Universitat Autònoma de Barcelona

Facultat de Ciències

Using deep learning to search for dark matter

Bachelor degree thesis presented by

Marco Praderio

Under the tutelage of

Mario Martinez

July 2018

Thanks

Many thanks to Rachel Rosten for having provided the inputs needed for training the developed models, and to Mario Martinez for his help in writing and correcting this document.

A special thanks is due to Bee the Data company and specially Jordi Zanca for providing the servers without which the trainings would have been impossible or, at least, a lot more complicated.

Contents

1 Introduction.

In this document we study the results of applying original Deep Convolutional Neural Networks for the task of classifying between $Z(\rightarrow \nu\overline{\nu}) + \text{jets}$ and $W(\rightarrow \tau\nu) + \text{jets}$ events generated via a Montecarlo simulation with particle level information.

The document focuses in developing and exploring neural network models designed for this classification task.

It is structured as follows.

In section 2 the physical motivations for the studies performed are explained.

In section 3 a detailed description of the original implemented deep learning models is given as well as a general introduction to deep neural networks.

In section 4 the inputs used for training and studying the implemented models are described together with the original pre-processing algorithms implemented.

In section 5 training results are presented together with a detailed analysis of the best performing models. This section includes a study of the improvements that using the best performing models would bring when if applied to events distributions as the ones described in [1].

Finally in section 6 the most important results are summarized and in section 7 further work is listed.

2 Motivation.

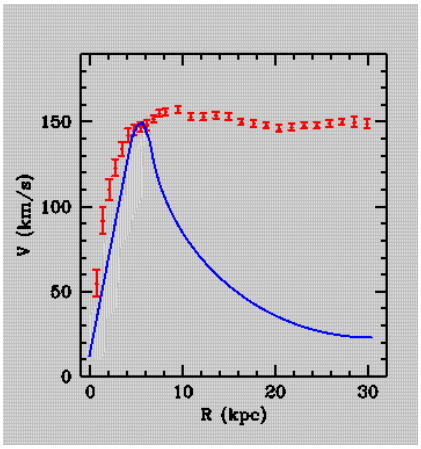


Figure 1: Rotation curve of a typical spiral galaxy. Image original from [2]. In red is reported the average observed rotation velocities of stars V on a spiral galaxy as a function of the the distance from the center of the galaxy R. In blue we can observe the graphic of the function $V(R)$ as predicted by the universal gravity law and the observed mass.As the distance from the galaxy's center increases the observed velocity becomes much greater than the predicted velocity.

The existence of dark matter was first postulated on year 1933 by the astronomer Franz Zwicky. The hypothesis of dark matter was born as a result of the measures Zwicky took regarding rotation velocities and light intensities of the Coma Cluster and similar measures regarding the stars in Messier 31 and the Milky Way. According to these measures, the rotation speed of stars far away from the galaxy's center, was far greater than expected (see figure 1).

Zwicky noticed that this discrepancy between the observed and predicted velocities could be explained by introducing the existence of a "missing mass" (dark matter) that constitutes the majority of the mass on the known universe.

The existence of dark matter is nowadays commonly accepted and is used to describe all the universes matter that cannot be explained via the standard model.

The currently dominant hypothesis for explaining dark matter's nature identifies it with a non-baryonic Weakly Interactive Massive Particle (WIMP) [3] having a mass between a few GeV and one TeV and an electroweak scale interaction cross section. The main reason for the WIMP hypothesis popularity is the fact that it predicts the correct relic density for non relativistic matter in the early universe [4] as reported by multiple measures [5, 6].

It may be possible to pair-produce WIMPs accompanied by a jet (i.e. narrow cone of hadrons and other particles produced by the hadronization of a quark or gluon) in proton-proton collisions at the LHC as shown in figure 2. Since this events would produce a signature of a jet and large missing transversal momentum E_T^{miss}, many studies have been conducted searching for dark matter in events with such signatures [1, 7, 8].

The backgrounds events found when searching for this signatures are: $Z(\rightarrow \nu\overline{\nu}) + \text{jets}$, $W(\rightarrow \tau\nu) + \text{jets}$, $W(\rightarrow \mu\nu) + \text{jets}$, $W(\rightarrow e\nu) + \text{jets}$ and, in smaller measure, $Z/\gamma*(\rightarrow l^+l^-) + \text{jets}$ ($l = e,\ \mu,\ \tau$), multijet, $t\bar{t}$, single-top, and diboson (WW, WZ, ZZ) events.

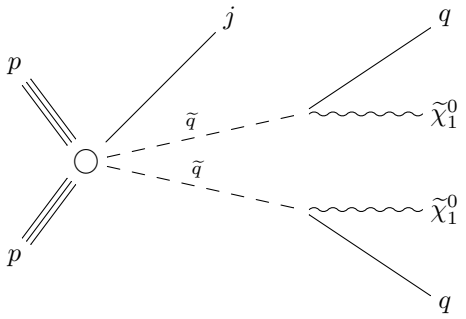
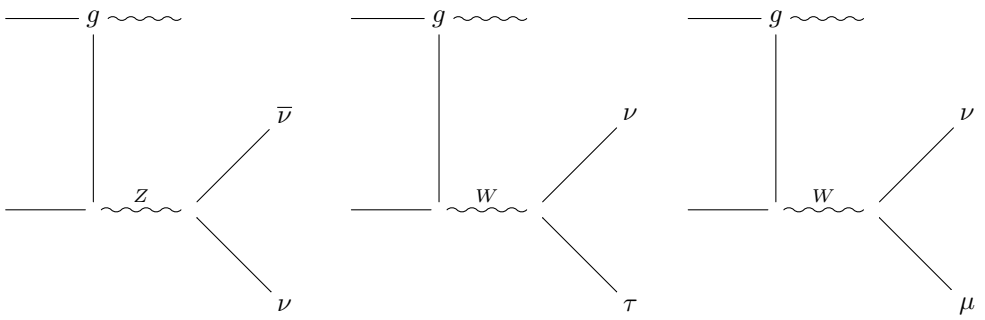


Figure 2: Feynman diagram for a possible model of pair-production of WIMPs ($\widetilde{\chi}_1^0$) and jet. Diagram obtained from [1].



(a) $Z(\to \nu\overline{\nu}) + \text{jets}$ (b) $W(\to \tau\nu) + \text{jets}$ (c) $W(\to \mu\nu) + \text{jets}$

Figure 3: Feynman diagrams for background events.

We focus on the 3 most common of this background events:

$Z(\to \nu\overline{\nu}) + \textbf{jets:}$ The collision produces jets and a Z boson that decays into a pair of neutrino and anti-neutrino (see figure 3a).

$W(\to \tau\nu) + \textbf{jets:}$ The collision produces jets and a W boson which decays in a tau, neutrino pair (see figure 3b).

$W(\to \mu\nu) + \textbf{jets:}$ The collision produces jets and a W boson which decays in a muon , neutrino pair (see figure 3c).

Given that both the neutrinos produced in $Z(\to \nu\overline{\nu}) + \text{jets}$ events and the WIMPs produced in the events shown in figure 2 cannot be detected then both events should result in the exact same signature (containing only the jets and a large E_T^{miss}). Because of this we consider $Z(\to \nu\overline{\nu}) + \text{jets}$ to be an irreducible background event and, therefore, any tool able to distinguish between $Z(\to \nu\overline{\nu}) + \text{jets}$ events and other background events should, if the distinction is not done using the jets information, be able to distinguish between WIMPs production and that same background events.

In this document we use deep neural networks to distinguish between $Z(\to \nu\overline{\nu}) + \text{jets}$ events and the second most common background events ($W(\to \tau\nu) + \text{jets}$) under the hypothesis of perfect detector performance.

We also use the same trained neural networks to see wethe this can be used to distinguish between $Z(\to \nu\overline{\nu}) + \text{jets}$ events and $W(\to \mu\nu) + \text{jets}$. This would allow to determine if the neural net model could be applied in order to reduce the remaining $W(\to \mu\nu) + \text{jets}$ background.

3 Neural Net models.

3.1 Introduction to Deep Learning.

A Deep Neural Network (also called DNN, Neural Network, Neural Net or NN) is a computing system that tries to emulate the biological neural networks of animal brains[9]. Neural nets were first conceptualized by Alan Turing in the earliest stages of informatics. However it hasn't been until recently that technology has advanced enough for NN models to be computationally viable. In the last years NN models have been showing increasing performances in fields as diverse as natural language processing [10], computer vision [11] and physics [12, 13].

3.1.1 Neural Net architecture.

In order to give a complete description of how a neural net works we first give a high level picture of a general neural net's architecture and then proceed with a more detailed description of each one of its components until we reach the most basic components of a neural net (the weights and biases of a neuron). In order to avoid lingering too long in this section, we focus solely on the notions necessary for describing our neural net models.

High level picture. The neural net models we are interested in are *feed-forward neural networks*. This neural networks take as input one or more real valued tensors (in our case 1) apply to them a series of *layers* followed by *activation functions*[1] resulting in one or more real valued output tensors (in our case 1).

Activation functions. *Activation functions* are, as the name suggests, functions taking a real valued tensor as input and returning a real valued tensor as output. This functions can be very diverse, however, in order for the NN model to perform correctly they must be non-linear functions. It can be proven [15] that, because of this non-linearity NN, models are able to approximate any continuous functions from $\mathbb{R}^n$ to $\mathbb{R}^m$. In our case we use the NN models to approximate the function assigning an event to the probability of it being a $Z(\to \nu\overline{\nu})$ + jets.

