UNIVERSIDAD DE GRANADA

Facultad de Ciencias

GRADO EN FÍSICA

TRABAJO FIN DE GRADO

Implementation and applications of Machine Learning algorithms in science and industry.

Uso y aplicaciones de Machine Learning en el ámbito científico y de mercado.

Presentado por:
D. Rodrigo Castellano Ontiveros

Curso Académico 2019/2020

Resumen

El objetivo de este TFG es estudiar los principios y la metodología de algunos de los principales algoritmos provenientes del área del Aprendizaje Máquina. También se hace hincapié en Aprendizaje Profundo (Deep Learning), concretamente en redes neuronales hacia adelante y redes neuronales convolucionales. Estos conocimientos son aplicados al análisis de señales de rayos cósmicos para identificar qué partículas los producen. Para finalizar, se implementa una red neuronal en un ordenador cuántico y se comparan resultados con redes neuronales clásicas.

Abstract

The aim of this thesis is to study the principles and methodology of different machine learning algorithms. Deep Learning is also studied, in particular feedforward neural networks and convolutional neural networks. This knowledge is applied to the analysis of cosmic rays signals, in order to indentify what particles produces them. Lastly, a neural network is implemented on a quantum computer and the results are compared with classical neural networks.

I would like to thank my supervisors, Luis Javier Herrera and Carmen García, for giving me the opportunity of doing this thesis and reviewing it. I also want to thank Carlos Todero for allowing me to work with his data.

Contents

1 Introduction

The purpose of this thesis is to get to know Machine Learning (ML) from a theoretical and practical approach, as well as its state of art. This knowledge is used to create models that perform classification tasks.

ML is the core of Artificial Intelligence (AI) and Data Science. It is widely studied due to its potential to deal with several complex real applications in which it has provided amazing results. Some of them include face-recognition, autonomous cars, online fraud detection, product recommendations, traffic predictions, virtual person assistants and so forth. The base of autonomous cars are convolutional neural networks, which are explained and applied along the thesis.

Most of the ML algorithms were already well known since decades ago. For instance, neural networks were known since the 60's. Given the improvements in computation (both software and hardware) in the last years, together with the evolution of storage and collection capabilities, that have provided a huge amount of data available to work with, ML has provided excellent results in a wide range of areas.

It is successful not only in the industry but science as well. In this thesis, data from different simulators has been analyzed to help to distinguish what particles produce cosmic rays. Apart from physics, other fields that ML works on are, for example, drug discovery, bioinformatics, and a long etc.

For the first part of this thesis, structured data from Ultra-High Energy Cosmic Rays obtained using the CORSIKA simulator is analyzed, so as to classify primary-particles. The theoretical bases of supervised learning algorithms, such as Support Vector Machines, random forest or neural networks, among others, are introduced. Then, the models are applied and their results compared.

For the second part, simulations of cosmic rays from the Cherenkov Telescope Array are analyzed as images. Convolutional neural networks are introduced and later applied to this dataset, showing different comparisons between particle types.

In the last part of the thesis, a neural network is implemented on a quantum computer. Its results and the results of a classical neural network are compared.

Finally, conclusions are drawn. All the concepts and ideas learned throughout the thesis are exposed, as well as ML's impact on society. There is a discussion on how ML will evolve in the next years, and how quantum computers can help this task to be more efficient.

2 Classification of Ultra-High Energy Cosmic Rays

An analysis has been carried out in order to classify Ultra-high Energy cosmic rays [1]. The data has been obtained from the simulator CORSIKA (COsmic Ray SImulations for KAscade). This is a program that simulates extensive air showers. It can be used up to 100 EeV and even further. Air showers are produced because the particles react with the air nuclei, or they just decay [2]. This is explained in detail in section 3. An example of showers for photon, proton and iron are shown in figure 1.

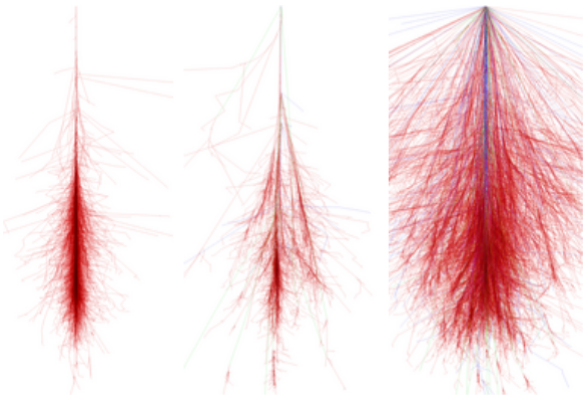


Figure 1: Air showers for photon, proton and iron [2].

The dataset is composed of about 40,000 examples, approximately 8,000 per particle. It is not a public dataset [1]. The features that were used for the analysis are the following:

- NALLParticlesTotal : Total number of particles generated by the event at the ground level.
- MUTotal: Total number of muons.
- ELTotal: Total number of electromagnetic particles.
- Zenith: Zenith angle of the particle (degrees).
- Energy: Particle energy (GeV).

The possible outcomes that were predicted are:

- 0: photon.
- 1: proton.
- 2: helium.
- 3: nitrogen.
- 4: iron.

For the classification, supervised learning algorithms were applied. This work will make use of the following well-known heterogeneous techniques: K-nearest neighbors, support vectors machines, decision tree, random forest, logistic regression and neural networks. The main bibliography used for this section is [3] and [4].

Let's have a glimpse at the data. The correlation between the different variables can be observed graphically in figure 2 (note that in the diagonal the distribution of each variable is shown). Thanks to this plot the distribution of the variables can be examined, together with how they are related. It is remarkable that the total number of electromagnetic particles and the total number of particles generated by the event at the

ground level are completely correlated. This implies that one of the two variables may be dispensable. Beyond, Energy has an 80% correlation with all the variables except for Zenith. Principal component analysis (PCA) could be applied to reduce the amount of data, keeping most of the information at the same time [5]. Since there are few variables, this is not needed. After a theoretical introduction to the algorithms, the results will be presented and compared.

2.1 Data splitting and methods assessment

During the data analysis is fundamental to follow procedures that ensure good results. In this section, the methodology needed to work on data is presented.

First of all, the data has to be randomly shuffled. Then, it has to be divided into train set and test set. In this way, once the model is created with the train set, it can evaluate non-biased data (test set). Since there might be variables with different units, the data has to be normalized, both train and test sets. They need to have the same distribution (equal mean and standard deviation). Generally, the train set is 75% of the dataset and the test set is 25%. This is not a fixed value, it depends on the size of the dataset.

Besides, a technique called cross-validation is applied to ensure that the results are consistent by doing many times train/test split of the data and combining the results.

2.1.1 K-fold Cross-Validation

This method uses the train set to check the performance of a model. It is less biased than the conventional train-test splitting method. The main idea is to split the train set into K parts (the data should be equally divided). A selected part is used as the test set and the other (K-1) parts are used as the train set (figure 15). Then, the model is trained with the train part and assessed with the test part. This is done for every fold (K times in total). At the end, the estimation of the errors from every iteration is combined [6]. A typical value of K is five or ten. A value of K=5 is chosen for the analysis.

2.1.2 Grid and random search

In every algorithm there are some parameters that can be arbitrarily chosen. To find these optimal values, **grid search** can be performed. These parameters that are optimized are called hyperparameters. A range of values is selected for each hyperparameter and then the model is trained and tested with every possible combination of them. For every hyperparameter there are some common values to start with. Usually, the logarithmic scale is preferred. This procedure has a very high computational cost. For m hyperparameters, the number of times that the model has to be trained grows as $O(n^m)$ [3].

There is another technique called **random search**. This is similar to grid search, but it tests values of parameters randomly. It might not perform a so thorough coverage of the domain of the hyperparameters considered, but in much less time it may perform very successfully [8]. Frequently, cross-validation and grid search are used for hyperparameter optimization.

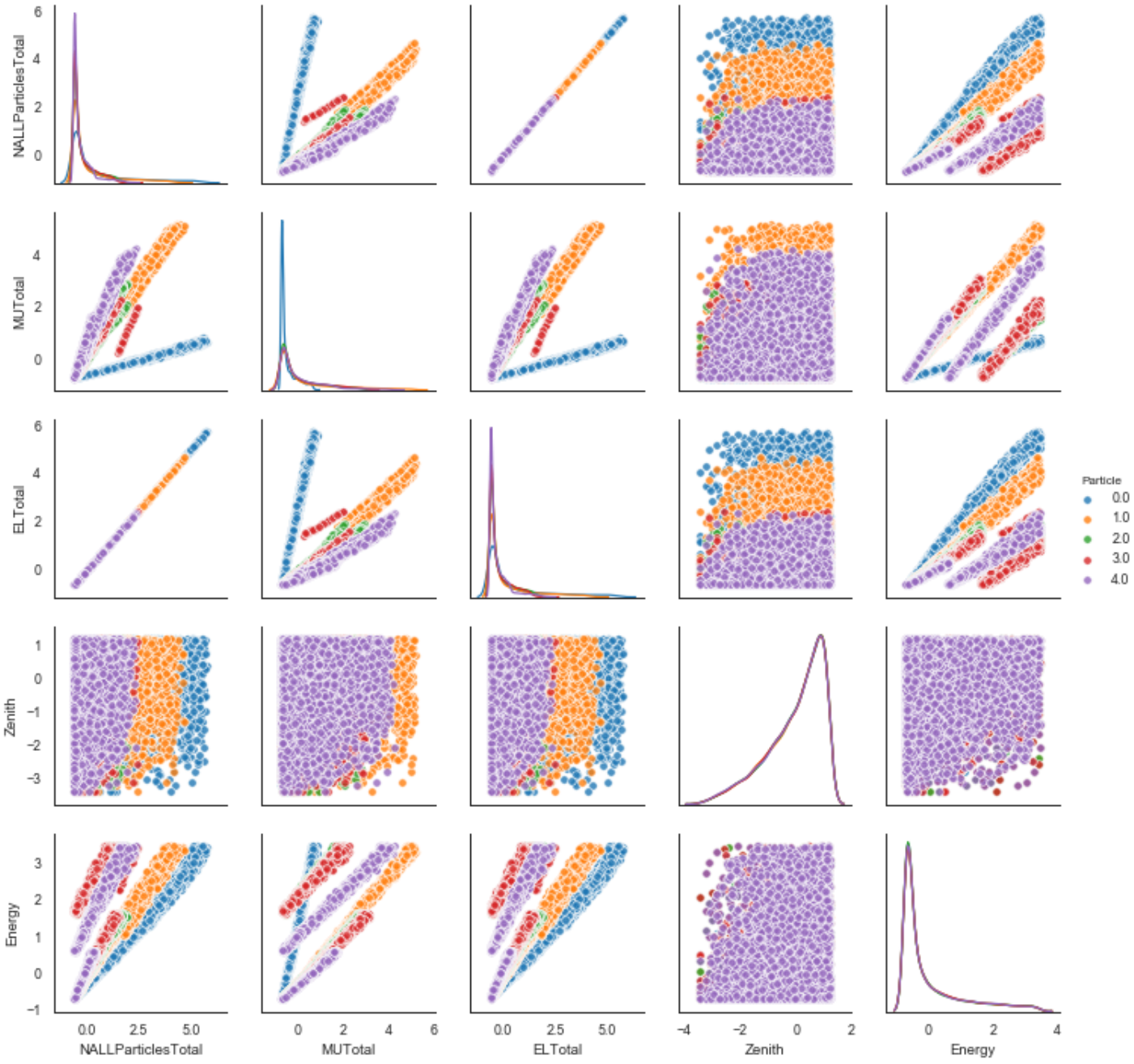


Figure 2: Pair plot.

