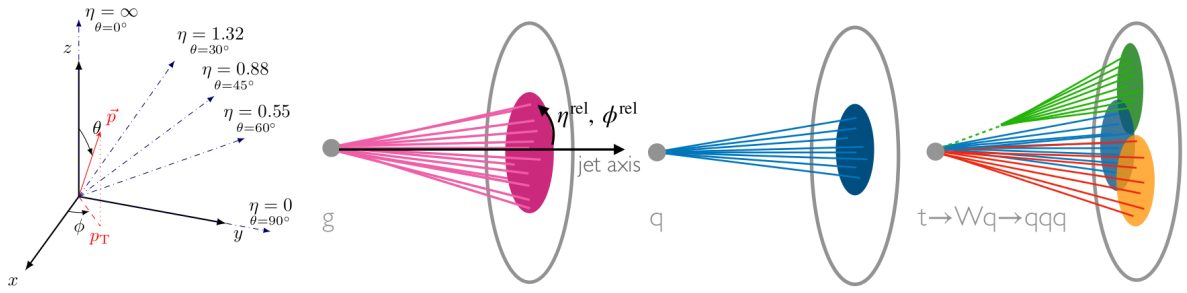

Figure 1: The collider physics coordinate system defining (p_T, η, ϕ) (left). Three types of jet classes in the dataset (right). Gluon (g) and light quark (q) jets have simple topologies, with q jets generally containing fewer particles. Top quark (t) jets have a complex three-pronged structure.

Derived from [4], JetNet consists of simulated particle jets with transverse momenta $p_T^{jet} \approx 1$ TeV, originating from gluons, light quarks, and top quarks produced in 13 TeV proton-proton collisions in a simplified detector. We limit the number of constituents to the 30 highest p_T particles per jet. For each particle, we provide the following four features: the relative angular coordinates $\eta^{rel} = \eta^{particle} - \eta^{jet}$, $\phi^{rel} = \phi^{particle} - \phi^{jet}$ (mod 2π), relative transverse momentum $p_T^{rel} = p_T^{particle} - p_T^{jet}$, and a binary mask feature classifying the particle as genuine or zero-padded.

The dataset is composed of 880 000 frames and five types of particle labels: Gluon (g), Top Quark (t), Light Quark (q), W boson (w), and Z boson (z). An example of how a frame is stored in memory is shown on the left in Fig.(2), while its visualization as an image is displayed on the right.

	etarel	phirel	ptrel
0	0.062908	0.055121	0.364357
1	-0.065864	-0.059077	0.124402
2	-0.065396	-0.053375	0.094154
3	-0.064592	-0.052434	0.079220
4	-0.067126	-0.048498	0.066350
5	0.076064	0.050911	0.042008
6	-0.069592	-0.052434	0.041139
7	-0.037092	-0.024920	0.022501
8	0.025666	0.055227	0.020087
9	0.077453	0.047536	0.016923

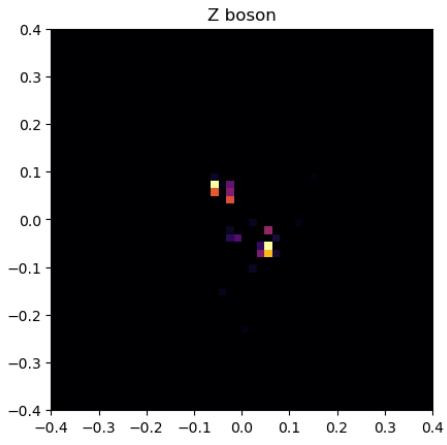

Figure 2: On the left we see part of the numerical representation of the first frame of the dataset, on the right we can see its visual representation.

To obtain meaningful results, we decided to use a single train/validation/test split in our data: we decided to use 200 000 frames for the train set (about 68% of the total number

of frames used), 50 000 frames for the validation set and the same number for the test set. In Fig. (3) is presented the histogram describing the training used. The distribution of the data is balanced across all classes, so there is no need to adjust the validation methods to account for any skewness in the label distribution.

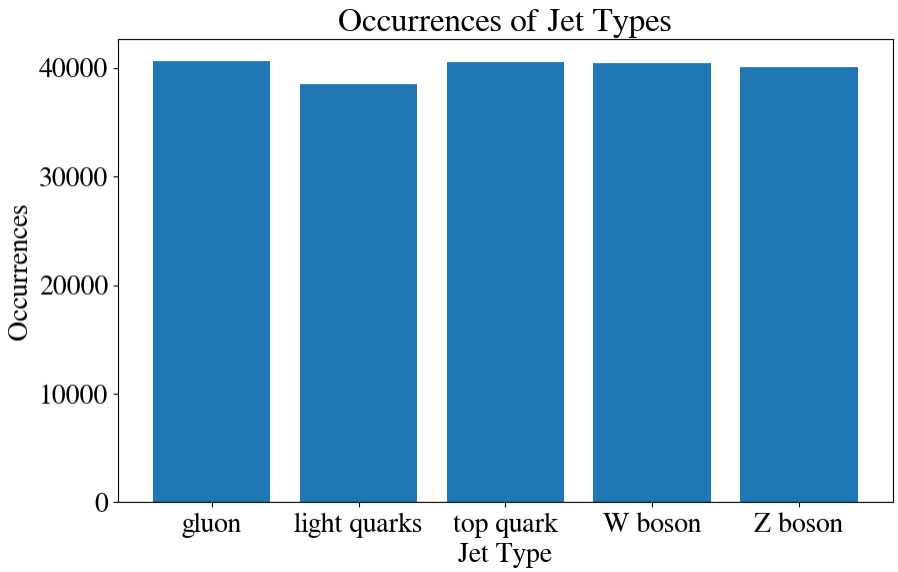

Figure 3: Histogram of jet label distribution in the training set

2.1 Evaluating Two Data Preprocessing Methods

The JetNet dataset can be studied in two different ways: the first method consists of studying the data as particle-based features (as described in the previous section); tThe second approach involves converting each frame into an image and then applying deep learning techniques to extract insights from the data. One of our goals is to analyze how the final model's performance depends on these two different methods of representing the same data.