Most of the activation functions can be though simply as functions from $\mathbb{R}$ to $\mathbb{R}$ applied element-wise to every value of the input tensor. That is the case of the *relu* (rectified linear unit, see figure 4) activation function. This is the most used activation function used in our model and can be defined as

$$\text{relu}(x) = \max(0,\, x),\ \forall x \in \mathbb{R}.$$

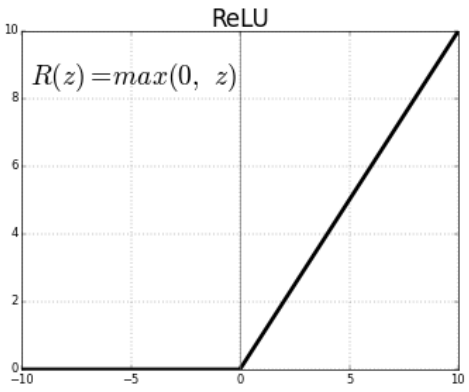


Figure 4: Plot of the relu function. Image extracted from [14].

There are however other types of activation functions as the max-pooling function (often called max-pool layer). The max-pooling function is, besides the relu function, the only other activation used in our models. As we use it the max-pooling function acts on an input 3-dimensional tensor by dividing it in disjoint squares of shape $2 \times 2 \times 1$ (padding the input tensor with zeros if necessary) and returning a 3 dimensional tensor whose value in the cell $(i,\, j,\, k)$ is the maximum value contained in the square $(i,\, j,\, k)$ of the input tensor. A more graphical and intuitive description of the max-pool layer can be seen in figure 5.

[1] A more detailed description of "layer" and "activation function" is given next. For the moment they can be thought as functions that take one tensor as an input and return another tensor.

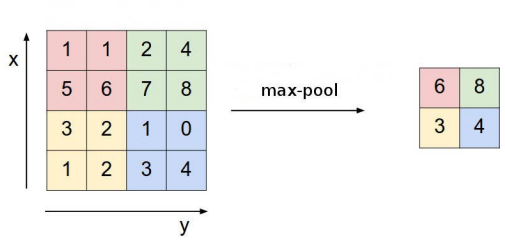


Figure 5: Max pool activation function, as it is used in our NN models, acting on a $4 \times 4 \times 1$ input tensor. Image obtained by editing an image in [16].

Layers. As activation functions, layers can be thought as functions taking as inputs real valued tensors and returning real valued tensors. However there are two main differences between them. First, a Layer is a set of linear (instead of non linear) functions called *neurons*. Second, as is later explained, the linear functions are not fixed but they can change as the NN model is being trained.

Among the possible types of layers two of them are the most common and are the only ones used on our models. This layers are: *fully connected* (or *dense*) layers and *convolutional* layers.

Dense layers. The architecture of a dense layer is defined entirely by the number of *neurons* (m) it has and the size (n) of the input tensor. Each one of the neurons of a dense layer takes as input the n values of the input tensor and returns a linear combination of them. A 1-dimensional tensor of size m is then generated by joining the outputs of each one of the neurons.

Convolutional layers. Unlike dense layers, convolutional layers can take as input only 3-dimensional tensors and their output is also a 3-dimensional (instead of 1-dimensionl) tensor.

The architecture of a convolutional layer can be defined by 3 parameters:

1. **filters:** An integer (m) representing the number of neurons. It coincides with the depth of the output tensor.
2. **Kernel shape:** A tuple of integers (h, w) used to define the size of the subset of values of the input tensor that each neuron takes as inputs.
3. **Strides:** A tuple of integers (Δh, Δw) that, together with the kernel shape, completes the description of the subsets of values of the input tensor that each neuron takes as inputs.

Given this parameters and denoting by n the depth of the input tensor we can define the action of a convolutional layer on an input tensor as follows.

First we divide the input tensor in (possibly intersecting) blocks of shape $h \times w \times n$. These blocks must be such that the distance between the top left cell of a block and that of the one immediately to its right must be of $(0, \Delta w)$ cells while the distance with the one immediately below it must be of $(\Delta h, 0)$. It is also necessary for the top left cell of the top left block to be the top left cell of the input tensor and the bottom right cell of the bottom right block to be the bottom right cell of the input tensor. If it is not possible to create such a set of blocks the input tensor is padded with zeros until such a division is possible.

Once this division is done the values of the block (i, j) are used as input of the neuron number k , for every possible i, j and k. That neuron performs a linear combination of those values and the result is stored in cell (i, j, k) of the output tensor.

A graphical explanation of the above described process can be observed in figure 6.

Convolutional layers are often used in NN models designed for image processing problems [17, 18] us the one we are treating [19]. In this type of problems convolutional layers present three main advantages compared to dense layers.

First, convolutional layers have a, by far, lower number of neurons which makes the training much cheaper computationally speaking. This allows to use the saved computational power to add more layers to the model, which has been shown to lead to performance improvements [17, 18].

Second, convolutional layers are able to preserve spatial information of the input image and aggregate information of neighbor pixels. This way a NN formed by convolutional layers followed by activations functions is able to combine pixel level information to obtain information about edges on an input image, combine this to get information relative to object parts in an image, use this to get information of objects in an image, ...

and so on until the NN "sees" a complete picture of the input image. This process can be observed in figure 7. This does not happen with Dense layers which "see" images as they would "see" a one dimensional tensor thus adding the problem restoring the original spatial information.

Third is their robustness to image rotations and translations [20]. This is of particular interest in the problem at hand since particles can be generated with equal probability for any azimuthal angle. As is later explained (4.1) this is equivalent to a random translation of the model's input values over an axis. Therefore, it is of great importance for our model to be robust under translations.

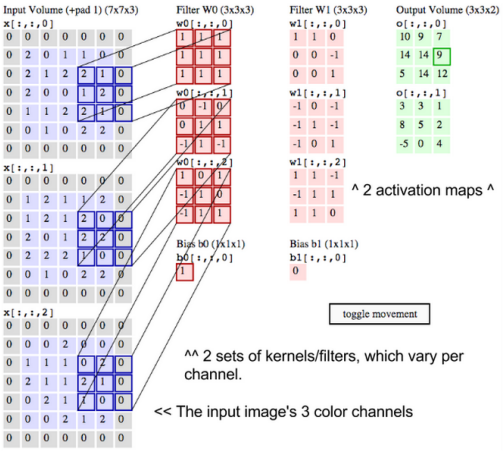


Figure 6: Image extracted from [16]. Graphical example of how a convolutional layer acts on $5 \times 5 \times 3$ input tensor. In blue is shown the input tensor, in red we can observe the values of weights and bias of each neuron and in green is the output.

Neurons. The neurons and, more specifically, its weights and bias, are the lowest level components of a neural net. The structure is defined by a finite set of variables called weights $W = w_1, \ldots, w_n$ and another single variable called bias b. A neuron acts as a linear function on a set of real input values $X = x_1, \ldots, x_n$ having the same cardinality as the set W. More precisely the result of applying a neuron N_e to the set or real values X is the linear combination

$$N_e(X) = \sum_{i=1}^{n} w_i x_i + b.$$

The fact that makes this very simple function the most important for the good performances of neural nets is that the weights and bias are not fixed. Therefore the neurons can be "trained" to make the neural net "learn" to approximate a wide variety of continuous functions [2].

3.1.2 Neural net training.

Once understood the general architecture of a neural net all that remains to be explained is how a neural net "learns" the adequate weights and biases for every neuron. This learning process can be divided in three main steps: choosing of a *loss function*, implementation of an *optimization algorithm*, choosing a *stopping condition*.

Loss function. A loss function is a "distance" between the neural net outputs and the outputs of the function we want the neural net to learn (i.e. the function returning the probability of some event being a $Z(\to \nu\overline{\nu}) + \text{jets}$ event).

In our case the output of the neural net is a tuple of two real values (x'_1, x'_2) which, after applying a softmax function

$$(x_1,\, x_2) = \frac{\left(e^{x'_1},\, e^{x'_2}\right)}{e^{x'_1} + e^{x'_2}},$$

gives us a probability distribution with the probability of the input data[3] corresponding to a $Z(\to \nu\overline{\nu}) + \text{jets}$ event (x_1) or to a $W(\to \tau\nu) + \text{jets}$ event (x_2). Therefore we want the output $x = (x_1,\, x_2)$ to match the ground truth $y = (y_1,\, y_2)$, where $y = (1,\, 0)$ if the input data corresponds to a $Z(\to \nu\overline{\nu}) + \text{jets}$ event and $y = (0,\, 1)$ otherwise. The loss function we have decided to use is a cross entropy loss function which is defined as

$$l(x, y) = \sum_i -\log\left(1 - (x_i - y_i)\right).$$

[2] Changing the neural net model we wold be able to approximate any continuous function [15].

[3] In section 4 this input data is described in detail.

We have chosen this function since it is the most commonly used function to solve classification problems as the one at hand [21, 17, 18].

To this loss function we have added (with a weight of 0.01) an $l2$ regularization loss. This regularization loss is simply the square sum of every weight in every convolutional layer of the neural net. This regularization loss is commonly used in machine learning to avoid the over-fitting phenomenon (i.e. the neural net performs very well on training set but poorly on testing set) [22]. The idea behind this function is that, by minimizing it, we are forcing the neural net neurons to not give extremely high weights to any input value. This should prevent the neural net from excessively depending on a small set of properties of the input data (that could be characteristic of the training set).

