Intelligent low-level signal detection in raw LArTPC waveforms through deep learning techniques

Lorenzo Uboldi

October 11, 2019

Contents

Introduction

DUNE is an experiment that uses a huge LArTPC as far detector. It aims to study neutrinos from accelerator and rare events neutrinos like the one from supernovae, proton decay, ecc...

Without some kind of data selection, the theoretical throughput from the DUNE DAQ will be of the order of TB/s.

For accelerated neutrinos, one has to save data only when the beam is incoming. But when it comes to rare events neutrinos, to have some chances to detect an interaction, one has to take data continuously with a live time the closest as possible to 100%. It is clear that TB/s is absolutely not manageable and some kind of data suppression has to be implemented. The current solution is called "zero suppression": only raw waveforms that have an ADC count higher than a certain threshold are saved. Furthermore, in those waveforms, everything below that threshold is set to zero to save data.

If the threshold is too low, the throughput will be over any computer capacity; if the threshold is too high, then too much physics risks to be lost.

I spent my internship trying to develop a reliable intelligent method able to discriminate signal from noise in order to reduce the throughput without having to discard possible information that is below the threshold. I developed methods based on deep learning techniques.

Chapter 1 describes the process of designing a deep learning architecture. It is quite technical and if the reader is interested in physical results only, it may be read very briefly and, maybe, skipped. It is to be noted that every time I designed a neural network architecture, a similar process has been done. For brevity, I reported it only for the first neural network.

Chapter 2 and 3 describe the application of the developed neural networks to a realistic DUNE dataset and the physical outcome.

Chapter 4 is about ICARUS, another LArTPC detector. It suffers from a complex noise spectrum and, due to the low signal-to-noise level, the actual classical analysis technique struggles to discriminate the true signal from noise. I tried to apply the deep learning method I developed to the ICARUS problem and in Chapter 4 the studies and results are presented.

CHAPTER 1

Architecture design and model selection

In machine learning, the path to follow is usually the following: understand data, decide the algorithm and design a basic architecture of the network. Then, after a baseline model has been found and proved to work, one has to refine it and find the best possible model. The basic baseline should be used as a reference: any change in the model has always to beat it in performance.

The main characteristic of our signal is the spatial/temporal independence: the feature we are trying to detect is a particular waveform shape that the neutrino interaction produces in the detector. This shape does not depend on where it is located in the signal. The DAQ of DUNE will outputs 4492 temporal bins for each channel; the neutrino interaction could occur in the first 1000 or in the last, the precise position it is not a feature that characterizes the neutrino interaction and so we have to implement a network able to discard this information. It has been shown that, in deep learning, convolutional networks are very well fitted for detecting feature that are not dependent on the position in the data. Since our data is mono-dimensional, the convolutional layers operate a 1D convolution (usually in deep learning it is common the 2D, since it is used for image detection).

In machine learning, a typical approach is to divide the dataset in three sub-sets: one for training, one for validation and one for testing.

The training set is used for fitting the neural network. During each stage, called "epoch", the network "sees" all the dataset once and optimizes its weights with some algorithm (as the gradient descent) in order to reduce a loss function, that could be the mean squared error, the categorical cross-entropy, ecc... The learning process usually involves several epochs, where the network "adapt" itself to the data of the training set.

The validation set is used for making prediction on unseen data at each epoch and see how the learning process is doing on predicting new data. This helps to understand how many epochs are needed. Furthermore the validation set is used for optimizing hyper-parameters,

such as learning rate, dropout rate, regularization, etc...

The test set should be used only at a very last step. It is useful to see how the fitted networks predicts on completely new data, never seen during learning or during hyper-parameters optimization. The test set should simulate real-life applications, where the network tries to predict on new and unseen data. This is the set that should be used to compare performance for different model.

In the following subsections the networks have been trained on waveforms of the U plane, where the signal has a minimum collections of 2000 electrons. Half of the data is signal and half is noise.

The total number of waveforms is 25719. Of those, 5144 have been taken out in order to use them as a test set. Of the remaining 20575 waveforms, 80% has been used for training and 20% for validation. All the waveforms come from a dataset produced by the DUNE collaboration.

Models have been written using keras with TensorFlow backend. Data has been normalized: for each of the 4492 temporal bins, the mean value and the standard deviation between all the training data has been computed; then, those value has been used for rescaling both the training data and the validation and test one. Machine learning methods usually expects normalized feature in order to work properly: avoiding normalization can, for instance, cause exploding weights (or other problems such the vanishing gradient) and prevent correct learning.

All the architectures have as output a single neuron with a sigmoid activation function. The predicted value is a probability of being a signal (1 is a signal, 0 is noise).

1.1 Fully connected

Before testing the convolution, I wanted to compare with a simple fully connected deep neural network. This was to have a baseline to be convinced of the benefit of the convolutional layers: the cons of the convolution is that it is very computing intense and if no benefit in accuracy is found, there is no point in using a more complex model.

The neural network is structured as in Table 1.1. The total number of parameters is

Layer type	Neurons
Dense + ReLu	500
Dropout (50%)	
Dense + ReLu	500
Dropout (50%)	
Dense + ReLu	500
Dropout (50%)	
Dense + ReLu	500
Dropout (50%)	
Dense + Sigmoid	

Table 1.1: Fully connected NN architecture

2998501. The network has been trained and tested using a Nvidia Tesla K80 GPU with 16

Gb of RAM, obtaining the results in Table 1.2. The "inferencing time" is intended as the time taken to predict all the probabilities for the 20575 waveforms of the test set.

Optimizer	Adam
Batch size	512
Epochs	300
Training time	4min 44s
Inferencing time	0.58s
Training accuracy	0.9296
Validation accuracy	0.9157
Test set accuracy	0.9012

Table 1.2: Fully connected results

In Figure 1.1 the training and validation histories are plotted.

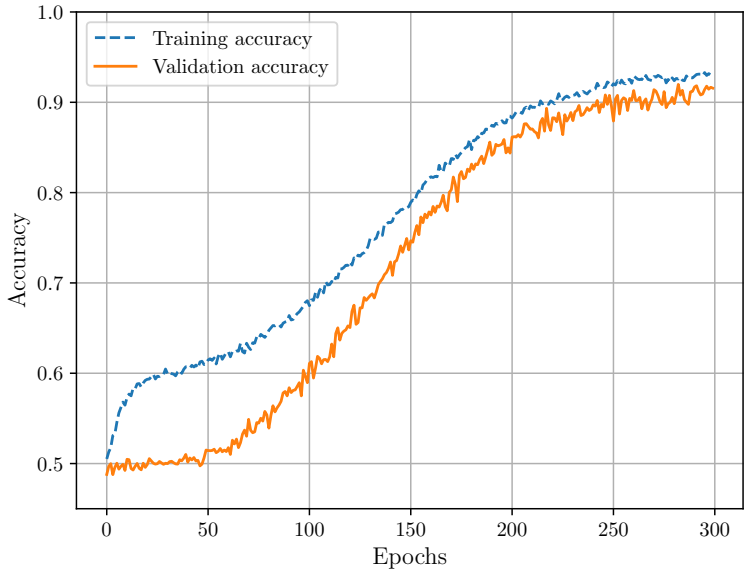


Figure 1.1: Training and validation history of fully connected NN

As clearly shown in Figure 1.1 the model is overfitting data: the accuracy on the validation set is always lower than the one of training set; the final accuracy on test set is even lower. This is a clear indication that the network is learning too much from the training set and it is not able to predict well unseen data (this is the definition of "overfitting"). Furthermore the number of parameters is very high and the network it is not astonishingly fast.

1.2 Convolutional Neural Network

The goal now was to develop a convolutional neural network to take advantage of the positional invariance of the peak. As stated before, the convolution operation tends to be slow. My aim

was to develop a fast and light network trying to keep the accuracy as higher as possible. Furthermore, at the beginning we had the idea to try to run the model on an embedded device: for a small ASIC the network must be as light as possible. See Section 1.3 for further details.

1.2.1 Convolutional Baseline

The baseline convolutional network I built is shown in Table 1.3.

Layer type	Filters/neurons	Kernel/pool size	Output shape
Conv1D + ReLu	128	10	(4483, 128)
Conv1D + ReLu	128	10	(4474, 128)
MaxPooling1D		2	(2237, 128)
Conv1D + ReLu	256	10	(2228, 256)
Conv1D + ReLu	256	10	(2219, 256)
GlobalAveragePooling			(256)
Dropout (50%)			(256)
Dense + Sigmoid	1		(1)

Table 1.3: Convolutional baseline architecture

The total number of trainable parameters is 701801. The network has been trained and tested using a Nvidia Tesla K80 GPU with 16 Gb of RAM, obtaining the results in Table 1.4. The "inferencing time" is intended as the time used to predict all the 20575 waveforms label.

Optimizer	Adam
Batch size	512
Epochs	10
Training time	15min 46s
Inferencing time	28s
Training accuracy	0.9351
Validation accuracy	0.9368
Test accuracy	0.9523717

Table 1.4: Convolutional baseline results

In Figure 1.2 is reported the training and validation history.

The network seems robust, with a good accuracy and no overfitting: the test accuracy is higher than both the validation and test ones. There are few drawbacks: even if this network has a good accuracy, it took a long time to be trained and did the same during the inferencing phase. Compared to the fully connected network, the time for predict all the data probabilities is almost 50 times higher.