However, transforming the data into images presents certain drawbacks. For instance, we increase the weight of the data without really increasing the amount of information stored, as a consequence we get longer training times and weaker performance for the same amount of useful information. On the other hand, we use a different representation of the data, meaning we adjust the significance of certain features within it. As an example, the distance between two particles will change depending on the representation: in the original dataset, it is represented by a function of the particle parameters `etarel` and `phirel`, while in the transformed dataset, it will be represented by the distance between the two pixels which contains the particles. Moreover, the second approach allows us to utilize a Convolutional Neural Network (CNN), to analyze the

data, which was not feasible with the first approach.

In order to transform the data into images, we used a method (called `to_image()`) which we import from `jetnet`, the same library which helps us download and load the data in the Colab environment. In these images, the position of the particle is defined from `etarel` and `phirel`, while the intensity of the pixel is directly proportional to the value of `ptrel`. We decide to save each frame as an image of dimensions 50x50 pixels.

Now, we might want to check the normalization of the data, so we can easily take the maximum pixel-value for every image in the training data and sort such list of values to obtain the Fig (4). We notice that the maximum value is already 1, so we conclude that no normalization is needed.

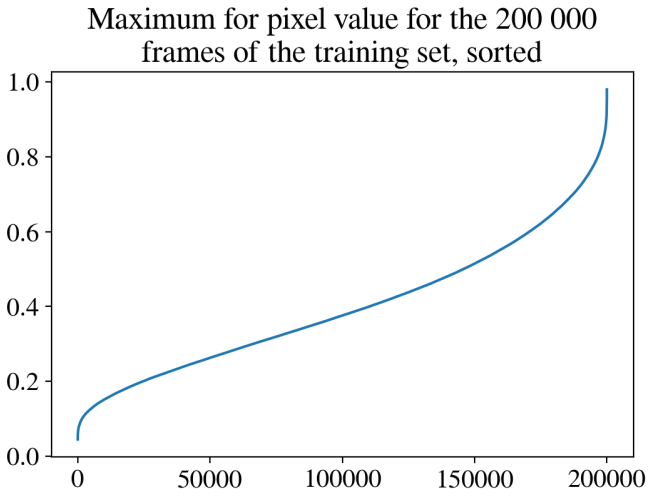

Figure 4: Ordered plot of the maximum pixel-value per image.

2.2 The radius parameter `maxR`

A very important parameter used to transform the dataset to images is called `maxR`. This variable defines the spatial window captured for each image. For instance, setting it to 0.4 in an image of 50x50 pixels means generating an image centered at zero, with the top-left pixel $(\bar{x}, \bar{y}) = (0, 0)$ placed in $(x, y) = (-\texttt{maxR}, \texttt{maxR})$. Our decision to initially set this parameter to 0.4 was guided by some preliminary insights into the data. Fig. (5) shows the pixel importance values displayed on both linear and logarithmic scales. The reason for choosing this specific value for `maxR` becomes clear, as it provides the best balance between file size, pixel density, and the amount of useful information stored in each image.

However, our very first results using various models of neural networks were not in

agreement with our expectations, so we decided to tweak this parameter. In the following Fig. (6), the image on the left is generated using `maxR` = 0.4, while the second image has `maxR` = 0.1. We can clearly see that the second image has a slightly higher number of features, compared to the first.

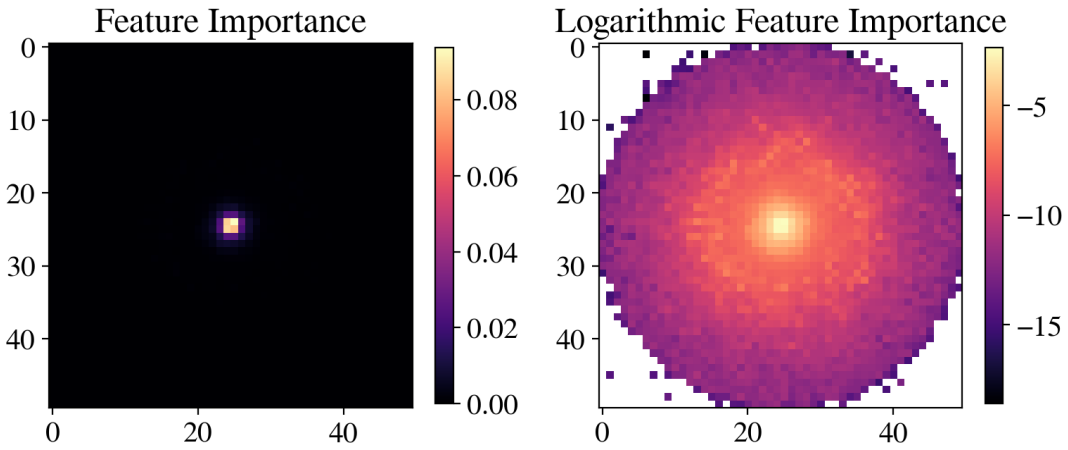

Figure 5: The feature importance in linear scale (left) and logarithmic scale (right). To evaluate these plots we used a RandomForest [1] model with 2000 estimators and 42 000 images.