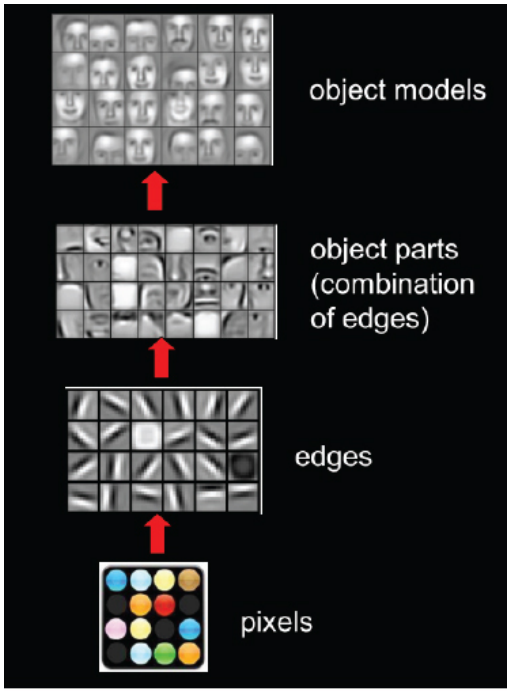


Figure 7: Image extracted from [23]. In the figure above it is shown an intuitive representation of how, after every convolutional layer, a NN is able to gain ever more global information on the input image.

Optimization algorithm. Before starting with the description of the optimization algorithm we introduce some conventions and notation.

All weights and biases of a NN model are thought as a single vector called weight vector and denoted by $v \in \mathbb{R}^n$.

The ideal loss function $F : \mathbb{R}^n \to \mathbb{R}$ is defined as the function that takes as input the weight vector v and returns the average of the previously described loss function over all (x, y) tuples that could be obtained by running the NN model, with fixed v, over all possible inputs.

Given a weight vector v we can use the back propagation algorithm explained in [24] in order to compute the direction δ (gradient) along which we should variate v in order to locally minimize the ideal loss function (gradient descent). Since it is impossible to compute the function F at every training step, we approximate F by averaging the previously defined loss over a random subset of the training data. We then use this approximation to compute the gradient δ.

The naive idea would now be that of initializing the vector v_0 at random and then fine tune it recursively by computing $v_{i+1} = v_i + \epsilon\delta_i$, where ϵ is a small positive number and δ_i is the computed approximation of $F(v_i)$. However this idea encounters multiple problems. The fact that ϵ is constant could result in the choosing of a value too big at some stages of the training, resulting in the vector v oscillating around an optimal value without ever approaching it. The same value could also bee too small at other training stages thus resulting in a huge increase if training time.

In order to solve this inconvenient we used as optimization algorithm the Adam algorithm [25]. This presents two main characteristics. First it decays the value of ϵ (initialized at 10^{-4}) over time, which makes it possible for the vector v to assume values arbitrarily close to its optimum value without endlessly oscillating around it. Second it stores information about the momentum at which the approximated ideal loss function is decreasing. It then uses this information to detect when the vector v is in a region where ideal loss function F is particularly flat. Under under this conditions the Adam algorithm increases ϵ thus making possible a fast escape from this flat regions and thus decreasing training time.

Weights and biases are initialized at random but following two different probability distributions. Weights are initialized using a normal distribution centered at 0 with standard deviation of 0.01 while biases are initialized following a uniform distribution in the interval $(-0.5, 0.5)$.

In order to prevent over-fitting , we have introduced Dropout between dense layers which is used only during training. Thus, during training, we randomly (with probability 0.1) replace a weight in a dense layer by a

zero, use the replaced zero value to compute the output x and then restore the original value. It has been shown [26] that such a technique can help preventing over-fitting.

Stopping condition. In order to decide when the training was finished we have applied two different methods.

The first method is *early stopping*. This method consists in observing, every few training steps (in our case 200), the accuracy (#correct predictions / #total inputs) that the NN obtains when detecting $Z(\to \nu\overline{\nu})$+jets events against $W(\to \tau\nu)$+jets events. When we observe that accuracy on the training set increases while that on testing set decreases (see figures 9 and 10) we interpret it as a sign of over-fitting and stop the training.

The second condition under which a training is stopped is met when, after a number of epochs (i.e. iterations over the training set) comparable to that needed by our best performing models in order to achieve maximum efficiency, the model shows no sign of improvement. Under this condition the training is stopped and the model is considered unable to solve the classification problem at hand. However we cannot completely discard the possibility that, some of the models meeting this condition, could achieve a performance comparable to that of our best performing models if given enough time (although it seems very unlikely).

3.2 Implemented models.

The two original implemented models were designed using as an inspiration the models described in [27, 13, 28, 17]. Our models are thus sequential models that take as input a single 3-dimensional tensor and apply to it a series of convolutional layers each followed by, at least, a relu activation. The result of this series of convolutions and relu activations is then fed as input to two consecutive dense layers with a relu activation in between.

The models can both be divided in four blocks, three of them containing only convolutional layers and the last one containing only dense layers. Both models can be described block by block as follows

Convolutional_block_1 Formed by 3 convolutional layers all of them followed by a relu activation. All layers have kernel shape (3, 3) while their filters are 64, 128 and 256 respectively. The first two layers have strides (1, 1) while the strides of the last layer differ between models. In the first model the third layer's strides are (2, 2) thus resulting in a factor 2 down-sample of height and width of the input tensor. On the second model the strides of the last layer are (1, 1) thus resulting in no down-sample. To compensate for this, after the last relu activation, the second model adds a max-pool layer. Because of this the shapes of the output of the first block in both models is the same.

Convolutional_block_2 Formed by 2 convolutional layers instead of 3. As in the first block all layers are followed by a relu activation and both layers have kernel shape (3, 3). The layers filters are 128 and 256 respectively and the first layer strides are (1, 1). The strides of the last layers differ from model to model in a completely analogous way than they did in the first block. Thus while the last layer in the first model has strides (2, 2) the same layer in the second model has strides (1, 1) but it is followed by a max-pool layer.

Convolutional_block_3 Formed by a single convolutional layer, followed by a relu activation. This layer is the same in both models. It has filters equal to 128, kernel shape (1, 1) and strides also (1, 1).

Dense_block This last block is formed by two dense layers, the first of them followed by a relu activation. This layers have a number of neurons equal to 128 and 2 respectively and are identical in both models.

As we can observe the only difference between the first and the second model is that the first increases the strides of the last convolution on the first and second blocks in order to down-sample the input tensor height and width, while the second model performs the same down-sample by applying a max-pool layer. We therefore refer to the first model as "strides model" and to the second model as "max-pool model". A graphic description of the strides model can be found in figure 8 while a table describing both models can be seen in table 1.

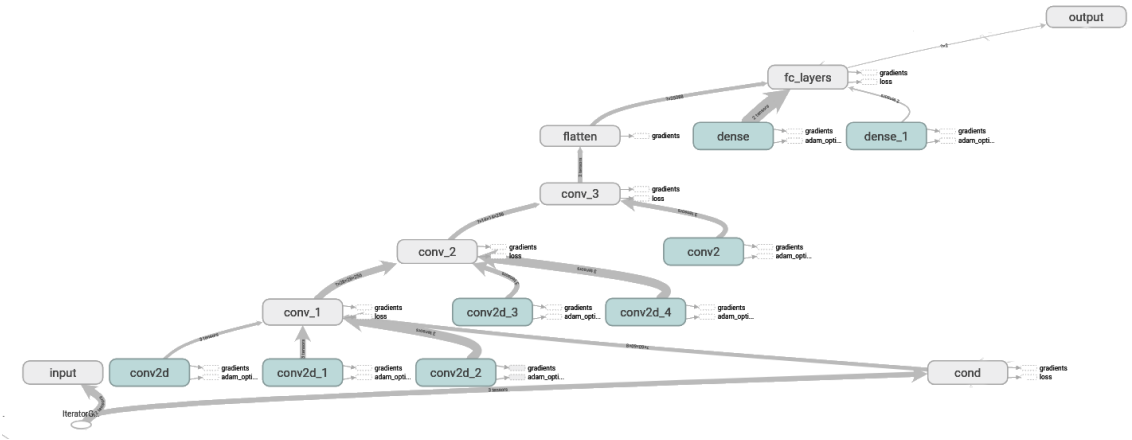


Figure 8: Diagram of the strides model as extracted from tensorboard. As we can observe the model can be divided in three convolutional blocks with 3, 2 and 1 convolutional layer respectively, followed by a block with 2 dense layers. The activation functions are not shown in the diagram but every layer except from the "dense_1" layer (the last one) is followed by a relu activation. We can observe in the diagram a block called "flatten" that has not been mentioned earlier. This is because this block's only function is that of reshaping a tensor in order to feed it to a dense layer as defined by tensorflow (the only difference with our definition is that dense layers in tensorflow can only take as input 1-dimensional tensors). The "cond" block that can be observed on the bottom right side of the diagram is used exclusively to distinguish between training and testing sets and, therefore, is not considered as a part of the NN model.

	strides model			max-pool model		
	kernel shape	strides	neurons/filters	kernel shape	strides	neurons/filters
Convolution 1_1	(3, 3)	(1, 1)	64	(3, 3)	(1, 1)	64
Convolution 1_1	(3, 3)	(1, 1)	128	(3, 3)	(1, 1)	128
Convolution 1_1	(3, 3)	(2, 2)	256	(3, 3)	(1, 1)	256
max-pool 1	no			yes		
Convolution 1_1	(3, 3)	(1, 1)	128	(3, 3)	(1, 1)	128
Convolution 1_1	(3, 3)	(2, 2)	256	(3, 3)	(1, 1)	256
max-pool 2	no			yes		
Convolution 1_1	(3, 3)	(1, 1)		(3, 3)	(1, 1)	
Dense 1	-	-	128	-	-	128
Dense 2	-	-	2	-	-	2

Table 1: Description of the layers of the strides and max-pool models. Each layer, except for the last, is followed by a relu activation function.

4 Inputs and pre-processing algorithms.

In this section are described the inputs used to train and test the NN models.

In sub-section 4.1 we give a brief description of the data generation process and a more complete description of the data topology.

In sub-section 4.2 we describe three different pre-processing algorithms applied to the described data.

