JOINT INSTITUTE FOR NUCLEAR RESEARCH
Dzelepov Laboratory of Nuclear Problems

FINAL REPORT ON THE SUMMER STUDENT PROGRAM

Investigation of Deep Learning methods for the classification of events in the NOvA experiment

Supervisor:
Chris Kullenberg
Oleg Samoylov

Student:
Denis Uzhva, Russia
Saint Petersburg State University

Participation period:
July 01 – August 11

Dubna, 2018

Contents

Abstract

With the rise of questions concerning the problems of the theory behind the phenomenon of neutrino oscillations, more and more neutrino experiments are under development and underway. One such experiment, NOvA, is supposed to tell the scientific comunity more about $\nu_\mu \to \nu_e$ (as well as $\bar{\nu}_\mu \to \bar{\nu}_e$) oscillations, ν_μ and $\bar{\nu}_\mu$ disappearance channels, to determine the order of neutrino masses, CP-violation phase in the lepton sector, and to measure precisely mixing angles, as well as Δm^2 (squared mass differences). A brand new approach of using convolutional neural networks has been applied in order to improve the quality of NOvA's data analysis, bypassing the standard event reconstruction procedures. The idea of combining the reconstruction with Convolutional Neural Networks (CNN) is interesting, as it may help with increasing the degree of precision and perfomance speed. Since the data is generally composed of particle tracks, emitted from an interaction point, we have attempted to train NOvA CNN on images that have had a polar transformation applied to each event, with the transformation origin located at the event's interaction vertex. In this way tracks emitted from the vertex will be represented by a horisontal line, perhaps presenting an easier data set for the network to learn from. We have shown that such a dataset decreases the overtraining of the network, though, slightly reducing the validation accuracy. Therefore, the future development of this method may be fruitful. Moreover, after removing parallel inception layers, a simplified version of the original NOvA network has been shown to be much faster with little cost in accuracy. Therefore, simpler networks should also be investigated further.

1 Introduction

1.1 The history of the neutrino

The premises for the introduction of neutrinos arised from the problem of beta decay, which was the focus of such great physicists of the XX century as Wolfgang Pauli and Niels Bohr. Among the three types of radioactivity: gamma, alpha and beta, the latter produced the most mysteries because of its nature; being based only on the weak interaction [1].

During the investigations around the phenomenon of beta decay Lise Meitner, Otto Hahn, Wilson and von Baeyer, and James Chadwick showed in 1914 that, contrary to expectations, the particles of beta radiation emitted by ^{210}Bi have a continuous spectrum. This observation seemed to contradict one of the fundamental postulates in physics – the law of conservation of energy, which is the basis of all the theories about our universe: from classical to quantum mechanics, electromagnetics, relativistic theories – all of them submit this absolute rule of invariability of energy [2]. By the way, not only did the conservation of energy seem to be broken: the conservation of momentum appeared to not be satisfied also. The results were so surprising and extraordinary that even N. Bohr dared to propose rejecting or modifying the fundamental laws mentioned above [3].

However, Bohr's bold idea, fortunately or not, did not eventually acquire as much preferences of the scientific community as the hypothesis of Pauli, which postulated a new particle called a "neutron". It is necessary to mention that Pauli's "neutron" and the "actual" neutron, a part of atomic nuclei, are not the same particle. The thing is, the "actual" neutron was proposed a little later than the neutrino – and the name "neutrino" was given afterwards to Pauli's "neutron" by adding the "-ino" ending in order to emphasize its smallness.

The one who named the new particle as "neutrino" was Enrico Fermi himself [2]. In 1933 he developed the first theory of beta decay based on second quantization, similar to that used earlier to describe the emission and absorbtion of photons. Fermi treated beta decay as a transition, that was dependant upon the strength of coupling between the initial and final states of a particle system, and the main result of his investigation is a relationship, which is now referred to as Fermi's Golden Rule [4]:

$$\lambda_{ij} = \frac{2\pi}{\hbar}|M_{ij}|^2\rho_j,$$

where i and j are the initial and the final states respectively, λ_{ij} is the transition probability, M_{ij} is the matrix element for the interaction, and ρ_j is the density of final states. In fact, the Golden Rule says that the transition rate is proportional to the strength of the coupling between the initial and final states, factored by the density of final states available to the system. Despite the success of Fermi's theory, the nature of the interaction which led to beta decay would remain unknown for about 20 years until a more advanced theory was developed.

As it was mentioned previously, beta decay occurs due to the weak interaction. This fact leads to the idea that the neutrino is the most abundant particle in the entire universe, since it interacts so rarely with matter and is produced in such large amounts in solar nuclear interactions. By the way, beta decay is the process which has made it possible for life to exist on on Earth: in the synthesis of four nucleons into one nucleus of helium, two electrons and two neutrinos inside the Sun release 27 MeV of energy, which allows the Earth to be warm enough to harbor life [5].

1.2 Neutrino oscillations

The neutrino is associated with many mysteries of modern physics. One of the most puzzling phenomenon is the existence of neutrino oscillations, or the ability of these particles to "transform" into each other by change of their lepton flavor (i.e. a neutrino created with a specific flavor can be measured as a neutrino with some different flavor).

The history of the investigation of neutrino oscillations originated with the "solar neutrino problem". In 1946 italian-soviet physicist Bruno Pontecorvo proposed a method for detecting neutrinos using chlorine atoms [6]. Although this method of experimental investigation had been developed in the 40s, the first detector based on Pontecorvo's principle was constructed only in 1966 by Raymond Davis [2]. Having been situated at the Homestake mine is South Dakota, this system was supposed to detect solar neutrinos, as the Sun is the best local natural source of these particles (several billions of them penetrate through each squared centimeter of Earth per second – here we can imagine how rarely they interact with matter as we cannot feel how they affect us). After all the required measurements were perfomed, Davis compared the experimental data of interaction rate with theoretical expectations. The result, however, was surprising: the experimental rate was 2.5 times less than the one suggested by the theory – such a discrepancy shocked the scientific community, which strongly believed in the success of their predictions.

According to the Standard Model, there are three flavors of the neutrinos: electron (ν_e), muon (ν_μ) and tau (ν_τ) neutrinos. In 1957 Pontecorvo suggested the idea of neutrino oscillations [7] – before any other neutrino particles, besides ν_e, were discovered. Although his first mode of thinking was directed towards neutrino-antineutrino oscillations, just like it appears to occur for K-mesons, he soon changed his mind in favor of the oscillations of flavor (the same year as Sakata, Nakagawa and Maki came to the same hypotesis) [2] – the idea today we still believe in. The hypothesis (afterwards a theory) of neutrino oscillations, by the way, was proposed when, besides ν_e, no other neutrinos were discovered yet, which makes this theory a quite strong one by the fact it predicts the solar neutrino puzzle.

The idea of neutrino oscillations allowed scientists to explain the solar neutrino problem described above: the remaining 2/3 of all the sun's neutrinos simply were not registered as they were muon and tau neutrinos. Moreover, the oscillation hypothesis gives neutrinos nonzero mass – and this contradicted to the Standard Model at the time, according to which those particles were massless.

A special 3x3 unitary matrix was named after the mentioned three japanese scientists and Pontecorvo (Pontecorvo-Maki-Nakagawa-Sakata or just PMNS matrix) – this matrix can be often found in papers about the subject – it contains information on the mismatch of quantum states of neutrinos when they either propagate freely or take part in weak interactions (sometimes this matrix is refered to as "neutrino mixing matrix") [8]. The PMNS matrix, in fact, connects masses and flavors of three active massive neutrinos, and is usually parameterized by so-called "mixing angles" θ_{12}, θ_{23}, θ_{13}, a CP-violating phase parameter δ_{CP} (it is worth remembering that in terms of weak interaction there is no CP-symetry), and additional parameters representing the squared-mass differences $\Delta m^2_{ji} = m^2_j - m^2_i$, where m_i is the mass of i-th eigenstate of neutrino mass. The models utilizing Pontecorvo-Maki-Nakagawa-Sakata paradigm are successfully confirmed by the experimental progress; furthermore, at present, nearly all the mixing parameters have been measured. Nevertheless, there are still unknowns remaining: mass hierarchy (figure 1 shows some of the possible situations), absolute mass, phase parameter, θ_{23} octant, existence of other neutrino types, and thus, there are many subjects of investigation. The neutrino masses and mixing angles are supposed to be fundamental constants, therefore measuring their values is a very important task for physics.

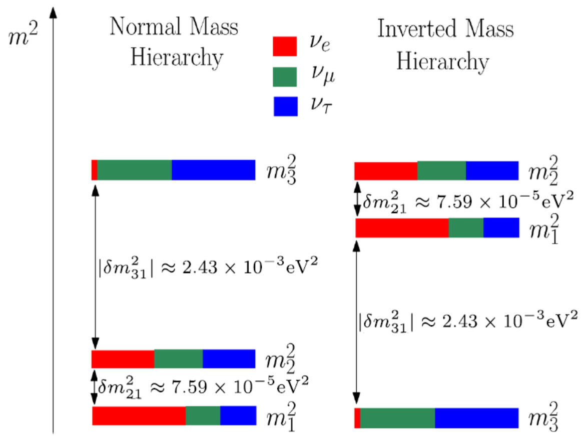

m^2
Normal Mass Hierarchy
Inverted Mass Hierarchy
ν_e
ν_μ
ν_τ
m_3^2
$|\delta m^2_{31}| \approx 2.43 \times 10^{-3}\text{eV}^2$
m_2^2
$\delta m^2_{21} \approx 7.59 \times 10^{-5}\text{eV}^2$
m_1^2
m_2^2
$\delta m^2_{21} \approx 7.59 \times 10^{-5}\text{eV}^2$
m_1^2
$|\delta m^2_{31}| \approx 2.43 \times 10^{-3}\text{eV}^2$
m_3^2

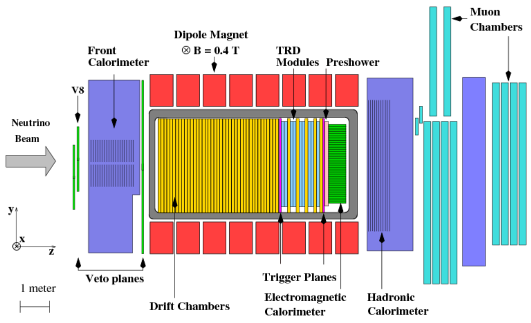
Figure 1: Neutrino mass hierarchy models

1.3 Experimental investigation of the neutrino

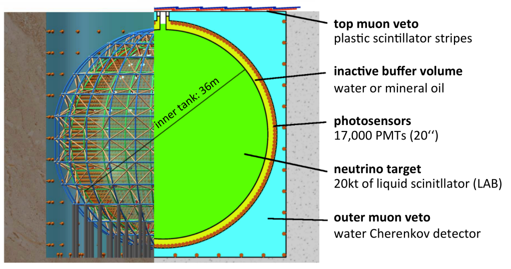
All the theories about the nature of the neutrino would have been nothing more than fiction without appropriate experimental verification. As described above, the first neutrino detector was constructed by Raymond Davis in 1966 according to Pontecorvo's idea about using chlorine atoms. The objectives of Davis' detector were relatively simple – simply count the rate of neutrino current from the Sun. However, the results of this experiment allowed the scientific community to accept the hypothesis about neutrino oscillations suggested by the italian physicist.

Further experiments are basically aimed at much more rigorous tests of the oscillation theory: the parameters of the PMNS matrix are of particular interest. More and more results confirm the original intuition of Bruno Pontecorvo: discoveries of atmospheric and solar neutrino oscillations by the Super-Kamioka Neutrino Detection Experiment (Super-Kamiokande) [9], Sudbury Neutrino Observatory (SNO) [10], Kamioka Liquid Scintillator Antineutrino Detector (KamLAND) [11] experiment between 1998 and 2002 have been recognized by the 2015 Nobel Prize in Physics awarded to Arthur Mc Donald and Takaaki Kajita.

However, not only naturally acquired neutrino fluxes are in use for experiments: there are several detectors exposed to artifically produced beams. One of those detectors is NOMAD [12], belonging to CERN, – for this experiment the ν_μ current from SPS ring was used; NOMAD obtained very precise data comparable with that gained out of bubble chambers, which gave an opportunity for accurate reconstruction. Another facility is JUNO – a spherical detector externally similar to KamLAND – this experiment uses a neutrino flux from Yangjiang and Taishan Nuclear Power Plants. Daya Bay, RENO and Double Chooz are also exposed to neutrinos produced by nuclear stations. Finally, the NOvA experiment exploits the NuMI beam (which is also used by MINOS, MINERvA, ICARUS, SBNE and DUNE experiments) produced by Fermilab – this experiment is described in the second section of this paper.



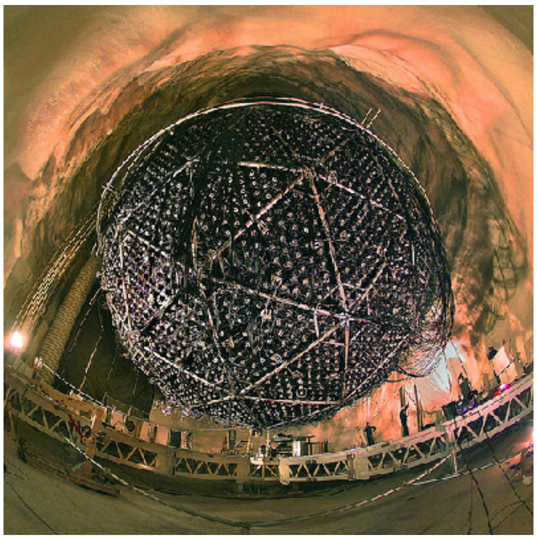
(a) Scheme of NOMAD experiment



(b) Scheme of JUNO experiment

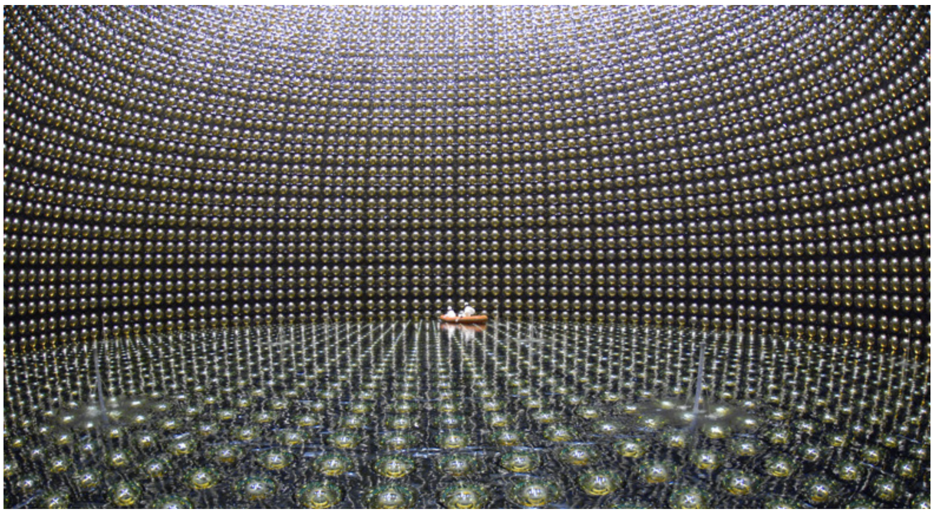
Figure 3: Detectors exposed to artifically produced beams

Although the main aim of modern neutrino physics is the investigation of neutrino oscillations, there exists a number of practical implementations of neutrinos. One of them deifnetly worth mentioning is neutrino astronomy. In this field of study physicists were highly pleased during the summer of 2018: the collaboration of the IceCube detector, localed in South Pole, have published results of observations of a high-energy neutrino event from an extragalactic object. This result shows desirable prospects for the use of neutrinos from distant astronomical objects for observations.