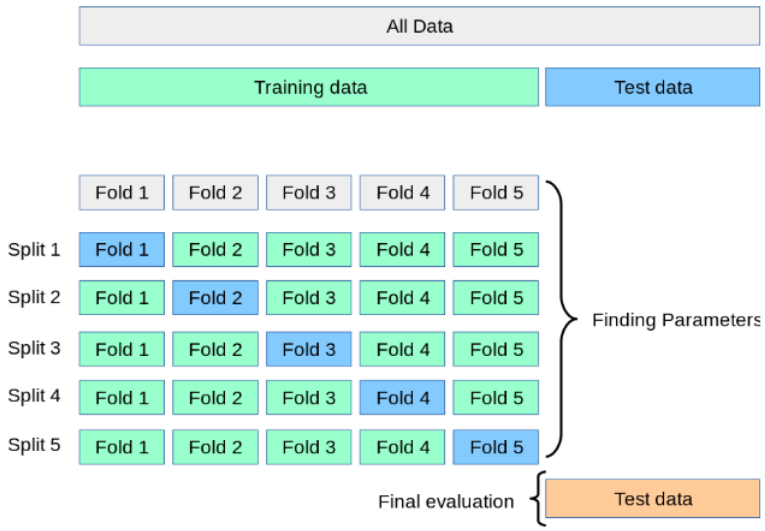


Figure 3: Cross-validation process for K=5 [7].

2.1.3 Model performance

To evaluate the results, for every algorithm a classification report is presented. The most relevant concepts are the following:

- TP is true positives (number of samples that were predicted to be true and were actually true).
- FN is false negatives (number of samples that were predicted to be false and were actually true).
- FP is false positives (number of samples that were predicted to be true and were false).
- Precision = TP/(TP+FP) (Precision measures the accuracy of positives predictions).
- Recall = TP/(TP+FN) (Recall measures the positives correctly predicted).
- F1 Score = 2*(Recall * Precision) / (Recall + Precision).
- Support: Number of samples used from a certain class belonging to the dataset.
- In weighted average, the relative number of examples from each class is weighted according to the portion of data that it represents in the dataset. On the other side, in macro average, each class is weighted equally.

An example is shown in table 1

2.2 K-nearest neighbors

K-nearest neighbors (KNN) is an algorithm that chooses a class for a sample based on the K closest neighbors. For a point x_0 to be predicted, the goal is to find the K closest

	precision	recall	f1-score	support
0.0	0.96	0.99	0.98	2109
1.0	0.75	0.80	0.77	1900
2.0	0.62	0.56	0.59	2247
3.0	0.61	0.57	0.59	2280
4.0	0.76	0.82	0.79	1962
accuracy			0.74	10498
macro avg	0.74	0.75	0.74	10498
weighted avg	0.74	0.74	0.74	10498

Table 1: Example of a classification report that will be used for every algorithm. In this case there are four classes.

points to it belonging to the train set, and then apply majority vote to the classes from these K closest points. In case of an equal number of classes, the class belonging to the closest point is chosen. If still there is a tie, the class is randomly chosen [6].

Talking about distance implies choosing a metric, and in this case the model has been evaluated with the euclidean metric. For two vectors $\mathbf{x}$, $\mathbf{y}$ in an n-dimensional vector space,

$$d = \sum_{i=1}^{n} \sqrt{(x_i - y_i)^2} \tag{2.1}$$

Other metrics are also plausible, such as the Manhattan metric $d = \sum_{i=1}^{n} \mid x_i - y_i \mid$.

2.3 Support Vectors Machines

Support Vectors Machines (SVMs) separates classes by using n-dimensional hyperplanes. First, let's discuss the algorithm for binary classification; later, multi-class classification will be approached.

A sample in the dataset is $X = \{x^t, r^t\}$. $r^t = 1$ if $x^t \in C_1$ and $r^t = 2$ if $x^t \in C_2$. The aim is to find the pair $\{\mathbf{w}, w_o\}$ that satisfies

$$\begin{cases} \mathbf{w^T x^t} + w_o \geq +1 & for \;\; y = +1 \\ \mathbf{w^T x^t} + w_o \leq +1 & for \;\; y = -1 \end{cases} \tag{2.2}$$

This is equivalent to

$$y^t(\mathbf{w^T x^t} + w_o) \geq +1 \tag{2.3}$$

w_o describes the location of the plane with respect to the origin and $\mathbf{w}$ describes its orientation. There is an optimization problem that implies maximizing the margin. The margin is the distance between the hyperplane and the closest vectors to the hyperplane (Support Vectors), as shown in figure 4 [4].

To maximize the margin, $\frac{1}{2} \left\|\mathbf{w}^2\right\|$ has to be minimized. This can be done by using Lagrange multipliers α^t, such that

$$L = \frac{1}{2} \left\|\mathbf{w}^2\right\| - \sum_{t=1}^{N} \alpha^t [y^t(\mathbf{w^t x^t} + w_o) - 1] \tag{2.4}$$

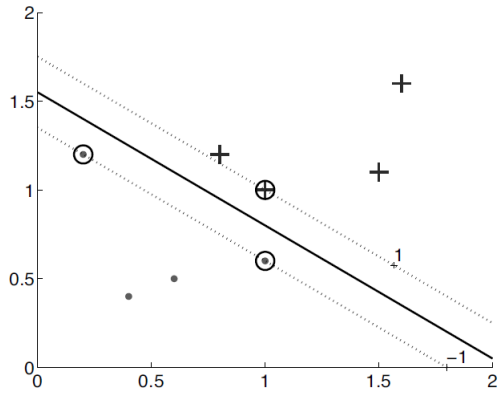


Figure 4: SVM for binary classification [4].

Thanks to this procedure, we can also get the support vectors, the ones that give information (the closest to the discriminant). These support vectors satisfy the condition $\alpha_i \neq 0$.

In case of non-separable data, a penalty term $C\sum_t \xi^t$ is added to the Lagrangian. ξ tells whether there is a incorrect classification, and C is a hyperparameter to be tuned, that helps to control overfitting [4].

2.3.1 Kernels

To use the model as non-linear, kernel functions are used, by mapping every point into a higher dimensional space. Kernel functions correspond to a inner product in a certain expanded feature space. The problem is mapped from d dimensions to a new larger k-dimensional space, called dual space, by means of non linear transformations, and then in the new space linear transformations can be applied [4].

The basis function used is $\boldsymbol{\phi}(\boldsymbol{x})$. Instead of $g(\mathbf{x}) = \mathbf{w^T x^t} + w_o$, the discriminant is now

$$g(\mathbf{x}) = \mathbf{w^T}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{j=1}^{k} w_j \phi_j(\mathbf{x}) \tag{2.5}$$

The so-called kernel trick in SVMs, involves replacing $\boldsymbol{\phi}(\boldsymbol{x^t})^T\boldsymbol{\phi}(\boldsymbol{x})$ by $K(\mathbf{x}^t, \mathbf{x})$ in the lagrangian and the discriminant. The discriminant results in

$$g(\mathbf{x}) = \mathbf{w^T}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_t \alpha^t y^t \boldsymbol{\phi}(\boldsymbol{x^t})^T\boldsymbol{\phi}(\boldsymbol{x}) = \sum_t \alpha^t y^t K(\mathbf{x^t}, \mathbf{x}) \tag{2.6}$$

This is faster because $\mathbf{x^t}$ and $\mathbf{x}$ are no longer transformed to the dual space. A kernel function is directly applied (same for the Lagrangian) in the original space [4].

A well-known effective kernel is the Gaussian kernel

$$K(x^t, x) = exp[-\frac{\|\mathbf{x^t} - \mathbf{x}\|}{2s^2}] \tag{2.7}$$

where $\mathbf{x^t}$ is the center and s is the radius. Under the operation using this kernel, two hyperparameters have to be optimized to train an SVM, C and s.

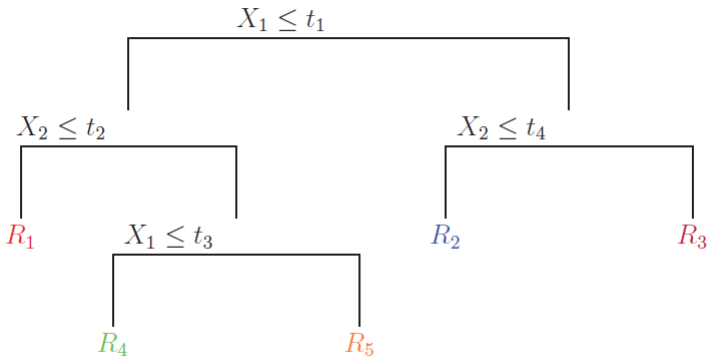


Figure 5: Structure of a tree [9].

2.3.2 Multiclass SVMs

For multi-class classification, the technique one vs. all is commonly used. This is based on doing binary classification for each class, by comparing every class against the rest of the classes. Also, a confidence value is given with each classification, so that that class with the highest confidence is selected [4].

2.4 Decision Tree

Decision tree is an algorithm that splits the data and classifies it by emulating the structure of a tree. In figure 5 a graphical description of a decision tree is shown. It has nodes, branches and leaves. In each node a feature is tested, i.e., in every node a threshold t_i is applied to a feature X_i. In the branches an outcome is given, based on the threshold that allows to give the best classification. The leaves are the outer nodes, and in them a class is assigned to the arriving sample.

For each feature, there is a list of thresholds to be tested. The algorithm looks for information gain for every threshold in every feature. Once the threshold that gives the maximum information gain is found, the node is split into two branches based on that threshold. For each branch the procedure mentioned above is repeated. This is done until the information gain does not increase.

When optimizing the tree there are other reasons to stop this process, such as a certain depth of the tree or a minimum amount of data in each leaf. When the process comes to an end, based on the portion of samples selected for each class in the last node (leaf), the class labeled with more data is assigned to the sample [9].

Information gain is equivalent to purity (contrary of impurity). For binary classification C can be divided into two groups of data, C_1 and C_2. It can be easily generalized for multi-class classification. For every sample in C_1 and C_2, it belongs either to one class or the other, i.e., C^+ or C^-. The threshold of a feature when classifying successes if the impurity is zero ($C_1^+ = C^+$ and $C_1^- = \varnothing$, or all the way round). For a dataset with n^+ and n^- samples, the impurity is defined as $\dot{p} = \frac{n^+}{n^+ + n^-}$ [10].

The two main expressions to measure the impurity of a split are the Gini index and entropy [10]:

$$\begin{cases} Gini\ index: \ \dot{p}(1-\dot{p}) \\ Entropy: \ -\dot{p}log_2\,\dot{p} - (1-\dot{p})log_2\,(1-\dot{p}) \end{cases} \tag{2.8}$$

Pruning is a technique used to reduce overfitting, when for example there is not enough data in a leaf to train and classify based on it. Taken decisions when there is not enough data turn out in high variance, so this technique cares about these cases by not including them in the model [4].

2.5 Random forest

Random forest is an ensemble method based on decision trees. The difference is that in random forest many trees are built. The performance of all the trees is combined by the majority vote. Another important difference is that in every tree a random number of features is considered. Usually, the square root of the total number of features is considered. This is done to decorrelate the trees, i.e., if there is a variable with a strong power of prediction, most of the trees will have that variable in the top split [11].

Some of the most important hyperparameters to tune are the number of trees, the maximum number of features considered when splitting a node, the maximum depth of the trees, and the minimum number of samples in a node before it is split.

2.6 Logistic regression

This is a prelude for the analysis of a neural network. In multi-class classification, for each class, the probability of a sample $\{\mathbf{x}^{(i)}|i \in (1,...,m)\}$ belonging to a class C_k (out of K classes) is defined as

$$\hat{p}(C_k \mid \mathbf{x}^{(i)}) = y_k^{(i)} = \frac{exp(\mathbf{a}_k^{(i)})}{\sum_{j=1}^{K} exp(\mathbf{a}_k^{(i)})} \tag{2.9}$$

where $\mathbf{a_k}$ is defined by $\mathbf{a_k} = \mathbf{w_k^T x} + \mathbf{b}$.