1.2.2 Advanced convolutional models

The goal here is enhance the performance and make the network as lighter as possible.

There are many tricks to enhance the speed of a network. For instance, a thin but deeper net, usually, may perform better than one thick with few layers and may be faster to compute

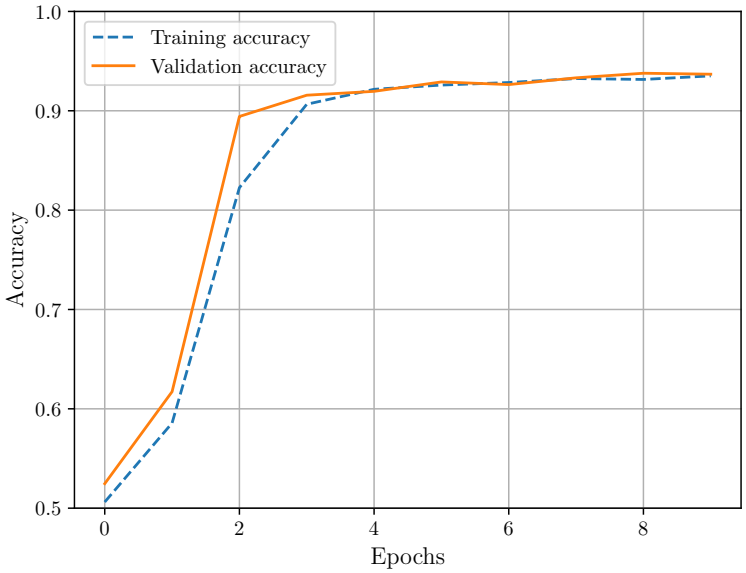


Figure 1.2: Training and validation history of convolutional baseline

both in training and inferencing stages. Another good rule is to keep the number of neurons as power of two: this is to take advantage of the structure of buses and memory of the hardware.

Furthermore, our baseline is not learning very easily: in the first two epochs the accuracy stays quite low: this suggests that the structure of the network struggles to adapt to our data, even if the final accuracy is very high.

In image recognition problems, it has been show that convolutional networks take advantage by starting with few filters with small kernel dimension in the first layers. Then, going deeper, increasing the number of filters and their kernel dimension increases learning performance. This appears the be a kind of learning the "gross" features in the first layers with few filters and, then, learning finer and smaller characteristics with the layers with more and bigger filters. Furthermore, it has been shown that this pyramidal structure makes the gradient descent faster and reduces the memory usage during learning. In order to achieve better speed performance, a good choice of kernel size, pooling layers and, eventually, strides sizes helps reducing the data dimensionality: this can have a huge impact in the computing speed of gradient descent.

Following this guidelines I shrank the network and adapted to our data requirements. After several tries, I obtained a good structure with satisfying performance both in accuracy and speed. The network architecture is reported in Table 1.5.

In the "output shape" column, the first number is the dimension of data that outputs by each filter; the second is the number of filters. It is to be noted that I choose a kernel and stride sizes in order to reduce as much as possible the data dimensionality between each layer. I experimented a huge speed increase with this architecture.

The total number of trainable parameters is 271393. As usual, the network has been trained and tested using a Nvidia Tesla K80 GPU with 16 Gb of RAM. In the first run with a batch size of 512 the network was trained in just 19 seconds, so I choose a batch size that

Layer	Filters/neurons	Kernel size	Stride	Output shape
Conv1D + ReLu	16	3	2	(2245, 16)
MaxPooling		3		(748, 16)
Conv1D + ReLu	32	5	3	(248, 32)
MaxPooling				(82, 32)
Conv1D + ReLu	64	7	2	(38, 64)
MaxPooling		3		(19, 64)
Conv1D + ReLu	128	9		(11, 128)
Conv1D + ReLu	128	11		(1, 128)
Dense + Sigmoid	1			(1)

Table 1.5: Convolutional Advanced architecture

maximizes the learning performance. The results are in Table 1.6.

Optimizer	Adam
Batch size	64
Epochs	10
Training time	35s
Inferencing time	2.4s
Training accuracy	0.9600
Validation accuracy	0.9485
Test accuracy	0.9463

Table 1.6: Convolutional baseline results

The learning speed is almost 40 times higher than the one of the convolutional baseline network. The inferencing speed is more than 10 times higher. The network is enough light that can be trained and run on a CPU too.

The accuracy appears to be good as well, not much lower than the one achieved previously. Note that in the first epoch the validation accuracy is already very high (the architecture of this new network may be more adapted to our problem). The drawback is a slightly probable overfitting, that can be noted in Figure 1.2.

Another model I tried is this network with one or two dense fully connected layers (for instance with 64 neurons each): I found no sensitive increase on accuracy and, since speed is the main goal, I discarded those models.

Reduced overfitting

In order to reduce overfitting one possible solution is to add some dropout in each layers. Dropout is the process where, during each epochs, some of the neurons are randomly "shut off" to prevent them from learning. For instance, a dropout rate of 0.5 means that each epochs half of the neurons are not learning: those are chosen randomly. The dropout helps not to learn too much from the training set and to keep the ability to generalize well.

I noted that the network is very sensitive to perturbations in the first two layers: a small

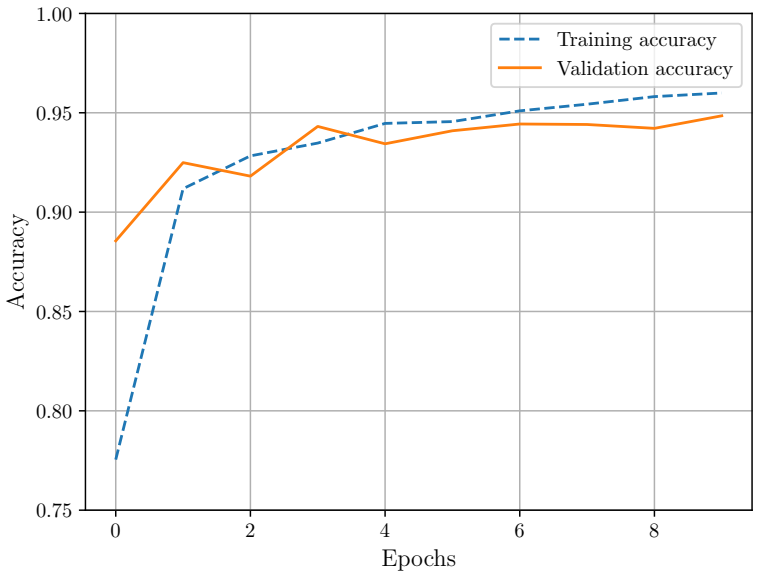


Figure 1.3: Training and validation history of convolutional advanced network

dropout in the first layer makes almost vanishing any attempt to learn something useful from data. So, I proceeded adding a dropout in the last three layers with an increasing rate.

Results are in Table 1.7 and learning history can be inspected in Figure 1.4.

Training accuracy	0.9473
Validation accuracy	0.9468
Test accuracy	0.9471

Table 1.7: CNN with dropout results

The overfit is clearly reduced and the accuracy is still high enough.

1.2.3 CNN shrinking

After running the network on the data of plane V and Z (on previous sections I used only data from U) I noticed that the network was able to go up to an efficiency of 99%. The fact that this happened after a long run with many epochs was a hint that maybe the network was bigger than required. I was able to achieve good performance discarding all the last two convolutional layer. However, removing only the last convolutional layer (and, so, keeping one with 128 filters) gave higher accuracy and, on the V plane data, even higher than keeping both layers. This is the model that I selected.

In Table 1.8 is reported the model architecture. The total number of parameters is 92321, almost 1/8 of the original baseline convolutional.

Training time is still about 35s. Inferencing time is 1.7s, slightly lower than before.

Training history is in Figure 1.5 and final results in Table 1.9

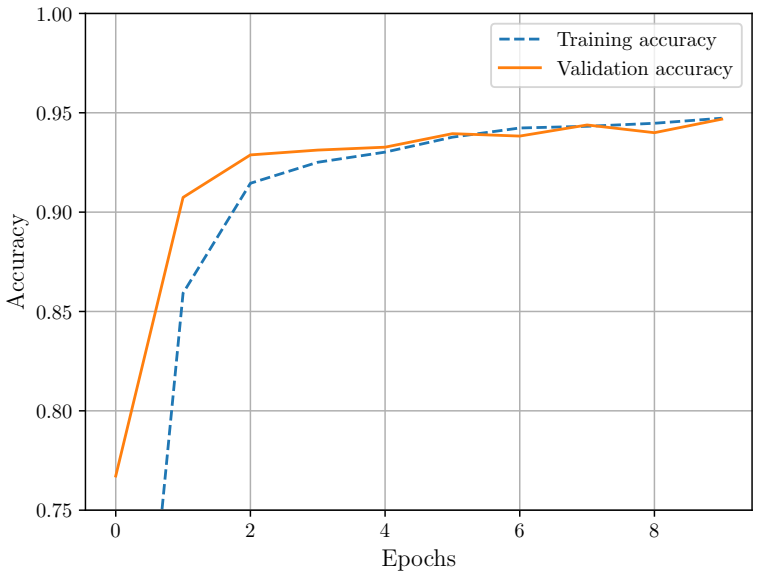


Figure 1.4: Training and validation history of advanced convolutional with dropout

Layer	Filters/neurons	Kernel size	Stride	Output shape
Conv1D + ReLu	16	3	2	(2245, 16)
MaxPooling		3		(748, 16)
Conv1D + ReLu	32	5	3	(248, 32)
MaxPooling				(82, 32)
Conv1D + ReLu	64	7	2	(38, 64)
Conv1D + ReLu	128	9		(11, 128)
Dense + Sigmoid	1			(1)

Table 1.8: Shrunk CNN

1.3 Embedded device

The search for a high speed inference method that could be placed directly in the DAQ brought us to considering the use of an embedded device developed for deep learning applications. In this framework, there are different options on the market as the Nvidia Jetson Nano, the Intel Movidius and the Google Coral EdgeTPU. The last one is the newest and, apparently, with the highest performance. Google claims a huge speed up, sometimes close to a 10x (https://coral.withgoogle.com/docs/edgetpu/benchmarks/), using only 5 watts at the cost of less than 100 dollars for each device.