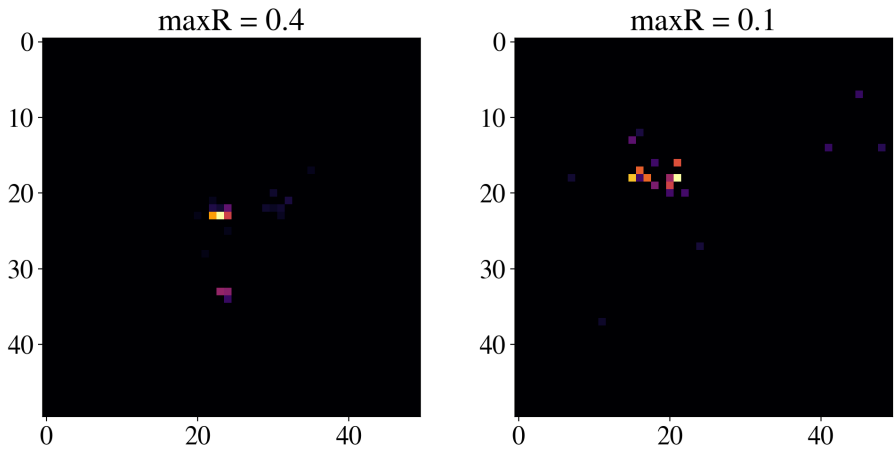

Figure 6: Comparison between two different values of `maxR`. The same image was used in both picture, the only difference is the zoom level.

2.3 Further analysis on JetNet

Based on the results from our initial tests using the image representation of the data, we were encouraged to gather more information to better understand the misbehavior we observed. To this aim, we further analyzed the distribution of the value of the maximum pixel per image (this is the same image used to check the normalization of the data in Fig. (4)). In the plot, we can see that the pixel intensity at the lower end is very low (below 0.3), which could potentially indicate

a problem. As a solution, we wrote the code to select only images with a greater value than a given threshold. To ensure that the dataset retains its key features, we examine the histogram after applying the filter. In the following Fig. (7), we present the result. It is evident that our filter significantly impacts the class distribution, leading us to conclude that it will not offer a substantial improvement over the previous model.

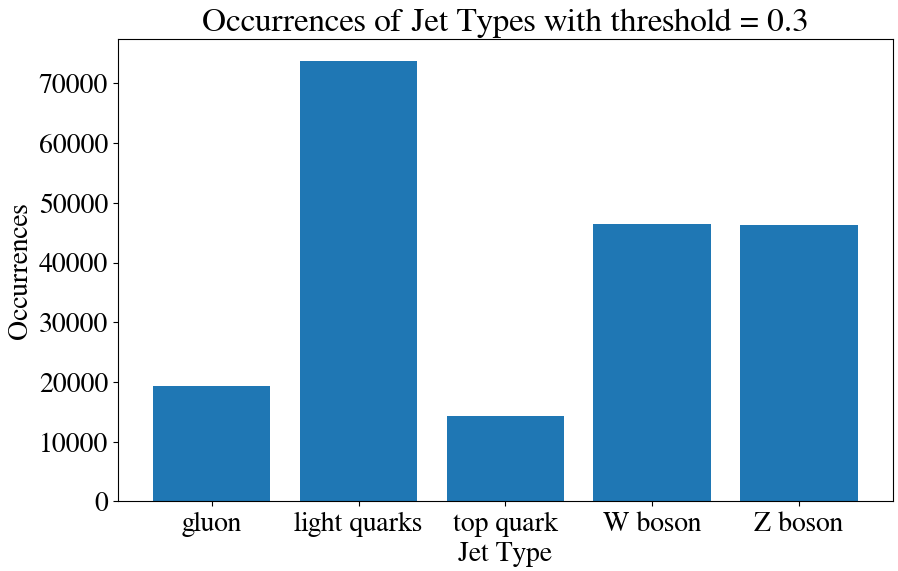

Figure 7: Histogram after the application of a threshold value equal to 0.3 to the dataset.

3 Model Architecture

3.1 Fully Connected Neural Network

We began our project by developing a fully connected neural network to be applied on the JetNet dataset. First, using the tensor representation and then, after preprocesing, we do the same using the image representation. After experimenting with various configurations and tuning all hyperparameters, we ended with an architecture that provided satisfactory results. The input layer takes in an (30×3) input when working with the original dataset (tensor representation) and $(50, 50, 1)$ inputs when using the image representation. A `Flatten` layer transforms the input tensor into a one-dimensional array, ensuring compatibility with the subsequent `Dense` layers. The model architecture consists of 8 blocks, each containing three layers; the first layer is a `Dense` layer with a `ReLU` activation function. This is followed by a `Dropout` layer, which randomly sets a fraction of nodes to zero during training, based on a specified rate. This regularization technique prevents overfitting by ensuring that the model doesn't rely too heavily on any particular neuron

[13]. The final layer of the block is a `Batch Normalization` layer, which normalizes the input by utilizing the mean and standard deviation of the current batch. The model ends with a classification `Dense` layer consisting of 5 nodes and a `softmax` activation function, which outputs a probability distribution over the five classes for the classification task.

Layers	Nodes	Dropout Rate
Block (1)	2500	0.4
Block (2)	2000	0.2
Block (3)	1800	0.2
Block (4)	1500	0.2
Block (5)	1500	0.2
Block (6)	800	0.2
Block (7)	200	0.2
Block (8)	150	0.2
Classification Layer	5	

Table 1: Each block consists of a Dense layer with ReLU activation, followed by a Dropout layer and a BatchNormalization layer. The classification Layer has a SoftMax activation function.