The error function (also called cost function) used for this algorithm is called cross-entropy error, mainly used for classification, and its expression for multi-class classification is

$$J(\mathbf{w}, \mathbf{b}) = \frac{-1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} [y_k^{(i)} log \hat{p}_k^{(i)}] \tag{2.10}$$

where y_k is 1 for the target class and 0 otherwise. For binary classification the cost function is shown in figure 6. If the predicted value is right, the cost function goes to zero [12].

Gradient Descent

The gradient of the cost function is performed to minimize said function. For a class k,

$$\nabla_{w_k} J(\mathbf{w}, \mathbf{b}) = \frac{1}{m} \sum_{i=1}^{m} [\hat{p}_k^{(i)} - y_k^{(i)}] \tag{2.11}$$

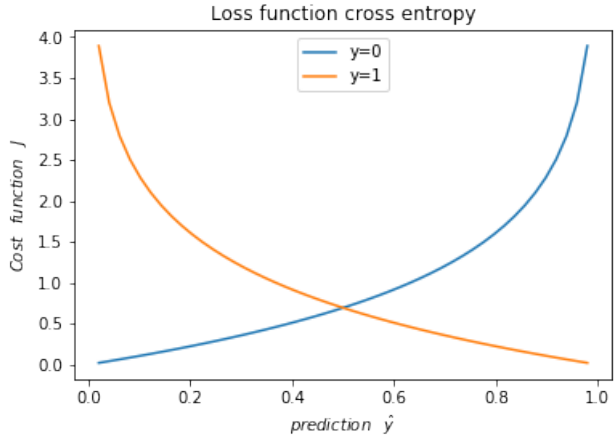


Figure 6: Cost function cross-entropy [12].

The parameters {w,b} can be updated as many times as desired

$$\begin{aligned} \mathbf{w}_k &= \mathbf{w}_k - \alpha \nabla_{\mathbf{w}_k} J \\ \mathbf{b}_k &= \mathbf{b}_k - \alpha \nabla_{\mathbf{b}_k} J \end{aligned} \tag{2.12}$$

α is the learning rate and it is a hyperparameter that can be tuned [13, 12].

2.7 Neural networks

Neural networks are widely used to approximate functions. Nowadays, under the so-called Deep Learning paradigm, they are outperforming many other top methods in a wide range of problems, due to three main reasons: increase of available data, development of more efficient computers and improvement of ML algorithms.

A neural network is an algorithm to work on linear and non-linear models, for both regression and classification. In a neural network there is first the input layer, with as many neurons as features in the dataset, and then as many layers as desired, followed by an output layer, that gives the prediction. The output layer, depending on the task, has different neurons; for example, for K-classes it has K neurons. Two important parameters to optimize are the number of neurons in each layer, and the number of layers. The structure of a layer i is the following:

$$z^{(i)} = \sigma^{(i)}(\mathbf{w}^{(i)T}\mathbf{x} + \mathbf{b}^{(i)}) \tag{2.13}$$

z is obtained after doing a linear transformation to $\mathbf{x}$ by means of $\mathbf{w}$, the weights, and $\mathbf{b}$, the bias (both learnable parameters). In the first layer, $\mathbf{x}$ is fed with the dataset. After the linear transformation, a non-linear transformation is performed thanks to the activation function σ [4].

There are many activation functions, the most common activation function is called *relu* (*relu* stands for rectified linear unit) such that $\sigma(z) = max\{0, z\}$. Depending on whether the model is created for regression or classification, the activation function in the last layer, g, is different. It is the identity function for regression, the sigmoid function for

binary classification and the softmax function for multi-class classification. The softmax function for a class k is described as

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}} \tag{2.14}$$

where K is the total number of classes.

All the layers except for the input and output layer are called hidden layers (the values of their parameters are hidden). If the activation function is the identity function, there is merely a linear model. Note that a neural network with one layer is very similar to logistic regression.

The cost function used depends again on whether working on regression or classification. For regression MSE is used and for classification either MSE or cross-entropy is used.

Gradient Descent

In order to minimize the cost function, gradient descent is used. In logistic regression this was applied to cross-entropy, and here it is applied to MSE. For a single hidden layer neural network, also called perceptron, there are p features or p units in the input layer, M units in the hidden layer and K-classes or K units in the output layer.

The set of biases and weights is composed of $(\alpha_{0m}, \alpha_m; m = 1, ..., M)$ for the hidden layer, and $(\beta_{0k}, \beta_m; k = 1, ..., K)$ for the output layer, which makes a total of $M(p+1)$ weights for the hidden layer and $K(M+1)$ weights for the output layer. Generally, if $z_{mi} = \sigma(\alpha_{om} + \alpha_m^T x_i),\ m = 1, ..., M$ and $z_i = \{z_{1i}, z_{2i}, ..., z_{Mi}\}$ the cost function is

$$R = \sum_{i=1}^{N} R_i = \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2 \tag{2.15}$$

and the derivatives

$$\begin{aligned} \frac{\partial R_i}{\partial \beta_{km}} &= -2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) z_{mi} \\ \frac{\partial R_i}{\partial \alpha_{ml}} &= -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) \beta_{km} \sigma'(\alpha_m^T x_i) x_{il} \end{aligned} \tag{2.16}$$

To update the parameters,

$$\begin{aligned} \beta_{km}^{(r+1)} &= \beta_{km}^r - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}} \\ \alpha_{ml}^{(r+1)} &= \alpha_{ml}^r - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}} \end{aligned} \tag{2.17}$$

where γ is the learning rate, a hyperparameter to be tuned [6].

2.7.1 Improving a neural network

2.7.1.1 Weight initialization

Note that the initial values of the weights cannot be zero, otherwise the same operations would be performed for every weight and there would be a symmetry in results (the same function would always be computed). The bias is usually set to be zero and the weights to small random values. It is easy to see why the weights are set to be small random values. For instance, in the sigmoid function, for large values the gradient tends to be very small (flat part of the sigmoid function). Therefore, the algorithm optimization is slow. A very effective initialization is the Xavier and He normal initialization [14].

2.7.1.2 Regularization

When there is overfitting a technique called regularization is applied. There are some important methods. Regularization consists of reducing the number of nodes in the network by removing weights from the layers.

The first one is called **early stopping**, which consists of avoiding to have too many parameters and stop training before reaching the global minimum. The problem with this is that the model might stop too early and thus being similar to a linear model [6].

The so-called **weight decay** is commonly used. It implies adding an extra term to the cost function [15], i.e.,

$$R(w) = MSE + \lambda \mathbf{w}^{\mathbf{T}}\mathbf{w} \tag{2.18}$$

where λ is a hyperparameter. A large λ makes the weights smaller. The results depending on λ for a high polynomial degree fit are shown in figure 7. If λ tends to zero, regularization is not applied anymore and therefore there is still the problem of overfitting. If λ is excessively large, there might be a problem of underfitting, which means that the model is too general. What is more, the term added can be seen as $\mathbf{\Omega}(\mathbf{w}) = \mathbf{w}^{\mathbf{T}}\mathbf{w}$, and other regularizers $\mathbf{\Omega}(\mathbf{w})$ can be applied [3].

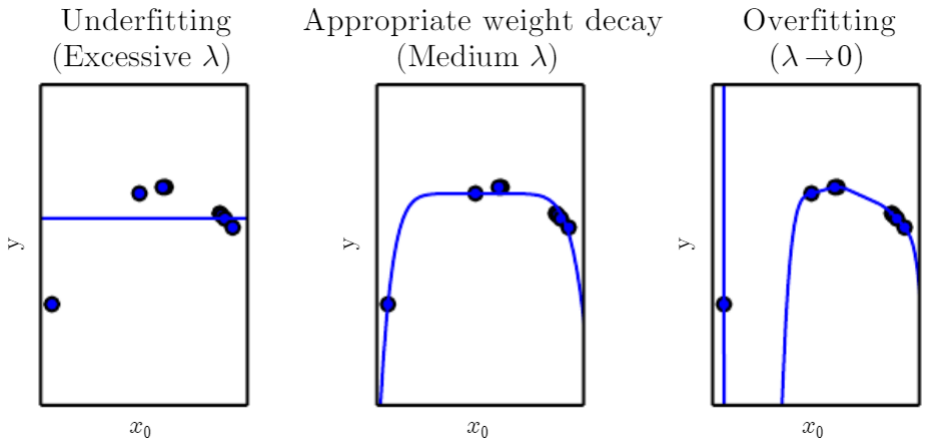


Figure 7: High degree polynomial fit depending on λ [3].

Another important method is called **dropout**. The concept is very simple. For each

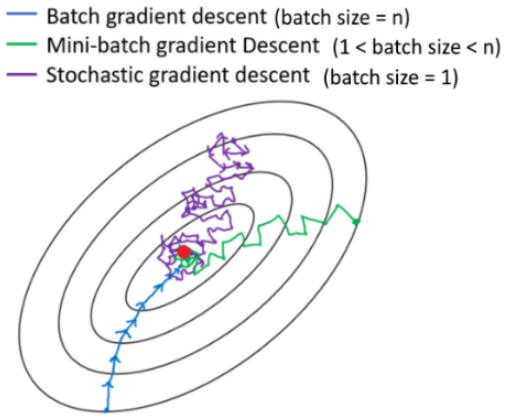


Figure 8: Reaching the optima depending on the mini batch size [17].

hidden layer, it sets a probability of removing nodes. For instance, if dropout is 0.5, approximately half of the nodes will be removed from each layer [16].

2.7.1.3 Epochs and mini batch

Another point that worths mentioning is epochs and mini batch, given that it makes the process be faster by using parallel computing.

An epoch is a whole iteration of the complete training dataset over the neural network, by calculating the cost function (feed-forward propagation) and minimizing it (back-propagation).

Mini batch consists of taking advantage of parallel computing and dividing the dataset into small groups, so that these groups of data can be analyzed at the same time. The conventional technique is called batch. Therefore, selecting as a mini batch the whole dataset is the same as the batch technique. On the other side, a mini batch size of one sample may lead to plenty of noise. This can be visualized in figure 8. There may be a gain in noise in favor of a gain of speed. The reason why is that each loss function is different from the previous one [17].

Power of 2 batch offers a better runtime, especially when using GPU [3].

2.7.1.4 Momentum

This is created to speed up the gradient convergence, by taking into account not only the current gradient, but also the previous ones. This is done by adding the term *velocity* (V). In this way the fit can be smoother [18]. The expression to update the weights is

$$\begin{aligned} w_{t+1} &= w_t - \alpha V_t \\ V_t &= \beta V_{t-1} + (1-\beta)\frac{\partial L}{\partial w_t} \end{aligned} \tag{2.19}$$

The update is accelerated when the velocity goes in the same direction of the gradient, and it is slowed down when the gradient changes its sign. β is a hyperparameter

frequently set to 0.9. α is another hyperparameter, the learning rate. The velocity is calculated for every parameter $\{w_t, b_t\}$[19].

If the bias is initialized to zero, a correction has to be applied such that V_t is replaced by $\frac{V_t}{1-\beta^t}$. As the iteration t gets large $(1-\beta^t)$ tends to zero [17].

2.7.1.5 Adam optimization algorithm

Adam optimization algorithm makes use of momentum and introduces a new characteristic; it also takes into account the square gradients, thanks to S. The new updates are

$$\hat{V}_t = \frac{V_t}{(1-\beta_1^t)} \tag{2.20}$$

$$S_t = \beta_2 S_{t-1} + (1-\beta_2)[\frac{\partial L}{\partial w_t}]^2 \tag{2.21}$$

$$\hat{S}_t = \frac{S_t}{(1-\beta_2^t)} \tag{2.22}$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{S}_t + \epsilon}}\hat{V}_t \tag{2.23}$$

Typical values are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. ϵ is added not to divide by zero [20, 19]. α is usually a hyperparameter to be tuned.