(a) Inside the Sudbury Neutrino Observatory

(b) Inside the Kamioka Liquid Scintillator Antineutrino Detector



(c) Inside the Super-Kamiokande (this room is usually filled with water)

Figure 2: Atmospheric and solar neutrino detectors

Another experiment worth mentioning is Baikal Deep Underwater Neutrino Telescope, one of the world's largest neutrino detectors. It was constructed to study high-energy muon and neutrino fluxes and search for new types of elernentary particles: magnetic monopoles, WIMPs (massive particles which can be considered as candidates to "dark" matter), and others.

2 NOvA experiment

2.1 Introduction to the experiment

The name NOvA stands for "NuMI Off-Axis ν_e Appearance". This experiment is designed in such a way that the detectors are penetrated by a directed artifically-created neutrino flux – the main benefits of the choice of using a human-made beam instead of cosmic radiation is its controllability and the fact that 'physicists have information about the kind of particles being produced as well as their direction and energy.

NuMI (Neutrinos at the Main Injector) presently provides the most powerful artifical neutrino current for many experiments [13]. Among them MINOS, MINERvA, ArgoNeuT, MINOS+, MiniBooNE and, of course, NOvA. In order to produce the beam of neutrinos the protons from Fermilab's Main Injector are used: they strike a target made of carbon, which causes the production of mesons, kaons and pions, primarily. Then, with the help of magnetic horns, the mesons are focused toward the beam axis, and eventually they decay into muons and muon neutrinos with corresponding antiparticles during a travel in a long (almost 700 meters) pipe. At the end of NuMI muons are stopped by layers of rock, which leaves us a nearly pure ν_μ flux aimed slightly downward at 3.3° in order not to let neutrinos fly into space.

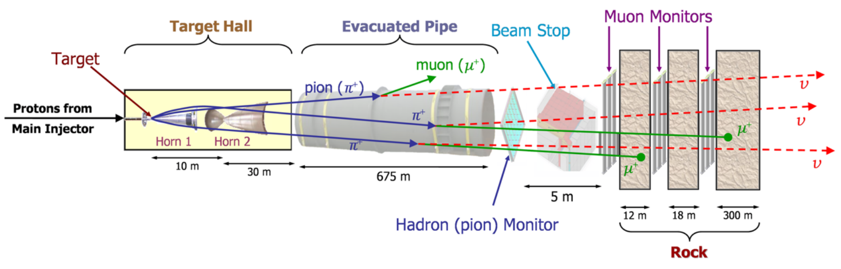


Figure 4: Scheme of Neutrinos at the Main Injector

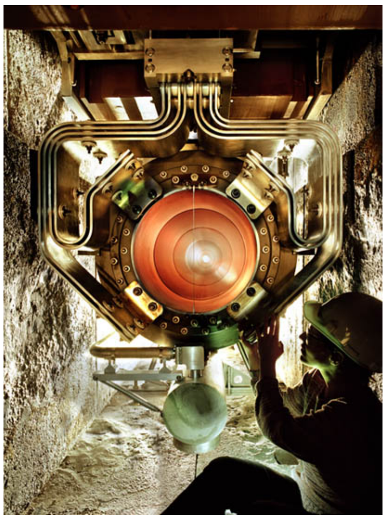


Figure 5: A photo of a NuMI horn

After the neutrino beam is created it travels 735 kilometers under the surface of Earth due to its curvature, where it meets MINOS' detector. However, the further the beam propagates, the wider it becomes – this fact allows us to use not only the stright moving neutrinos, but the ones deviating by small angles. NOvA has two detectors – "near" and "far" – both of them use slightly off-axis (by 14 milliradians off of the main beam line) part of the NuMI flux, and while the near detector is placed under the ground, the far detector is situated on the surface at the end of the beam, so that the distance between the start point of neutrinos and the far detector appears to be 810 km (see figure 6). As the energy of protons being injected is 120 GeV and as the NOvA detectors are exposed by the 14 mrad off-axis part of the beam, the energy of neutrinos registered by NOvA reach 2 GeV, which optimizes oscillation analysis in the first oscillation maximum.

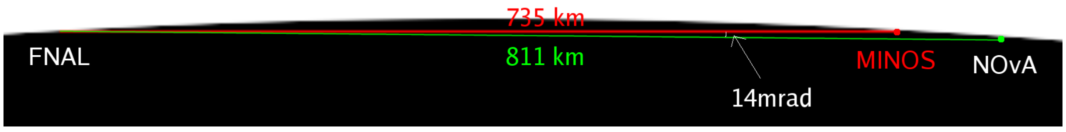


Figure 6: Propagation of NuMI beam towards MINOS and NOvA research objects

Each of the two detectors of the NOvA experiment are, in fact, plastic parallelepipeds filled with liquid scintillator. While the near detector is not very large and heavy – only 300 metric-ton – the far detector is huge: it is a 14 metric-kiloton structure, the size of which is comparable with the size of an airbus – and, by the way, the far detector is the largest manmade free-standing plastic structure in the world. Both detectors are constructed using highly reflective plastic PVC cells. Charged particles born in a neutrino collision event inside a cell produce light. With the help of wavelengt-shifting fiber, the light then is collected by photo-detectors, thus the scientists of the NOvA collaboration can reconstruct events as pictures [14]. Besides the difference between the sizes of the detectors, there are some technical distinctions: being placed under the surface, the near detector is less exposed to cosmic background noise than the far detector; it is also worth mentioning that the relatively small size of the near detector costs the resolution of picture, representing the data, as this detector simply has fewer cells.

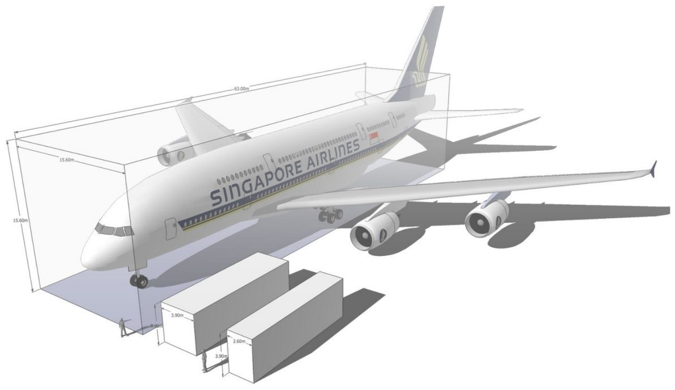


Figure 7: A comparison of the NOvA far and near detectors (and the NDOS test detector) with the size of an Airbus A380 and with an average human

2.2 The goals of NOvA

The NOvA experiment was organized primarily in order to investigate experimental evidence in favor of the theory behind neutrino oscillations. The measurements of the mixing angles of the PMNS matrix is of particular interest: non-zero value of θ_{13} allows NOvA to see ν_e oscillation events and determine precisely neutrino mass mixing parameters. Initially the NuMI beam consisted mainly of muon neutrino particles, therefore detection of ν_e with certain energy and direction implies $\nu_\mu \to \nu_e$ oscillation (the same is true for antiparticles also), and θ_{13} itself helps define the frequency for this kind of oscillation. By the way, comparing $\nu_\mu \to \nu_e$ with $\bar{\nu_\mu} \to \bar{\nu_e}$ is only sensitive to specify δ_{CP}, because it modifies the probabilities of oscillation in opposite ways for neutrinos and anti-neutrinos. In turn, measuring the δ_{CP} parameter will help us understand the nature of the matter-antimatter antisymmetry

Another useful result is expected during investigation of the mass hierarchy. Nowadays scientists are not sure if the ordering of neutrino masses follow the same pattern as the other particles: generally, with an increase of generation a mass of a particle also increases.

Aside from the main tasks, there are several minor goals of the NOvA experiment. One of them is measuring neutrino interaction cross sections. The investigations of sterile neutrinos, supernova neutrinos, magnetic monopoles and non-standard neutrino interacions are also of interest to physicists.

3 Data analysis in NOvA

3.1 Data science and artifical neural networks

Machine learning, the most popular branch of computer science today, is spread in a great many areas of life. Aside from the purposes for consumers, there are plenty of ways of exploiting machine learning methods in fundamental sciences, and it has found its role in particle physics as well [15]. However, in order to explain how to apply the developments of computer science to physics, it is necessary to understand the basic primary concepts of machine learning.

The idea of machine learning is usually associated with artifical intelligence, because, in this paradigm, computers gain the ability to "learn" how to solve certain tasks rather than solving them with clear, precise programmed instructions. The learning itself is supposed to be similar to what we are used to understand as learning: a machine repeatedly guess an answer to a certain question, and if the answer is wrong, we tell the machine it is wrong – then the computer adjust its internal state in order to be more correct the next time we ask a question. It is supposed that there is some dependency between questions and answers, which is not known initially, but which is the goal of the computer to discover. The most strict definition of a machine that learns was provided by Tom Michael Mitchell, a computer scientist of the second half of the XX century, and now this definition is considered as a formal one: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E". There one can feel the difference between understanding the learning of a computer as a cognitive process and as an algorithmic operation, however, totally dependent on data and less intuitively understandable. This reasoning may lead one to thoughts about perception and consciousness of artifical intelligence, though computer scientists tend to distinguish biological methods of thinking and their computer models: in Alan Turing's paper "Computing, Machinery and Intelligence" the question "Can machines think?" is replaced with the question "Can machines do what we (as thinking entities) can do?" [16].

A biological brain works according to the following principle: a brain is composed of so-called

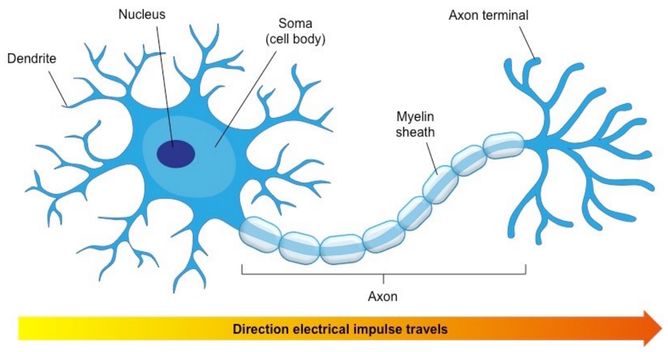


Figure 8: A model of a biological neuron. Signals are received by dendrites, "process" inside the cell body and then transport along the axon to the axon terminals, connected to other cells

neurons – they are cells, the purpose of wich is to receive an input signal and generate an output based on everything that was sent to the input. These cells are connected with each other in a certain way in order to receive and send signals from and to each other and other cells of an organism. For example, when we hear musical sounds, the vibration of air forces the tympanic membrane to vibrate also, then the vibrations, through the flexile system of tiny bones, are transmitted to the cochlea where the signal is being converted into neural impulses. Dozens of such impulses after the special treatment inside neurons eventually add up in our brains to what we perceive as music. Technically, a similar principle produces consciousness: the information from the world outside us as well as our own "inner voice", passing signals from the inside to the inside of the brain, allows us not only think, but observe, imagine, and create.

In order to provide computers the ability to learn, we generally use models of real biological brains – they are called artifical neural networks (ANN or just NN). The neurons of an ordinary ANN are usually some functions of several variables, called "activation functions", and the dendrites, called "weights", are numbers. It turns out that in a neural network the inputs of a function is either somehow combined outputs of another functions or the input data (like the perception cells in out eyes); in turn, this function produces an output that is used by other functions, or to check the response of the network to the data.

Usually neural networks are constructed with the layer architecture. Three types of layers can be identified: input (to where the data is fed), output (from where we receive the answers of certain network) and hidden (the layers between the previous two) layers. The input data is usually represented as a set of vectors or matrices, elements of which pass as arguments of the functions in the input layer. Let's suppose we have only one hidden layer, for simplicity. The outputs of the functions of the input layer are summed with some weight, and then fed into the functions of the hidden layer. It is important to notice, that for each function of the hidden layer a unique set of weights is supposed. The same procedure is performed when the signals pass further to the output layer. A good visual representation of such a model can be seen on figure 9, where the network is supposed to guess the result of the forthcoming exams according to how much the student slept and studied.

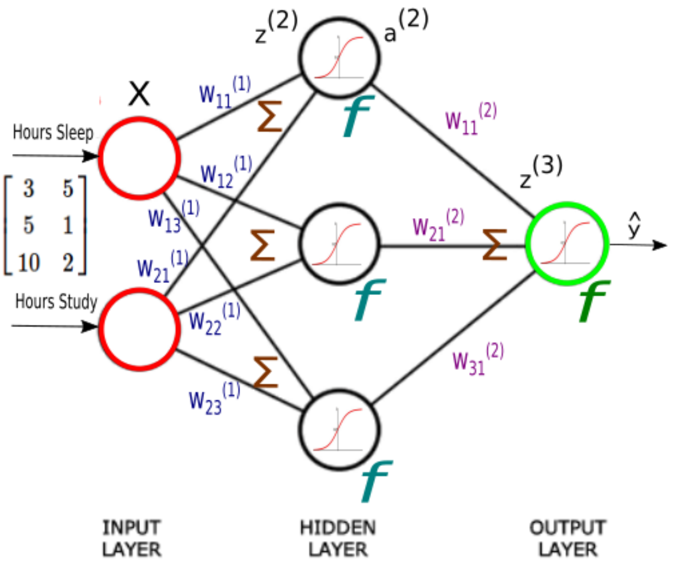


Figure 9: A model of an artifical network. X here is an input matrix of data; $W_{ij}^{(k)}$ is a set of weights from the i-th previous neuron to the j-th next neuron, k is a number of the weight matrix W_{ij} between layers k and $k+1$; Σ means summation of the weights; $z^{(k)}$ and $a^{(k)}$ are the arguments and the values respectively of the activation functions f of the k-th layer; $\widehat{y}$ is the output answer of the network

The main task of machine learning can be formulated as follows: for each given input the network should give an answer, as correct as possible. This may be understood as the task of approximating a function $F(X)$, where X is a set of inputs and F are the correct answers to a question. The learning process, technically, is the operation of adjusting the weights of a given network – not the activation functions. The reason learnig is performed in this way is that it imitates the real brain: all the activation functions are the same, like the biological neurons, and the values of the weights can be compared with the strength of connections between certain brain neurons. There are several methods of learning and ways of their realization, however, the most effective and popular today is backpropogation: by comparing the output answer of the network and the correct answer the so-called error funtion is determined, then the weights are adjusted in such a way that the error function will be minimized (it is called gradient descent).