To compile the model, we used `Categorical Crossentropy` as the loss function and the `Adam` optimizer with a learning rate of 0.001. The model was then trained for 20 epochs with a batch size of 256. During training, we monitored the validation loss and employed `Early Stopping` to prevent overfitting, stopping training if the validation loss did not improve for 5 consecutive epochs. In the following section, we will briefly evaluate the performance of this architecture on the two datasets.

3.1.1 Particle-based representation

When working with the dataset in its original array representation, the model achieves a training accuracy of 0.75 and a validation accuracy of 0.70. As shown in Fig. (8), we observe that, while the training loss steadily decreases, the validation loss levels off and begins to rise after the 10th epoch. This behavior indicates that the model starts to overfit at this stage.

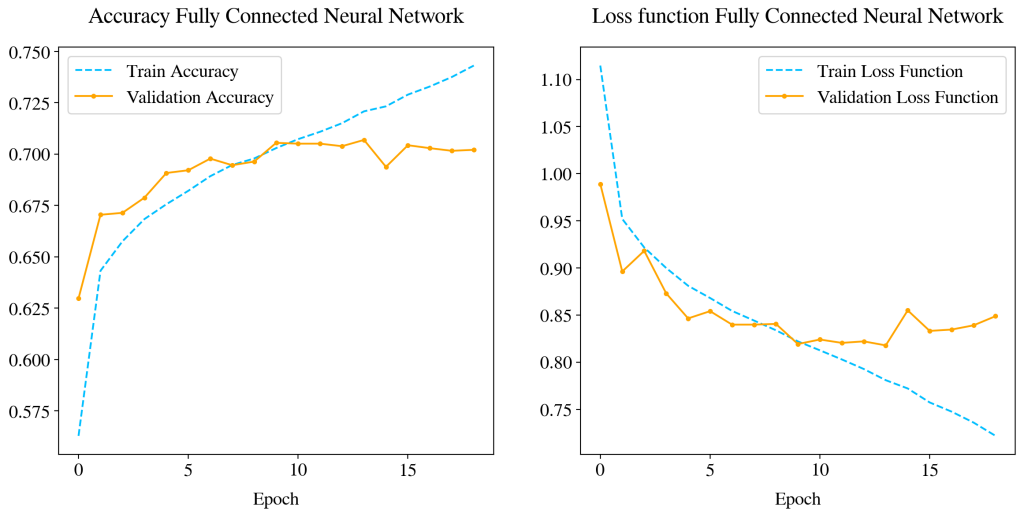

Figure 8: The history of accuracy (left) and loss (right) during the training of the FNN using the data in array format. The final accuracy on the test set is 0.705 and the loss on the test is 0.842.

Moreover, Fig. (9) shows that the model generalizes fairly well. The large diagonal values in the confusion matrix indicate a high number of correct predictions. However, misclassification is more pronounced in certain classes. For instance, the Z boson is often misclassified as the W boson. The area under curve (AUC) scores are close to one, demonstrating that the model can effectively distinguish between all five different classes.

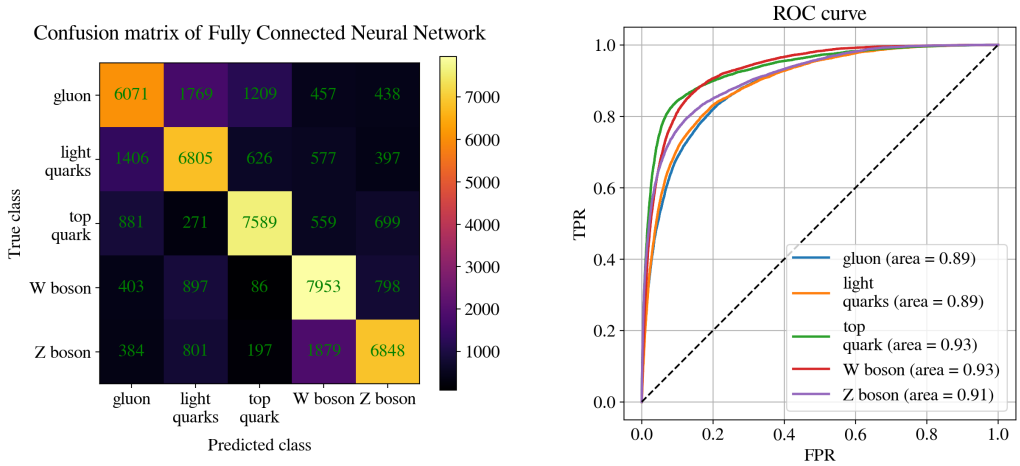

Figure 9: The confusion matrix and the ROC curves for every class in a One-versus-All setup of the FNN using the data in array format.

3.1.2 Image-based representation

When the dataset is represented as images, the model's performance declines significantly. As shown in Fig. (10), the training accuracy increases, but the validation accuracy levels off at 0.575. It is noteworthy that the validation loss begins to increase after the fourth epoch, indicating that

the model starts overfitting much earlier using this representation with respect to the tensor representation.