The Google Coral EdgeTPU supports only TensorFlowLite (from now on "TFlite") quantized models. Normal TensorFlow models use floating point 32 bits variables and operations (supported by all the modern Nvidia GPUs). A TFlite quantized model uses unsigned 8 bits integer. Operations involving that data type are much faster. This new implementation, indeed, has born for mobile applications. It has been shown that for many situations cutting

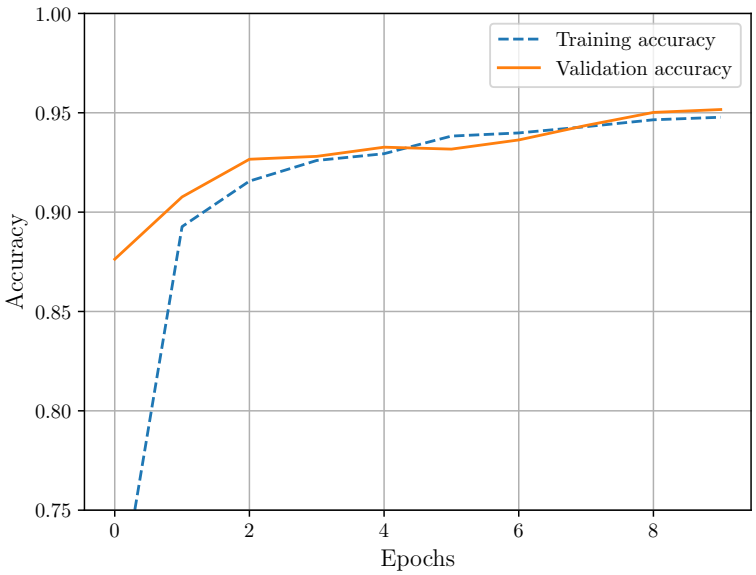


Figure 1.5: Training and validation history final model

Epochs	10
Training accuracy	0.9478
Validation accuracy	0.9516
Test accuracy	0.9425

Table 1.9: U plane results, final model

the precision from floating point to short integer doesn't overkill the accuracy and boost up speed performance.

Once converted, the important parameters to monitor are the difference in speed and accuracy of the models run on CPU/GPU versus the ones run on the EdgeTPU.

The hardware setup on which I tested the models are:

- **CPU** an Intel i3-6100U CPU 2.30GHz with 12GB of RAM;
- **CPU+TPU** EdgeTPU Accelerator connected via USB 3.0 to an Intel i3-6100U CPU 2.30GHz with 12GB of RAM;
- **EdgeTPU DevBoard** EdgeTPU directly connected on the SoC of an ARM board[1].

The time is the wall clock time used to infer all the test set (5144 waveforms) one waveform at time. For the CPU test, this restricts to use one core only.

The accuracy comparison is performed on the test sets for each model/hardware. For a better analysis, I evaluated the quantized TFlite model on the CPU only too[2]. Results are in Table 1.11.

[1] https://coral.withgoogle.com/products/dev-board/

[2] This test is for the accuracy only. The time performance is not to be taken into account: TensorFlow

CPU	TPU+CPU	DevBoard
8.4s	2.6s	4.3s

Table 1.10: Inference time for 5100 waveforms

	Keras	TFlite CPU	EdgeTPU
U plane	0.9553	0.9537	0.9541
V plane	0.9998	0.9764	0.9752
Z plane	0.9951	0.9871	0.9841

Table 1.11: Test sets accuracy

Systematically, the accuracy on the TPU dropped by a few point percentage. The difference, however, is almost negligible.

Even if there isn't a big accuracy drop, the gain in speed is very disappointing. The CPU inference has been done on a low-end mobile Intel CPU but, still, is almost fast as the TPU. It turned out that, with high probability, the Google's benchmarks have been performed with the TensorFlowLite models on CPU. The TFlite models are optimized for ARM architectures and don't benefits of the x86 optimizations making them slower.

Due to this results we decided to discard the use of the EdgeTPU: if the deep learning methods we are developing turns out to be reliable and useful, a custom ASIC/FPGA can be developed for this specific neural network to be placed in the experiments.

Lite is optimized for ARM architectures. On a AMD64 the time performance are much lower than a pure TensorFlow model, even after quantization.

CHAPTER 2

DUNE realistic dataset: supernovae neutrinos and radiological background

In the framework of low energy neutrinos, DUNE aims to study the ones from supernovae. This is a very hard task, not only because the signal is very close to the noise, but also because there are several radiological backgrounds with overlapping energy spectrum to the neutrinos one. In particular, some of the backgrounds are intrinsic in the detector and it is impossible to eliminate the radioactivity even with ultra-pure material.

In DUNE, the main intrinsic radiological backgrounds are depicted in Table 2.1. I have reported the percentage of each type of signal. Radiological events are interactions of the decays products (α and β).

Neutrinos	0.38 %
Neutrons	0.37 %
APA	0.37 %
CPA	3.81 %
^{85}Kr	3.81 %
^{222}Rn	58.40 %
^{39}Ar	32.82 %

Table 2.1: Signal type

The first goal is to demonstrate that a neural network is able to differentiate the signal type. I wanted the network to be able to distinguish neutrinos from the main backgrounds: ^{222}Rn, ^{39}Ar and ^{85}Kr.

In the first stage, I did not care about the accuracy: in order to make the learning much powerful on the training set, I avoided the use of any kind of regularization and dropout. I used a dataset composed by half of neutrinos events and half events of the other classes totalling twenty thousands waveforms. For neutrinos event I used monoenergetic 5 MeV of the previous chapter. The background is generated using DUNE radiological model.

In all the datasets (neutrinos vs ^{222}Rn, neutrinos vs ^{39}Ar and neutrinos ^{85}Kr) the validation set accuracy was not able to go further than 65-70%. The fact that it went over 50% suggests that there is the possibility to distinguish the classes with the network. The low accuracy on the validation set, easily exceeded by the accuracy on the training set, strongly indicates overfitting. This is easily pointed out looking at the validation loss in Figure 2.1: it slightly decreases until epochs 5, then it starts increasing, indicating overfitting. This can be solved used a bigger dataset for training.

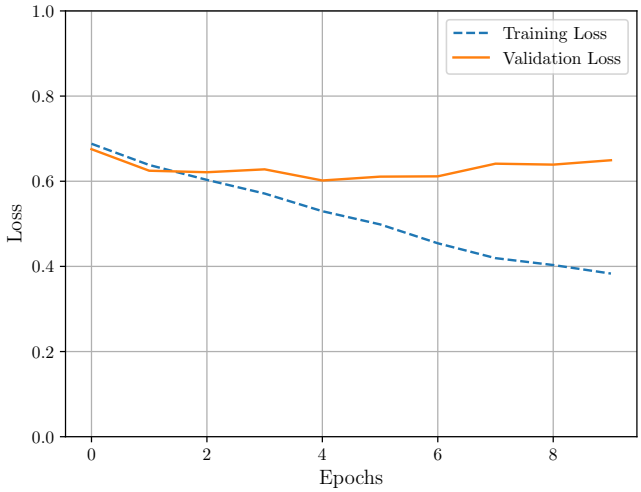


Figure 2.1: Model loss[1]for neutrinos versus ^{222}Rn.

The next step was to move to a realistic and sensitive situation: neutrinos from supernovae (not 5 MeV monoenergetic) using a much bigger dataset in order to avoid overfitting.

2.1 Dataset description

To generate the new realistic dataset I used the current LArSoft[2] production version, v08.30. The process involved in the simulation is:

- One of the possible generators simulates a physical events. The generators are "ar39Gen", "kr85Gen", "rn222Gen" and "Marley". The first three are part of the DUNE radiological model. The Marley event generator produces neutrino events in accord with the GVKM supernovae flux model.
- The second stage involves Geant4 and simulates the interaction of the particles in the detector. This phase stops at the production of the ionizing electrons from the liquid argon.
- The third step involves DetSim and simulates the drifting of the electrons and the response of the induction and collection wires.

[1]When two classes are involved, it has been used the binary cross-entropy loss.
[2]The software produced for all the LArTPC detector (DUNE, ICARUS, etc...).

- Finally the electronic chain is simulated.

In order to produce noise events, the zero suppression flag has been turned off. 10000 marley events and 2000 radiological[3] were generated. Then, using SimChannel, all wires with a minimum of 2000 electrons deposited were identified: the produced waveforms were labelled with the name of the generator in order to know if they belong to radiological or neutrinos events. All waveforms with zero collected electrons were labelled as noise.

From the Monte Carlo, information on the number of collected electrons, on the energy deposited in the detector and the position of the peak were saved. In Figure 2.2 there is the spectra of energy deposited by each class of events in the detector.