2.7.1.6 Learning rate decay

If a fixed value of α is always used, there might be no convergence, thereby the solution is to slowly reduce alpha. There are many possibilities, such as $\alpha = 0.95^{epoch\_num}\alpha_0$ [17].

2.7.1.7 Batch normalization

As the input layer is normalized (the dataset), the same can be done with the hidden layers. This technique consists of normalizing every mini-batch to speed up learning. Basically it takes the variance and mean in every mini batch in order to have $\mu = 0$ and $\sigma = 1$. It can be done to the function z or the activation function. If applied to z,

$$\mu = \frac{1}{m}\sum_i z^i \qquad \sigma^2 = \frac{1}{m}\sum_i (z_i - \mu)^2 \tag{2.24}$$

$$z^i_{norm} = \frac{z^i \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{2.25}$$

$$\tilde{z}^i = \gamma z^i_{norm} + \beta \tag{2.26}$$

γ and β are learnable parameters [17, 21].

There are plenty of parameters that can be tuned, such as the learning rate, the number of epochs, the batch size, the number of hidden layers and hidden units, the choice of

activation function, the regularization technique, etc.
For high bias (underfitting), a bigger network can be trained, a larger dataset can be used or a different neural network architecture. For high variance (overfitting), generally more data is a good solution, as well as implementing regularization.

2.8 Results

In this section, results from every algorithm are discussed and then compared. For the analysis of the data python was used (version 3.7). The open-source web application Jupyter is where the code is written. The main libraries used are numpy, pandas, matplotlib, sk-learn, TensorFlow 2.0 and Keras. The code can be found in my personal GitHub repository [22].

KNN

In KNN the hyperparameter that was tuned is K (the metric can also be tuned). To choose the K value with the highest accuracy, grid search was performed. Figure 9 shows the obtained results. The accuracy gradually descends as K is larger. The highest accuracy is given by K=1.

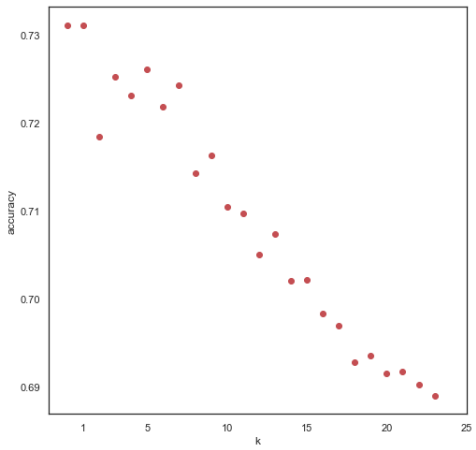


Figure 9: Most accurate values for K from 1 to 24.

The decision boundary can be visualized in 2D by taking two variables. This is useful to visualize to what class a point belongs to, depending on where it ”falls” on the plot. With the variables Energy and ElTotal, the boundary decision is shown in figure 10. The classification report can be consulted in table 2.

SVM

To build the model, a gaussian kernel was used. Two hyperparameters were tuned. The first one is C. It controls how many unclassified samples might be tolerated, and thus it is tuned to avoid underfitting. The second parameter (in the case of the gaussian kernel) is the radius *s*. *s* gives an idea of the influence of each train sample in the input space. Depending on the literature, *s* can be also called *Gamma*. The results for grid search are shown in figure 11.

The best values of s and C are 80 and 50,000, respectively. There was a resulting accuracy of 97%. No further search has been done, given that the improvement in accuracy was

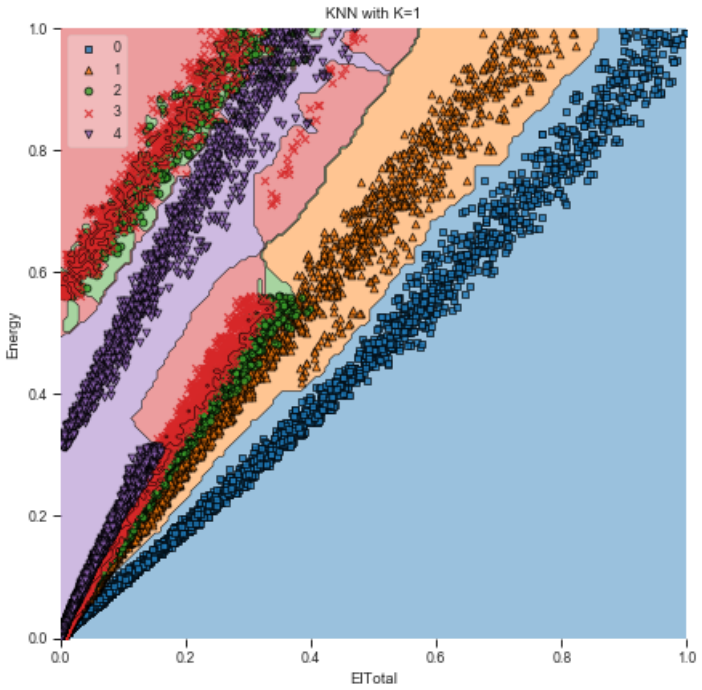


Figure 10: Decision boundary for ElTotal and Energy.

	precision	recall	f1-score	support
0.0	0.96	0.99	0.98	2109
1.0	0.75	0.80	0.77	1900
2.0	0.62	0.56	0.59	2247
3.0	0.61	0.57	0.59	2280
4.0	0.76	0.82	0.79	1962
accuracy			0.74	10498
macro avg	0.74	0.75	0.74	10498
weighted avg	0.74	0.74	0.74	10498

Table 2: Classification report for KNN

not very important. The obtained classification report results are shown in table 3.

Decision Tree

Entropy was proven to give more accuracy than Gini index. There are many hyperparameters to be tuned in decision trees, but the ones that were taken into account are the maximum depth (range of [3,100]) and the criterion (entropy and Gini index). The chosen depth was 19, but for the sake of visualization, a tree with depth 3 is shown in figure 12. In each node is exposed:

1) The chosen feature and its threshold.

2) The resulting entropy.

3) The number of samples used in the node.

4) The number of samples classified in each class.

Another interesting plot is presented in figure 13, with the importance of each variable

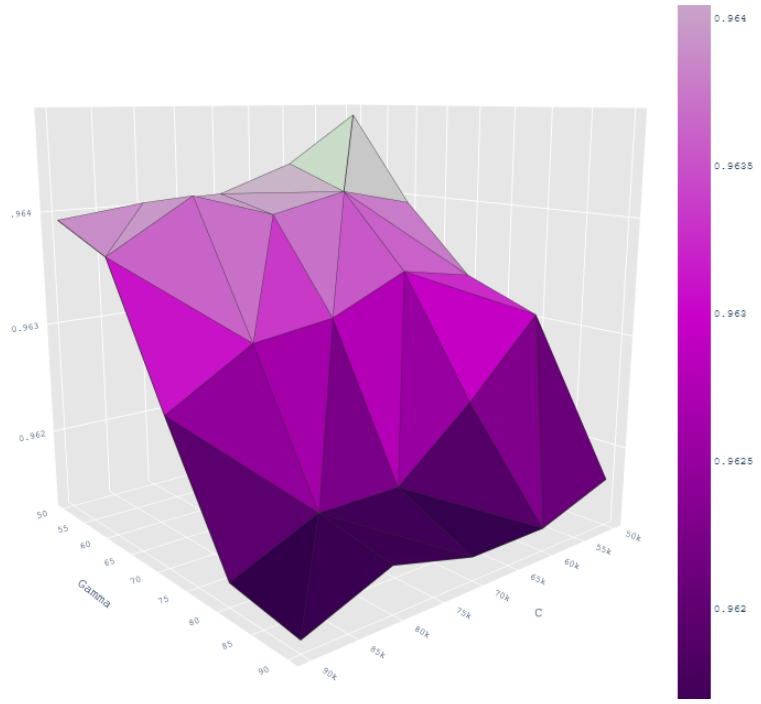


Figure 11: Grid search hyper-parameter optimization of the SVM model.

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	2099
1.0	0.99	0.99	0.99	2075
2.0	0.94	0.93	0.93	2096
3.0	0.93	0.94	0.94	2120
4.0	0.98	0.98	0.98	2108
accuracy			0.97	10498
macro avg	0.97	0.97	0.97	10498
weighted avg	0.97	0.97	0.97	10498

Table 3: Classification report for SVM.

in the decision tree. A variable is important depending on its information gain. The resulting accuracy was 95%. The classification report is shown in table 4.

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	2114
1.0	0.97	0.96	0.97	2129
2.0	0.90	0.92	0.91	2113
3.0	0.92	0.91	0.91	2130
4.0	0.97	0.97	0.97	2012
accuracy			0.95	10498
macro avg	0.95	0.95	0.95	10498
weighted avg	0.95	0.95	0.95	10498

Table 4: Classification report for Decision tree

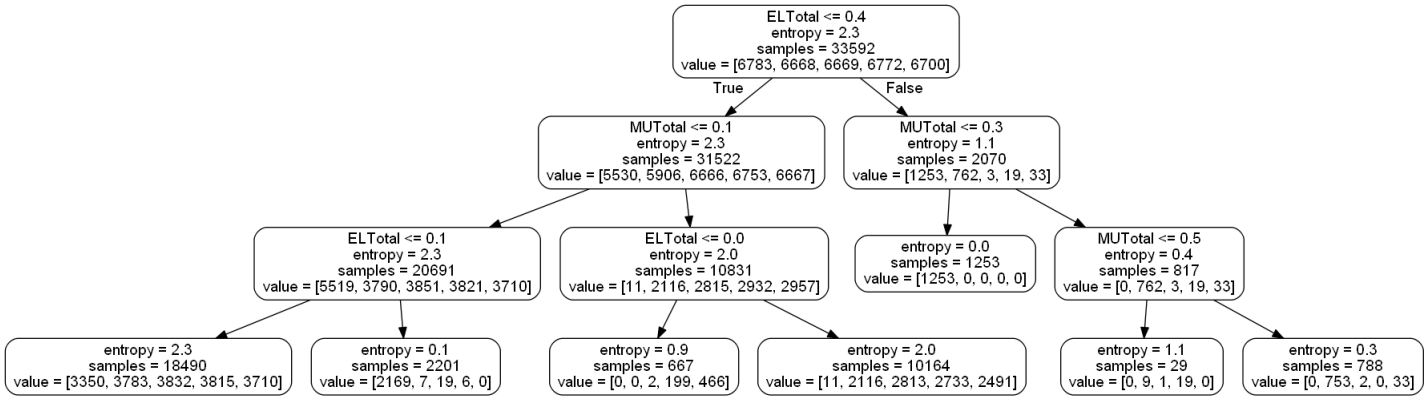


Figure 12: Tree obtained for depth 3.

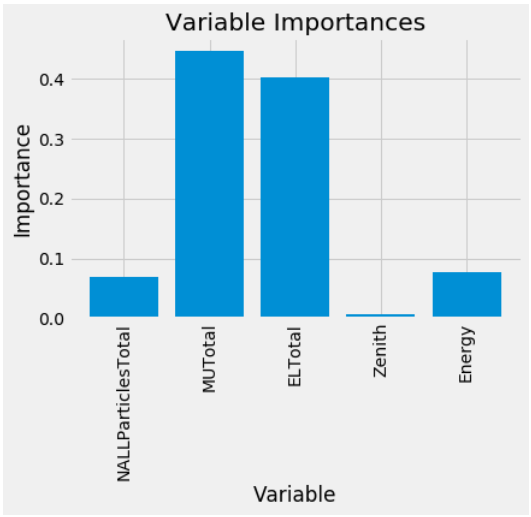


Figure 13: Importance of the variables in the decision tree.