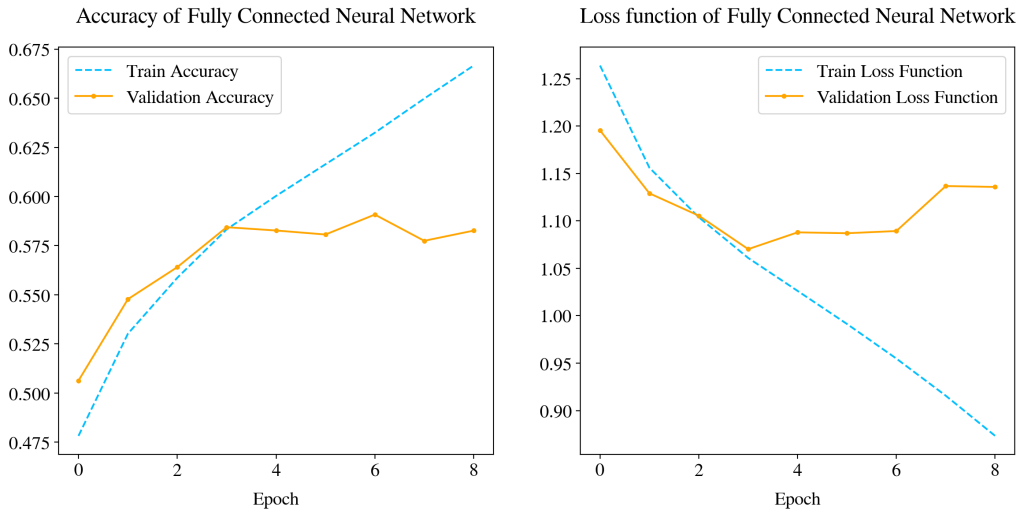

Figure 10: The history of accuracy (left) and loss (right) during the training of the FNN using the data in image format. The final accuracy on the test set is 0.579 and the loss on the test is 1.085.

Although the diagonal values in the confusion matrix are high (11), several off-diagonal elements are also larger in this case. In particular, the model struggles to distinguish a Z boson. Additionally, the lower AUC scores compared to the array representation indicate that the architecture does not generalize well when applied to the image-represented dataset.

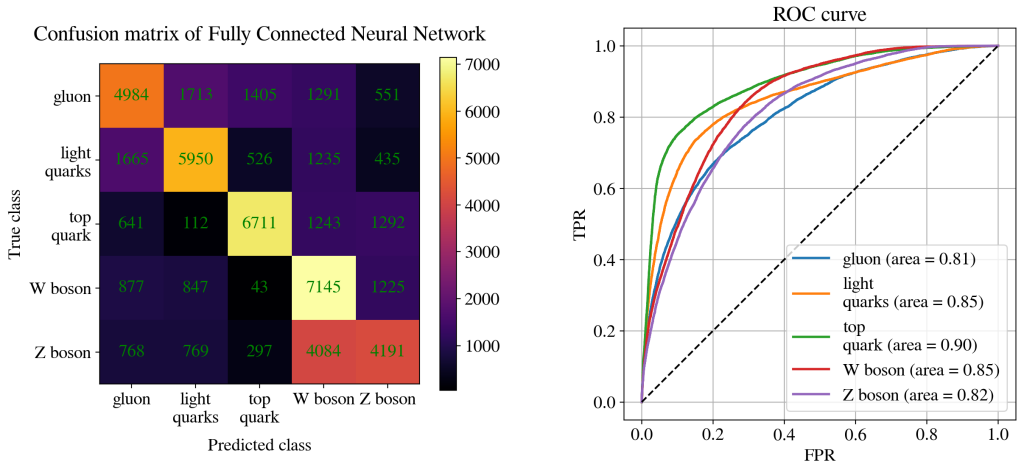

Figure 11: The confusion matrix and the ROC curves for every class in a One-versus-All setup of the FNN using the data in image format.

3.2 Convolutional Neural Network

A key advantage of using an image-based dataset lies in effectively employ Convolutional Neural Networks (CNNs) for feature extraction. In a CNN, instead of relying on `Dense` layers, `Convolutional layers` are employed. The key difference between a fully connected layer and a convolutional layer is that, while the dense layer learns patterns in their input features space, convolutional layers specialize in learning local pattern or features. This ability to detect local structures makes CNNs particularly effective for image-related tasks, as they can capture features such as edges, textures, and more complex patterns [2]. Our model receives input data with shape $(50, 50, 1)$, representing 50×50 single-channel images. The architecture consists of multiple blocks, each composed of a `Conv2D` layers with `ReLU` activation, followed by a `MaxPooling2D` layer. The number of filters increases progressively through the blocks, while the kernel size and the number of strides remained constant for each convolution and pooling layer respectively. The final layer is a `Dense` layer with 5 nodes and a `softmax` activation function.

Layers	Output Size	Convolutional Neural Network
Input layer	$50 \times 50 \times 1$	
Block (1)	$40 \times 40 \times 16$	6×6 conv , filters=16 6×6 max pool
Block (2)	$30 \times 30 \times 32$	6×6 conv , filters=32 6×6 max pool
Block (3)	$20 \times 20 \times 64$	6×6 conv , filters=64 6×6 max pool
Block (4)	$10 \times 10 \times 128$	6×6 conv , filters=128 6×6 max pool
Classification Layer		5 dense, softmax

Table 2: Each block is composed of a `Conv2D` and a `Maxpooling2D` layers, the strides are fixed to 1.

As before, `Categorical Crossentropy` was used as the loss function, and the Adam optimizer was used with a learning rate of 0.005. The model was trained for 40 epochs with a batch size of 128.

As seen from Fig. (12) the accuracy during training keeps growing while the validation

accuracy remains constant to around 0.6. In fact, after the tenth epoch, the validation loss began to increase, indicating that the model was overfitting.