The weights are usually applied by multiplying them by the corresponding outputs of the previous layer. This mechanism may lead to the understanding of neural networks as models based on linear algebra with simple matrix and vector operations. However, quite often $F(X)$ is not simple enough to be approximated by a linear regression. Here activation functions may provide assistance: the functions such as the logistic function or tanh, so-called sigmoid functions, bring the desired nonlinearity – they simply squeeze extremely high or low values.

In fact, the considered case is extremely simple in comparison to the modern networks that are used in, for example, computer vision, cybersecurity, speech recognition and, of course, data analysis in physics. These ANNs have lots of hidden layers (so-called deep neural networks), and the principles of the propagation of data through some networks are different from those described above. In the next subsection one of such modern networks will be considered in detail.

3.2 Convolutional neural networks

In the framework of the computer vision problem a method was developed for the viewing of images by machines with an "understanding" of what is contained within, as humans do, – convolutional neural networks (CNN). The main distinctive feature of this kind of network is their dimensionality: as we deal with, for example, 2D images with one grayscale color layer, the input data is supposed to be a matrix. The main idea of convolutional networks is to extract the desired features using convolutional layers, reducing the data flowing into the NN. In these layers a picture from the dataset is being convolved with a set of kernels – matrices of smaller size than the picture. The convolution in this sense is element-wise multiplication of a certain part of the image (of the kernel size) with a kernel, then the summation of the results of multiplication gives the inputs for the activation functions. Generally, a kernel is placed on some corner of the given picture, and during the data processing the kernel strides on the picture with certain step in that way it eventually is convolved with each part of the picture. Visually this process is shown on the figure 10. The purpose of each kernel is to contain a certain feature (either a part of a line, or a whole, for example, picture of a number), their abstractness increases as we move from the input to the output, as the network is required to provide a relatively simple answer based on a complex data like photos. By the way, the values in kernels are the only parameters being adjusted in the convolutional portion of the network while learning.

ReLU (Rectified Linear Units) are used as an activation function instead of sigmoids. The formula of ReLU is $f(x) = \max(0, x)$, and although it does not provide that much nonlinearity as, for instance, tanh, it still works well. The main pros of ReLU are its computational simplicity, which allows for the creation of deeper, better, but at the same time fast training networks; and it is a solution to the vanishing gradient problem: with sigmoids, the very deep layers are trained very slowly because of exponential decay of the gradient as the error function is minimized.

Besides the described convolution and ReLU layers, there exist several methods of data enhancement. For example, pooling layers allow the network to get rid of unnecessary data after a convolution or ReLU are applied: the output of convolution is divided into segments, and in each segment the maximum or an average number are taken, so as to reduce the size of the matrix, while retaining important activation informationfrom the pattern recognition (kernel convolution) process (see figure 11). Another interesting kind of layer is the dropout layer. Sometimes the problem of overfitting occurs in the network: this is the situation when the network has learned the specific features of the training data too well, and it is unable to properly process any new data set. In order to provide more robust training, dropout layers are exploited: while taining they randomly set some of the weights to zero (this is illustrated on the figure 12). This will tend to produce a network that is more redundant, and better able to generalize to new data sets.

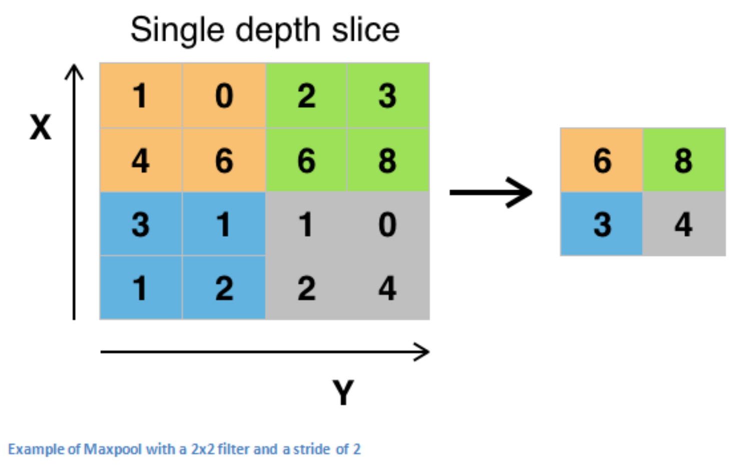


Figure 11: A pooling layer, which passes max number in a segment, reducing then the size of the matrix

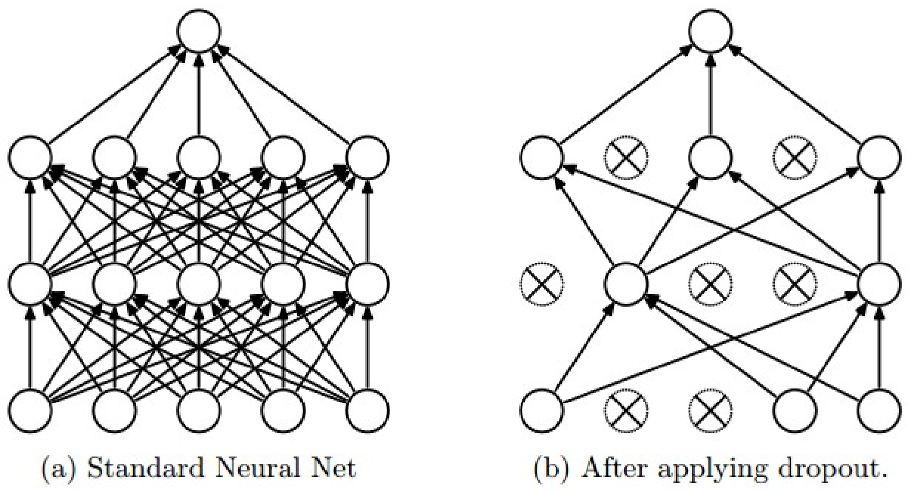
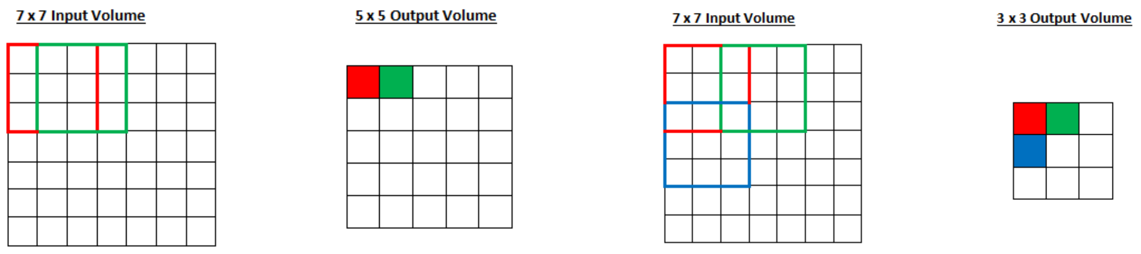


Figure 12: Dropout in a simple network



(a) A convolution with a step equal to 1 (b) A convolution with a step equal to 2

Figure 10: Schemes of convolutions with kernels

3.3 The task of classification in the NOvA experiment

Convolutional neural networks are widely implemented in solving the task of classification. The goal of classification is for each picture provided, the network should be able to determine the category it belongs to: for example, determine if there is cat or dog on a picture from the dataset. The same problem is arisen in the NOvA experiment.

The NOvA collaboration has already created a CNN [17]. It is able to classify neutrino events based on the overall event topology by converting 12-bit energy readings from the detector cells into unsigned 8-bit integers, and then processing these grayscale images through the network. As the data from the detectors is represented by X-Z and Y-Z plane views, one data sample is a "picture" with two layers for X and Y plane each, and the resolution of 100x80 (one of the samples is shown on the figure 13, one layer only). As there are 2 layers, the data sample initially is separated in the network on two branches, then it is merged, as the answer is given with respect to both of the layers. The diagram of NOvA's network is shown on figure 25 in the appendix. For our studies we will be using a CNN to classify neutrino events into 5 categories: ν_e CC, ν_μ CC, ν_τ CC, NC (+other), cosmic background. Because the emergent lepton escapes the detector without leaving a trace in neutral current interactions, it is not generally possible to separate interaction types, and so all NC interactions will be lumped together. But for both CC and NC the simulation samples for each neutrino flavor have comparable proportions. This is to allow the network to learn features of each event type, regardless of their proportions in the data sample.

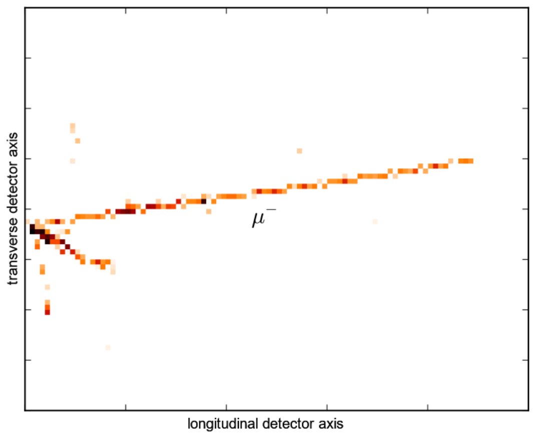


Figure 13: A simulated muon neutrino charged current event in the far detector

4 Improvements of the deep learning methods in NOvA

4.1 Motivation

Despite the fact that NOvA's network copes with its task quite well, there are still many prospets for the improvement of classification. This convolutional neural network was modeled after GoogLeNet [18] – the deep CNN of Google made for the purpose of image classification – by its simplification and adoptation to the 2 layer samples of the datasets.

GoogLeNet has proved to be an excellent image classifier: it has been tested on the ImageNet dataset [19]. In order to gain that degree of precision it was decided to construct this CNN using mini-networks as modules of the GoogLeNet – so-called inseption layers; this architectue utilizes network-in-network layers, or simply, NIN [20]. This approach, basically, allows for the reduction of the dimensionality while still augmenting the learning capacity. The inception modules are, in fact, complete conventional convolutional neural networks, each of them is supposed to recognize some complex features. By the way, in order to reduce the risks of being stuck in a local minimum, GoogLeNet uses a method called local response normalization (LRN), which normilizes the responce of a cell in a kernel map accoring to the activity of adjacent kernel maps.

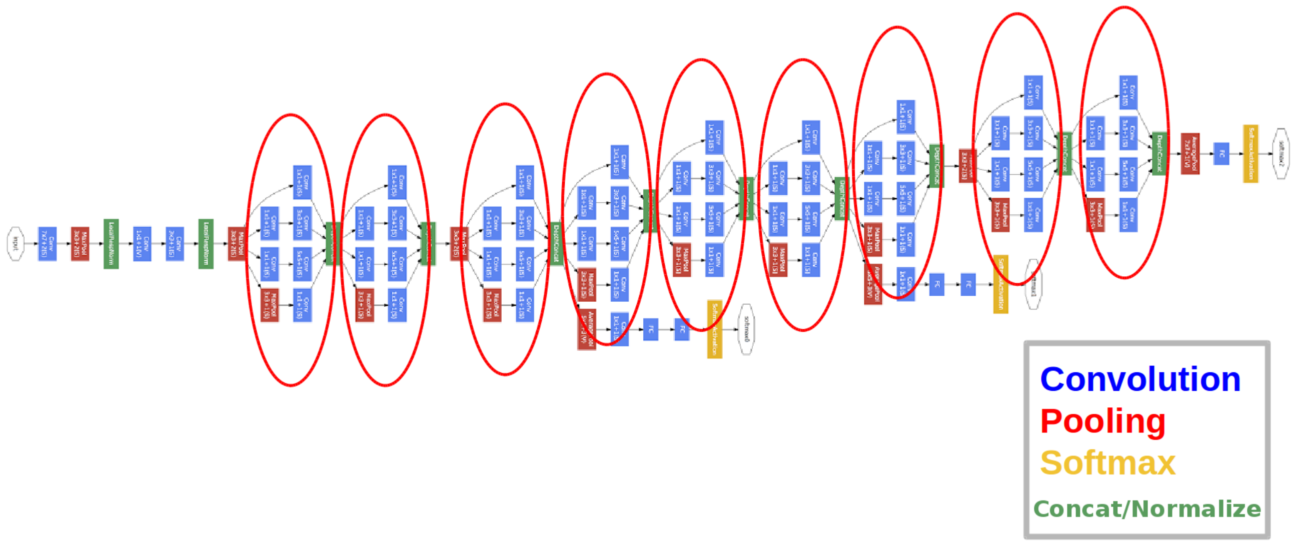


Figure 14: Diagram of the GoogLeNet, the inception layers are circled. The input is in the left side. The network has three sofrmax outputs (three right white blocks)

The main simplification of GoogLeNet for the purpose of neutrino event calssification in NOvA is the reduction of the number of the inception layers (see 25). However, this is not necessarily the final version of NOvA's network. In order to provide training data Monte-Carlo simulations are used. And in order to provide better training, huge datasets are required. However, the enlargement of the datasets leads to enormous training times: for example, in order to train the NOvA's network on a dataset of 300000 data samples for 30 epochs (full training cycles) it takes approximately 8 hours on a single Tesla K40 GPU. For the best quality of training the network needs datasets of several millions of pictures, but then the training time would increase up to several days. Therefore, the investigation of methods of optimization and further simplification are of interest.

4.2 Caffe framework and NVIDIA DIGITS

As neural networks are computer models, they are created with the help of special frameworks, which support fast and convenient programming languages such as C++ or python. The most popular neural network platforms are Tensorflow, PyTorch, MXNet and, of course, Caffe – the one which has been implemented for developing NOvA's CNN.

In order to create a network in Caffe, one should define the layers as a kind of structured data using protocol buffers (.prototxt files) – an alternative solution of Google to replace the less compact and slower XML format. Some examples of layers defined using protocol buffers can be seen in the listing 1 in the appendix.