4.1 Inputs.

The inputs used for training the neural net was generated via a Montecarlo simulation that replicated proton to proton collisions leading to the three different types of background events discussed in section 2. The processes where simulated assuming perfect detector response. Therefore the ultimate performance of the trained NN models would probably degenerate due to imperfect detector effects. However the results obtained can give us an idea on how the NN architecture would perform if trained with inputs generated including a detailed detector simulation.

The inputs corresponding to each one of the three events is separated in different slices depending on the norm of the generated boson's momentum (P_t). The slice with high P_t contains events with $P_t > 1000\,GeV$, the slice with medium P_t contains events with $500\,GeV < P_t < 1000\,GeV$ and the slice with low P_t contains events with $280\,GeV < P_t < 500\,GeV$.

Since new physics is likely to concentrate at very high P_t the neural net models are trained using the $Z(\rightarrow \nu\overline{\nu}) + \text{jets}$ and $W(\rightarrow \tau\nu) + \text{jets}$ events from the high P_t slice while the remaining slices are used for studying the resistance of the trained models to P_t variation. Good performance in all P_t slices would indicate that a single NN model could be applied to all P_t range.

The training set is formed by 70K $Z(\rightarrow \nu\overline{\nu})+$jets events (from now on signal) and 70K $W(\rightarrow \tau\nu)+$jets events (from now on background) from hight P_t slice. The testing set used during training to check for over-fitting and trigger the stopping condition is formed by 20K signal events and 20K background events from the same P_t slice. Finally for every remaining P_t slice and event we use an 8K events dataset for further studying the best performing models.

The inputs corresponding to every event are presented to the the NN as $100 \times 126 \times 8$ tensors build as follows.

For every particle its pseudorapidity η [4] and the azimuthal angle ϕ of the particle's momentum is computed.

The interval $[-2.5,\, 2.5) \times [-3.15,\, 3.15)$ of the $\eta \times \phi$ plane is then uniformly divided in a grid with cells of shape 0.05×0.05. For each one of these cells the following 8 different physical magnitudes are computed:

pt_{all}**:** The sum of the momentums of all particles pointing that cell.

n_{part}**:** The total amount of particles in that cell.

charge: The sum of the electrical charge of all particles in that cell.

pt_{μ}**:** The sum of the momentum of all muons in that cell.

pt_{e}**:** The sum of the momentum of all electrons in that cell.

EMF: The electromagnetic fraction, defined as the amount of energy deposited in an electromagnetic calorimeter by the particles pointing in that direction (e, γ, π^0).

pt_{jet}**:** The sum of the momentums of every jet detected in that cell.

jet_{width}**:** The width of the detected jet.

[4] Pseudorapidity is computed from the angle θ that the particle's momentum forms with the positive direction of the beam axis as $\eta = -\ln\left[\tan\left(\frac{\theta}{2}\right)\right]$.

The values of the 8 physical magnitudes registered in the cell (i, j) of the grid are copied on the cell (i, j, k) of a tensor T of shape $100 \times 126 \times 8$ for every possible i, j and every $k = 1, \ldots, 8$. Thus the cell $(i, j, 1)$ of T contains the pt_{all} registered in cell (i, j) of the grid, the cell $(i, j, 2)$ contains the registered n_{part}, the cell $(i, j, 3)$ the charge,....

The grid cells for which no particle or jet is registered have an associated default pt, charge, EMF and jet_{width} values of 0. At first this default value could seem to be confusing since it is possible for a cell to register a charge of 0 while one or more particles are in that cell (i.e. neutrally charged particles or particles with opposite charges). However, as it is observed in [19], NN are able to learn high level physic information if enough low level physic information is given. It is therefore reasonable to assume that, since our NN modelsare fed with information relative to the number of particles in each cell, they should be able to learn to distinguish between situations where a zero charge comes from the absence of detected particle and the case where it comes from a total charge of 0.

The units of the magnitudes listed above (except for the n_{part} magnitudes) are standardized. More precisely, before feeding the input tensor to the NN each layer is divided by the mean value of its non zero entries computed by iterating over the training set. The non-zero entries of all of the eight magnitudes assume therefore values on the order of unit. According to [29] the described operation should ease the learning process of the NN increasing training speed and improving training results.

4.2 pre-processing algorithms.

Due to the sparse nature of the inputs (i.e. most of the values registered in the input tensors are zeros) one more pre-processing step is needed before training the NN models with the described inputs. Without this last pre-processing step the data sparsity could lead to an increased difficulty during the training process as explained in [30].

In order to solve this problems three different original compression algorithms are implemented. The NN models are then trained with the outputs obtained from each of this algorithms.

The pre-processing algorithms can be divided in three steps:

1. Two layers are added to the initial tensor thus converting it to a $100 \times 126 \times 10$ real valued tensor T_{init}. This two layers contain respectively the number of row and column of the cell (i.e. the η and ϕ of the cell's center). This step is performed in order to feed the neural net with enough information to revert the compression process. In other words, by doing this we make sure that the compression algorithms do not imply any information loss. During this step we also create a tensor T of shape $100 \times 126 \times 10$ filled exclusively with zeros

2. This step is algorithm dependent and is later explained in detail for every compression algorithm.

3. The first 60 columns and rows of the tensor resulting from the previous step are cropped from it thus obtaining a new tensor of shape $60 \times 60 \times 10$. This tensor is then returned as the compression algorithm's output. At first sight it would seem that this step is inevitably linked with some information loss since, almost three quarters of the original tensor, are discarded. However, we have been careful to check that no information was actually lost and, in fact, we have confirmed that, most of the time, all information is compressed in the first 30 rows and columns (as we can clearly observe in figures 9 and 10). We can therefore confirm that, this step does not imply any information loss either.

The three algorithms are named as: *empty cell compression*, *empty row and column compression* and *empty row and column stochastic compression.* Algorithms are listed from highest to lowest obtained data density after compression.

4.2.1 Empty cell compression.

Pseudo-code for this compression algorithm can be found in appendix A.

Figure 9: Compression results of applying *empty cell compression* algorithm over over inputs corresponding to a $W(\rightarrow \tau\nu) + \text{jets}$ event. Only the EMF information is shown, the rest of magnitudes result in similar figures.

Figure 10: Compression results of applying *empty row and column compression* algorithm over inputs corresponding to a $W(\rightarrow \tau\nu) + \text{jets}$ event. Only the EMF information is shown, the rest of magnitudes result in similar figures.

This algorithm can be simply thought as iterating over every row and removing empty cells from them. Once the process of erasing empty cells is finished we add zero padding to the resulting rows and columns until we reach a tensor shape of $60 \times 60 \times 10$.

This algorithm is the one able to condense the greatest amount of information (non-empty cells) in the least number of cells. However this algorithm results in a more efficient compression over the ϕ axis than the η axis and the extreme compression makes it more complicate to intuit the original shape.

The compression results can be observed in figure 9 were it can be clearly seen how the non-empty cells are extremely compressed in the top left part of the images and no non-empty cells are in a row or column above the 30-th.

4.2.2 Empty row and column compression.

The *empty row and column compression* algorithm slightly modifies the previous algorithm in order to avoid a such an extreme compression that seemly looses spatial information. Moreover the current algorithm, unlike the last, compresses uniformly over both the ϕ and η axes. Pseudo-code for implementing this algorithm can be found in appendix A.

Intuitively explained this algorithm removes first all empty rows and then all empty columns and returns the first 60 rows and columns of the resulting tensor, padding it with zeros if necessary until the shape $60 \times 60 \times 10$ is reached. As shown in figure 10 this algorithm seems to preserve spatial information better that the empty cell compression algorithm (i.e. non-empty cells are placed further apart and the non-empty cell distribution resembles more that of an uncompressed input). However, as with the empty cell compression algorithm, all non-empty cells can be found in the first 30 rows and columns of the final tensor which results in having a great amount of unused space.

To avoid this the next compression algorithm was implemented.

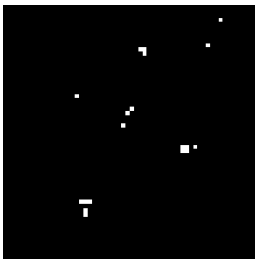


Figure 11: Compression results of applying *empty row and column stochastic compression* algorithm over inputs corresponding to a $W(\rightarrow \tau\nu)+$jets event. Only the EMF information is shown, the rest of magnitudes result in similar figures.

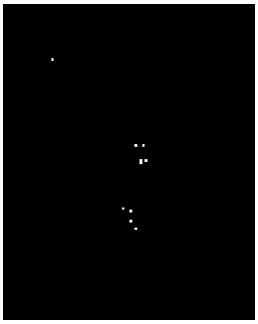


Figure 12: The image shows the EMF information of $W(\rightarrow \tau\nu)+$jets event, the rest of magnitudes result in similar figures. Each pixel corresponds to a 0.05×0.05 cell on the grid of the interval $[-2.5,\ 2.5)\times[-3.15,\ 3.15)$ on the $\eta\times\phi$ plane as explained in section 4.1.

4.2.3 Empty row and column stochastic compression.

The *empty row and column stochastic compression* algorithm is very similar to the previous one. The main difference between the two algorithms is that the current algorithm does not remove all empty rows and columns but only a controlled number of them chosen at random. Doing this we avoid having all non-empty cells gathered in a small portion of the resulting tensor. In other words we are able to better distribute the event information over all the tensor. Pseudo-code for implementing this algorithm can be found in appendix A.

As we can observe from figure 11 this algorithm distributes the non-empty cells over the whole tensor more efficiently than the two previous algorithms. However it shows two major disadvantages with respect to them.

The first disadvantage is the introduction of a stochastic component that could difficult the training of the NN model.