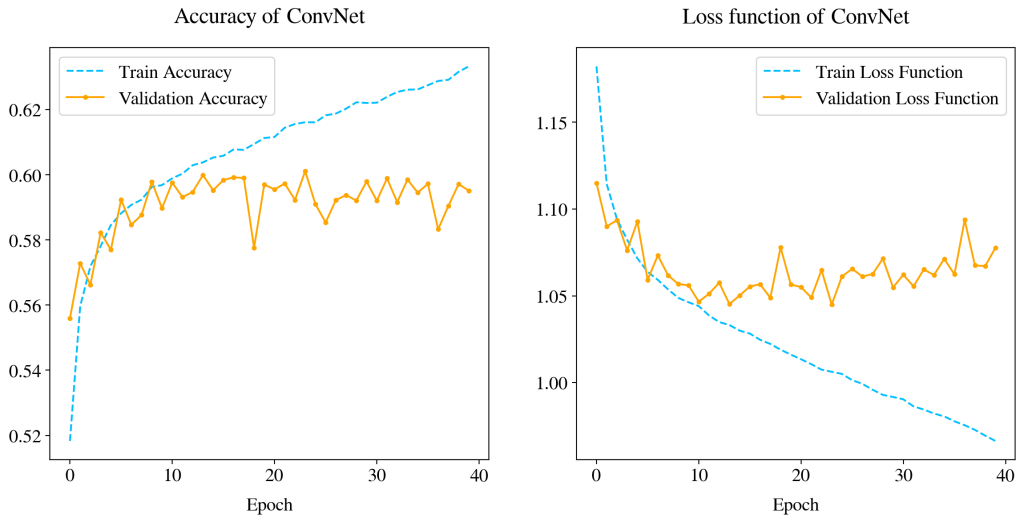

Figure 12: The history of accuracy (left) and loss (right) during the training of the ConvNet using the data in image format. The final accuracy on the test set is 0.5916 and the loss on the test is 1.081.

Moreover, from the confusion matrix and the AUC scores shown in Fig. (13), we see that the model is capable of distinguish the different classes. However, it misclassifies certain classes; for instance, it struggles to accurately differentiate between W and Z bosons.

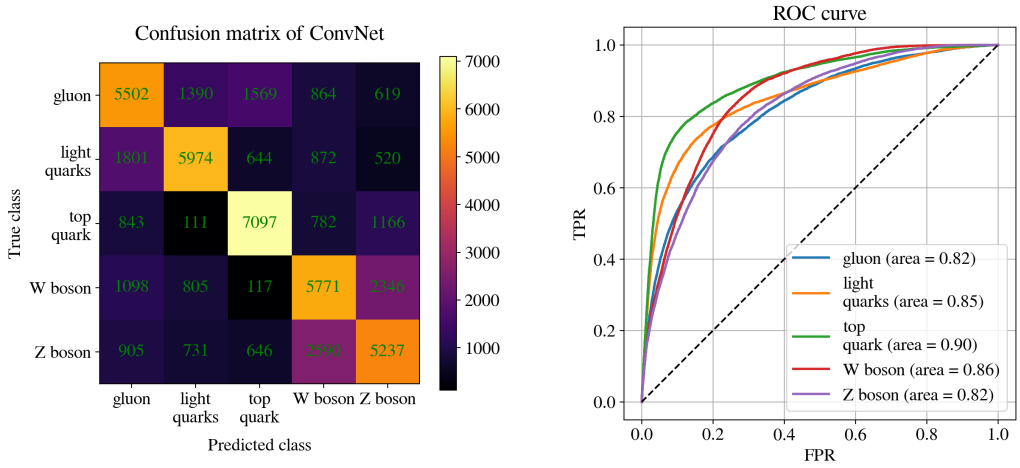

Figure 13: The confusion matrix and the ROC curves for every class in a One-versus-All setup of the ConvNet using the data in image format.

3.3 DenseNet

The following model is the result of combining different ideas from traditional architectures describing feed-forward networks like ResNet [6], which connect the output of the ℓ^{th} layer as

input to the $(\ell+1)^{th}$ layer, namely $\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1})$, where $\mathbf{x}_\ell$ corresponds to the output of ℓ^{th} layer and H_ℓ may be a set of operations like Batch Normalization (BN), Rectified linear units (ReLU), Pooling or Convolution (Conv). In particular, ResNets add a skip-connection that bypasses the non-linear transformation with an identity function, i.e.

$$\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}) + \mathbf{x}_{\ell-1}. \tag{1}$$

This facilitates smoother gradient flow and mitigate vanishing gradients [7]. We, instead, follow a different and interesting connectivity approach introduced by [7], which imposes a direct connection from any layer to all subsequent layers, namely

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}]), \tag{2}$$

where $H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}])$ refers to the concatenation of the feature maps produced in layers $0, ..., \ell-1$. Due to this dense connectivity, this architecture is referred to as a Dense Convolutional Network (DenseNet) [7]. As shown in the following figure, we define dense blocks separated by a set of operations, the so called transition blocks [7], which perform convolution and pooling, facilitating down-sampling in our architecture.