There are several parameters which are required to set up the learning process. As the back-propagation proceeds, the error function is being minimized, and at each training iteration the rate of minimization, namely, weight adjustment, is defined by the corresponding parameter of the "base learning rate". The word "base" here means that the learning rate is actually decreasing during the training process in order not to leap around the minimum point. The methods of decreasing learning rate are, for example, step decrease (the learning rate discretely drops by a constant value at a set frequency during training), smooth exponential decrease, polynomial decrease etc.

Suppose the error function is E, then its negative gradient is $\partial E/\partial w_i$, where w_i is the value of a weight on the i-th learning iteration. Then the process of updating a weight can be represented by the following formula:

$$w_{i+1} = w_i - \eta \frac{\partial E}{\partial w_i} = w_i - \theta_i, \qquad (1)$$

where η is the learning rate and θ_i is a name for the partial derivative times the rate. A good solution in order to prevent overfitting is to penalize very large weights, which is done by adding a parameter of the weight decay λ, which works as follows:

$$w_{i+1} = w_i - \eta \left(\frac{\partial E}{\partial w_i} + \lambda w_i \right). \tag{2}$$

The λ is usually a relatively small number, however, if w_i is large, the penalty will be significant. This formula is received by the differentiation of the regularization of the function E:

$$\widetilde{E}(\boldsymbol{w}) = E + \frac{\lambda}{2}\boldsymbol{w}^2.$$

Another learning enhancement is derived with including the momentum. Suppose θ_1 is the first update of the weighs, then the second one would be θ_2. However, with the momentum parameter μ it becomes $\theta_2 + \mu\theta_1$ – this means that if the speed of the gradient was fast on step 1 then, even if the minimum is reached, the speed will still be pretty high, like if there is a descending object having some mass. The momentum is introduced with a simple purpose: if we descend towards a local minimum, we will probably overcome it as we came there with some initial speed, but if the minimum is global, then the continuation of our way will be too steep even for such a speed.

There are some other special parameters of the network such as the snapshot rate – if the network is learning too long or if the dataset is of bad quality, we are able to choose one of the previous states of the network, as it was snapshoted during learning. Another parameter is the rate of testing the network upon training: during a testing process the network does not learn, but it is being tested by passing another, usually smaller, dataset through it in order to check how accurate the network predicts answers on a certain step of the training. The maximum number of training iterations is also defined as one of the parameters.

After the network is defined, the next step is to specify the path to the training and testing datasets. The training supposes passing a certain amount of data samples – a "batch", the size of which is predetermined. Whilst the data is being processed, the wrong decisions of the network are being accumulated. Then, with respect to those errors, the adjusting of weights is performed. Generally, the whole training dataset, named "epoch", passes the network several times during learning. Eventually we obtain a trained CNN, which is capable of solving our initial task with certain accuracy.

The Caffe framework supports parallel GPU computation as well as CUDA. NVIDIA provides the Deep Learning GPU Training System, or DIGITS, which exploits the Caffe framework, has a graphical interface, supports prototxt and is capable of showing real-time plots of the training process, as well as pictures of kernels while testing on a single data sample. DIGITS was chosen for realization of the ideas concerning enhancement of the classification.

In order to present the capabilities of DIGITS, a simple default network called LeNet was trained on the MNIST database, composed of 28x28 grayscale images with handwritten digits from 0 to 9. The architecture of LeNet is presented on figure 15, and the learning plot can be seen on figure 16. On figure 17 the visual representation of some layers of LeNet during a one-sample testing is shown.

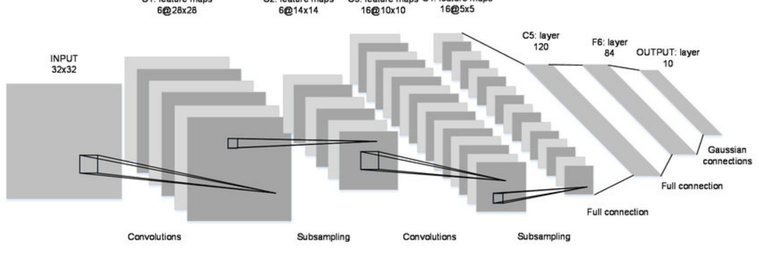


Figure 15: Architecture of LeNet

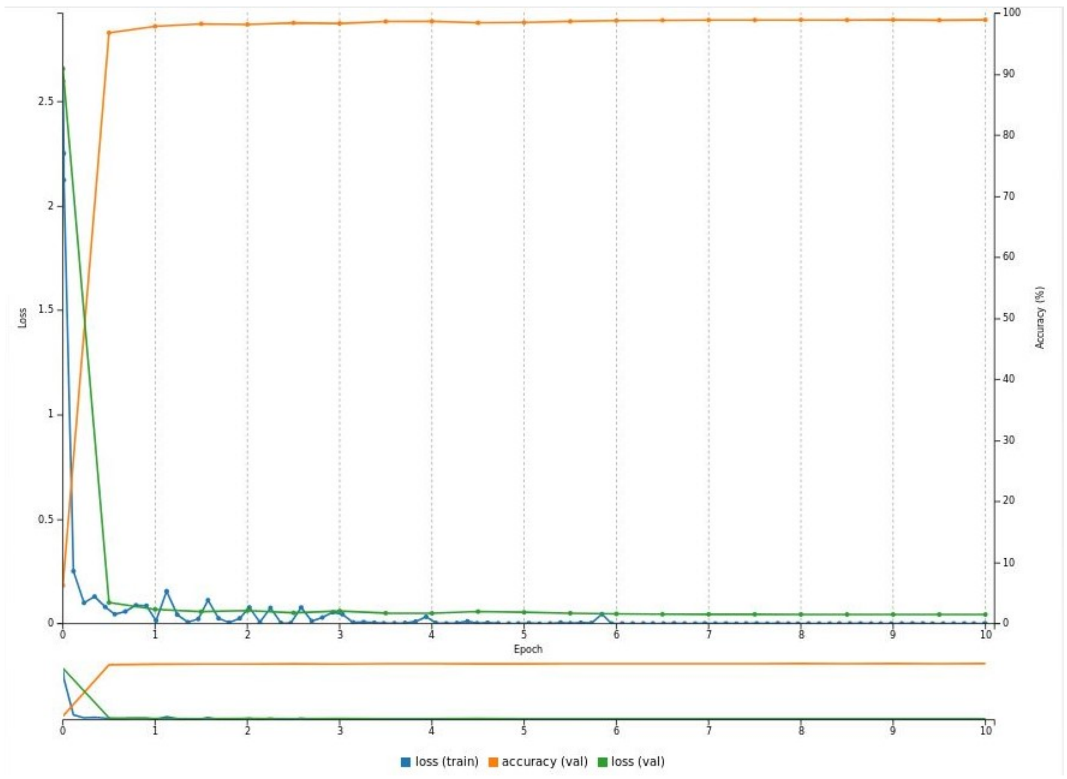
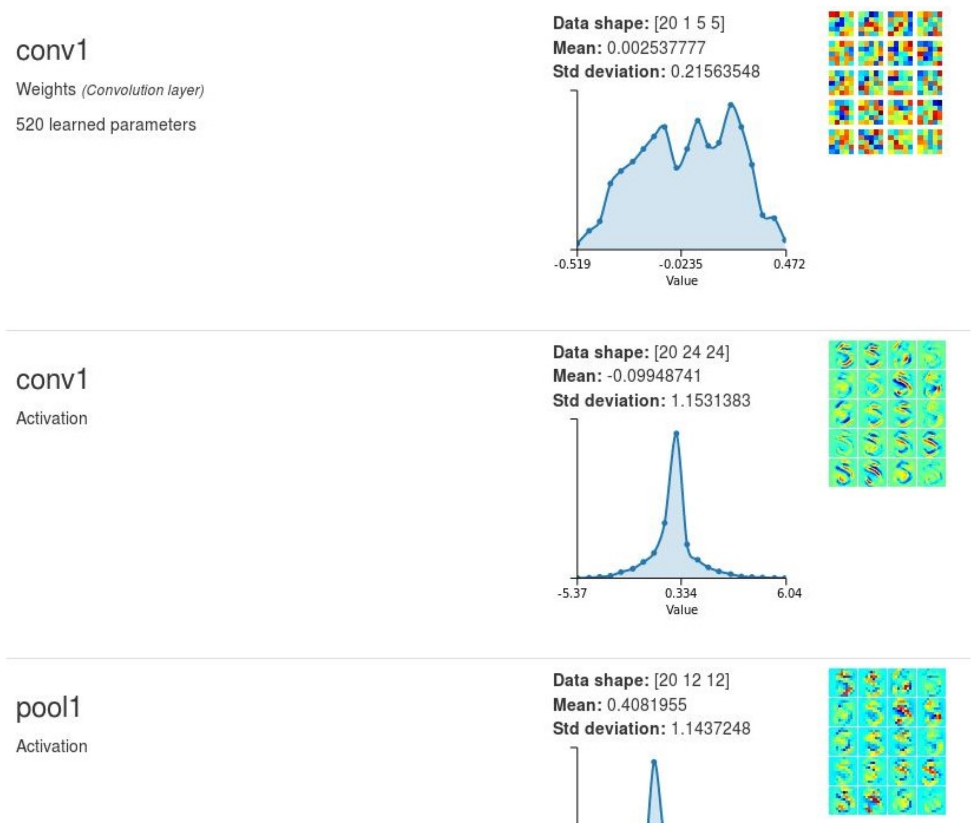


Figure 16: The result of training of the LeNet on the MNIST database


conv1
Weights *(Convolution layer)*
520 learned parameters

Data shape: [20 1 5 5]
Mean: 0.002537777
Std deviation: 0.21563548

conv1
Activation

Data shape: [20 24 24]
Mean: -0.09948741
Std deviation: 1.1531383

pool1
Activation

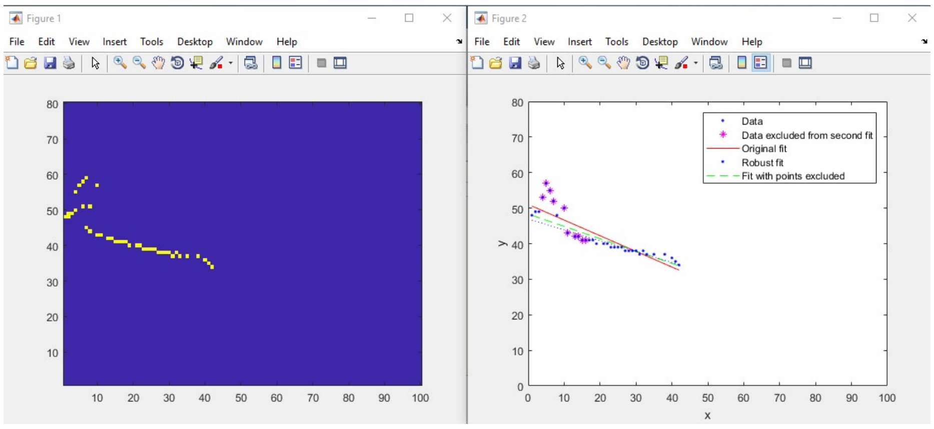
Data shape: [20 12 12]
Mean: 0.4081955
Std deviation: 1.1437248

Figure 17: Some of the layers of the LeNet during a one-sample testing (for a handwritten "5")

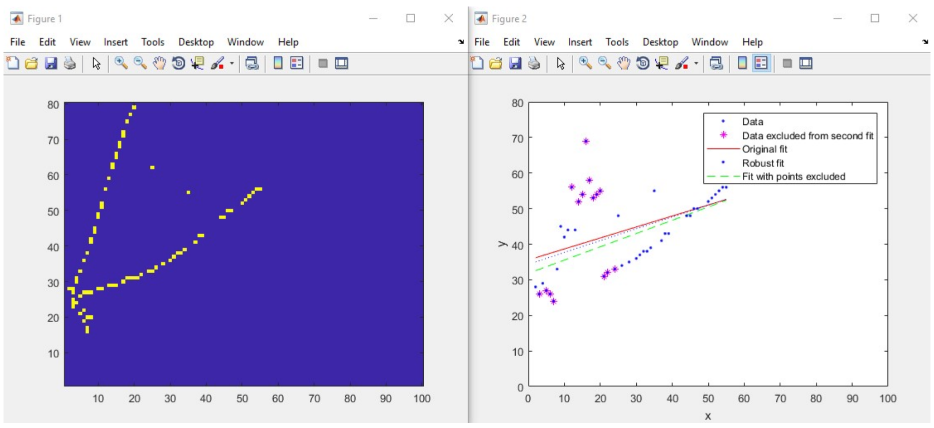
4.3 Polar transformation of the datasets

The purpose of NOvA's CNN is to classify neutrino events without resorting to track reconstruction. However, it is interesting to test a hybrid model: apply certain enhancements to the initial data in order to improve the training accuracy and time. One such enhancement could be the polar transformation of NOvA's data samples – this idea has come form the fact that the pictures of the events are, roughly speaking, lines, outspreading from a certain point – a point of an interaction of the neutrino with the matter of the scintillator. These lines and a point from where they range along are called tracks and a vertex respectively. As the initial pictures require the network to persue rotational invariance of the pictures relative to the vertex point, the transformation of the dataset into the polar coordinates also replaces the rotations with the translations along the angle axis. The hypothesis is, the CNN should adopt to the slight shifts of an image along one axis better than to the indifference to the rotations.

In order to perform the polar transformation, one, foremost, needs to find the vertex point. To catch it, it was decided to first reconstruct one track, or just a line that leads to that point. The first step was to consider a picture as a cartesian plane and the points as values of $y_i = f(x_i)$. In order to test the simplest least square method of linear approximation of a set of points the averages of y_i, if there were several of them for a single x_i, were taken. The approximation equation is given by $k * x + b$, and everywhere below the variables k and b mean the coefficients of this equation. Apart from the basic method the robust solution was also tested: after the initial "simple" linear fit, the algorithm determined the points that are too far and considered them as a noise, then, throwing away those points, another "simple" linear fit occured. As a result, more precise approximation was developed (see figure 18).