The second disadvantage is that the maximum final density of non-empty cells is much lower than it was after applying the two previous algorithms. In other words the compressed tensor may still present the sparse data problem we had prior to applying any compression algorithm.

4.2.4 Uncompressed.

In order to perform a complete analysis on the optimum pre-processing of inputs needed to feed the NN models we also trained both models with uncompressed inputs. A visual representation of the uncompressed inputs similar to that shown in figures 9, 10, 11 can be observed in figure 12

5 Training results.

In this section are reported the training results for every possible combination of pre-processing algorithm and NN models among those described above.

First we study the progress of the models performances over training time.

Then the obtained results are used to choose two of the best performing models and a more complete examination of this models performances is presented.

During this section the convention of referring to $Z(\rightarrow \nu\overline{\nu})$+jets events as signal events and to $W(\rightarrow \tau\nu)$+jets events as background events is maintained.

5.1 Training curves.

To evaluate the model during training time we computed at every step the precision (#correct signal predictions/#signal predictions) and recall (#correct signal predictions/#signal ground truth) over a random sub-sample of training and testing set. Taking the mean value between precision and recall we obtained the accuracy used to evaluate our models. An input was classified as a predicted signal event if the predicted probability of it being a signal was above 0.5. We refer to this choice as having chosen a threshold value of 0.5.

In this sub-section we show for every compression algorithm the curves showing the progress of the accuracies of both models over training time in testing and training sets.

5.1.1 Empty cell compression.

As we can observe from figure 13 the NN models show promising results when trained with inputs pre-processed with the empty cell compression algorithm. This allows us to formulate two possible hypothesis regarding how NN are able to learn in classification problems where inputs show very sparse data such as the one at hand.

The spatial information derived by the position of the non-empty cells in the unprocessed data is not of great importance to the neural net, or, at least, is not so important as the number of non-empty cells, the values of pt, the electromagnetic fraction and the rest of information we have fed the NN with. Therefore the possible information loss derived from the compression algorithm does not have a negative effect on the NN performance.

The spatial information is indeed important but the NN is able to completely restore it from the two extra layers we added at the algorithm's first step.

A retraining of the NN models with different inputs[5] should be performed in order to determine the correct hypothesis.

In either case the surprisingly good results obtained with this compression algorithm suggest that a convolutional NN (used for preservation of spatial information) is not really needed for this kind of classification problems. Instead, it could be attempted a NN model that combined a compression algorithm compressing all non-empty cells into a relatively small 1-dimensional tensor and a series of dense layers that would process this tensor before returning a probability distribution.

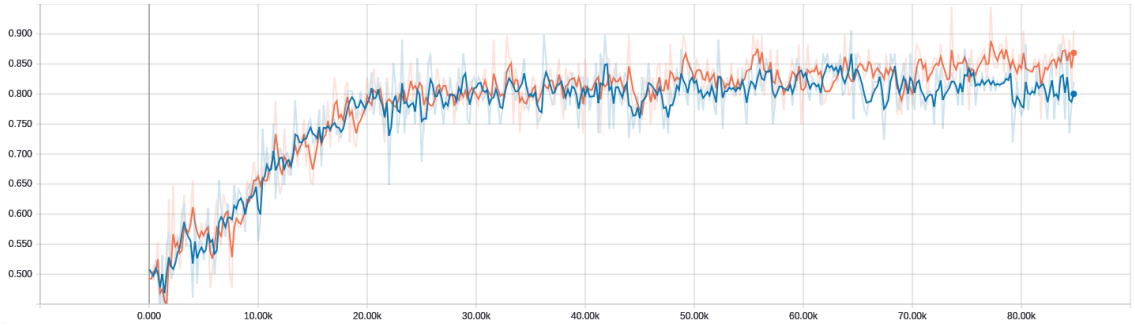
However these experiments escape the purpose of this document and is therefore left as future work.

5.1.2 Empty row and column compression.

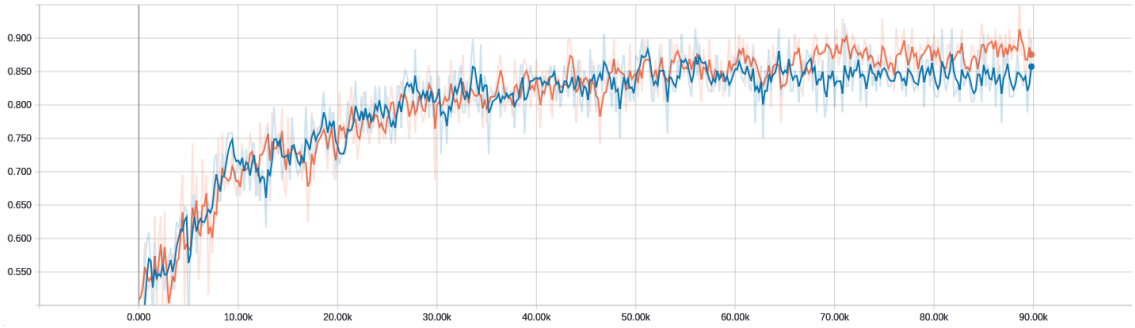
As we can observe from figure 14 the NN models also show promising results when trained with data pre-processed with the empty row and column compression algorithm.

This allows us to deduce once again that, either the spatial information does not take a great role in the NN training process, or that the NN is able to reverse the compression algorithm.

[5] If, after re-train the NN without the last two layers it is still able to perform well then the first hypothesis would be more likely to be correct. Otherwise we should accept the second hypothesis.

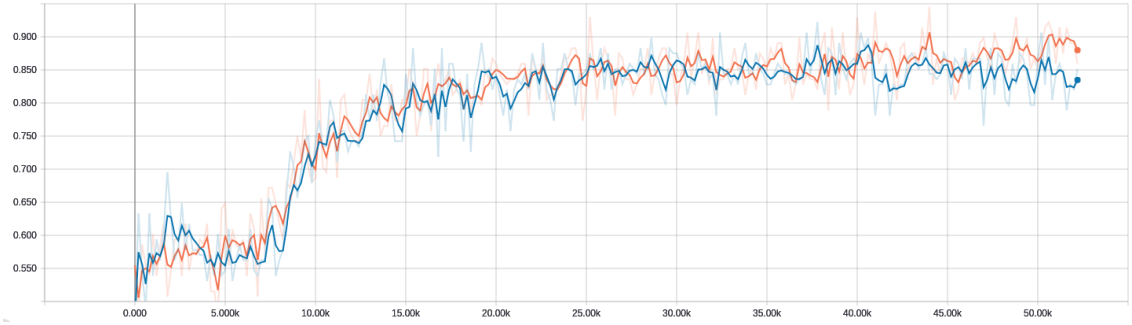


(a) max-pool model

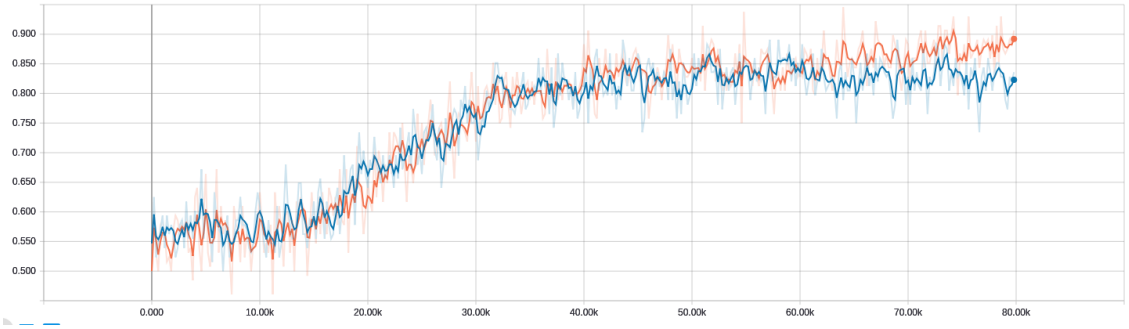


(b) strides model

Figure 13: Training results of applying both models to inputs pre-processed with the *empty cell compression* algorithm. We can observe in the images above the accuracy in training set (orange) and in testing set (blue). As we can observe the strides model is able to obtain an accuracy in testing set approximately 5% better than that obtained by the max-pool model.

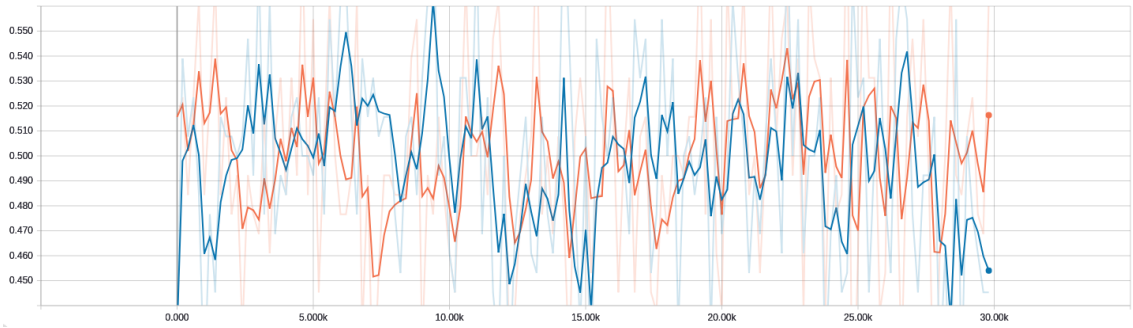


(a) max-pool model

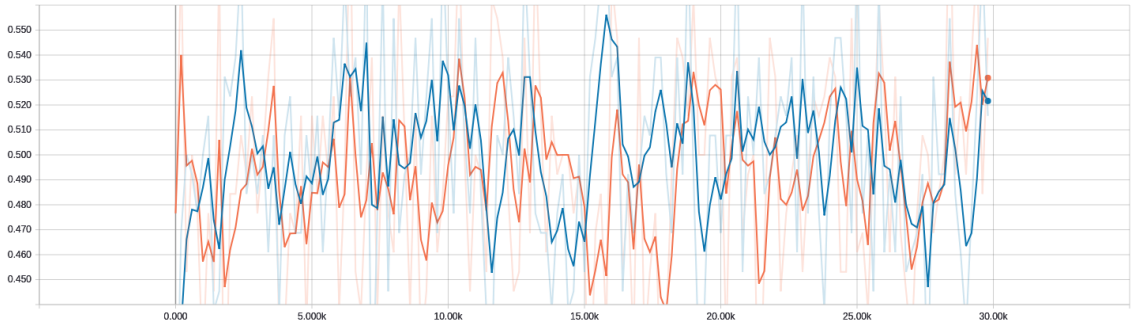


(b) strides model

Figure 14: Training results of applying both models to inputs pre-processed with *empty row and column compression* algorithm. We can observe in the images above the accuracy in training set (orange) and in testing set (blue). As we can observe the max-pool model is able to reach the same accuracy than the strides model with less training steps and with not so pronounced over-fitting signs.