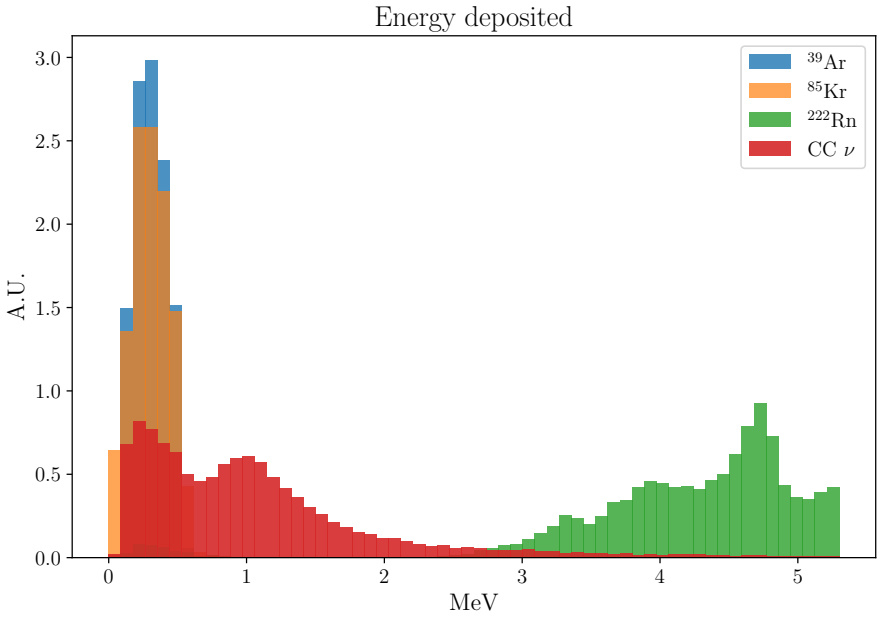


Figure 2.2: Spectra of energy deposited in the DUNE far detector.

In the dataset preparation I picked randomly from all the events a total number of 778979 waveforms. Of those, 80% has been used for training and validation, 20% for testing. The proportion for each class of events in the dataset is depicted in Figure 2.3.

2.2 Strategy

The final goal is to be able to discriminate signal from noise and then neutrino events from the radiological ones. There are, at least, two possible ways to proceed: one is to do a single multi-class inference, the other is to use two sequential neural networks.

In the first case, one can substitute the last neuron activation function with a softmax. This function is the generalization of the sigmoid with more than two classes. Then, one label as "0" the noise, as "1" the radiological background and as "2" the neutrinos. With a

[3]The DUNE radiological model has been used

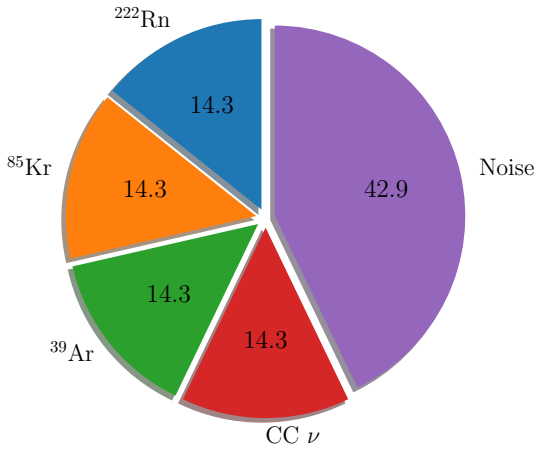


Figure 2.3: Class partitioning in the dataset

method called "hot-k-encoding", the network is taught to predict the correct label between all the possibles.

In the second case the strategy is to train the first network with only two classes: noise and any signal (all the possible: neutrinos and radiological). Then, a second network is trained to distinguish the type of signal: neutrino or background.

In all the cases the architecture of the networks was the one in Table 1.8 (where, in the three classes strategy, the sigmoid activation function is switched to the softmax).

The second method turned out to have a higher accuracy on detecting neutrinos. Furthermore, using two networks it is easier to tune parameters in each stage, in order to prefer neutrinos. Due to those considerations, I decided to stick with the two networks approach.

2.2.1 First NN: transfer learning

The first neural network aims to discriminate signal from noise. Even if the accuracy is very high (over 99%) there is a rate of false negative. A false negative is a waveform that is actually signal but that is classified as noise.

I immediately noted that the majority of those false negative was actually neutrino events. That means: almost all of the mistakes were the event that we don't want to lose.

I wanted to completely invert the situation: if the false negatives were radiological events that would not be a problem.

To revert the situation I had to teach the network to prefer neutrino to radiological: not to make mistake is impossible, but if a mistake has to be done, than it be a radiological.

The solution resembles the idea of "transfer learning". This method consists in training with a datasets and get the knowledge out of it, then retrain the network in a different dataset to "transfer what has been learnt". The difference with a standard training is that, normally, the weights of each neurons are initialize randomly; in this situation the weights are initialized with the knowledge of the previous dataset. The gradient descent (the optimization algorithm

of the weights) does not start from a random point: the local minimum that will be found may be different and more useful[4].

The idea of transfer learning comes from image and object recognition, where the networks may be very huge with millions of parameters. For those situations, fitting the entire network to a dataset may take many hours, or even days, with the most powerful GPUs. Researchers showed that "freezing" all the layers but the last one and refitting only this, saves many hours of computation and in many situation can achieve sensitive results[5].

My strategy was the same, but not for timing reasons: I explicitly wanted the network to keep some knowledge from another dataset. I fitted the network to a dataset with only noise and neutrino waveforms. Then, I froze all the convolutional layers, leaving trainable only the sigmoid layer. I retrained the last layer for few epochs (3-4) on a dataset made by all the waveforms: any signal and noise.

I was able able to reduce the number of neutrino false negative: the overall percentage of false negative resulted to be 12%, of those only 4% were neutrinos.

2.2.2 Optimization: Bayesian search

In deep learning, the design of a neural network it is not only about choosing the number of layers, neurons, filter size, ecc... i.e. the architecture; there are also parameters concerning the fitting stage. Those are, for instance, the learning rate, the number of epochs and the regularization rate.

The two enemy, while training, are always underfitting and overfitting: the goal is to achieve the situation between those.

Underfitting is when the network does not learn enough from the training set. This could be because the architecture is too simple to catch all the features or because it was fitted not for enough epochs. Overfitting is when the the network learns too much from the training set. Over-adapting to this set turns out to ruin the ability to generalize predictions over new examples.

Both those cases can be recognized looking at the losses function over the training and the validation set[6].

A first approach is to manually modify the hyper-parameters to find the best setting, but, for a fine tuning, this is not enough. When the hyper-parameters space is quite big, one can perform a grid-search that consists in sampling uniformly spaced points.

My aim was to find the best setting of learning rate, number of epochs and dropout (for the last two layers) for the neutrino vs noise and the radiological vs neutrino networks. The space I wanted to explore was epochs between 10 and 50, learning rate between 0.0001 and 0.002 and dropout (for each layer) between 0 and 0.8. Sampling, for instance, ten configurations

[4] Where the complexity of the problem space is big (and for deep learning it is always so) there are many local minima. Any optimization algorithm has high chances to stuck in one of those and hardly the global minimum is found. Each local minimum is different and some are better than others. The method I describe is to "guide" the network in choosing a more fitted minimum than the others.

[5] Obviously the accuracy would be lower, since the firsts layers are fitted to a different dataset. But, in image recognition, the first layers are the one that detect "gross features" as angles, surfaces, ecc... Those, usually, don't differ so much between different image datasets

[6] A good article that explains tricks to detect and correct those undesirable situations is https://arxiv.org/abs/1803.09820

for each parameters gives a total of 1000 possible settings. With a mean training time of 10 minutes it is easy to understand that this way it is not to be pursued.

A better approach is the probabilistic Bayesian search through the Gaussian Process[7]. The basic idea is to sample very few points trying to minimize the uncertainty over new possible points using the Bayesian posterior. There is not the certainty to find the best hyper-parameter setup, but a good approximation should be found. I was able to increase the accuracy of all the networks of at least 1-2% sampling only 50 settings per net.

2.3 Results

The first network, after the transfer learning, gave the results in Figure 2.4. The image is called "confusion matrix": the rows represent the true value and the columns the predicted ones. On the diagonal there are the correct predictions; on the upper off-diagonal the false positives and on the lower the false negatives.

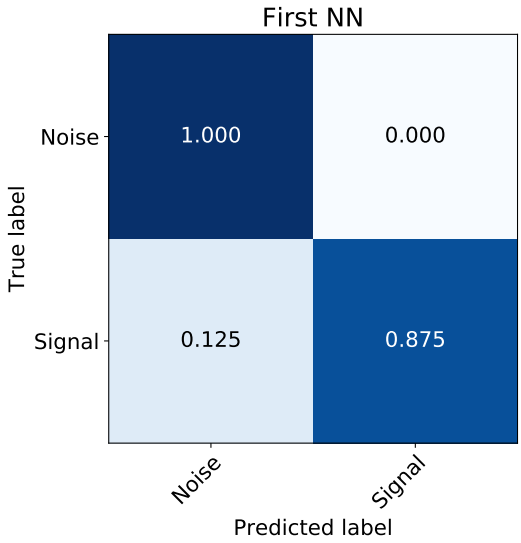


Figure 2.4: Confusion matrix for the first neural network.