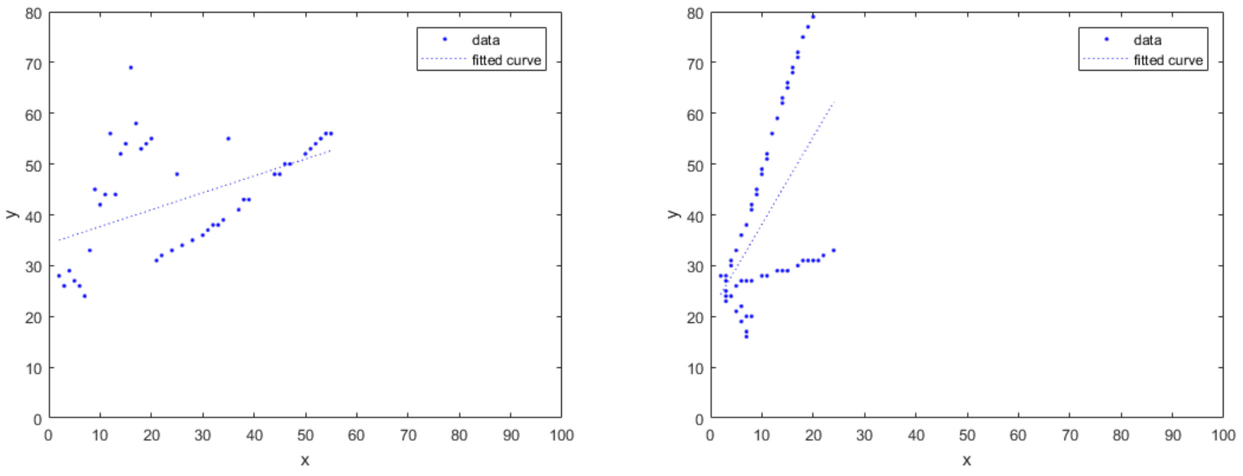
(a) "Original": k = -0.4405, b = 51.04; far points excluded: k = -0.3468, b = 48.33; MATLAB bisquare robust: k = -0.3056, b = 46.93



(b) "Original": $k = 0.3089$, $b = 35.58$; far points excluded: $k = 0.3732$, $b = 31.81$; MATLAB bisquare robust: $k = 0.3327$, $b = 34.36$

Figure 18: Linear approximations of two events (1 layer). There are three kinds of fitting used: "original" least square method – the red line; a method of cutting the outliers that are farther than a certain multiple of the standard deviation of the fitting – the green line; default MATLAB's method of robust fitting (same as the previous, but the outlying points are being chosen with the bisquare method) – the blue line

However, the solution of taking the averages of y_i appeared less effective than fitting the points as they are (figure 19).



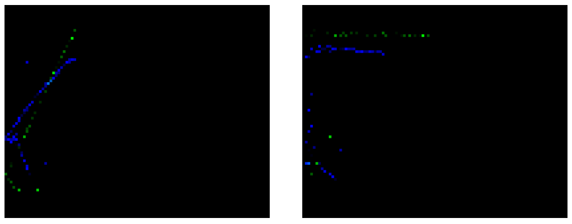
(a) Using averages: $k = 0.3327$, $b = 34.36$ (b) Using initial points: $k = 1.721$, $b = 20.93$

Figure 19: Comparison between uning and not using average points of the vertical axis, MATLAB's robust fitting with the bisquare penalty

This algorythm was eventually rewritten as a python script. In order to fit points, the function "least_squares" of the package scipy was exploited with the value of "loss" set as "cauchy", which provides the strongest penalty to the outliers. Another step of obtaining better fits was the 2-iteration fitting: after the first robust fit, some of the outliers are being cut, then the same second robust fitting is applied. It is worth mentioning, that the approximations were performed for the pictures with their right half being cut: sometimes the secondary interactions appear farther, therefore some of the tracks can be curved or even broken.

In the case of NOvA's training images, the tracks on the pictures are usually begin on the leftmost part of the image, so that there is no need to fit a second track (which, in fact, does not even always exist) in order to find the intersection of the lines. The algorithm of picking a vertex point is the following: for the function of an approximation $t(x)$ the points closer than some small value are being found, and the search is being conducted from the very leftmost part of the picture; then the leftmost nearby point, x_0, is put into the approximation function; finally, the rounded value of $t(x_0)$ is considered as y_0, so that the pair (x_0, y_0) are the coordinates of the vertex point.

Caffe networks use high-perfomance Lightning Memory-Mapped Databases (LMDB). The data stored in a .lmdb database can be easily obtained with the help of the pyCaffe package for python, and it is represented as arrays with their labels (labeled pictures). In order to perform the polar transformation with visualization the corresponding training and testing datasets were extracted



(a) Non-transformed (b) Transformed

Figure 20: Two samples from a standard and a transformed datasets. Two X and Y layers represented as two PNG color layers

as PNGs. Then, after the modification, the pictures were converted back to a new dataset. Two of the datasamples, one from a standard and one from a transformed database, are shown on figure 20. The python polar transformation function can be examined in the listing 2 in the appendix.

4.4 Python layers

There is a possibility of using the polar transformation on-the-fly: it is feasible to convert the function, which takes a NOvA data sample and outputs a polar transformed version of it, into a new layer inside the CNN. In DIGITS these python layers can be simply implemented. The python layers are kept in one python script as classes, each containing functions, wherein all of them are used in a certain part of the training. There are several required functions to be defined in a layer class: forward() and backward() – the former determines how to process data in the layer, while the latter reacts to the gradient descent and is supposed to be used in learning.

4.5 Simplification of the NOvA's network

Alongside with the methods of combining very basic reconstruction with CNN, the network itself can be modified into a simpler version and then compared to the original by their learning capabilities. This is particularly interesting, as such simplification could reduce the learning time. It is also interesting to compare how the original and a simple networks react to the polar transformation of the datasets: how this modification of the images affects the accuracy and stability.

The first simplification of NOvA's CNN was the removal of the parallel inception modules. The resulting diagram of the network can be seen on figure 26 in the appendix. The other ways of simplifying the CNN have not been performed yet.

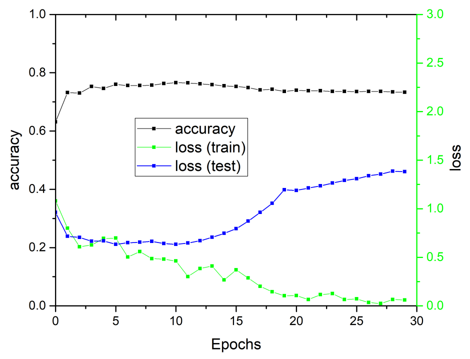
5 Results

5.1 Application of polar transformation

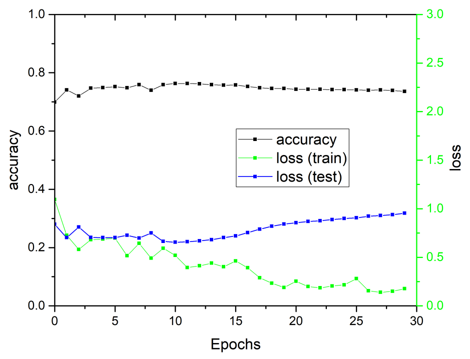
In order to compare the behaviour of NOvA's network with the transformed and non-transformed datasets, the corresponding train-test operations were performed in DIGITS. In fact, there were three kinds of non-transformed databases: small – with 2,000 samples, medium – where their count equals 20,000, and large, or "full", – 291,962 samples. For each of these datasets their transformed analogs were prepared. The purpose of the smallest dataset was primarily to test new features, while the full LMDB was used as the main dataset for serious trainings.

For the first stage, the default NOvA CNN was tested, the correspondong plots are presented below (figure 23 in the appendix). It can be seen, that polar transformation appears to reduce the risk of overtraining, while only slightly reducing the overall accuracy. We can compare the two plots with the most significant difference (see figure 21) and calculate the corresponding differences of accuracies (Δa) and testing losses (Δl_{val}) at the final iteration of the training (the numbers are from table 2). The quoted relative differences will simply be the difference between values, divided by the average. The relative difference between the test and train loss (Δl_{tt}) for each plot will give some measure of the network's over training.

The next step was to test the diminished CNN, without the inception modules. The result is shown on figure 24 in the appendix. According to the plots, the polar transformation helps the simpler CNN in the same way as it augments the data for the standatd network (see figure 22, where the most qualitative example is shown), however, not that significantly. This data modification again slightly reduces the overall training accuracy. We can also see, that when the standard network is simplified, the relative drop in accuracy is very small, especially for the largest datasets: $\Delta a = 1.0\%$.

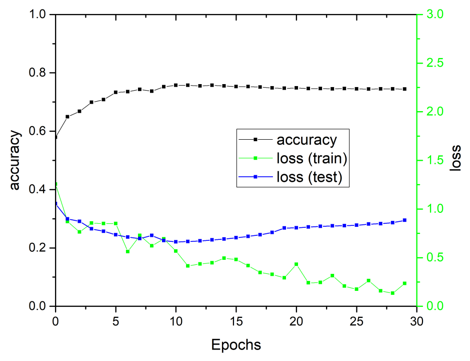


(a) Standard CNN trained on the medium LMDB. Training time 31 minutes, 57 seconds. $\Delta l_{tt} = 183.5\%$

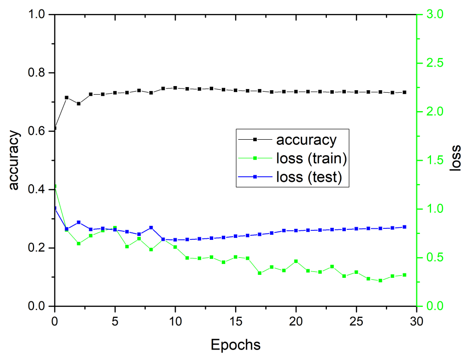


(b) Standard CNN trained on the medium transformed LMDB. Training time 32 minutes, 4 seconds. $\Delta l_{tt} = 25.8\%$

Figure 21: Improvement of the validation loss with polar transformation (standard CNN). Relative differences between standard and transformed datasets: $\Delta a = 0.4\%$, $\Delta l_{val} = 36.6\%$



(a) Simplified CNN trained on the medium LMDB. Training time 22 minutes, 8 seconds. $\Delta l_{tt} = 116.0\%$



(b) Simplified CNN trained on the medium transformed LMDB. Training time 22 minutes, 41 seconds. $\Delta l_{tt} = 87.0\%$

Figure 22: Improvement of the validation loss with polar transformation (simplified CNN). Relative differences between standard and transformed datasets: $\Delta a = 1.5\%$, $\Delta l_{val} = 8.2\%$

The reason the accuracy reduction happens could be the data loss while the polar modification is processed: some of the points far from the vertex, even if they are visually separated from each other, can have the same rounded polar angle. At the same time, two pixels close to each other and also to the vertex may differ on relatively large angles, so that on the polar transformed image of these pixels appear to be rather far apart. Also, it is necessary to mention that the decision to dispose of the incepton modules did not dramatically affect the accuracy and the validation loss.

5.2 The main problem of python layers

While the implementation of the python layers seems to be a great solution, there is one major problem: these layers causes bottlenecks, because now it is not possible to process those layers using the high perfomance GPU. Instead, the python layers tend to prefer CPU. Thus, despite the attractiveness of this approach, it appears to be unworkable. However, this problem is, no doubt, solvable (by using PyCUDA, for example).

6 Conclusions

We have applied a polar transformation to the NOvA training MC, in the hopes of creating a dataset that will allow the network to learn more quickly, or in a more robust manner. The hypothesis is that the translational symmetry of such a dataset will provide simpler patterns for the network to learn than the rotationally symmetric raw event data. We have showed that this modification to the data reduces the risk of over training the network, making it more robust, and better able to generalize when given new data. However, this modification does slightly reduce the overall accuracy of the network, but only by a marginal amount (<2%). A simple vertexing method was developed in order to perform the polar transformation of the events. It would be interesting to investigate improvements in the vertexing, as well as improvements to the transformation method (for example, rescaling the transformed data samples to higher resolutions in order to avoid the loss of data when mapping pixels to polar coordinates).

We also attempted to simplify NOvA's CNN structure. Removing parallel layers in the network affected the accuracy by a very small amount, while reducing the training time by a factor of 1.5 or more. Such modifications would be useful for the NOvA collaboration to investigate, as one of the main issues with trying new CNN methods is the training time required for each test.

Another interesting possibility is the implementation of python layers in the network that exploit the GPU, allowing for on-the-fly calculations or modification of the data (raw or within the network). Currently our Python layers produce a bottleneck due to their reliance on the CPU for calculation. Additionally, the datasets themselves could be augmented with an additional third layer, which could be filled with basic reconstruction information that may help the network to learn more quickly or more robustly (track fuzziness, number of tracks, or hits per track, for example).

7 Acknowledgements

A very special gratitude goes out to Chris Kullenberg, a member of the NOvA team, for his expert advice and encouragement during this project, as well as to Oleg Borisovich Samoylov, also a member of NOvA, for accepting and inviting to the JINR Summer Student Program. Also, I would like to thank the JINR organizing committee for providing the opportunity to participate in SSP. In addition, I wish to give thanks to JINR for providing the facilities and funding.

8 Bibliography

References

[1] T. D. Lee (1987). *History of the weak interactions.* CERN Courier, **January/February 1987**, pp. 7-12

[2] C. Kullenberg (2018). *History of Neutrino Physics.* `http://astronu.jinr.ru/wiki/index.php/File:Ckullenberg-phil-essay.pdf`

[3] D. Verkindt (1999). *Neutrino history.* `https://lappweb.in2p3.fr/neutrinos/anhistory.html`

[4] K. S. Krane (1955). *Introductory Nuclear Physics.* ISBN-13: 978-0471805533. ISBN-10: 047180553X

[5] G. Rajasekaran (2014). *Fermi and the Theory of Weak Interactions.* Resonance (Indian Academy of Sciences, Bangalore), vol **19**, No 1., pp. 18-44 arXiv:1403.3309

[6] M. Blennow (2007). *Theoretical and Phenomenological Studies of Neutrino Physics.* ISBN 978-91-7178-646-3

[7] B. Pontecorvo (1967). *Neutrino Experiments and the Problem of Conservation of Leptonic Charge.* Sov. Phys. JETP **26** 984-988, Zh. Eksp. Teor. Fiz. **53** 1717-1725 `http://inspirehep.net/record/51319`

[8] C Giganti, S Lavignac, M Zito (2017). *Neutrino oscillations: the rise of the PMNS paradigm.* Prog. Part. Nucl. Phys. **98**, pp. 1-54 arXiv:1710.00715

[9] Y. Fukuda et al. (1998). *Evidence for oscillation of atmospheric neutrinos.* Phys. Rev. Lett., vol. **81**, pp. 1562–1567 arXiv:hep-ex/9807003

[10] Q. Ahmad et al. (2002). *Direct Evidence for Neutrino Flavor Transformation from Neutral-Current Interactions in the Sudbury Neutrino Observatory.* Phys. Rev. Lett., vol. **89**, p. 011301 arXiv:nucl-ex/0204008

[11] K. Eguchi et al. (2003). *First results from KamLAND: evidence for reactor anti-neutrino disappearance.* Phys. Rev. Lett, vol. **90**, p. 093004 arXiv:hep-ex/0212021

[12] F. Vannucci (2014). *The NOMAD Experiment at CERN.* `http://dx.doi.org/10.1155/2014/129694`