(a) max-pool model



(b) strides model

Figure 15: Training results of applying both models to results from *empty row and column stochastic compression* algorithm. We can observe in the images above the accuracy in training set (orange) and in testing set (blue).

Another interesting result that we can observe from figure 14 is that the max-pool model seems to perform slightly better than the strides model while we observed the opposite behavior when the training was performed with inputs compressed with the empty cell compression algorithm.

A possible explanation for this behavior is that, because of the definition of the max-pool layer, this layer tends to attribute importance only to cells containing the highest values while ignoring neighbor cells. If non-empty cells are extremely close (empty cell compression) this could result in information lost while if they are more further apart (empty row and column compression) this could help solve the data sparsity problem.

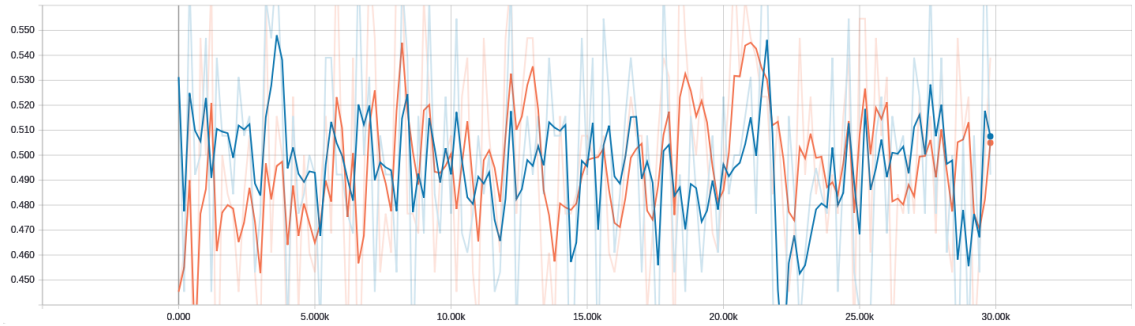
5.1.3 Empty row and column stochastic compression.

In figure 15 we can observe the accuracy curve of the NN models when trained with data pre-processed with the empty row and column stochastic compression algorithm. The poor performance of the NN models could be caused, as we suspected in section 4.2, by either the sparsity of data or because of the stochastic component of the compression algorithm.

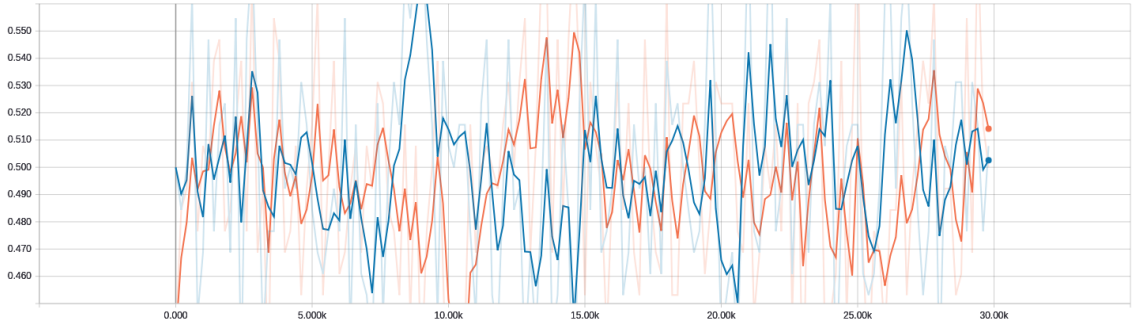
5.1.4 Uncompressed.

In figure 16, we can observe how the NN models trained on uncompressed data perform very poorly.

This is probably caused by the already mentioned sparsity of the data or by the different shape (greater size) of the input tensor which is too big to allow the convolutional layers to actually "see" all the data.



(a) max-pool model



(b) strides model

Figure 16: Training results of applying both models to uncompressed data. We can observe in the images above the accuracy in training set (orange) and in testing set (blue).

5.2 Model evaluation.

Given that the two best performing models are the combination of empty cell compression algorithm and strides model (cell + strides) and the combination of empty row and column compression algorithm and max-pool model (row-col + max-pool), we evaluate both this models on the validation sets described in sub-section 4.1. More specifically we evaluate those models as they were at iterations 48k (row-col + max-pool) and 60k (cell + strides) when they reach their peak accuracy in the testing set.

First, we find the optimal threshold for which the NN models are better able to classify between signal and background events. The models precision, recall, accuracy and mean average precision (precision·recall) is thus evaluated on the testing set at 21 different threshold values uniformly distributed in the interval [0, 1].

Results of this process can be observed in table 2 and figure 17.

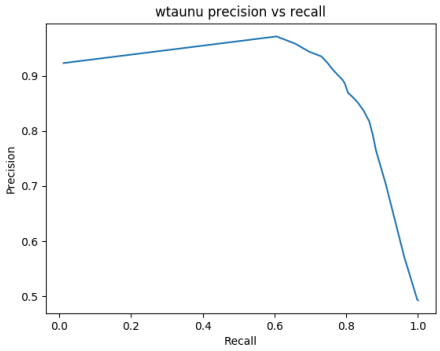
Using the obtained result we can deduce that the optimal threshold values (were peak mean average precision is reached) are 0.25 for row-col + max-pool model and 0.4 for cell + strides model.

With these threshold values we evaluate the model on all described P_t slices and events. This is done by calculating for every P_t slice the fraction of events correctly classified as being signal or background events, in the case of signal and background events. In the case of $W(\rightarrow \mu\nu)$ + jets events instead, the fraction of events classified as non-signal events was computed.

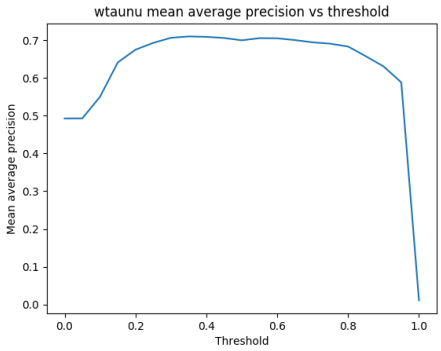
The results of the described analysis are reported for both models in table 3.

The high performance of the neural net in classifying between signal and $W(\rightarrow \mu\nu)$ + jets events indicates that the neural net has learned to distinguish between signal and background events based on the presence of other particles besides those of the jets. Because of this it is very likely that the trained models would be able to distinguish between WIMPs+jets events (that like signal events show only jets) and $W(\rightarrow \tau\nu)$ + jets and $W(\rightarrow \mu\nu)$ + jets background events.

Finally the only slightly decreasing performance of the NN on lower P_t slices indicates that the NN models are resistant to P_t variations and, therefore, a unique NN model could be used for all P_t ranges.

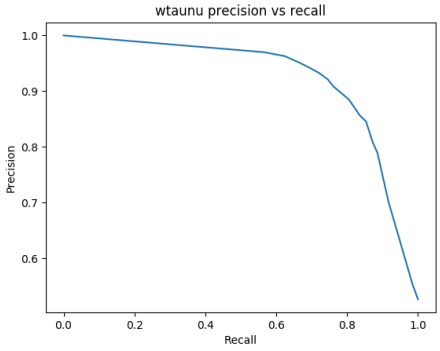


(b) Precision vs recall

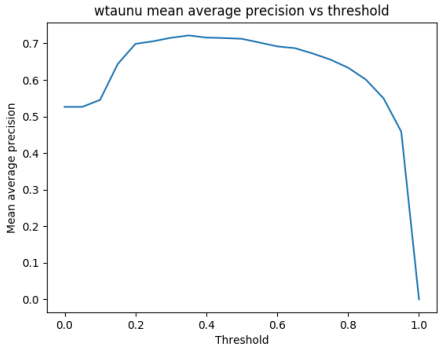


(c) Accuracy vs threshold

(c) row-col + max-poolmodel



(e) Precision vs recall



(f) Accuracy vs threshold

(f) cell + strides model

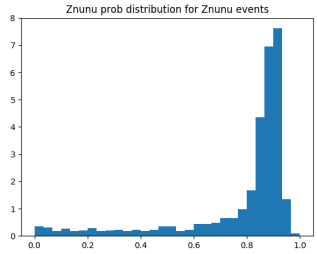
Figure 17: The above plots show the precision vs recall curve built while variating threshold and the accuracy vs threshold curve for both models. The seemingly strange behavior of the mean average precision curve for small threshold values is, in fact, completely normal. The NN usually outputs probabilities above this small threshold values and, therefore, for these values almost all of the events are predicted to be signal events. Therefore the NN predicts correctly all signal events to be signal events (recall$\simeq$ 1) but since signal and background events are found in a 1:1 ratio and the NN is predicting every event as being a signal only half of its predicted signals are indeed signals (precision$\simeq$ 0.5). This results in a mean average precision of approximately 0.5 for small threshold values as can be observed in the plots above. We can also use what we have just explained to notice that the NN models begin to be discriminative above threshold of 0.1.