Random Forest

With grid search the same hyperparameters have been tested, but this time the number of estimators was 1000. This was the best value in [100, 500, 1000]. The maximum number of features used chosen was 3, in the range [2, 5]. With this set of hyperparameters, 97% accuracy was obtained. Regarding the importance of the variables, the plot obtained was very similar to the one in decision tree.

The classification report is presented in table 5. This algorithm, along with SVM and neural network, gave the best results.

Logistic regression

For this dataset, the number of iterations was 5000, with a learning rate $\alpha = 0.1$.

The accuracy was 63%, but it could be known beforehand that the accuracy would not be great. Better results can be obtained, but not a great accuracy as other models do. The purpose was to use it as an introduction for a neural network.

Neural network

The model was built with 3 hidden layers and an output layer with 5 nodes and the

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	1705
1.0	0.99	0.98	0.99	1652
2.0	0.94	0.94	0.94	1683
3.0	0.93	0.95	0.94	1676
4.0	0.99	0.97	0.98	1683
accuracy			0.97	8399
macro avg	0.97	0.97	0.97	8399
weighted avg	0.97	0.97	0.97	8399

Table 5: Classification report for Random forest

softmax activation function. For the input layers, they had 128 nodes, the relu activation function and the initializer Xavier and He uniform. The number of epochs was set to 500 and the batch size to 128. This comes from doing random grid search. The tuned parameters were:

Batch size: [128, 264], init mode: [he normal, he uniform], neurons: [32, 64, 128, 256], optimizer: [Adam, Nadam, Adagrad].

The accuracy for the train set and the validation set is shown in figure 14. It is clear that for the test set there is more noise than for the train set. It might be due to different factors, such as the batch size or the initialization method.

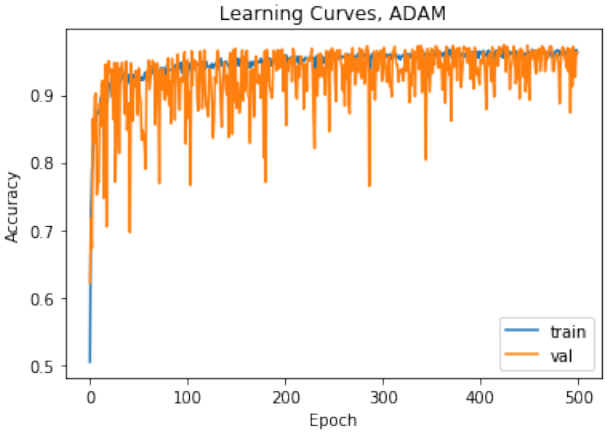


Figure 14: Accuracy for train and test sets.

The best given result was chosen for the final model, and a accuracy of 97% was obtained. The results are presented in table 6.

2.8.1 Comparison of the results

Table 7 compares all the results. The algorithms that gave the best results are SVM, Random forest and the neural network. The performance of random forest was slightly better than decision tree, given that random forest might be seen as an optimized version of decision tree. On the other side, KNN and logistic regression gave the worst result, with a 74% and a 63%, respectively. The result of logistic regression might be because

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	2151
1.0	0.96	1.00	0.98	2087
2.0	0.92	0.96	0.94	2108
3.0	0.97	0.91	0.94	2091
4.0	1.00	0.97	0.98	2061
accuracy			0.97	10498
macro avg	0.97	0.97	0.97	10498
weighted avg	0.97	0.97	0.97	10498

Table 6: Classification report for the neural network.

it is a shallow model, compared to other models such as neural network. In terms of efficiency, there are two parts to take into account: first, the optimization of the model and then the training. For all the algorithms, the training has been quite fast, i.e., few minutes on a average current laptop. For the optimization it can take as much as desired. It depends on the number of hyperparameters and their range of values to optimize.

Algorithm	Accuracy (%)
KNN	0.74
SVM	0.97
Decision tree	0.95
Random forest	0.97
Logistic regression	0.63
Neural network	0.97

Table 7: Comparison of the different algorithms.

3 Cherenkov Telescope Array. Particle classification

Cherenkov Telescope Array (CTA) is a set of gamma rays detector located in Chile and La Palma (Spain) [24]. When a cosmic ray (a high energy particle) reaches the atmosphere, there is an interaction leading to cascades of subatomic particles. Since light travels 0.03 times slower in air than in vacuum, the particles go faster than the speed of light in the air, creating the so-called Cherenkov light. This happens in billions of a second, but CTA is able to detect it. The energy that the telescopes cover ranges from 20 GeV to 300 TeV [24].

The distribution of telescopes of different sizes is shown in figure 16, as well as the image provided by the telescope, that is fed into the model. The values of the pixels in every image is given by the number of photoelectrons, that can be converted to energy. The data can be analyzed as hexagonal images, therefore a convolutional neural network can be used as a model [24].

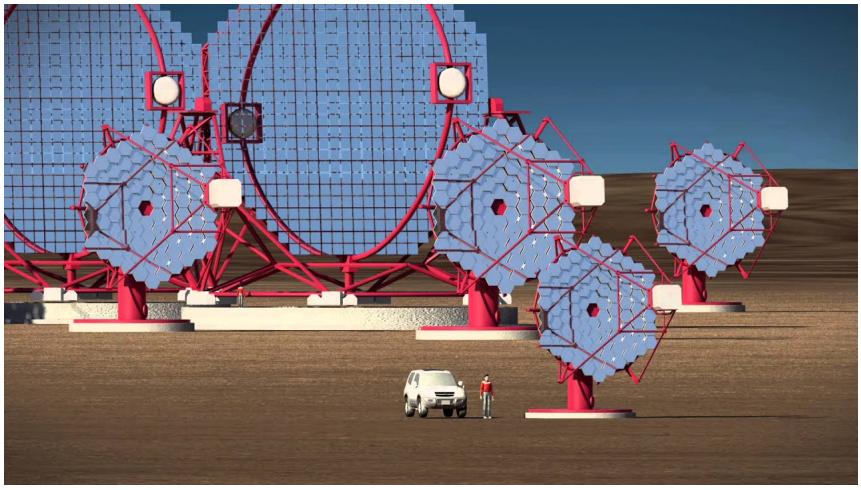
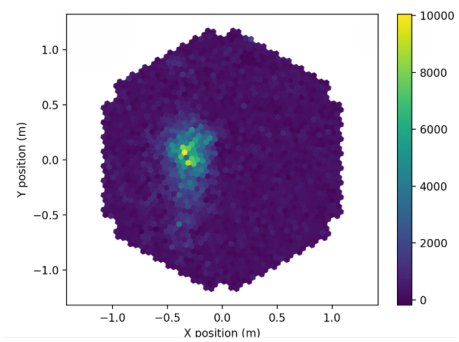
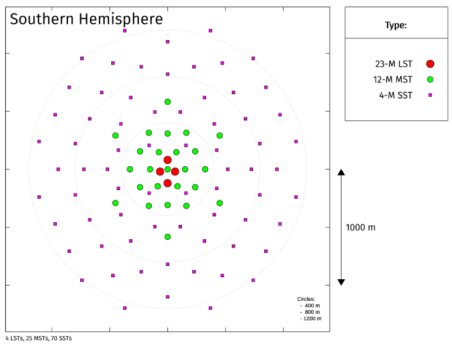


Figure 15: Cherenkov Telescope Array [25] .



(a) Hexagonal image that is fed into the detector.



(b) Distribution of CTA.

Figure 16: CTA [24].

The 7 possible outcomes to classify are the following:

- 1: gamma.
- 2: electron.

- 3: proton.
- 4: helium.
- 5: nitrogen.
- 6: iron.
- 7: silicon.

The dataset is confindential [23]. It is composed of 23,000 photos, and 12,000 belong to gamma. The rest are more or less equally split into the other particles.

The features are:

- Number of event.
- X position in the hexagonal grid.
- Y position in the hexagonal grid.
- Intensity of the pixel in (X,Y).
- Whether a pixel is activated (0 for deactivated pixels, which consist of background noise, and 1 for activated pixels).

There are three types of telescopes: Large, Medium and Small sized Telescope (LST, MST, SST). Simulated data from a LST telescope is used. Its main characteristics are the 23 m in diameter and the 4.5 degrees of vision [24].

3.1 Data cleaning

The general procedure explained in the last section was also done for this dataset.

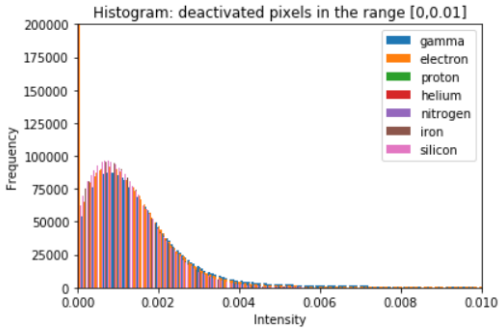
For every image, the positions X,Y were reescaled by a factor of 333 and 192, respectively. The images resulted in a range [1,55] for X and [1,93] for Y. Negative intensities were set to zero, and then all of them normalized between 0 and 1. The hexagonal grid was passed to a 55x93 image, as presented in figure 16 (a).

The distributions of gamma and other particles were compared. In figure 17 (a) there is the histogram for deactivated pixels, and in 17 (b) for activated pixels. The background noise for all the particles is very similar. For activated pixels, the main difference is the maximum value for every particle, presented in figure 17 (c). In general terms, gamma has higher values of intensity. In the range [0.6,1] gamma is the only one particle that has a significant number of points, whereas in the range [0.2,0.6] gamma and electron are the only particles that play a relevant role.

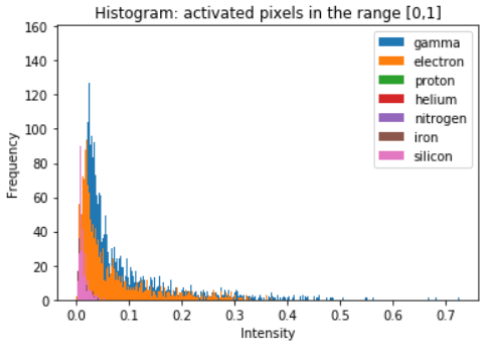
An interesting comparison can be done between gamma and electron. As said before, the noise is very similar, as well as the histogram of activated pixels. In figure 18, it can be seen how for the range of intensity [0.6,1], most of the points belong to gamma and not to electron. This might help the model to distinguish between gamma and electron.

3.2 Convolutional neural networks

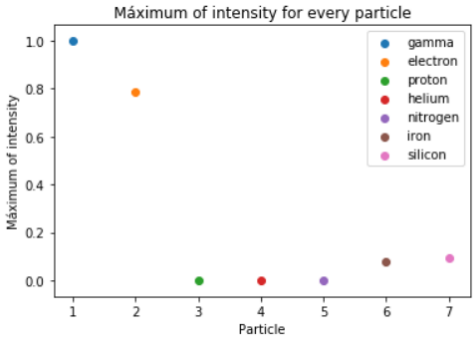
Convolutional neural networks (CNN) are a sort of neural networks that have some convolutional layers, i.e., layers where a convolution is applied, instead of a matrix multiplication. In this case, 2D data is analyzed (a hexagonal grid of pixels), but also 1D data or 3D data can be analyzed with this sort of network [3]. The main bibliography used for the theoretical introduction is [26] and [27].



(a) Histogram for activated pixels.

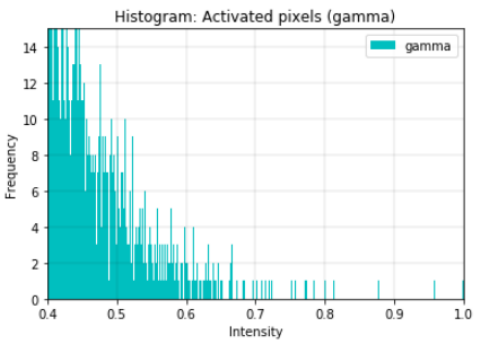


(b) Histogram for deactivated pixels.