[13] P. Adamson et al. (2015). *The NuMI Neutrino Beam.* Nucl. Instrum. Methods Phys. Res. A, vol **806**, pp. 279-306 arXiv:1507.06690

[14] B. Behera, G. Davies, F. Psihas (2017). *Event Reconstruction in the NOvA Experiment.* FERMILAB-CONF-17-513-E arXiv:1710.03772

[15] L. Teodorescu (2008). *Artificial neural networks in high-energy physics.* C06-03-06.2, p.13-22. C05-02-23.1, p.13-22 `http://cds.cern.ch/record/1100521/files/`

[16] S. Harnad (2006). *The Annotation Game: On Turing (1950) on Computing, Machinery, and Intelligence.* Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer. Evolving Consciousness Springer, pp. 23-66

[17] A. Aurisano et al. (2016). *A Convolutional Neural Network Neutrino Event Classifier.* JINST 11, no. **09**, p. 09001 arXiv:1604.01444

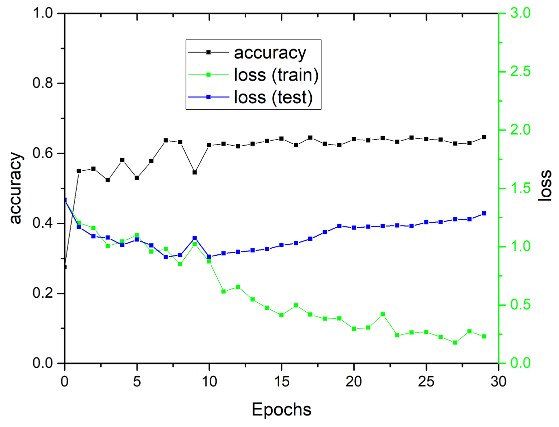
[18] C. Szegedy et al. (2014). *Going Deeper with Convolutions.* 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) arXiv:1409.4842

[19] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma et al. (2015). *ImageNet Large Scale Visual Recognition Challenge.* International Journal of Computer Vision (IJCV) **115**, pp. 211–252

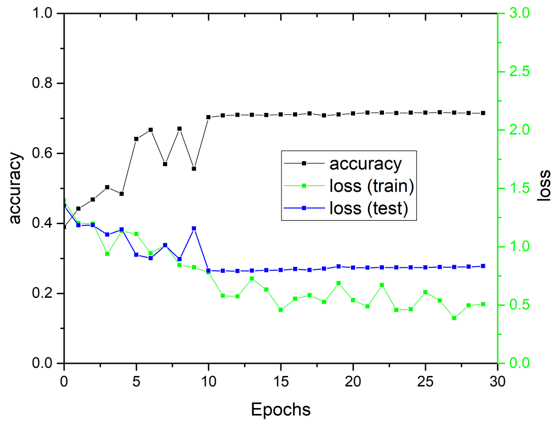
[20] Min Lin, Qiang Chen, Shuicheng Yan (2014). *Network In Network.* arXiv:1312.4400

9 Appendix

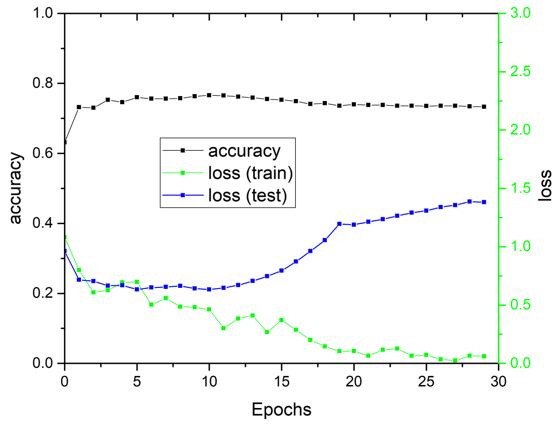
9.1 Figures



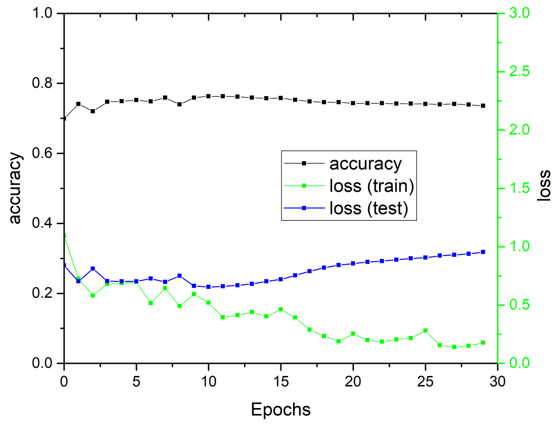
(a) Standard CNN trained on the small LMDB. Training time 11 minutes, 35 seconds



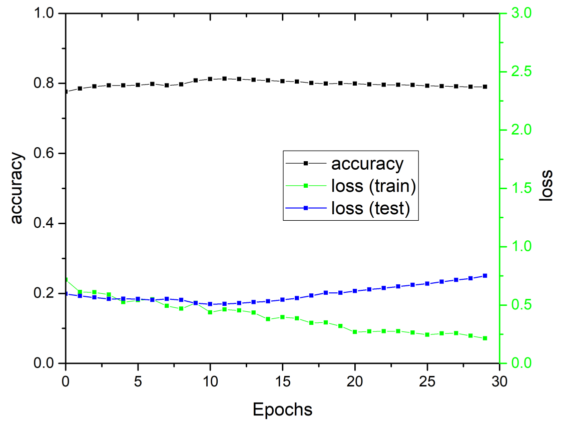
(b) Standard CNN trained on the small transformed LMDB. Training time 21 minutes, 7 seconds



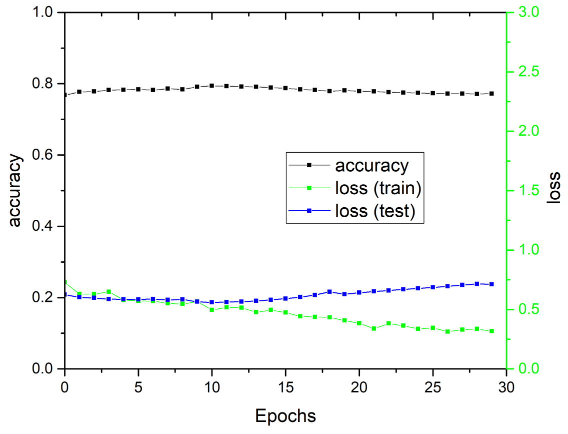
(c) Standard CNN trained on the medium LMDB. Training time 31 minutes, 57 seconds



(d) Standard CNN trained on the medium transformed LMDB. Training time 32 minutes, 4 seconds

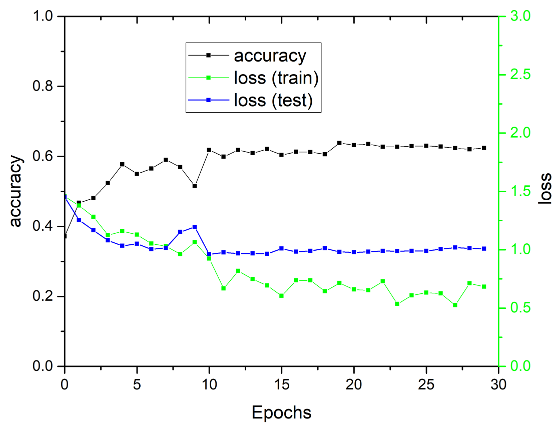


(e) Standard CNN trained on the full LMDB. Training time 7 hours, 2 minutes

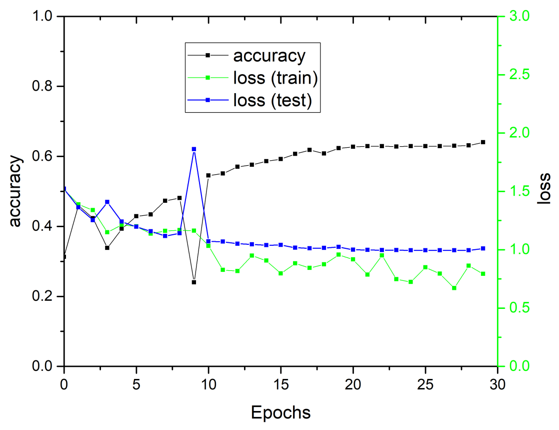


(f) Standard CNN trained on the full transformed LMDB. Training time 7 hours, 0 minutes

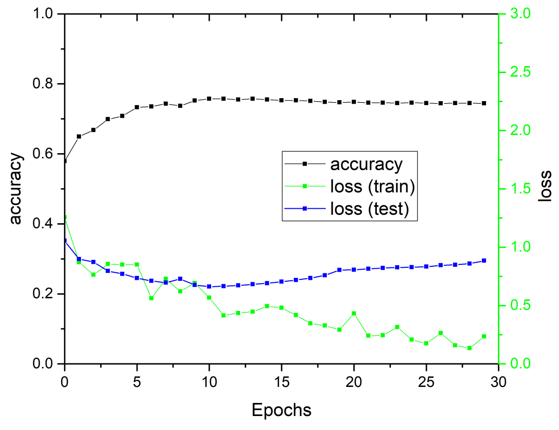
Figure 23: The results of training operations on the standard NOvA's CNN



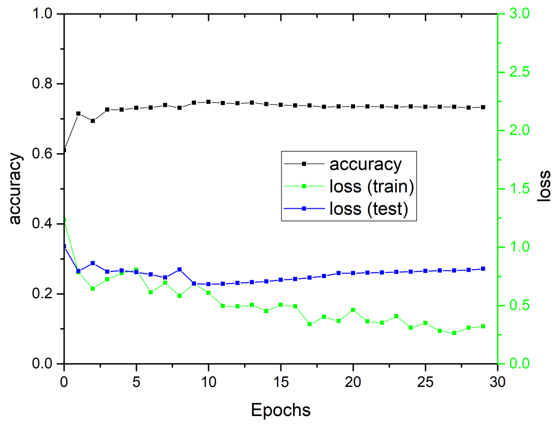
(a) Simplified CNN trained on the small LMDB. Training time 2 minutes, 23 seconds)



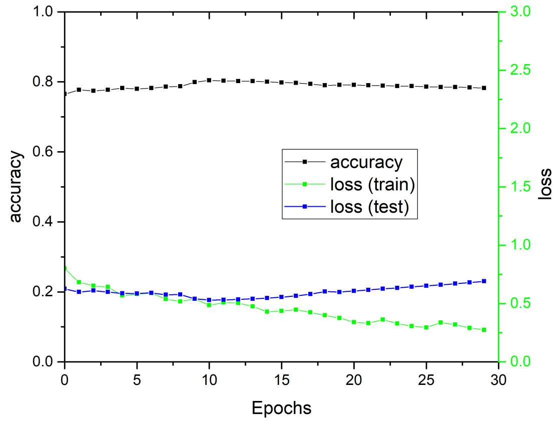
(b) Simplified CNN trained on the small transformed LMDB. Training time 8 minutes, 49 seconds



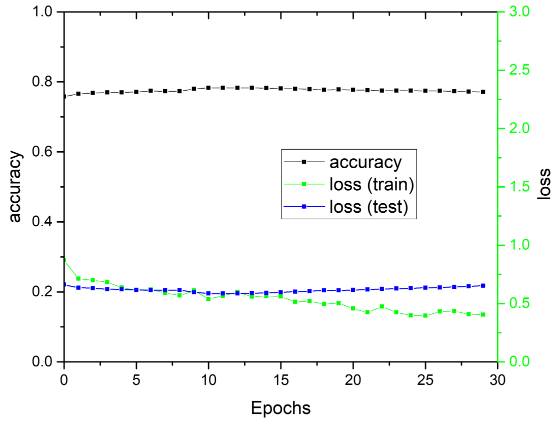
(c) Simplified CNN trained on the medium LMDB. Training time 22 minutes, 8 seconds



(d) Simplified CNN trained on the medium transformed LMDB. Training time 22 minutes, 41 seconds

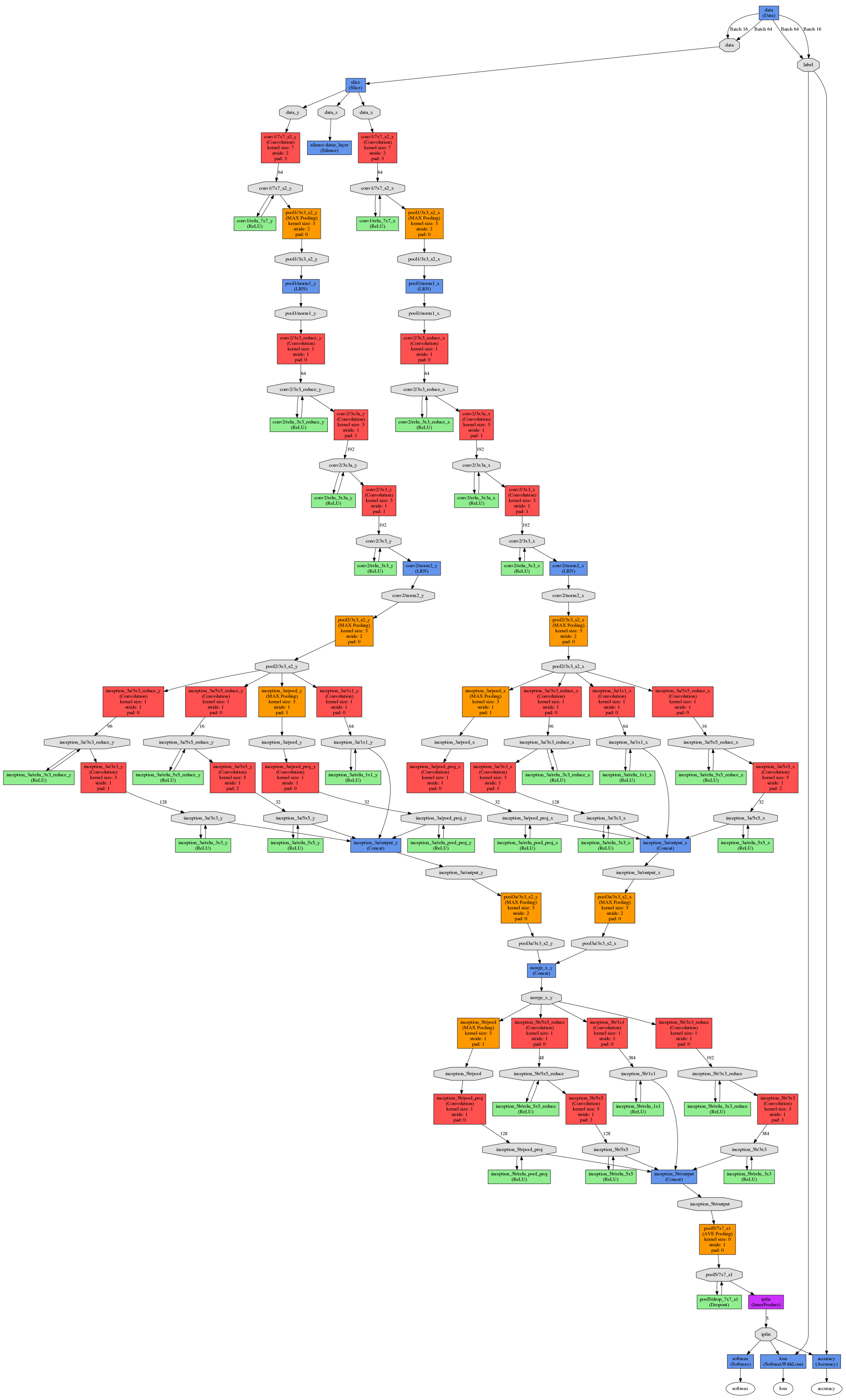


(e) Simplified CNN trained on the full LMDB. Training time 4 hours, 40 minutes



(f) Simplified CNN trained on the full transformed LMDB. Training time 4 hours, 41 minutes