The network never classifies noise to be signal (the false positive). Of the 12.5% false negative only 4 % are neutrinos (the rest is radiological events).

Of the total number of waveforms of neutrino events in the test set 1.9% are lost in this stage of prediction.

The confusion matrix is in Figure 2.5.

Of the 0.2% of false positives, 57% are krypton events, 26% radon and 17% argon.

Finally, passing through the first and then the second neural network, 92.28% of the incoming neutrinos are recognised to be so.

[7]A complete explanation of this method can be found on Bayesian reasoning and Machine learning, David Barber; or Machine Learning: a probabilistic perspective, Kevin P. Murphy.

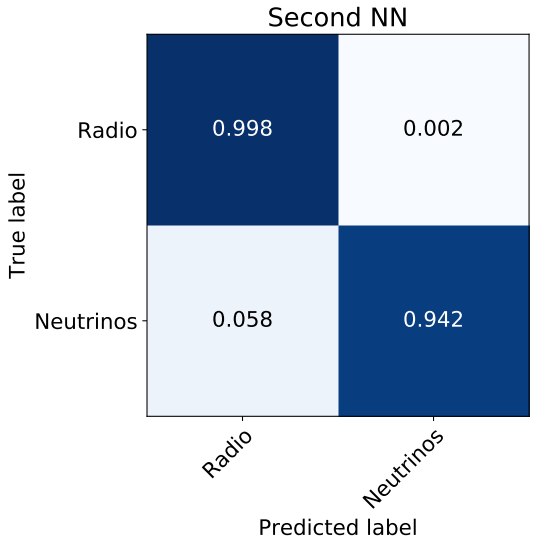


Figure 2.5: Confusion matrix for the second neural network.

In Figure 2.6 the energy deposited spectra for correct neutrinos predictions and false negatives are depicted. It can be seen that most of the mistakes are made with less than 1 MeV of energy deposited per channel.

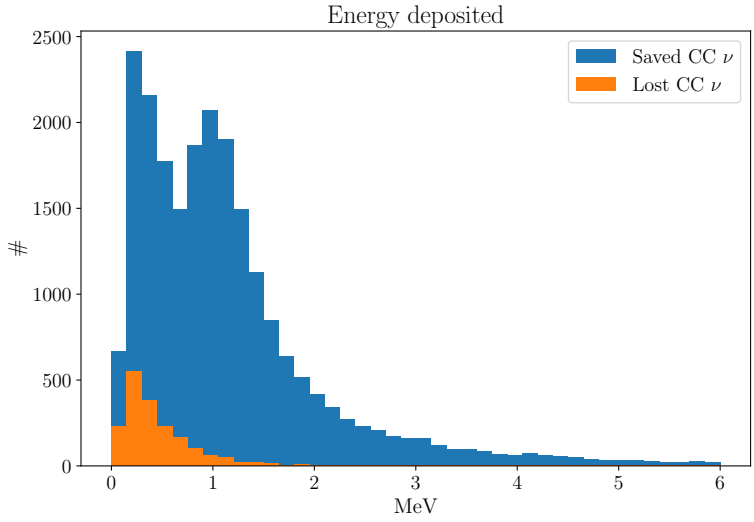


Figure 2.6: Deposited energy spectra for lost and recognised neutrinos.

CHAPTER 3

Region of interest: peak localization

As mentioned, the reason of the zero-suppression algorithm is to save data reducing the throughput from the DAQ. This is accomplished saving only the waveforms where the peak is higher than a certain threshold. Furthermore, only the region where the peak is present is saved, everything else is set to zero.

As seen in the previous chapter, the method I developed is able to do something more sophisticated than a "threshold trigger" and it is able to recognize also events where the peak is comparable to the noise level. Locating the peak, however, it is still important: after a waveform is recognized to be a signal, everything that is not the peak can be cancelled in order to save data space. This is important also for physical reason: knowing the exact position of the peak gives information about the exact timing of the interaction.

When the peak has an amplitude comparable to the noise level, trying to locate it with classical methods is very compelling and, frequently, cannot be done.

I decided to develop another deep learning technique to achieve this goal.

3.1 Algorithm

I took inspiration from the Region-Based CNN strategy that has been implemented for object detection and localization. In RB-CNN a first network makes "class agnostic proposal" for possible region in the image where a generic object could be. Then, a second network classifies the "possible object" in each proposed region.

The first stage is very complex and it is usually achieved with huge networks; our method, instead, must be fast and not too computationally heavy. I decided to drop the region proposal stage for a simpler algorithm: a small window scans the waveform and a simple convolutional network decides if inside there is a peak or not. In that way, the peak can be located with a precision equal, at least, to the length of the scanning window.

3.2 Data preparation and CNN architecture

The total length of each waveform from the DUNE DAQ is 4492 bins, I decided to have the window width of 100 bins and to move it by 10 bins at time during the scanning. As can be seen in Figure 3.1, most of the peaks are less than 100 width.

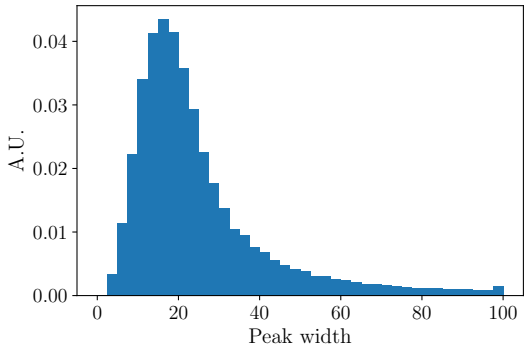


Figure 3.1: Peak width distribution for DUNE neutrino signal.

For the training stage I prepared a dataset with 10000 windows with a peak inside and 13000 without; I labelled the firsts as 1 and the seconds as 0. A small portion of 10% of the total has been kept out for the testing stage.

The CNN structure is very simple: one convolutional layer with five filters, one hidden dense layer with ten neurons and one output neuron. A schematic structure is reported in Table 3.1.

Layer	Filters/neurons	Kernel size	Stride
Conv1D + ReLu	5	10	2
Dense + ReLu	10		
Droput (0.5%)			
Dense + Sigmoid	1		

Table 3.1: CNN for peak location

Training for five epochs gave 98% of accuracy on the test set made of approximately half windows with peak and half without.

3.3 Results

In a real application, the CNN has to infer the position of the peak in an entire waveform that, as explained, is sliced in windows. Since the last neuron is a sigmoid, the output is the probability of having a peak in the inferred window. On clear and high peak, the CNN works perfectly, as it should be. In Figure 3.2 I have depicted two "hard" case. The red dotted line is the position given by the Monte Carlo simulation. Below the probability of the peak predicted by the CNN.

The waveform on the left has a small peak, but still can be recognized and, indeed, the CNN is almost sure about the position. The waveform on the right has a peak of the same level of the noise, it can be hardly recognized. Indeed, the CNN is less sure but, still, made the correct prediction with a probability of 75%.

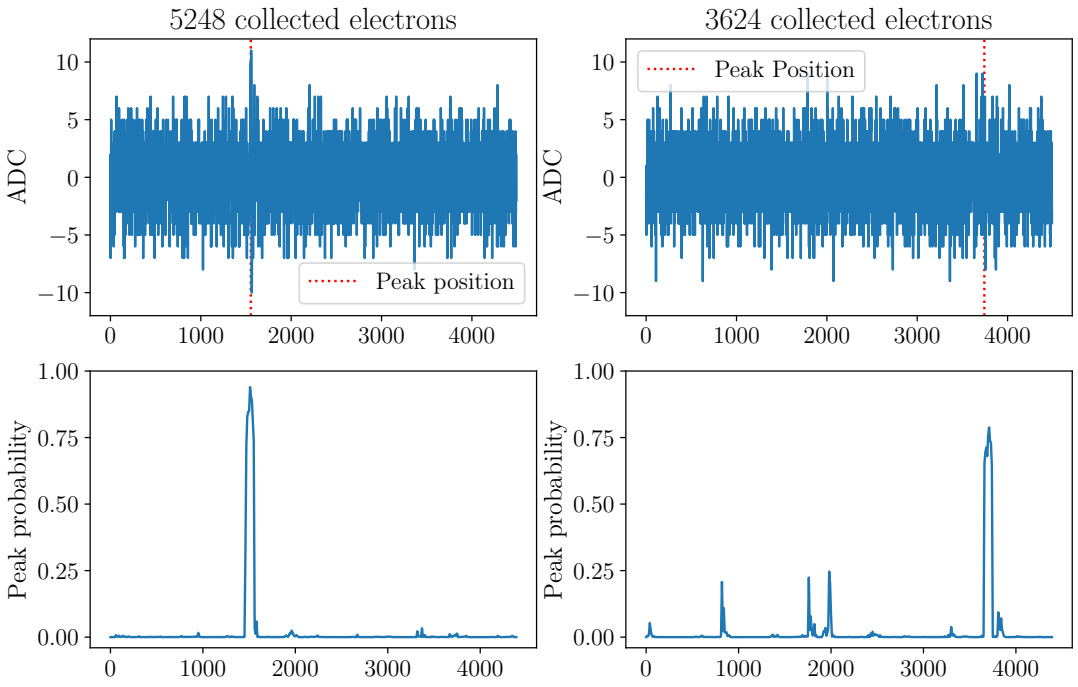


Figure 3.2: Two waveforms and the predicted peak location probabilities.