row-col + max-pool				cell + strides				
precision	recall	accuracy	prec · rec	precision	recall	accuracy	prec · rec	threshold
0.508	1.000	0.754	0.508	0.481	1.000	0.741	0.481	0.00
0.718	0.984	0.851	0.706	0.636	0.992	0.814	0.631	0.05
0.745	0.975	0.860	0.727	0.673	0.983	0.828	0.661	0.10
0.762	0.964	0.863	0.735	0.705	0.975	0.840	0.687	0.15
0.778	0.953	0.866	0.741	0.726	0.966	0.846	0.702	0.20
0.789	**0.942**	**0.866**	**0.743**	0.743	0.958	0.851	0.712	**0.25**
0.796	0.930	0.863	0.740	0.755	0.944	0.850	0.713	0.30
0.804	0.919	0.862	0.739	0.768	0.931	0.850	0.715	0.35
0.811	0.910	0.861	0.738	**0.780**	**0.919**	**0.850**	**0.717**	**0.40**
0.816	0.899	0.858	0.734	0.790	0.899	0.845	0.710	0.45
0.818	0.882	0.850	0.722	0.806	0.887	0.847	0.715	0.50
0.826	0.866	0.846	0.716	0.714	0.866	0.790	0.705	0.55
0.833	0.850	0.842	0.708	0.826	0.846	0.836	0.698	0.60
0.846	0.829	0.838	0.701	0.834	0.826	0.830	0.689	0.65
0.853	0.805	0.829	0.687	0.841	0.806	0.824	0.678	0.70
0.858	0.774	0.816	0.665	0.845	0.779	0.812	0.658	0.75
0.861	0.724	0.793	0.623	0.850	0.741	0.796	0.630	0.80
0.871	0.624	0.748	0.544	0.866	0.569	0.718	0.492	0.85
0.877	0.296	0.587	0.260	0.881	0.120	0.501	0.105	0.90
0.778	0.007	0.393	0.005	1.000	0.000	0.5	0.000	0.95
1.000	0.000	0.5	0.000	1.000	0.000	0.5	0.000	1.00

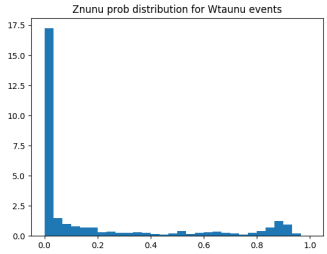
Table 2: Precision, recall and mean average precision variating threshold on validation set for both row-col + max-pool and cell + strides models. The results are computed for the validation set in high P_t slice.

row-col + max-pool			
P_t slices	$Z\,(\to \nu\overline{\nu}) + \text{jets}$	$W\,(\to \tau\nu) + \text{jets}$	$W\,(\to \mu\nu) + \text{jets}$
high	0.951	0.760	0.823
medium	0.948	0.697	0.781
low	0.942	0.679	0.763
cell + strides			
P_t slices	$Z\,(\to \nu\overline{\nu}) + \text{jets}$	$W\,(\to \tau\nu) + \text{jets}$	$W\,(\to \mu\nu) + \text{jets}$
high	0.914	0.779	0.833
medium	0.923	0.730	0.793
low	0.940	0.702	0.782

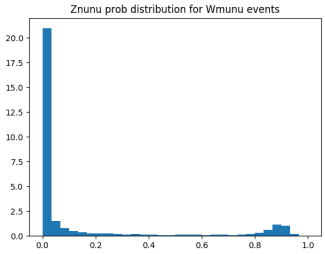
Table 3: The table shows predictions given by both NN models for all different P_t slices. In the $Z\,(\to \nu\overline{\nu})$+jets columns are shown the portion of correctly classified $Z\,(\to \nu\overline{\nu}) + \text{jets}$ events. In the $W\,(\to \tau\nu) + \text{jets}$ columns are shown the portion of correctly classified $W\,(\to \tau\nu) + \text{jets}$ events. In the $W\,(\to \mu\nu) + \text{jets}$ columns are shown the portion of $W\,(\to \mu\nu) + \text{jets}$ events correctly classified as not being $Z\,(\to \nu\overline{\nu}) + \text{jets}$ events.



(b) Signal probability distribution over signal events.

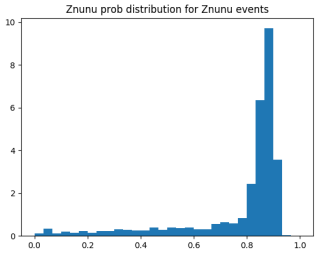


(c) Signal probability distribution over background events.

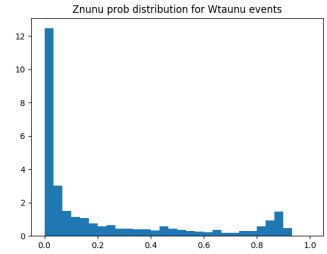


(d) Signal probability distribution over $W(\to \mu\nu) + \text{jets}$ events.

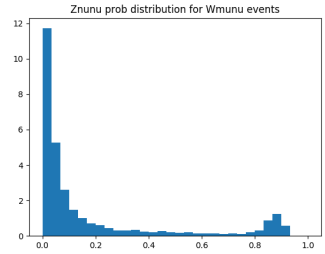
(d) row-col + max-poolmodel



(f) Signal probability distribution over signal events.



(g) Signal probability distribution over background events.



(h) Signal probability distribution over $W(\to \mu\nu) + \text{jets}$ events.

(h) cell + strides model

Figure 18: Predicted probability distributions according to both models for signal, background and $W(\to \mu\nu) + \text{jets}$ events in the test set of the high P_t slice. Notice the small peak on background and $W(\to \mu\nu) + \text{jets}$ events shown above a probability threshold of 0.8. This peak concentrate most of the incorrectly classified events.

	Correctly predicted %	
event type	row-col + max-pool	cell + strides
$Z\,(\to \nu\overline{\nu}) + \text{jets}$	96.3	96.4
$W\,(\to \tau\nu) + \text{jets}$	73.0	77.7
$W\,(\to \mu\nu) + \text{jets}$	86.2	90.4

Table 4: The table shows predictions given by both NN models for a joined set of P_t slices to which the same event selection criteria of [1] was applied.

background event	number of events
$W(\to e\nu) + \text{jets}$	20600
$W(\to \mu\nu) + \text{jets}$	20860
$W(\to \tau\nu) + \text{jets}$	50300
$Z/\gamma*(\to e^+e^-) + \text{jets}$	0.11
$Z/\gamma*(\to \mu^+\mu^-) + \text{jets}$	564
$Z/\gamma*(\to \tau^+\tau^-) + \text{jets}$	812
$Z(\to \nu\overline{\nu}) + \text{jets}$	137800
$t\overline{t}$, single top	8600
Diboson	5230
Multijet	700

Table 5: Number of events with missing transversal momentum (E_T^{miss}) greater than 250 GeV for each one of the background events after applying the selection criteria described in [1]. Table is a modification of a table in [1]. Uncertainties are not considered and only results for $E_T^{miss} > 250\,\text{GeV}$ are reported.

5.3 Performance on ATLAS results.

We present the improvement in sensitivity of WIMPs production events that the trained models would suppose if applied to input data as the one specified in [1]. The sensitivity, assuming that statistical uncertainty dominates in the background determination, can be defined as $S = \frac{\#S}{\sqrt{\#B}}$ where $\#S$ indicates the number of WIMPs production events and $\#B$ indicates the sum of all background events, listed in section 2. It is assumed that the NN models are able to select WIMPs production events with the same efficiency with which they select $Z(\to \nu\overline{\nu}) + \text{jets}$ events while all background events other than $Z(\to \nu\overline{\nu}) + \text{jets}$, $W(\to \tau\nu) + \text{jets}$ and $W(\to \mu\nu) + \text{jets}$ are not affected by the NN decision.

Models are reevaluated on a validation set mimicking the events distributions of the data presented in [1]. This validation set is obtained by combining all validation sets described in 4.1, respecting the cross section of events of each set, and applying to the resulting set the same selection criteria used in [1]. That is we select $Z\,(\to \nu\overline{\nu}) + \text{jets}$, $W\,(\to \tau\nu) + \text{jets}$ and $W\,(\to \mu\nu) + \text{jets}$ events with $E_T^{miss} > 250\,\text{GeV}$, a leading jet with $P_{T,j1} > 250\,\text{GeV}$ and $|\eta| < 2.4$ and a maximum of four jets with $P_T > 30\,\text{GeV}$ and $|\eta| < 2.8$. A separation in the azimuthal angle between the missing transverse momentum direction and each one of the last jets is also required to be greater than 0.4 in absolute value. Results of the reevaluations can be observed in table 4. After event selection, the cell+strides model obtains a clearly better performance than the row-col+max-pool model. Thus the predicted increased sensitivity is shown only for the cell + strides model.

Using the reevaluated efficiencies and statistics from [1] data shown in table 5 we calculated sensitivity increase (sensitivity after NN/sensitivity before NN) to be of 1.118. Which indicates that the inclusion of the NN would improve, by about 10%, the performance of the analysis for an inclusive selection with $E_T^{miss} > 250\,\text{GeV}$. It is reasonable to expect, given the better results for high P_t, that results would improve in inclusive sections for larger E_T^{miss}.