Figure 24: The results of training operations on the NOvA's CNN without the inception layers



data (Data)
Batch 16
Batch 64
data
label
slice (Slice)
data_y
data_z
data_x
silence-dataz_layer (Silence)
conv1/7x7_s2_y (Convolution) kernel size: 7 stride: 2 pad: 3
conv1/7x7_s2_x (Convolution) kernel size: 7 stride: 2 pad: 3
conv1/relu_7x7_y (ReLU)
pool1/3x3_s2_y (MAX Pooling) kernel size: 3 stride: 2 pad: 0
pool1/norm1_y (LRN)
conv2/3x3_reduce_y (Convolution) kernel size: 1 stride: 1 pad: 0
conv2/3x3_y (Convolution) kernel size: 3 stride: 1 pad: 1
conv2/norm2_y (LRN)
pool2/3x3_s2_y (MAX Pooling) kernel size: 3 stride: 2 pad: 0
inception_3a/output_y (Concat)
inception_3a/output_x (Concat)
pool3a/3x3_s2_y (MAX Pooling) kernel size: 3 stride: 2 pad: 0
pool3a/3x3_s2_x (MAX Pooling) kernel size: 3 stride: 2 pad: 0
merge_x_y (Concat)
inception_5b/pool (MAX Pooling) kernel size: 3 stride: 1 pad: 1
inception_5b/5x5_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
inception_5b/1x1 (Convolution) kernel size: 1 stride: 1 pad: 0
inception_5b/3x3_reduce (Convolution) kernel size: 1 stride: 1 pad: 0
inception_5b/output (Concat)
pool5/7x7_s1 (AVE Pooling) kernel size: 0 stride: 1 pad: 0
pool5/drop_7x7_s1 (Dropout)
ipfin (InnerProduct)
softmax (Softmax)
loss (SoftmaxWithLoss)
accuracy (Accuracy)
softmax
loss
accuracy

Figure 25: Diagram of NOvA CNN, obtained with NVIDIA DIGITS

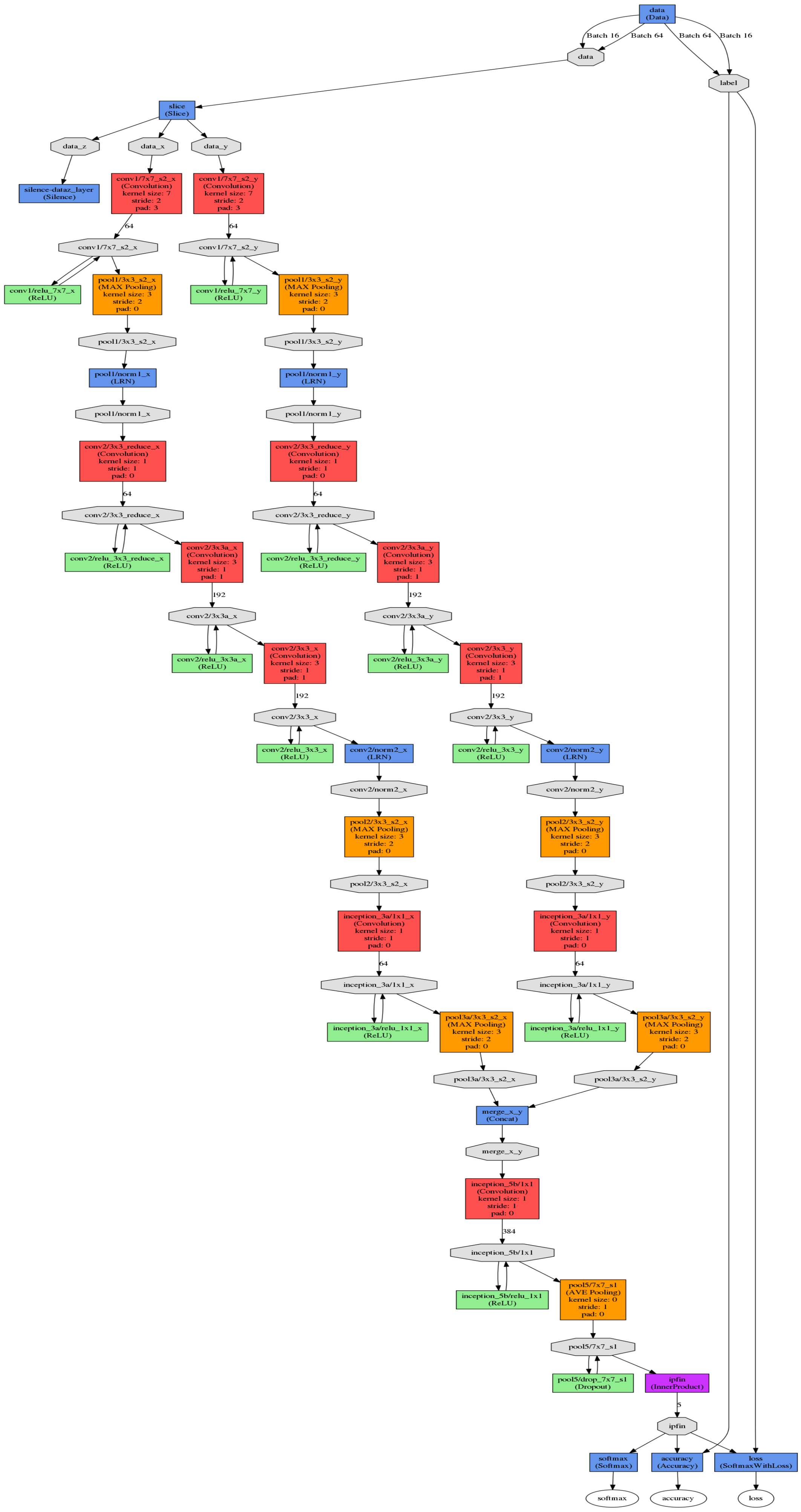

data (Data)
Batch 16
Batch 64
Batch 64
Batch 16
data
label
slice (Slice)
data_z
data_x
data_y
silence-dataz_layer (Silence)
conv1/7x7_s2_x (Convolution) kernel size: 7 stride: 2 pad: 3
conv1/7x7_s2_y (Convolution) kernel size: 7 stride: 2 pad: 3
64
64
conv1/7x7_s2_x
conv1/7x7_s2_y
conv1/relu_7x7_x (ReLU)
pool1/3x3_s2_x (MAX Pooling) kernel size: 3 stride: 2 pad: 0
conv1/relu_7x7_y (ReLU)
pool1/3x3_s2_y (MAX Pooling) kernel size: 3 stride: 2 pad: 0
pool1/3x3_s2_x
pool1/3x3_s2_y
pool1/norm1_x (LRN)
pool1/norm1_y (LRN)
pool1/norm1_x
pool1/norm1_y
conv2/3x3_reduce_x (Convolution) kernel size: 1 stride: 1 pad: 0
conv2/3x3_reduce_y (Convolution) kernel size: 1 stride: 1 pad: 0
64
64
conv2/3x3_reduce_x
conv2/3x3_reduce_y
conv2/relu_3x3_reduce_x (ReLU)
conv2/3x3a_x (Convolution) kernel size: 3 stride: 1 pad: 1
conv2/relu_3x3_reduce_y (ReLU)
conv2/3x3a_y (Convolution) kernel size: 3 stride: 1 pad: 1
192
192
conv2/3x3a_x
conv2/3x3a_y
conv2/relu_3x3a_x (ReLU)
conv2/3x3_x (Convolution) kernel size: 3 stride: 1 pad: 1
conv2/relu_3x3a_y (ReLU)
conv2/3x3_y (Convolution) kernel size: 3 stride: 1 pad: 1
192
192
conv2/3x3_x
conv2/3x3_y
conv2/relu_3x3_x (ReLU)
conv2/norm2_x (LRN)
conv2/relu_3x3_y (ReLU)
conv2/norm2_y (LRN)
conv2/norm2_x
conv2/norm2_y
pool2/3x3_s2_x (MAX Pooling) kernel size: 3 stride: 2 pad: 0
pool2/3x3_s2_y (MAX Pooling) kernel size: 3 stride: 2 pad: 0
pool2/3x3_s2_x
pool2/3x3_s2_y
inception_3a/1x1_x (Convolution) kernel size: 1 stride: 1 pad: 0
inception_3a/1x1_y (Convolution) kernel size: 1 stride: 1 pad: 0
64
64
inception_3a/1x1_x
inception_3a/1x1_y
inception_3a/relu_1x1_x (ReLU)
pool3a/3x3_s2_x (MAX Pooling) kernel size: 3 stride: 2 pad: 0
inception_3a/relu_1x1_y (ReLU)
pool3a/3x3_s2_y (MAX Pooling) kernel size: 3 stride: 2 pad: 0
pool3a/3x3_s2_x
pool3a/3x3_s2_y
merge_x_y (Concat)
merge_x_y
inception_5b/1x1 (Convolution) kernel size: 1 stride: 1 pad: 0
384
inception_5b/1x1
inception_5b/relu_1x1 (ReLU)
pool5/7x7_s1 (AVE Pooling) kernel size: 0 stride: 1 pad: 0
pool5/7x7_s1
pool5/drop_7x7_s1 (Dropout)
ipfin (InnerProduct)
5
ipfin
softmax (Softmax)
accuracy (Accuracy)
loss (SoftmaxWithLoss)
softmax
accuracy
loss

Figure 26: Diagram of simplified NOvA CNN with the inception modules removed, obtained with NVIDIA DIGITS

9.2 Tables

NumIters	accuracy	loss
0	0.063	2.654
352	0.968	0.101
704	0.978	0.068
1056	0.983	0.057
1408	0.982	0.062
1760	0.984	0.052
2112	0.983	0.060
2464	0.987	0.050
2816	0.987	0.049
3168	0.984	0.058
3520	0.985	0.054
3872	0.987	0.050
4224	0.988	0.046
4576	0.988	0.045
4928	0.989	0.045
5280	0.989	0.044
5632	0.989	0.044
5984	0.989	0.044
6336	0.989	0.044
6688	0.989	0.044
7040	0.989	0.044

Table 1: The results of the validation testing procedures during training the LeNet on the MNIST dataset. “NumIters” is the number of the training iterations, “accuracy” and “loss” stand for the validation accuracy and loss

stdSmall				stdSmallT				stdMed			
Epoch	accuracy	loss (v)	loss (t)	NumIters	accuracy	loss (v)	loss (t)	NumIters	accuracy	loss (v)	loss (t)
0	0.275	1.402	1.399	0	0.389	1.350	1.394	0	0.631	0.963	1.080
1	0.549	1.169	1.205	1	0.442	1.183	1.202	1	0.732	0.716	0.799
2	0.556	1.087	1.160	2	0.468	1.184	1.196	2	0.730	0.705	0.608
3	0.523	1.077	1.006	3	0.503	1.103	0.938	3	0.753	0.666	0.627
4	0.581	1.014	1.043	4	0.484	1.147	1.135	4	0.746	0.668	0.694
5	0.530	1.060	1.100	5	0.641	0.930	1.108	5	0.760	0.634	0.697
6	0.578	1.010	0.958	6	0.667	0.901	0.944	6	0.756	0.650	0.502
7	0.637	0.912	0.979	7	0.569	1.011	1.015	7	0.756	0.655	0.559
8	0.632	0.929	0.850	8	0.670	0.893	0.842	8	0.757	0.663	0.486
9	0.545	1.075	1.022	9	0.556	1.156	0.824	9	0.763	0.641	0.481
10	0.623	0.913	0.872	10	0.703	0.795	0.779	10	0.766	0.633	0.461
11	0.627	0.942	0.615	11	0.708	0.793	0.579	11	0.765	0.647	0.301
12	0.620	0.955	0.656	12	0.710	0.791	0.573	12	0.762	0.670	0.384
13	0.627	0.967	0.546	13	0.710	0.794	0.726	13	0.759	0.706	0.410
14	0.635	0.979	0.475	14	0.709	0.798	0.631	14	0.755	0.746	0.266
15	0.642	1.013	0.414	15	0.711	0.799	0.458	15	0.753	0.795	0.371
16	0.623	1.029	0.495	16	0.711	0.808	0.554	16	0.749	0.873	0.287
17	0.645	1.067	0.417	17	0.714	0.800	0.584	17	0.741	0.963	0.201
18	0.627	1.125	0.382	18	0.708	0.811	0.527	18	0.743	1.056	0.144
19	0.623	1.177	0.384	19	0.711	0.830	0.686	19	0.736	1.195	0.104
20	0.640	1.162	0.296	20	0.714	0.820	0.541	20	0.740	1.188	0.105
21	0.637	1.171	0.305	21	0.716	0.820	0.487	21	0.738	1.213	0.065
22	0.643	1.176	0.419	22	0.716	0.822	0.670	22	0.738	1.235	0.115
23	0.633	1.181	0.239	23	0.715	0.822	0.456	23	0.736	1.265	0.126
24	0.645	1.177	0.264	24	0.716	0.821	0.463	24	0.736	1.291	0.065
25	0.640	1.208	0.267	25	0.716	0.822	0.610	25	0.735	1.309	0.071
26	0.639	1.212	0.226	26	0.717	0.825	0.538	26	0.736	1.339	0.035
27	0.628	1.233	0.177	27	0.716	0.825	0.387	27	0.736	1.356	0.024
28	0.629	1.234	0.274	28	0.715	0.828	0.496	28	0.734	1.386	0.065
29	0.646	1.284	0.229	29	0.715	0.833	0.506	29	0.733	1.382	0.060