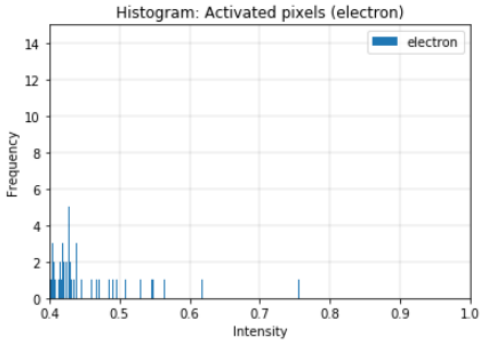


(c) Maximum values of intensity for all the particles.

Figure 17: Histograms for activated and deactivated pixel and maximum values of intensity.



(a) Histogram of activated pixels for gamma.



(b) Histogram of activated pixels for electron.

Figure 18: Comparison of intensity between gamma and electron in the range [0.4,1].

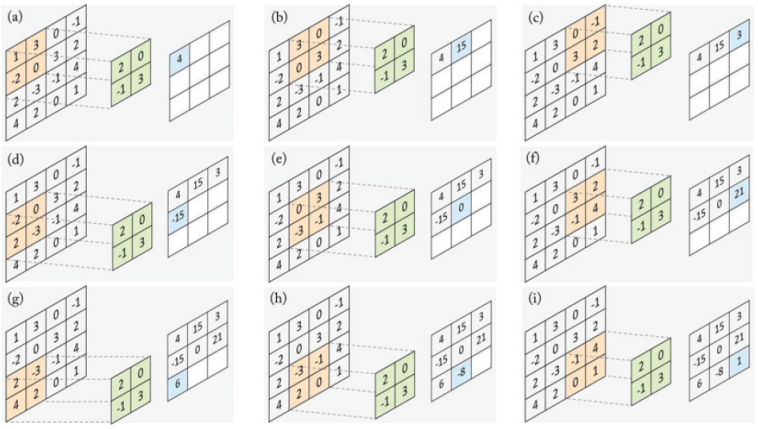


Figure 19: Convolution operation on an image [28]. The kernel $2x2$ is in green, the pixels from the image used in every operation are in orange, and the resulting feature map is in blue.

A dense layer (conventional layer) learns global patterns, whereas a convolutional layer learns local patterns, such as edges or textures of an image. These patterns are translation invariant given that they can be recognized anywhere in the image, while a dense layer has to learn it again. On top of that, convolutional layers can learn more and more abstract and complex concepts [26].

3.2.1 Convolution operation

Convolution can be seen as an operation in which two functions give a third one depending on how the shapes of the first and second function (reversed and translated) are overlapped. In one dimension, the 1D convolution is defined as [27]

$$y(t) = (x * w)(t) = \sum_{t=-\infty}^{+\infty} x(\tau)w(t-\tau) \tag{3.1}$$

for the continuous domain the expression is different

$$y(t) = (x * w)(t) = \int_{t=-\infty}^{+\infty} x(\tau)w(t-\tau)d\tau \tag{3.2}$$

In ML, the input (function x in Eq. (3.1)) is the dataset and the kernel is the weighting function w in Eq. (3.1), whose parameters will be learned by the network. The feature map is the output, referred to as y in Eq. (3.1) [3]. The dataset is a tensor with shape $nxmxc$, where nxm is the size of the images and c is the number of channels. If the image is in black and white, the number of channels is one. Likewise, if the image is in color, there are three channels (RGB), which means three matrices storing values of intensity for the colors blue, red and green.

In the case of an image, the convolution operation is shown in figure 19. It should be known that in ML cross correlation is applied instead of convolution (the only difference is that the kernel is not flipped). For further details, consult [3].

3.2.2 Filters

A filter is a matrix composed by numbers. Filters are typically applied to images to detect features, and in ML they are very similar to kernels. Kernels are matrices whose parameters are learnable. In an image with three channels, the convolution is done with three kernels (one per channel) and the set of three kernels is called filter [29]. In ML, filters might detect corners or borders, and advanced ones (or cascades of basic ones) could detect the ears of a cat or its nose.

There are many kind of conventional filters. For edges detection, a filter that is used is

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \tag{3.3}$$

Median filter. This filter is used to remove pepper and salt noise by taking a neighbourhood and replacing all its values by its mean. An example of a vertical filter is shown in figure 20.

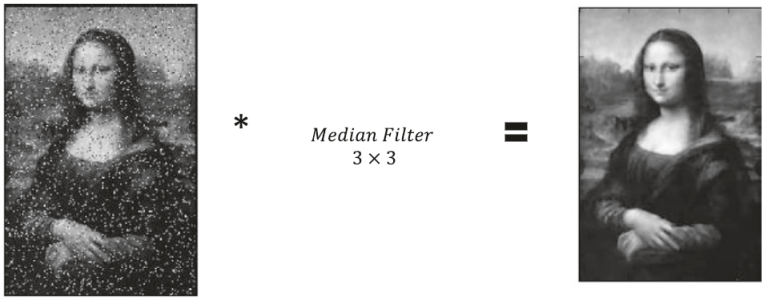


Figure 20: Median filter applied to an image [27].

Gradient-based filter. A gradient based filter takes advantage of the gradient of the function $I(x, y)$, representing the intensity of the pixel belonging to (x, y). The gradient can be applied to the horizontal and vertical direction.

3.2.3 Padding, Strides and Pooling

Padding is used to keep the same size of the image when the operation of convolution reduces it. A fxf filter on a nxn image results in a $(n - f + 1)x(n - f + 1)$ output. It is done by adding zeros to the boundary of the output. Padding can be *valid*, which means no padding, or *same*, which means that the size of the image is kept. Padding (defined as p) adds two dimensions to a matrix, i.e., adding p=1 to a $3x3$ matrix results in a $5x5$ matrix.

Strides consists on the shift of a certain number of pixels s every time convolution is performed. Usually, as in figure 19, the stride is s=1, despite it can take different values depending on the characteristics of the images.

After a convolution is performed to a nxn image, the size of the output is $n'xn'$

$$n' = \frac{n + 2p - f}{s} + 1 \tag{3.4}$$

Pooling. The most frequently used expressions are max pooling and average pooling. Max pooling consists of a kernel (usually $2x2$) that takes the maximum value in a locality of the image, while average pooling takes the average value. This method is usually performed to reduce the size of the image, hence this is not commonly used at the same time as padding. A typical size of the kernel used is $2x2$ [17].

3.2.4 Structure of a CNN

A CNN usually follows a pattern. Based on that pattern, plenty of different architectures can be created, from shallow to very complex modern models. Some of these architectures will be introduced later. The main components of a CNN are the following:

Input layer $\longrightarrow$ Convolutional layer $\longrightarrow$ Pooling layer $\longrightarrow$ Fully connected layer (FC) $\longrightarrow$ Output layer

Then many parameters can be changed, such as the number of convolutional layers before the pooling layer, how many times convolution and pooling is repeated, the number FC layers and so forth. Apart from this, the features introduced in section 2.7.1 are still valid, such a dropout, batch normalization or weights initialization. A common activation function is relu, but there are others that work good as well.

A typical structure in convolutional layers is shown in figure 21. Frequently, the shape of the kernels shrinks, whereas the number of filters grows. In this way, the net detects easier patterns, from more global to more particular and concrete.

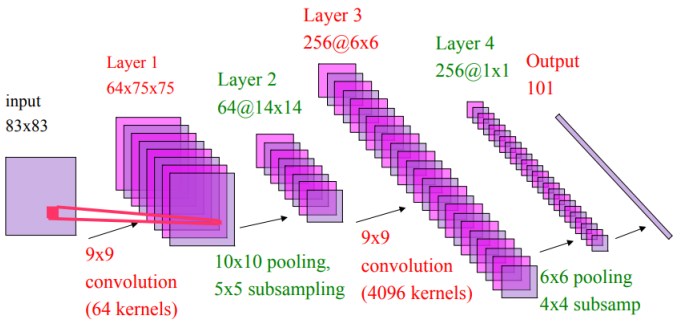


Figure 21: Structure of convolutional layers [30].

Transfer learning takes advantage of this structure that goes from global to particular, given that complex models (with many parameters) that have been already trained for certain problems can be reused for similar problems. This is done by training only the last layers and leaving the other frozen.

Transfer learning allows to use pretrained weights from other models instead of training the weights from scratch. With keras this can be easily implemented for a wide range of models, with weights from the *Imagenet* dataset. It also can be used just to implement an architecture.

3.2.5 Architectures in CNN

Over the years, a wide range of architectures has been developed. Here, some of them are presented and briefly explained.

Visual Geometry Group (VGG)

In 2012, AlexNet was very successful after implementing some novelties such as the relu activation, max pooling instead of average pooling and a pattern consisting of a convolutional layer fed not into a pooling layer, but to another convolutional layer [19, 31].

Two years later, in 2014, VGG was published. Its main features are that the filters are smaller (3x3), followed by relu activation functions and max pooling (2x2). The filters are smaller given that they allow more non-linearities and a reduction of parameters. Its structure is shown in figure 22. Convolutions have pad 1 and stride 1, while max pooling has stride 2. The structure of the feature maps are 64, then 128, 25 and 512 filters. At the end, connected layers and a 0.5 dropout are applied. A softmax layer is the last layer [27, 32].

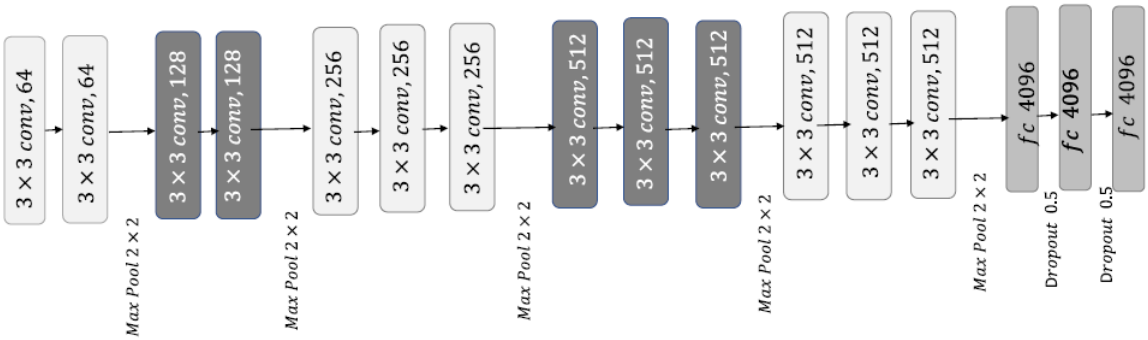


Figure 22: VGG architecture [27].

Resnet

This convolutional neural network gave a very good performance in the ILSVRC 2015 competition. The main feature that it presents is the so-called residual blocks. This consists of adding in each block of the convolutional network the input to the output. This is shown in figure 23. The residual blocks are repeated along the net. Another important characteristic is that Resnet is a 152 layer model, which is much deeper than VGG [33].

3.3 Results

For this section, the same libraries of the last section have been used. The main goal was to distinguish between gamma and other particles. Besides, a general model was created to classify all the particles. First, gamma was compared to proton, given that it is an easy task. Then, the comparison was done with electron, and finally gamma was classified versus all the particles.

The structure of the convolutional network is very similar to the VGG net. There is a

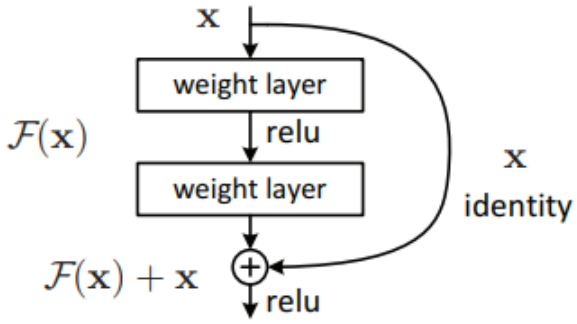


Figure 23: How a residual block works [33].

structure of blocks composed of two convolutions and one max pooling. Padding is *same*, pooling is 3, stride 1 and there are 64 filters 3x3 (in every block the number of filters goes as 2^n). There are two hidden layers with 24 neurons and dropout 0.3. The optimizer is Adadelta. For more details, my GitHub repository is available in [22].