6 Conclusions.

Two different neural net models are trained to classify between $Z\,(\rightarrow \nu\overline{\nu}) + \text{jets}$ events and $W\,(\rightarrow \mu\nu) + \text{jets}$ events.

Training was performed over 140K events simulated considering detector with particle level information.

Models seem to be robust to variations of the boson P_t of input events which indicates that a single neural net could be used for all P_t range.

Results suggest that trained models could be used for separating WIMPs productions events and $W\,(\rightarrow \tau\nu) +$ jets and $W\,(\rightarrow \mu\nu) + \text{jets}$ background even if the later was not used during training.

Application of best performing model over data described in [1] results in an increase of sensitivity to WIMPs production events of 1.118. Which indicates that the inclusion of the NN would improve, by about 10%, the performance of the analysis for an inclusive selection with $E_T^{miss} > 250\,\text{GeV}$.

7 Further work.

The surprisingly good results obtained when training the NN models with inputs pre-processed using the empty cell compression algorithm suggest that results could be improved if compression algorithms were modified to output a 1-dimensional tensor of small size and a NN architecture having only dense layers was used for training. The same models used in this document may also show some sort of improvement if trained with sliced tensors of shape $30 \times 30 \times 10$ instead of $60 \times 60 \times 10$ since unneeded information would be removed.

Other deeper NN models more similar to VGG16 y ResNet50 models (which have shown very good results in image classification problems) should be studied when more computational power is available.

A better model efficiency when identifying $Z(\rightarrow \nu\overline{\nu})+$jets against $W\,(\rightarrow \mu\nu)+$jets events (unused for training) than against $W\,(\rightarrow \tau\nu) + \text{jets}$ events (used for training) suggest that the same NN architecture could be used to differentiate between this two $W + \text{jets}$ events. Moreover identification of $Z(\rightarrow \nu\overline{\nu}) + \text{jets}$ against multiple background events could result in an overall improvement of the NN models and should therefore be studied.

Studies of sensitivity improvement for an inclusive section of $E_T^{miss} > 1\,\text{TeV}$ should be performed.

Training of the NN models should be performed using as inputs events simulated under non ideal detectors.

A Pre-processing algorithms pseudo-code.

In this appendix is presented pseudo-code for each of the three implemented compression algorithms. The naming conventions for tensors obtained from step 1 described in 4.2 is adopted.

Empty cell compression

1. Initialize the variables $i = 1$ and $i' = 1$.
2. Initialize the variables $j = 1$ and $j' = 1$.
3. Check if cells (i, j, k), with $k = 1, \ldots, 8$, of the tensor T_{init} are empty (i.e. contain only the value 0).
4. If at lest one of the cells is not empty then copy the content of the cells (i, j, k) of tensor T_{init} to the cells (i', j', k) of tensor T for $k = 1, \ldots, 10$ and add 1 to the variable j'.
5. Add 1 to the variable j.
6. If $j <= 126$ continue from step 2.
7. If $j' > 1$ add 1 to the variable i'.
8. Add 1 to the variable i.
9. If $i <= 100$ continue from step 2.
10. return tensor T.

Empty row and column compression.

1. Initialize the variables $i = 1$ and $i' = 1$.
2. Check if cells (i, j, k), with $j = 1, \ldots, 126$ and $k = 1, \ldots, 8$, of the tensor T_{init} are empty.
3. If at lest one of the cells is not empty then copy the content of the cells (i, j, k) of tensor T_{init} to the cells (i', j, k) of tensor T for $j = 1, \ldots, 126$ and $k = 1, \ldots, 10$ and add 1 to the variable i'.
4. Add 1 to the variable i.
5. If $i <= 100$ continue from step 2.
6. Create a tensor T' of shape $100 \times 126 \times 10$ and fill it with zeros.
7. Initialize the variables $j = 1$ and $j' = 1$.
8. Check if cells (i, j, k), with $i = 1, \ldots, 100$ and $k = 1, \ldots, 8$, of tensor T_{init} are empty.
9. If at lest one of the cells is not empty then copy the content of cells (i, j, k) of tensor T_{init} to the cells (i, j', k) of tensor T for $i = 1, \ldots, 100$ and $k = 1, \ldots, 10$ and add 1 to the variable j'.
10. Add 1 to the variable j.
11. If $j <= 126$ continue from step 8
12. Return tensor T'.

Empty row and column stochastic compression.

1. Create a set L containing all values of $i \in \{1, \ldots, 100\}$ such that exists a couple of values of $j \in \{1, \ldots, 126\}$ and $k \in \{1, \ldots, 8\}$ such that the cell (i, j, k) of tensor T_{init} contains a non zero value.
2. If L contains less than different 60 values chose at random $i \in \{1, \ldots, 100\}$ such that $i \notin L$ and add it to the set L.
3. Repeat step 2 until the set L contains 60 different values.
4. Sort the elements in set L and initialize $i' = 1$.
5. Select i the first element of the ordered set L and copy the content of cells (i, j, k) tensor T_{init} to the cells (i', j, k) of tensor T for $j = 1, \ldots, 126$ and $k = 1, \ldots, 10$.
6. Add 1 to the variable i' and remove the element i from the ordered set L.
7. If the set L is not empty continue from step 5.
8. Create a tensor T' of shape $100 \times 126 \times 10$ and fill it with zeros.
9. Create a set L' containing all values of $j \in \{1, \ldots, 126\}$ such that exists a couple of values of $i \in \{1, \ldots, 100\}$ and $k \in \{1, \ldots, 8\}$ such that the cell (i, j, k) of tensor T contains a non zero value.
10. If L' contains less than different 60 values chose at random $j \in \{1, \ldots, 126\}$ such that $j \notin L'$ and add it to the set L'.
11. Repeat step 10 until the set L' contains 60 different values.
12. Sort the elements in set L' and initialize $j' = 1$.
13. Select j the first element of the ordered set L' and copy the content of cells (i, j, k) of tensor T to the cells (i, j', k) of tensor T' for $i = 1, \ldots, 100$ and $k = 1, \ldots, 10$.
14. Add 1 to the variable j' and remove the element j from the ordered set L'.
15. If the set L' is not empty continue from step 13.
16. Return tensor T'.

References

[1] The ATLAS Collaboration. Search for dark matter and other new phenomena in events with an energetic jet and large missing transverse momentum using the atlas detector. *JHEP 01 (2018) 126*, February 2018. arXiv:1711.03301.

[2] Sergio A. Cellone. Materia oscura: un responsable en hechos de gravedad?

[3] Gary Steigman and Michael S.Turner. *Cosmological constraints on the properties of weakly interacting massive particles*, volume 253. Nuclear Physics B, 1985.

[4] Edward W. Kolb and Michael S.Turner. *The Early Universe*. ADDISON-WESLEY, 1990.

[5] G. Hinshaw, D. Larson, E. Komatsu, D. N. Spergel, C. L. Bennett, J. Dunkley, M. R. Nolta, M. Halpern, R. S. Hill, N. Odegard, L. Page, K. M. Smith, J. L. Weiland, B. Gold, N. Jarosik, A. Kogut, M. Limon, S. S. Meyer, G. S. Tucker, E. Wollack, and E. L. Wright. Nine-year wilkinson microwave anisotropy probe (wmap) observations: Cosmological parameter results. *arXiv:1212.5226*, 2012.

[6] Plank Collaboration. Planck 2015 results. i. overview of products and scientific results. *arXiv:1502.01582*, 2015.

[7] CMS Collaboration. Search for new physics with a mono-jet and missing transverse energy in pp collisions at sqrt(s)=7 tev. *Phys. Rev. Lett. 107, 201804 (2011)*, Juny 2011. arXiv:1106.4775.

[8] *Search for New Physics woth Monojet plus missing transverse energy at CMS*. Sarah Alam Malik, October 2011. arXiv:1110.1609.

[9] lodzislaw Duch, Janusz Kacprzyk, Erkki Oja, and Slawomir Zadrozny. *Artificial Neural Networks: Biological Inspirations - ICANN 2005*. Springer, 2005.

[10] *A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning*, 2008.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *arXiv:1609.00607*, 2012.

[12] Sander Dieleman, Kyle W. Willett, and Joni Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *arXiv:1503.07077*, 2015.

[13] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *arXiv:1402.4735*, 2014.

[14] Kanchan Sarkar. Relu : Not a differentiable function: Why used in gradient based optimization? and other generalizations of relu., May 2018.

[15] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 1989.

[16] skymind. Convolutional neural network (cnn).

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556v6*, April 2015.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv:1512.03385v1*, December 2015.

[19] Patrick T. Komiske, Eric M. Metodiev, and Matthew D. Schwartz. Deep learning in color: towards automated quark/gluon jet discrimination. *arXiv:1612.01551v2*, February 2017.

[20] Eric Kauderer-Abrams. Quantifying translation-invariance in convolutional neural networks. *arXiv:1801.01450v1*, December 2017.

[21] John E. Shore and Rodney W. Johnson. Properties of cross-entropy minimization. *IEEE*, 1981.

[22] James McCaffrey. Test run - l1 and l2 regularization for machine learning. 2015.

[23] Nathan Lintz. Exploring computer vision (part i): Convolutional neural networks, February 2016.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[25] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[26] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*, July 2012.

[27] Wahid Bhimji, Steven Andrew Farell, Thorsten Kurth, Michela Paganini, Prabhat, and Evan Racah. Deep neural networks for physics analysis on low-level whole-detector data at the lhc. *arXiv:1711.03573v2*, November 2017.

[28] The ATLAS Collaboration. Quark versus gluon jet tagging using jet iimage with the atlas detector. July 2017.

[29] Image data pre-processing for neural networks, September 2017.

[30] Why are deep neural networks so bad with sparse data?, 2017.