If the final goal is to implement an algorithm that save only the peak and its position, a deep study should be performed in order to decide a probability threshold over which the bins should be saved. If, for instance, a probability of 0.5 is chosen, there may be a too high rate of false positive (bins saved without a peak) or false negative (peak lost). A threshold with a good balance of those situations must be found.

CHAPTER 4

ICARUS: lower signal to noise ratio

ICARUS is a LArTPC that will operate as the far detector in the Short Baseline Neutrino program at Fermilab. It is the only liquid argon time projection chamber with "hot" read-out electronics: this means that the electronic is not inside the detector, close to the very cold liquid argon, but it is placed outside. This makes the ICARUS DAQ very noisy compared to other experiments. The poor signal to noise ratio makes very hard to discriminate low-level events. It is not only the fact that the peak amplitude is low compared to the noise mean level, but also the shape of the noise resembles the typical peak shape and makes very compelling to discriminate signal from noise.

In Figure 4.1 are reported the power spectra density for DUNE and ICARUS.

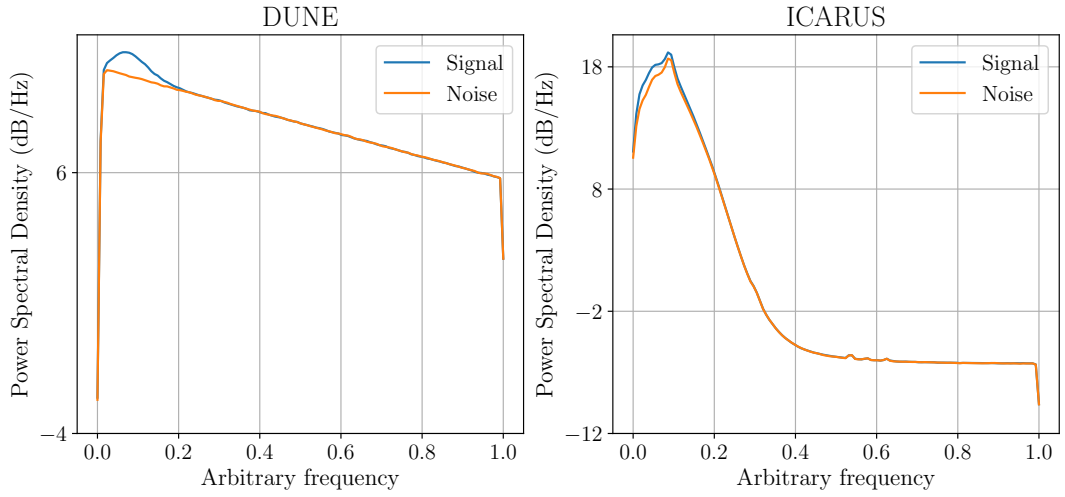


Figure 4.1: Power spectra density for DUNE and ICARUS. The frequencies are rescaled between 0 and 1.

The noise for DUNE is Monte Carlo generated and it has an almost perfect $1/f$ spectrum.

The signal is the one generated by the supernovae neutrinos: it can be seen that at low frequencies the signal differentiates quite clearly from noise.

The noise for ICARUS is a mixture of Monte Carlo and real data noise samples, measured ad CERN from the actual electronics; those samples have been implemented in the Monte Carlo. The signal comes from simulated muons. It can be observed that the spectra of the noise is quite involved, and the signal hardly differentiates from it (the shape at low frequencies is almost identical, only a little bit more powerful).

The actual method to detect signal is a classical approach that uses a complex de-convolution operation that is computationally very intense and must be ran offline. Furthermore, the accuracy of this method drops below 20000 electrons collected.

Since ICARUS is a LArTPC detector (as DUNE will be), we decided to apply the methods we developed to it.

4.1 Signal detection with CNN

ICARUS was born at Laboratori Nazionali del Gran Sasso (LNGS), here it took data for many years. Then it moved to CERN where the electronics was updated, but the detector never took data with the liquid argon. Now it is in commissioning stage at Fermilab.

Before this summer, the noise model in the Monte Carlo was mainly based on data taken at LNGS. With the new electronics the noise is expected to be different. A research group is working to build a noise model based on the real data taken at CERN (and at Fermilab) from the new hardware. I had the chance to inspect waveforms from the old model and from the new one. The last is completely different, with a much worse signal to noise ratio, but since it is expected to be more similar to what the real data will be, I decided to work on that data.

The ICARUS researchers were working on this noise model during my internship and this feature is not implemented in LArSoft yet: I used data that the group provided me. The drawback is that they were able to produce only a very small dataset of 60000 waveforms[1] equally divided between noise and signal from muon events. As usual, data was partitioned between training and test set, 80% and 20%.

The hope is that with a bigger dataset the results of this chapter may be improved reaching higher accuracy.

The network I used for discriminating signal from noise is the same used for DUNE, depicted in Table 1.8.

On the test set an overall accuracy of 93.64% was achieved[2]

The actual classic de-convolution method has an accuracy of almost 100% for waveforms with more than 20000 collected electrons[3]. Below this level, the accuracy rapidly drops under 70% and lower. For the CNN-based method, the accuracy is stable[4] in the range of collected

[1] Less than a tenth of the one I used for DUNE.

[2] The network was trained for only 6 epochs: the loss function was monitored in order to avid overfitting. Clearly, due to the small quantity of data, it was not possible to train more. Again, with a bigger training set, the hope is to be able to train more and achieve an higher accuracy.

[3] But it must be done offline, since it is too computationally heavy. Hence, the drawback is that all the output from the DAQ must be saved.

[4] It stays between 93 and 95%.

electrons between 2000 and 20000.

4.2 Region of interest: peak localization with CNN

When we presented out work to the ICARUS researchers, we were told that the main requirement is to be able to localize the peak and not only to tell if the waveform is signal or not. Since the detector will operate at the sea level, there will be a significant muon flux and most of the waveform will be a signal: to save space, one has to perfectly localize the peak and discard everything else.

4.2.1 Windows model

I firstly tried the moving windows method described in Chapter 3. Inspecting waveforms and their labels clearly points out that the ICARUS Monte Carlo has problems in the information about the peak positions: the peak is never exactly where the simulation says it should be. Furthermore, as shown in Figure 4.1, the noise for ICARUS data resembles the peak shape. The CNN of the windows method is able to find the peak with high accuracy, but it is prone to a high number of false positives: it thinks there are many more peaks. This can bee seen in Figure 4.2: the network perfectly finds the true peak but, if we set the threshold at 0.5, then it would find a second false one.

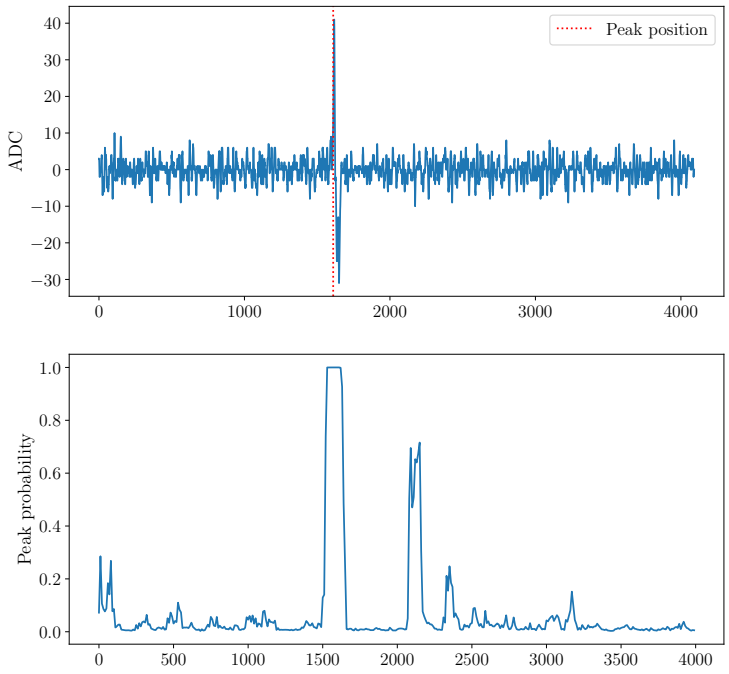


Figure 4.2: Output of the windows CNN method for a signal waveform.

This problem gets even worse with small number of electrons collected and lower peaks, as shown in Figure 4.3, where 2706 electrons were deposited.

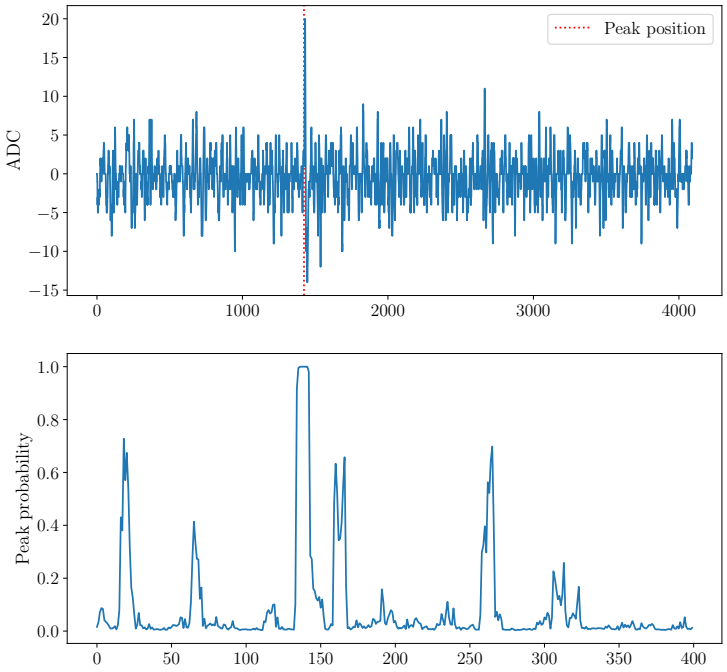


Figure 4.3: Output of the windows CNN method for a signal waveform.