In table 8, a summary of the results is presented.

Task	Accuracy (%)
Gamma vs proton	99.7
Gamma vs electron	96.7
Gamma vs all	99.0

Table 8: Classification performance for different cases.

Besides, the classification for all the particles resulted in a 57% accuracy. The confusion matrix [4] is shown below, and it is organized according to the order shown at the beggining of the section: gamma, electron, proton, helium, nitrogen, iron, and silicon.

$$\begin{pmatrix} 1192 & 6 & 2 & 1 & 0 & 0 & 0 \\ 35 & 320 & 81 & 32 & 27 & 19 & 25 \\ 5 & 94 & 380 & 157 & 63 & 39 & 25 \\ 0 & 38 & 211 & 226 & 79 & 45 & 37 \\ 0 & 8 & 62 & 63 & 125 & 85 & 71 \\ 0 & 5 & 36 & 26 & 75 & 181 & 80 \\ 0 & 8 & 46 & 37 & 84 & 130 & 68 \end{pmatrix}$$

When the model misclassifies gamma, most of the times it is done by mistaking it with electron. Moreover, the model confuses electron mostly with proton and gamma.

All in all, the classification for all the particles is not good, but for gamma the results are better. In general, the model distinguishes with a 99% accuracy between gamma and the rest of particles. In particular, with proton the model gave a 99.7% accuracy, whereas with electron it gave a 96.7% accuracy.

4 Quantum Machine Learning

4.1 Introduction to quantum computers

In 1936, Alan Turing presented a paper in which he described the model of a universal computer that can simulate any other programmable computer [34]. This is the so-called *Universal Turing machine*. This results in a machine that can simulate any algorithm that has been reproduced in another machine, leading to what is known today as computer science. Besides, the invention of transistors let computer science advance thanks to the development of hardware [35].

Moore's law predicts that computational power doubles every two years (the number of transistors is doubled), and since the 60's this has been true. By the time it comes to an end, quantum computing might play a significant role, giving place to a new type of computation, much more efficient on certain problems than the actual computation. Quantum computation is based on the theory of quantum mechanics, and later it will be briefly explained [35].

David Deutsch proposed an example to show that quantum computers might be more efficient than conventional computers, and Peter Shor proved that on a quantum computer the problem of finding the prime factor of a number was more efficiently solved [36].

4.1.1 Quantum bits

The quantum bit (or *qubit*) is different from a classical bit, whose states are two: 0 and 1. A qubit is represented with the *dirac notation*, $|\ \rangle$, and it can also be in the states 0 or 1 ($|0\rangle$and $|1\rangle$, respectively). The qubit differs in a bit in the sense that these are not the only possible states. Thanks to quantum mechanics, linear combinations of this states can occur,

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{4.1}$$

such that

$$|\alpha|^2 + |\beta|^2 = 1 \tag{4.2}$$

where α and β belong to $\mathbb{C}$, and they are called amplitudes. $|\psi\rangle$ represents the state of a qubit (called wave function), and it belongs to a two-dimensional vector space, in which $|0\rangle$ and $|1\rangle$ form an orthogonal base, which is called computational base and used, by default, for measurements [35].

Measuring in quantum mechanics means that the wave function goes from a state of superposition of eigenstates that form an orthogonal base to a single eigenstate. Measuring a qubit implies getting either the state $|0\rangle$ or $|1\rangle$ with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively. Nevertheless, before the qubit is measured, its state is a superposition state, Eq. (4.1), not 1 or 0.

Equations (4.1) and (4.2), ignoring a phase factor, can be expressed as

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle \tag{4.3}$$

where $\theta \in [0, \pi]$, $\phi \in [0, 2\pi]$. A typical way to see this expression is to use the *Bloch sphere*, shown in figure 24.

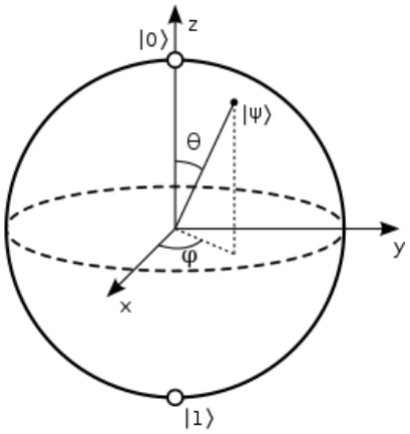


Figure 24: Bloch sphere. Figure taken from ref[37].

In a three qubits system, the orthogonal base is formed by $|000\rangle, |001\rangle, |010\rangle, |011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle$, which means that there are 2^3 orthogonal states. For a n qubits system, and the basis states $|x_1 x_2 ... x_n\rangle$, the state vector belongs to a $\mathbb{C}^{2^n}$ dimensional vector space, given that there are 2^n amplitudes. This implies that computational power improves with the number of bits as n in classical computers and with the number of qubits as 2^n in quantum computers, which is a huge difference [35].

4.1.2 Quantum logic gates

Quantum logic gates are analogous to logic gates. Qubits are useful to store information and quantum logic gates process that information by taking qubits to different states. In contrast with the classical gates, these gates are reversible (they are represented by unitary matrices), and this is the only restriction quantum gates have. In figure 25 some gates are shown, such as the identity gate, the Hadamar gate (H), which creates equal superpositions of a qubit, or the Pauli gates, that generate rotations [38, 35].

For classical computing with multiple bits, any algorithm can be generated by using only the NAND gate, together with the ability of copying or duplicating bits, hence it is a universal gate, while this cannot be done by a *NOT* and *XOR* gate altogether.

Gate	Pauli	Implementation	Matrix form
$\mathbb{1}$	σ_0		$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
X	σ_x	180°_x	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Y	σ_y	180°_y	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Z	σ_z	180°_z	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
$S = \sqrt{Z}$		90°_z	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$
$T = \sqrt{S}$		45°_z	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$
$\sqrt{X}$		90°_x	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$
$\sqrt{Y}$		90°_y	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$
H			$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$

Figure 25: Some quantum single qubit gates [38].

4.1.3 No-cloning theorem

An interesting theorem of quantum information states that it is not possible to copy an unknown quantum state, which means that it is not possible to copy information.

Theorem *In a complex Hilbert space H, it does not exist a unitary transformation U : $H \otimes H \rightarrow H \otimes H$ such that there exists a state $|s\rangle \in H$ satisfying*

$$U(|\psi\rangle|\rangle) = |\psi\rangle|\psi\rangle, \quad \forall|\psi\rangle \in H \tag{4.4}$$

The proof can be found in [39]. An easy example is the following: the state to copy is $|\psi\rangle = a|0\rangle + b|1\rangle$. By means of a CNOT gate, the copy of a $|0\rangle$ and a $|1\rangle$ can be done,

$$CNOT[\,a|0\rangle + b|1\rangle\,]|0\rangle = a|00\rangle + b|10\rangle \tag{4.5}$$

In the cases where $|\psi\rangle$ is 0 or 1, the information is properly copied, but for other cases, this is not the result. By *reductio − ad − absurdum*, if

$$|\psi\rangle|\psi\rangle = a^2|00\rangle + b^2|11\rangle + ab|01\rangle + ab|10\rangle \tag{4.6}$$

and $|\psi\rangle|\psi\rangle$, $CNOT|\psi\rangle|0\rangle$ are compared, the only solution is $ab = 0$, which means that information cannot be copied [35]. From a classical point of view, copying bits is one of the most common operations when it comes to generating classical algorithms, in circuits formalism. From a quantum point of view this is impossible.

4.2 Machine Learning in quantum computers

One of the areas in which quantum computers have applications is ML. As explained previously, a state with n qubits is a vector that belongs to $\mathbb{C}^{2^n}$. Therefore, logic operations on qubits multiplies the state vector by $2^n x 2^n$ matrices. For these transformations, quantum computers are much faster than classical computers [40, 41]. A quantum version of many supervised learning algorithms, such as KNN or SVM is introduced in [42]. Moreover, a quantum version of a neural network has been studied and implemented.

Quantum computers give better results with quantum data than with classical data. Quantum data is data created in a quantum system. What is particular from this data is that it shows superposition and entanglement. Some examples of quantum data are chemical simulations (drug research, computational chemistry...) or quantum matter simulations, mainly for exotic states, in which many-body quantum mechanics effects can be observed [43].

4.2.1 Quantum neural networks

Hybrid models are being currently implemented. In these models, an option is to implement a quantum layer as one more hidden layer of a neural network. This is done thanks to a *parameterized quantum circuit* (PQC). A PCQ can be seen as a quantum circuit in which the rotation angles are specified, in this case by the inputs of a classical hidden layer. Then, the outputs of this quantum layer are the inputs of a new hidden layer [44]. The structure is presented in figure 26.

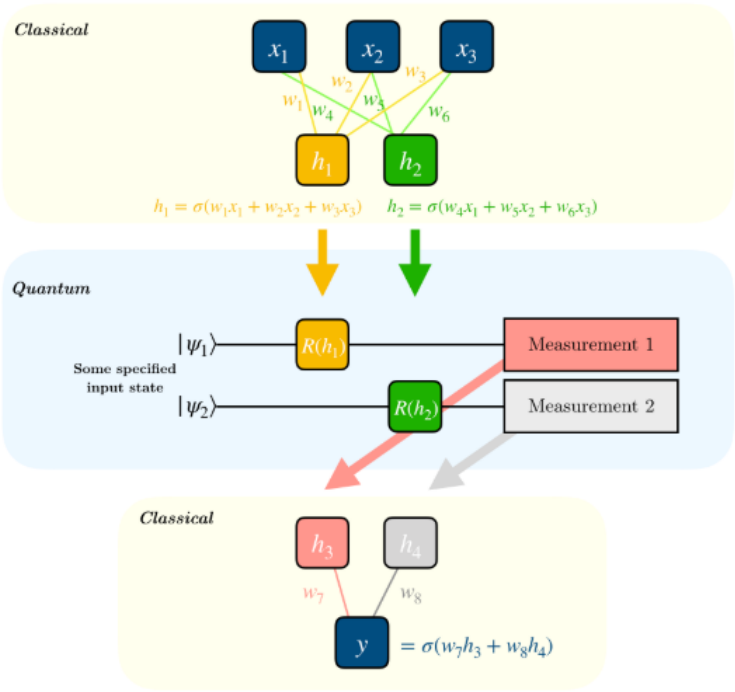


Figure 26: Structure of a quantum-classical neuronal network [44].

Backpropagation for quantum circuits is slightly different, there are no analytic formulas.

In general, the gradient of a quantum circuit can be calculated as [44]

$$\nabla[Quantum\ Circuit(\theta)] = Quantum\ Circuit(\theta + s) - Quantum\ Circuit(\theta - s) \quad (4.7)$$

4.2.2 Comparison of a quantum and classical neural network

A classification task for the MNIST dataset (classical data) was performed by a classical neural network and a quantum neural network. It was a binary classification of the numbers 5 and 6. For this comparison *TensorFlow quantum* was used. The code source can be found in my GitHub repository [22], based on [43].

First, the dataset was loaded, and then, the images were transformed from 28x28 to 4x4 pixels, so that they fit in a quantum computer. These images were converted to quantum circuits. Qubits with different states are associated with the pixels depending on the value of the pixels. The classical neural network was a convolutional neural network with 1.2 million parameters, and the quantum neural network 37 parameters, so also a convolutional neural network of 37 parameters was created, for a fair comparison [43]. The results are presented in figure 27.