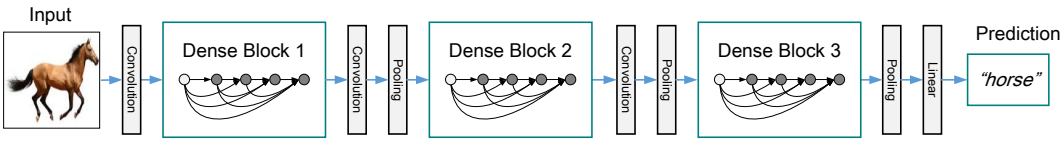

Figure 14: DenseNet architecture

Furthermore, the function $H_\ell(\cdot)$ is defined as a composite function of three main operations: batch normalization, ReLU, and a convolution. Each $H_\ell(\cdot)$ produces k feature maps. Thus, the ℓ^{th} layer produces $k_0 + k(\ell-1)$ feature maps. We refer to this hyperparamter as the grow rate of the network. In this way, each layer has access to all the preceding feature maps in its block, and therefore contributing to the global "collective knowledge". In Table (3.3) we summarize the structure of the architecture used.

Layers	Output Size	DenseNet 1.0
Convolution	$50 \times 50 \times 20$	7×7 conv, stride 2
Pooling	$25 \times 25 \times 20$	3×3 max pool, stride 2
Dense Block (1)	$25 \times 25 \times 100$	1×1 conv 3×3 conv
Transition Block (1)	$12 \times 12 \times 90$	1×1 conv 2×2 average pool, stride 2
Dense Block (2)	$12 \times 12 \times 170$	1×1 conv 3×3 conv
Transition Block (2)	$6 \times 6 \times 153$	1×1 conv 2×2 average pool, stride 2
Dense Block (3)	$6 \times 6 \times 233$	1×1 conv 3×3 conv
Classification Layer	233	6×6 global average pool
		5 Dense, softmax

Table 3: DenseNet architecture. Each dense block is composed of 8 blocks, and the growth rate was fixed to $k = 8$.

Furthermore, during training, we used the same splitting ratio as discussed before and the network was trained using stochastic gradient descend (SGD), a batch size of 64 and 40 epochs. The learning rate varies from 0.002 to 0.0002[1]. Unexpectedly, simpler models like the ones discussed in the previous sections produced better results in terms of accuracy and performance. As shown in Fig. (15), we addressed how the validation accuracy and loss vary considerably during each epoch. This was not surprise at all, similar results during training were obtained by [7] using datasets like CIFAR-10 and CIFAR-100 with a higher number of classes and training epochs.[2]

[1]We tried several recall methods for the learning rate, none of which lead to an increase of accuracy nor performance.

[2]Due to GPU memory constraints, our experiments using DenseNet faced some difficulties. On the one hand, we could have trained more like [7] did, study the accuracy trend, and maybe obtain better results; on the other hand, the small learning rate implies longer training time, namely longer GPU availability time to do so.

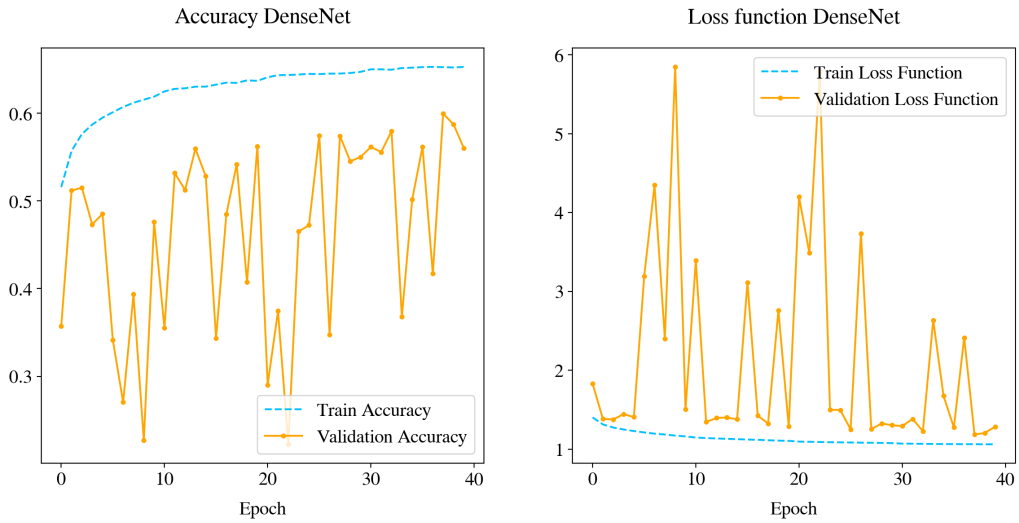

Figure 15: The history of accuracy (left) and loss (right) during the training of the ConvNet using the data in image representation. The final accuracy on the test set is 0.551 and the loss on the test is 1.304.

Moreover, the choice for the optimizer was first motivated by the original article [7]. But, since there is evidence that adaptive moment estimation (Adam) converges to dramatically different minima compared to SGD [14], we also tried training using Adam without obtaining any improvement in performance nor in accuracy.

Finally, as seen from figure (16), DenseNet is able to recognize (at least!) the top quark reasonably well, while may misclassify the other categories; for instance, all categories are often missclassified with gluons (specially light quarks), this being the highest source of inaccuracy. This misclassification of the Z boson as the W boson was also observed in previous models.