4.2.2 Heatmap

Clearly the previous method is failing and, since the theoretical peak position from the Monte Carlo is not a reliable information, I had to develop something else.

The network that distinguish signal from noise has an high accuracy and, supposing that it is looking at the right thing to find signal (i.e.: the peak), I took again inspiration from a trick developed in image recognition, called heatmap[5], that it is used to understand what the network is looking at when it does its prediction.

The strategy is the following. For each waveform I a created a batch of new dummy waveforms. On the first one, the starting 100 bins are set to zero. On the second one, the next 100 bins are set to zero, on the third...

In this way we are suppressing possible region of interest: if the network predicted the waveform to be a signal with a probability of almost one, suppressing the region that made the network thinking so will drop the probability when making inference. If we plot the results for the new modified batch, there will be a region where the inferred value drastically drops:

[5] https://arxiv.org/abs/1311.2901

here it is the region of interest that is crucial for the network. This region is what makes the network thinking that a waveform is signal. If the network is looking at the correct feature (and the high accuracy points out so) that region is where the peak is.

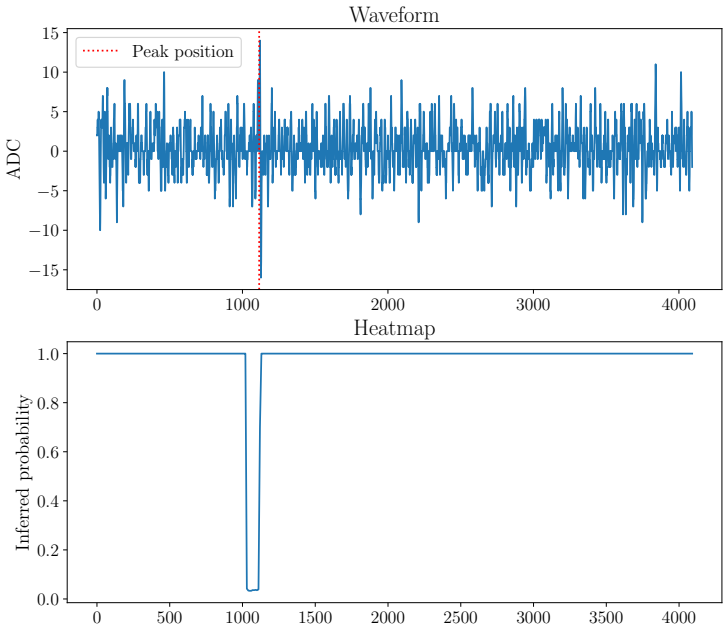


Figure 4.4: Heatmap for a waveform. The red line is the peak position given by the Monte Carlo.

In Figure 4.4 there is an output of the method previously described for a waveform. The graph below has to be read in the following way: if the first 100 bins are suppressed, the new waveform is predicted to be a signal with a probability of almost 1. The same is for the next 100 bins and so on. But if the bins around 1100 are suppressed, then the waveform is predicted to be a signal with probability close to 0 (so it is noise). This means that in that region there is what distinguishes a signal from pure noise. The Monte Carlo simulation tells us that the peak is there (and looking at the original waveform, indeed, it can be seen).

In Figure 4.5 I have reported a harder situation where the peak is not easily detectable by eyes. Still, the heatmap method is performing quite well. This methods clearly points out one of the powerful features of deep learning. I only gave to the network waveforms telling that some were signal and some noise. I never told the network what is different between the two. The network learnt itself to recognise noise and it automatically did learn to look at the correct feature, the peak, with an astonishingly precision.

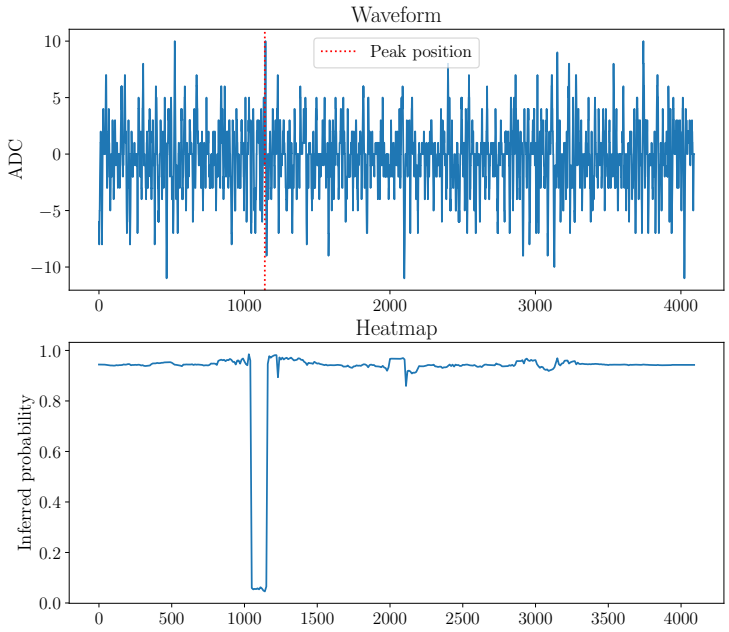


Figure 4.5: Heatmap for a waveform. The red line is the peak position given by the Monte Carlo.

4.3 Advanced noise filtering: autoencoder

The heatmap method is not perfect: when there is more than one peak in the waveform the method struggles to find the region of interest. This is easily explained: if one peak is suppressed, there is another and the CNN still recognises the waveform to be a signal. The result is that we never observe a drop in the predicted probability and we are not able to find the region of interest. Having more than one peak in one channel is quite a rare event, but still possible.

I wanted an algorithm reliable in all situations. All the previous methods were part of branch of the machine learning called "supervised learning". The term "supervised" means that we guide the network in what it should learn. Practically this is done with the "ground truth": the label that the network should learn to predict.

Another branch of machine learning is the so called "unsupervised learning". In this approach, we don't tell the network what to learn and we leave it free to automatically understand the feature to learn[6].

In the framework of unsupervised learning for feature extraction one powerful method is the autoencoder. The idea is to use a first algorithm to reduce the dimensionality of the

[6]Most of the applications of this techniques are in automatic suggestions, as film/music advise based on previous visualizations. Another developing application that uses unsupervised learning is the field of decision making, for instance in automatic driving cars.

data forcing to keep only the very important information; then, a second algorithm upsamples the original dimensionality trying to reconstruct the original data[7]. During the first stage, most of the information is lost and only the "very important" features are retained during the reconstruction of the full dimensionality. One possible application of the autoencoder is noise filtering, this is currently done in image processing for removing unwanted features as granularity and backgrounds.

In deep learning applications, the encoding part is generally a convolutional neural network that reduces the dimensionality throughout pooling layers; the decoding part is another convolutional network that increases the dimensionality throughout upsample layers. When enough data is present, the mixture of convolutional deep neural network and the idea of autoencoder is much more powerful than any classical methods.

4.3.1 Architecture

In reality I did something different: I used an encoder trained in supervised manner and a decoder left unsupervised.

As encoder I have used the CNN trained to distinguish signal form noise[8]; then, I dropped the last dense layer and I attached an equal but reversed CNN to work as decoder. For the training part, I froze the encoder weights and left the decoder to learn itself in unsupervised manner to reconstruct the waveform. The reason of the weights freezing is to keep the knowledge learnt in distinguishing signal from noise: I wanted to give some strong hints telling the autoencoder that the information to retain is what distinguishes signal from noise.

The unsupervised training is accomplished giving the network the same waveform as input and as target.

The autoencoder structure is in Table 4.1.

4.3.2 Results

In Figure 4.6a and 4.6b two waveforms and their encoded versions are depicted. In the first one a peak originated by $\sim$ 20000 collected electrons; the peak is clear but the autoencoder almost completely filters the noise. In 4.6b only 8159 electrons were collected, the peak is not easily detectable, but the autoencoders drop out the noise leaving the peak clear.

In Figure 4.7a I have shown a pure noise waveform, it can be seen that the autoencoder is not prone to false positives: the noise is left as pure noise.