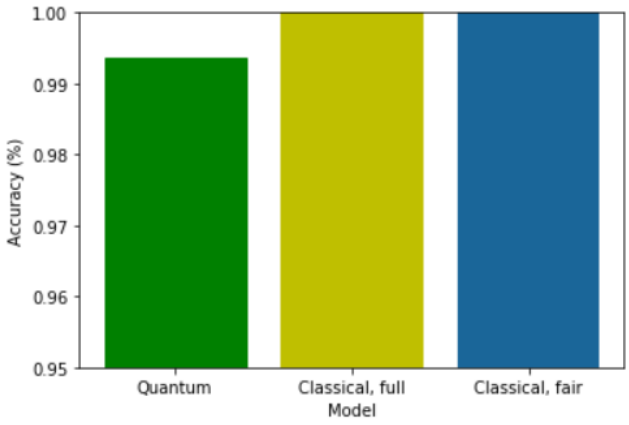


Figure 27: Results of the comparison between classical and quantum neural networks.

The results are quite good for all the neural netwoks, specially for the classical neural networks, that achieved a better performance, and faster than the quantum neural network. With other datasets, or even different number in the MNIST dataset, the results can be worse for the quantum neural network. The development of quantum computers might lead to more accurate and efficient models in the future.

5 Conclusions

For the conclusions, everything learned throughout the thesis is going to be discussed, just as applications in science and industry, nowadays and in the future.

Plenty of information has been acquired and this knowledge has been put into practice later. First of all, the main background needed for this area is based on mathematics and programming skills. A physicist fulfills these requirements, hence the first thing done was to study every model before implementing it. Most of the supervised ML techniques were studied, and for the first problem SVM, random forest and neural network gave the best results, with an 97% accuracy. The model could easily distinguish photon, proton and iron, although it had more difficulties with helium and nitrogen. Nevertheless, for each particle the accuracy was always above 90% in most of the algorithms. Therefore, this problem was solved with ML techniches successfully. Despite this, every dataset is different and for every case some algorithms might perform better than others. As a general rule, neural networks are successful in most of the cases.

For the second part CNN were studied and implemented to distinguish between different particles, focusing mainly on gamma. The results were quite good, even though the general classification of all the particles was not very accurate. The reason of the accurate classification of gamma is its values of intensity for activated pixels, higher than the rest of the particles. The toughest problem was to distinguish from gamma and electron, and the performance of the model resulted in a 96.7% accuracy. For the general case, it was hard to distinguish among helium, nitrogen, iron, and silicon. Gamma was mostly confused with electron, and electron with gamma and proton, despite they were successfully classified.

For the last section, a hybrid model of a neural network was implemented on a quantum computer, giving worse results than the classical neural network with the same number of parameters, but less than 0.1%. This is just a starting point, and in the future quantum neural networks are likely to outperform neural networks in many problems. Quantum neural networks will give better results when more complex models that can outperform current models are created. Also, they are more proper for quantum data.

As it has been shown in this thesis, neural networks are a very important part of ML. In fact, this algorithm is widely used to solve many kind of problems. The principal types of neural networks are feed-forward, convolutional and recurrent neural networks, giving surprising results in many approaches, such as creating music, generating text or translating. Open AI has recently released GPT-3, a natural language processing model with 175,000 millions parameters, the largest model created so far. This model can create articles very difficult to distinguish from the ones created by humans. Besides, with this model, natural language can be translated into programming code. What is more, code from a programming language, such as python or c++, can be translated into other programming languages [45].

ML is a revolutionary field very helpful for society, since it is the core of AI. Some dangers have to be taken into account, i.e., autonomous weapons, *deepfakes* and many processes in which AI plays a relevant role. AI is relevant nowadays due to the development of computers and the vast amount of data being created every day. Perhaps, what is today the most important issue is how is exposed everybody due to data. The availability

of so much data is new, and society does not cope with it properly yet. This issue concerns most of the population, given that many apps, internet navigators or webpages can obtain information of at what time we wake up, where we go to work and by what mean, what sort of clothes we like, the political party we vote for, our hobbies... This is not given too much importance, but society should care more about it, and set some rules that do not allow companies to take advantage of the current situation.

Hopefully, there will be some adjustments on this in the future, and AI will be more developed. For instance, some companies are working on robots that can help elderly people that live alone, and the basis of this is convolutional neural networks (computer vision). A relevant feature for the optimization of models is computational speed, and quantum computers will be very important for this. We have seen how a quantum computer can do this task, and even though this is currently done worse than classical computers, when quantum computers will be developed and powerful enough, some tasks will be done incredibly fast. This is an area which quantum computers might improve, but there are many others. Weather forecasting, cybersecurity, creation of new materials and so forth. Important companies such as Google and IBM are working to turn this into reality. In 2019, Google achieved quantum supremacy using a programmable superconducting processor, by doing in 200 seconds a task that would take about 10,000 years on a powerful classical computer [46].

Overall, AI is a very interesting field with a promising future. Quantum computing will bring plenty of developments to many areas, and one of them is AI. Therefore, there is still research to be done and hopefully, in the incoming years, society will witness a change for the better.

References

[1] Guillén A., Todero C., Martínez J.C., Herrera L.J.,
A Preliminary Approach to Composition Classification of Ultra-High Energy Cosmic Rays,
Ntalianis K., Vachtsevanos G., Borne P., Croitoru A. (eds) Applied Physics, System Science and Computers III. APSAC 2018. Lecture Notes in Electrical Engineering, vol 574. Springer, Cham, 2019.

[2] CORSIKA simulator,
`https://www.ikp.kit.edu/corsika/`

[3] Ian Goodfellow, Yoshua Bengio, Aaron Courville,
Deep Learning,
MIT Press, 2006.

[4] Ethem Alppaydin,
Introduction to Machine Learning, Second Edition (Adaptive Computation and Machine Learning),
The MIT Press, 2010.

[5] Svante Wold, Kim Esbensen, Paul Geladi,
Principal Component Analysis,
https://doi.org/10.1016/0169-7439(87)80084-9, 1987.

[6] Trevor Hastie, Robert Tibshirani, Jerome Friedman
The elements of statistical learning: Data mining, inference, and prediction,
Springer, 2009.

[7] Cross-validation,
`https://scikit-learn.org/stable/modules/cross_validation.html`

[8] James Bergstra, Yoshua Bengio,
Random Search for Hyper-Parameter Optimization,
Journal of Machine Learning Research 13 281-305, 2012.

[9] Kevin P. Murphy,
Machine Learning: A Probabilistic Perspective,
The MIT Press, 2012.

[10] Peter Flach,
Machine Learning: The Art and Science of Algorithms that Make Sense of Data,
Cambridge University Press, 2012.

[11] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani,
An Introduction to Statistical Learning with Applications in R,
Springer, 2013.

[12] Implementation of logistic regression for multi-class,
`https://github.com/bamtak/machine-learning-implemetation-python/blob/master/Multi%20Class%20Logistic%20Regression.ipynb`

[13] Christopher M. Bishop,
Pattern Recognition and Machine Learning,
Springer, 2006.

[14] Dmytro Mishkin and Jiri Matas,
All you need is a good init,
arXiv:1511.06422 [cs.LG], 2016.

[15] Anders Krogh, John A. Hertz,
A Simple Weight Decay Can Improve Generalization,
NIPS'91: Proceedings of the 4th International Conference on Neural Information Processing Systems, 1991.

[16] Srivastava, Nitish Hinton, Geoffrey Krizhevsky, Alex Sutskever, Ilya Salakhutdinov, Ruslan,
Dropout: A Simple Way to Prevent Neural Networks from Overfitting,
Journal of Machine Learning Research. 15. 1929-1958., 2014.

[17] Andrew Ng, Younes Bensouda Mourri, Kian Katanforoosh
Deep Learning specialization, online lectures,
`https://www.coursera.org/specializations/deep-learning`
Coursea webpage, unknown date. Accessed on March, 2020.

[18] Dmytro Mishkin and Jiri Matas,
On the momentum term in gradient descent learning algorithms,
https://doi.org/10.1016/S0893-6080(98)00116-6, 1998.

[19] Hisham El-Amir, Mahmoud Hamdy,
Deep Learning Pipeline: Building A Deep Learning Model With TensorFlow,
Apress, 2020.

[20] Diederik P. Kingma, Jimmy Ba,
Adam: A Method for Stochastic Optimization,
arXiv:1412.6980 [cs.LG], 2014.

[21] Sergey Ioffe,
Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models,
arXiv:1702.03275 [cs.LG], 2017.

[22] Rodrigo Castellano Ontiveros github repository,
`https://github.com/rodrigo-castellano/Bachelor-thesis`

[23] Private communication.
Carlos Jose Todero Peixoto ,
Universidad de São Paulo (USP).

[24] Description of an atmospheric Cherenkov telescope,
`https://www.cta-observatory.org/`

[25] Video presenting Cherenkov Telescope Array (CTA),
`https://www.youtube.com/watch?v=0QgnKA5AUDY`

[26] Francois Chollet,
Deep Learning with Python,
Manning, 2018.

[27] Santanu Pattanayak,
Pro Deep Learning with TensorFlow,
Apress, 2017.

[28] Salman Khan et al,
A Guide to Convolutional Neural Networks for Computer Vision,
Morgan and Claypool, 2018.

[29] Difference between filters and kernels,
`https://towardsdatascience.com/`

[30] Yann LeCun and Marc'Aurelio Ranzato,
Deep Learning tutorial,
ICML, Atlanta, 2013.

[31] Krizhevsky, Alex and Sutskever, Ilya and Hinton, Geoffrey,
ImageNet Classification with Deep Convolutional Neural Networks,
Neural Information Processing Systems. 25. 10.1145/3065386. , 2012.

[32] Karen Simonyan and Andrew Zisserman,
Very deep convolutional networks for large-scale image recognition,
arXiv:1409.1556 [cs.CV], 2014

[33] Kaiming He et al,
Deep Residual Learning for Image Recognition,
arXiv:1512.03385 [cs.CV], 2015

[34] A.M. Turing,
On computabl numbers, with an application to the entscheidungsproblem,
The Graduate College,Princeton University,New Jersey, 1936

[35] Michael A. Nielsen Isaac L. Chuang,
Quantum computation and Quantum information,
Cambridge University Press, 2010.

[36] Peter W. Shor,
Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,
SIAM REVIEW, Vol. 41, No. 2, pp. 303–332, 1999

[37] Bloch sphere,
`https://www.ibm.com/blogs/emerging-technology/dplaying-with-the-bloch-sphere/`

[38] Jonathan A. Jones, Dieter Jaksch,
Quantum information, computation and communication,
Cambridge University Press, 2012.

[39] Anirban Pathak,
Elements of Quantum computation and quantum communication,
CRC Press, 2013.

[40] Aram W. Harrow, Avinatan Hassidim and Seth Lloyd,
Quantum algorithm for linear systems of equations,
arXiv:0811.3171v3 [quant-ph], 2009.

[41] Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., Lloyd, S.,
Quantum machine learning,
Nature, 549(7671), 195,2017.

[42] Maria Schuld, Ilya Sinayskiy and Francesco Petruccione,
An introduction to quantum machine learning,
Contemporary Physics, 56:2, 172-185, 2015.

[43] Implementation in TensorFlow quantum of a quantum neural network,
`https://www.tensorflow.org/quantum/tutorials/mnist`

[44] Hybrid quantum-classical neural network,
`https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html#Create-a-%22quantum-class%22-with-Qiskit`

[45] Tom B. Brown et al,
Language Models are Few-Shot Learners.,
arXiv:2005.14165 [cs.CL], 2020

[46] Frank Arute et al,
Quantum supremacy using a programmable superconducting processor.,
https://doi.org/10.1038/s41586-019-1666-5, 2019