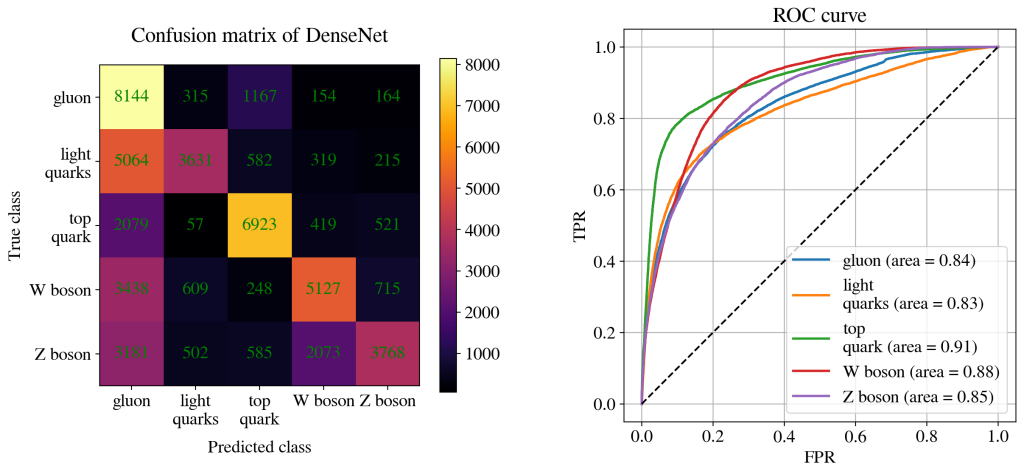

Figure 16: The confusion matrix and the ROC curves for every class in a One-versus-All setup of the ConvNet using the data in image format.

4 Results Overview

We now address the main results of the project by comparing the different models in terms of the accuracy and loss describing the validation and test sets obtained for each of the models.

/	FCNN on tensor data	FCNN on image data	ConvNet	DenseNet
Validation Accuracy	0.705	0.579	0.5916	0.551
Validation Loss	0.842	1.085	1.081	1.304

Table 4: Validation Accuracy and Validation Loss for every model summaried in a table.

In general, the most complex of our models, DenseNet, was the one less accurate when distinguishing between the five categories achieving an accuracy of 0.551 on the test set with a total loss of 1.304 as shown in Table (4). Our best model, namely the one which achieved the highest accuracy on the test set was, to our astonishment, the first fully connected model which classifies the jets using the tensor representation of their features. Initially, we expected to obtain better results by studying the jets using the image representation and CNNs. This was not the case at all; in fact CNNs (including DenseNet) reached the lowest accuracies of all of our models. The model classifies more accurately when inputs are represented as tensors, meaning that the individual physical properties of each jets[3] has a huge impact on the classification task, namely the prediction of the jet type.

5 Conclusions

We successfully studied the data by using two different representations and we were able to train several machine learning models to perform a classification task to distinguish between different jet types. Although the accuracies and performance of our models were not high enough in comparison with very advanced and efficient models for jet image classifications like MaxOut networks [5], we have gained valuable insights into preprocessing and training multi-class classification models. In particular, we showed that the particle-based representation is a very natural and generic way of representing jets and, by using this representation, we obtained the highest accuracy in our experiments with a fully connected neural network. We consider this project a solid first step into the expansive field of Machine Learning, a path that will allow us to refine these models so they excel in distinguishing between Z and W bosons, much like finding the straightforward answer to a seemingly complex riddle.

[3]The Physics of the process

References

[1] G. Aurélien. *Hands-on machine learning with scikit-learn, keras and tensorflow*. O'REILLY, 2019.

[2] François Chollet. *Deep Learning with Python, Second Edition*. Simon and Schuster, Dec. 2021.

[3] Luke De Oliveira et al. "Jet-images â deep learning edition". In: *Journal of High Energy Physics* 2016.7 (July 2016). DOI: 10.1007/jhep07(2016)069. URL: https://doi.org/10.1007/jhep07(2016)069.

[4] J. M. Duarte et al. "`hls4ml` LHC jet dataset (30 particles)". In: (2020). DOI: doi:10.5281/zenodo.3601436.

[5] Ian J. Goodfellow et al. "Maxout Networks". In: *arXiv (Cornell University)* (Jan. 2013). DOI: 10.48550/arxiv.1302.4389. URL: https://arxiv.org/abs/1302.4389.

[6] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *arXiv (Cornell University)* (Jan. 2015). DOI: 10.48550/arxiv.1502.01852. URL: https://arxiv.org/abs/1502.01852.

[7] Gao Huang et al. "Densely connected convolutional networks". In: *arXiv (Cornell University)* (Jan. 2016). DOI: 10.48550/arxiv.1608.06993. URL: https://arxiv.org/abs/1608.06993.

[8] *JetNet*. URL: https://github.com/jet-net/JetNet.

[9] Raghav Kansal et al. "JetNet: A Python package for accessing open datasets and benchmarking machine learning methods in high energy physics". In: *Journal of Open Source Software* 8.90 (2023), p. 5789. DOI: 10.21105/joss.05789. URL: https://joss.theoj.org/papers/10.21105/joss.05789.

[10] Raghav Kansal et al. "Particle Cloud Generation with Message Passing Generative Adversarial Networks". In: *NeurIPS* (2021). DOI: 2106.11535.

[11] Raghav Kansal et al. "Particle Cloud Generation with Message Passing Generative Adversarial Networks". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 23858–23871. arXiv: 2106.11535. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/c8512d142a2d849725f31a9a7a361ab9-Paper.pdf.

[12] Jovan Nelson and May. "Using Jet Images and Deep Neural Networks to Identify Particles in High Luminosity Collisions". In: 2017. URL: https://api.semanticscholar.org/CorpusID:140064314.

[13] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *Journal of Machine Learning Research* 15.1 (Jan. 2014), pp. 1929–1958. URL: https://jmlr.csail.mit.edu/papers/volume15/srivastava14a/srivastava14a.pdf.

[14] Ashia C. Wilson et al. "The marginal value of adaptive gradient methods in machine learning". In: *arXiv (Cornell University)* (Jan. 2017). DOI: 10.48550/arxiv.1705.08292. URL: https://arxiv.org/abs/1705.08292.