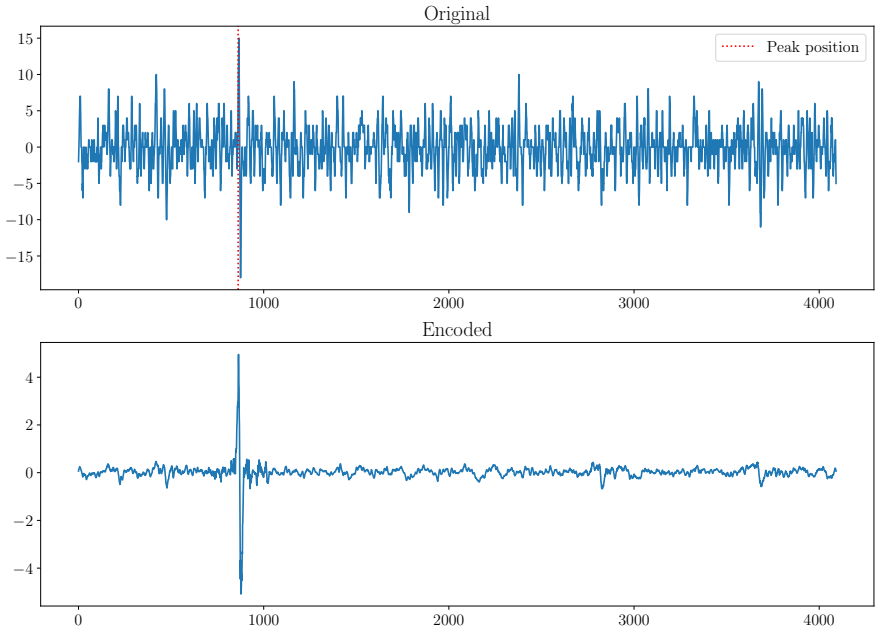
In Figure 4.7b the result for the situation where the heatmap was struggling: a double peak. The autoencoder works perfectly and it is able to reduce the noise also when more than one peak is present.

A possible application of the autoencoder is to use it as a first filtering stage. Then a second neural network may be applied to find the position of the peak[9]. Both the windows method and the heatmap may work better with filtered data.

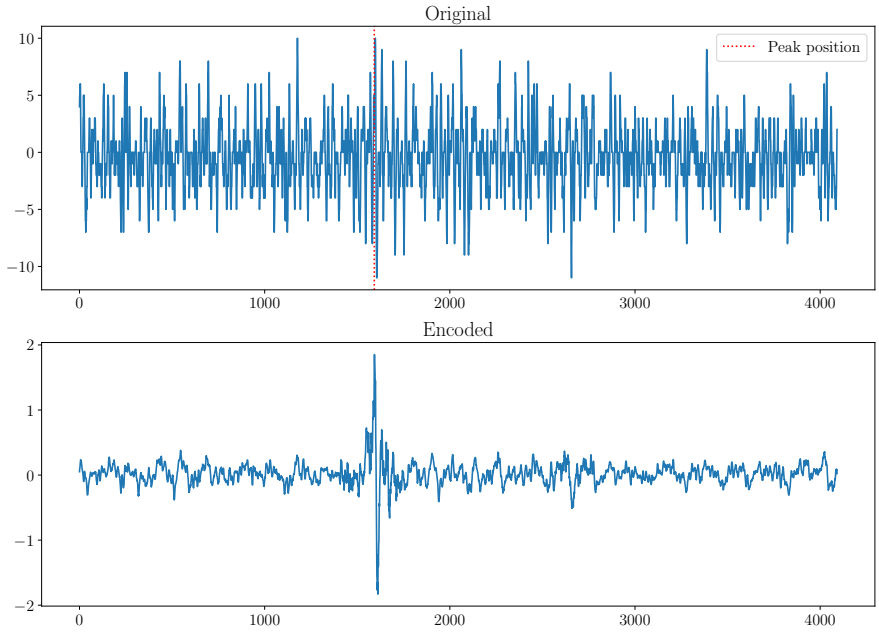
[7] In classical machine learning the most used algorithm is the Principal Component Analysis, maybe one of the reason of the success of the Spotify music client that became mostly appreciated for the correct music suggestions to their customers.

[8] Precisely, a slightly modified version with paddedd convolutional layers to easily match dimensionality.

[9] Actually, since after the autoencoder works very well in filtering, a classical method for peak finding may be considered.

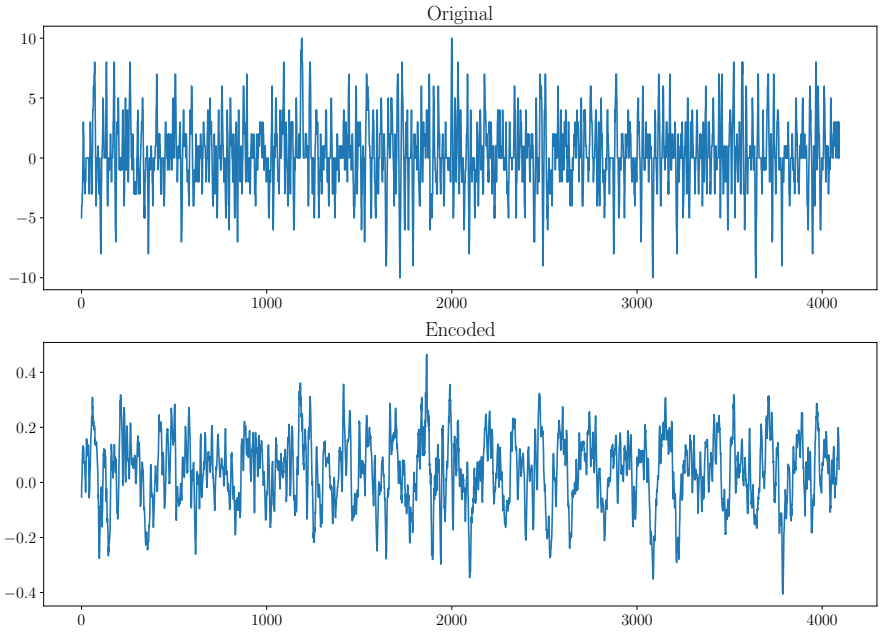


(a)

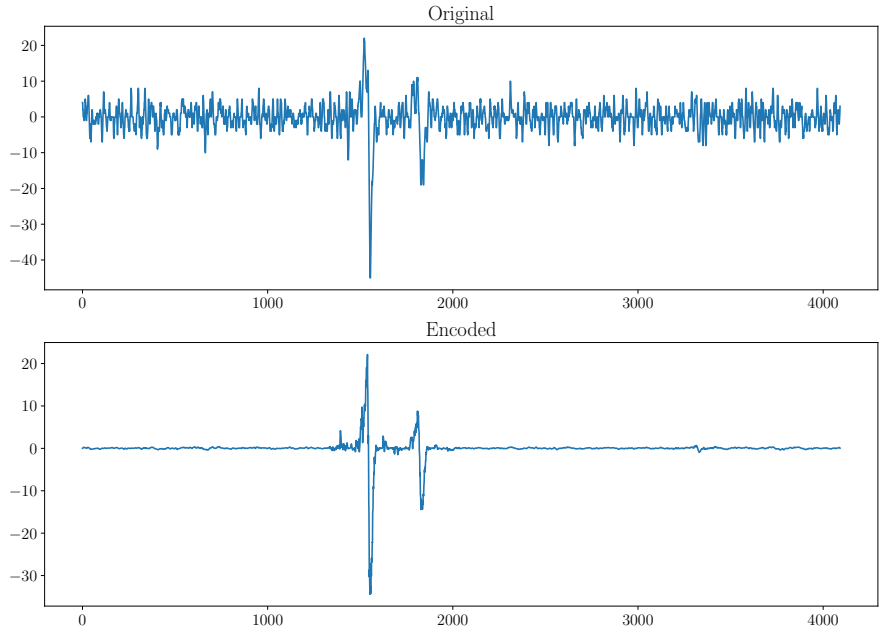


(b)

Figure 4.6: Encoded results for two waveforms. For the 4.6a the number of collected electrons is of the order of 20000 and the peak is clear. In 4.6b it is depicted a more compelling situation: the number of collected electrons is 8159 and the peak is very hard to detect.



(a)



(b)

Figure 4.7: Encoded noise in 4.7a. In 4.7b the result for a double peak waveform.

Layer	Filters/neurons	Kernel size	Output shape
(E) Conv1D + ReLu	16	3	(4092, 16)
(E) MaxPooling		3	(1364, 16)
(E) Conv1D + ReLu	32	5	(1364, 32)
(E) MaxPooling		3	(455, 32)
(E) Conv1D + ReLu	64	7	(455, 64)
(E) MaxPooling		2	(228, 64)
(E) Conv1D + ReLu	128	9	(228, 128)
(D) Conv1D + ReLu	128	9	(228, 128)
(D) UpSampling		2	(456, 128)
(D) Conv1D + ReLu	64	7	(456, 64)
(D) UpSampling		3	(1368, 64)
(D) Conv1D + ReLu	32	5	(1368, 32)
(D) UpSampling		3	(4104, 32)
(D) Conv1D + ReLu	4	5	(4100, 4)
(D) Conv1D + ReLu	1	9	(4092, 1)

Table 4.1: Autoencoder architecture. Each layer is labelled with (E) if it is part of the encoder, with (D) if it belongs to the decoder. Each convolutional operation is padded in order to have the output with the same dimension of the input. The number of trainable parameters (the decoder) is 215945. The encoder has 90912 non trainable parameters.

Conclusion

During my internship at Fermilab I was able to develop a deep learning method able to classify waveforms from a LArTPC detector.

The results I obtained are promising, but this is a preliminary work and many things are still to be studied. For instance, the autoencoder, used for the noise filtering, seems very interesting, but I did not implement a second algorithm that perfectly locate the peak from the filtered waveform. Furthermore, all my studies were done on simulations data; it may be worth to test the neural networks with real data from protoDUNE and compare the outcomes with the results of the actual approach.

After refining the deep learning algorithms, a depth comparison between the classic methods (de-convolution for ICARUS and zero suppression for DUNE) and the presented technique must be performed. This has to be done not only looking at accuracy performance, but also inspecting the costs of implementing the neural networks in each experiment. Only if the possible benefits outweigh the costs, this technique could have some real application.