stdMedT				stdFull				stdFullT			
Epoch	accuracy	loss (v)	loss (t)	NumIters	accuracy	loss (v)	loss (t)	NumIters	accuracy	loss (v)	loss (t)
0	0.699	0.840	1.095	0	0.776	0.595	0.717	0	0.768	0.624	0.728
1	0.741	0.702	0.727	1	0.785	0.578	0.611	1	0.777	0.602	0.630
2	0.720	0.811	0.581	2	0.791	0.565	0.609	2	0.778	0.597	0.629
3	0.747	0.705	0.681	3	0.794	0.552	0.589	3	0.782	0.587	0.649
4	0.749	0.702	0.687	4	0.794	0.554	0.524	4	0.783	0.585	0.579
5	0.752	0.703	0.697	5	0.795	0.551	0.541	5	0.784	0.584	0.572
6	0.748	0.726	0.516	6	0.798	0.544	0.549	6	0.782	0.588	0.571
7	0.759	0.698	0.643	7	0.794	0.551	0.493	7	0.786	0.580	0.551
8	0.740	0.750	0.491	8	0.797	0.544	0.468	8	0.784	0.584	0.546
9	0.759	0.663	0.593	9	0.808	0.516	0.517	9	0.791	0.565	0.567
10	0.763	0.655	0.519	10	0.812	0.508	0.437	10	0.794	0.560	0.496
11	0.763	0.660	0.394	11	0.813	0.510	0.462	11	0.793	0.562	0.517
12	0.762	0.669	0.412	12	0.812	0.516	0.454	12	0.792	0.566	0.515
13	0.759	0.681	0.439	13	0.810	0.524	0.435	13	0.791	0.572	0.477
14	0.757	0.703	0.403	14	0.808	0.532	0.378	14	0.789	0.581	0.495
15	0.758	0.719	0.461	15	0.806	0.545	0.396	15	0.787	0.591	0.473
16	0.753	0.754	0.393	16	0.805	0.558	0.386	16	0.784	0.604	0.442
17	0.748	0.790	0.288	17	0.801	0.580	0.347	17	0.782	0.622	0.437
18	0.746	0.820	0.233	18	0.799	0.605	0.350	18	0.779	0.649	0.433
19	0.746	0.842	0.189	19	0.800	0.605	0.320	19	0.781	0.628	0.407
20	0.743	0.855	0.254	20	0.799	0.620	0.268	20	0.779	0.641	0.383

21	0.743	0.869	0.199	21	0.797	0.634	0.274	21	0.778	0.651	0.338
22	0.743	0.876	0.185	22	0.796	0.645	0.276	22	0.776	0.659	0.382
23	0.742	0.888	0.204	23	0.796	0.658	0.276	23	0.775	0.669	0.363
24	0.742	0.899	0.216	24	0.795	0.672	0.263	24	0.774	0.677	0.337
25	0.741	0.906	0.280	25	0.793	0.683	0.245	25	0.773	0.686	0.345
26	0.740	0.923	0.154	26	0.792	0.699	0.256	26	0.772	0.695	0.313
27	0.741	0.930	0.139	27	0.791	0.714	0.259	27	0.772	0.706	0.330
28	0.739	0.938	0.150	28	0.790	0.729	0.237	28	0.771	0.715	0.336
29	0.736	0.954	0.176	29	0.790	0.750	0.214	29	0.772	0.712	0.317

simSmall				simSmallT				simMed			
Epoch	accuracy	loss (v)	loss (t)	NumIters	accuracy	loss (v)	loss (t)	NumIters	accuracy	loss (v)	loss (t)
0	0.371	1.452	1.458	0	0.313	1.522	1.518	0	0.579	1.055	1.255
1	0.467	1.253	1.378	1	0.460	1.363	1.388	1	0.649	0.898	0.870
2	0.481	1.166	1.281	2	0.423	1.253	1.339	2	0.668	0.873	0.764
3	0.524	1.079	1.125	3	0.338	1.408	1.149	3	0.699	0.796	0.856
4	0.577	1.033	1.159	4	0.393	1.240	1.214	4	0.708	0.771	0.850
5	0.550	1.051	1.128	5	0.429	1.196	1.199	5	0.733	0.735	0.851
6	0.565	1.003	1.052	6	0.434	1.158	1.137	6	0.735	0.712	0.562
7	0.590	1.015	1.030	7	0.473	1.117	1.160	7	0.743	0.696	0.728
8	0.569	1.152	0.963	8	0.481	1.140	1.168	8	0.737	0.728	0.622
9	0.515	1.195	1.064	9	0.240	1.862	1.162	9	0.752	0.676	0.693
10	0.618	0.961	0.923	10	0.545	1.070	1.032	10	0.757	0.662	0.567
11	0.599	0.975	0.667	11	0.551	1.069	0.826	11	0.757	0.666	0.415
12	0.618	0.968	0.818	12	0.570	1.051	0.817	12	0.755	0.672	0.435
13	0.609	0.968	0.748	13	0.576	1.046	0.949	13	0.757	0.682	0.447
14	0.621	0.964	0.693	14	0.586	1.038	0.906	14	0.755	0.691	0.494
15	0.604	1.009	0.604	15	0.592	1.040	0.796	15	0.753	0.704	0.480
16	0.613	0.982	0.736	16	0.607	1.017	0.883	16	0.753	0.718	0.417
17	0.612	0.989	0.736	17	0.618	1.012	0.844	17	0.751	0.735	0.347
18	0.606	1.012	0.643	18	0.608	1.014	0.874	18	0.748	0.758	0.327
19	0.638	0.981	0.714	19	0.623	1.023	0.957	19	0.747	0.804	0.293
20	0.632	0.977	0.659	20	0.627	0.999	0.916	20	0.748	0.807	0.431
21	0.635	0.982	0.651	21	0.629	0.998	0.785	21	0.746	0.815	0.241
22	0.627	0.990	0.728	22	0.629	0.997	0.951	22	0.746	0.821	0.244
23	0.627	0.987	0.534	23	0.628	0.997	0.746	23	0.745	0.827	0.315
24	0.629	0.989	0.608	24	0.629	0.995	0.723	24	0.746	0.829	0.207
25	0.630	0.989	0.631	25	0.629	0.995	0.849	25	0.745	0.833	0.175
26	0.628	1.004	0.624	26	0.629	0.995	0.795	26	0.744	0.845	0.264
27	0.623	1.018	0.524	27	0.630	0.995	0.671	27	0.745	0.849	0.159
28	0.620	1.012	0.711	28	0.631	0.995	0.863	28	0.745	0.859	0.134
29	0.624	1.008	0.683	29	0.640	1.009	0.792	29	0.744	0.884	0.235

simMedT				simFull				simFullT			
Epoch	accuracy	loss (v)	loss (t)	NumIters	accuracy	loss (v)	loss (t)	NumIters	accuracy	loss (v)	loss (t)
0	0.610	1.008	1.234	0	0.765	0.626	0.800	0	0.758	0.662	0.870
1	0.715	0.794	0.784	1	0.777	0.599	0.682	1	0.766	0.635	0.713
2	0.694	0.862	0.644	2	0.774	0.611	0.651	2	0.768	0.632	0.700
3	0.726	0.790	0.725	3	0.777	0.599	0.641	3	0.770	0.623	0.683
4	0.726	0.800	0.777	4	0.782	0.587	0.568	4	0.770	0.621	0.636
5	0.731	0.785	0.806	5	0.780	0.586	0.580	5	0.771	0.617	0.616
6	0.732	0.766	0.613	6	0.782	0.591	0.589	6	0.774	0.615	0.619
7	0.739	0.740	0.694	7	0.786	0.576	0.539	7	0.773	0.614	0.590
8	0.731	0.808	0.582	8	0.787	0.577	0.517	8	0.773	0.615	0.568
9	0.746	0.687	0.687	9	0.799	0.540	0.540	9	0.780	0.596	0.611
10	0.748	0.683	0.608	10	0.804	0.530	0.486	10	0.783	0.586	0.539
11	0.745	0.685	0.495	11	0.803	0.531	0.508	11	0.783	0.586	0.566
12	0.744	0.692	0.492	12	0.802	0.535	0.505	12	0.783	0.586	0.599
13	0.746	0.700	0.504	13	0.802	0.540	0.476	13	0.783	0.589	0.558
14	0.742	0.706	0.452	14	0.800	0.547	0.429	14	0.782	0.591	0.565
15	0.740	0.719	0.506	15	0.798	0.555	0.437	15	0.781	0.596	0.561
16	0.738	0.727	0.492	16	0.797	0.566	0.446	16	0.780	0.601	0.514
17	0.738	0.738	0.339	17	0.794	0.581	0.424	17	0.779	0.606	0.520
18	0.734	0.752	0.402	18	0.790	0.603	0.399	18	0.777	0.613	0.495
19	0.735	0.777	0.366	19	0.791	0.598	0.376	19	0.778	0.612	0.502
20	0.735	0.777	0.462	20	0.791	0.608	0.340	20	0.777	0.616	0.456
21	0.735	0.781	0.363	21	0.790	0.617	0.332	21	0.776	0.620	0.424
22	0.735	0.783	0.351	22	0.789	0.627	0.361	22	0.775	0.624	0.474
23	0.734	0.787	0.407	23	0.788	0.634	0.328	23	0.775	0.628	0.424
24	0.735	0.789	0.309	24	0.788	0.644	0.306	24	0.775	0.631	0.398
25	0.734	0.796	0.350	25	0.786	0.652	0.294	25	0.774	0.635	0.395
26	0.734	0.799	0.282	26	0.785	0.660	0.337	26	0.774	0.637	0.431
27	0.734	0.800	0.263	27	0.785	0.670	0.319	27	0.773	0.642	0.435
28	0.732	0.804	0.309	28	0.784	0.680	0.291	28	0.772	0.647	0.408
29	0.733	0.814	0.321	29	0.782	0.690	0.274	29	0.771	0.652	0.405

Table 2: The results of the validation procedures during training, shown on the plots 23 and 24. “std” stands for “standard NOvA’s CNN”, “sim” stands for “simplified NOvA’s CNN”; “T” stands for “transformed dataset”; “Small”, “Med” and “Full” mean the small, medium and full datasets. “Epoch” is the number of the epochs passed during iterations, “accuracy” and “loss” stand for the validation accuracy and loss: (v) is testing and (t) is training loss

9.3 Listings with code

Listing 1. Three layers of a neural network: convolution, ReLU and pooling

```
layer {
  name: "conv1/7x7_s2_x"
  type: "Convolution"
  bottom: "data_x"
  top: "conv1/7x7_s2_x"
  param {
    name: "conv1/7x7_s2_w"
    lr_mult: 1
    decay_mult: 1
  }
  param {
    name: "conv1/7x7_s2_b"
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 64
    pad: 3
    kernel_size: 7
    stride: 2
    weight_filler { type: "xavier" }
    bias_filler {
      type: "constant"
      value: 0.2
    }
  }
}
layer {
  name: "conv1/relu_7x7_x"
  type: "ReLU"
  bottom: "conv1/7x7_s2_x"
  top: "conv1/7x7_s2_x"
}
layer {
  name: "pool1/3x3_s2_x"
  type: "Pooling"
  bottom: "conv1/7x7_s2_x"
  top: "pool1/3x3_s2_x"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
```

Listing 2. The function of the polar transformation

```
def polarTransformFunc(im_i):


    ##########################
    # Track Reconstruction #
    ##########################


    ### Initialization

    im = im_i

    # Get dimensions
    width = im.shape[1]
    height = im.shape[0]

    # Cut unnecessary far interactions (track reconstruction only)
    im_tracking = np.delete(im, np.s_[width/2:width], axis=1)

    # Find nonblack pixels (nonzero elements)
    x_arr = np.nonzero(im_tracking)[1]
    y_arr = np.nonzero(im_tracking)[0]

    # Define initial parameters for fitting
```

```
x0 = np.array([0, height/2])

# Define fitting function (linear function k*x+b)
def fun(x, t, y):
    return x[0]*t + x[1] - y


### Fitter

def fitFunc(x_arr_i, y_arr_i, fun, x0, mul):

    # Use least_squares from scipy.optimize with 'cauchy' (log) robustness
    fit = sp.optimize.least_squares(fun, x0, loss='cauchy', f_scale=1,
            args=(x_arr_i, y_arr_i))

    # Get k and b (see def func)
    k = np.asscalar(fit.x[0])
    b = np.asscalar(fit.x[1])

    #k = 0.1
    #b = 0.1

    # Get residuals and their standard dev
    f_res = fit.fun
    sigma = np.std(f_res)   # squared

    # Cut outliers
    x_arr_o = np.delete(x_arr_i, np.where(f_res > mul * np.sqrt(sigma)))
    y_arr_o = np.delete(y_arr_i, np.where(f_res > mul * np.sqrt(sigma)))

    # Send k, b and image pixels w/o outliers
    return [k, b, x_arr_o, y_arr_o]


### Fit Iterations

mul = 1    # multiplier for mul*sigma cut
num_iters = 2   # number of fitting iterations
x_arr_pass = x_arr
y_arr_pass = y_arr

# Iterations
for iter in range(0, num_iters):
    [k, b, x_arr_pass, y_arr_pass] = fitFunc(x_arr_pass, y_arr_pass,
        fun, x0, mul)


###################
# Find The Vertex #
###################


### Defining The Vertex

# Shift image if b is too big or too small (cosmic background events)
if b > 79:
    deltaX = int(round((b-79)/k))
    im = np.roll(im, deltaX, axis=1)
    im[..., width+deltaX-1:width] = 0
    b = 79
elif b < 0:
    deltaX = int(round(b/k))
    im = np.roll(im, deltaX, axis=1)
```

```
        im[..., width+deltaX-1:width] = 0
        b = 0

    # Initialize vertex coords as (0, b)
    vertexX = 0
    vertexY = int(round(b))

    # p1 and p2 are some points on the line k*x+b, p3 is a pixel being measured
    p1 = np.array([0, b])
    p2 = np.array([1, k+b])
    p3 = np.array([0, 0])

    # Sort arrays of pixels with according to x coordinate
    x_arr_sorted = np.sort(x_arr)
    y_arr_sorted = np.array([x for _,x in sorted(zip(x_arr, y_arr))])

    # Find norm distances from p3 to k*x+b in order to define the vertex point
    i = 0
    while i<len(x_arr_sorted):

        # Check only first 8 x pixels
        if x_arr_sorted[i] >= 12:
            break

        # Pick the coordinates of the pixel
        p3[0] = x_arr_sorted[i]
        p3[1] = y_arr_sorted[i]

        # Check if the pixel is near enough to the line
        if np.linalg.norm(np.cross(p2-p1, p1-p3))/np.linalg.norm(p2-p1) < 0.5:
            vertexX = x_arr_sorted[i]
            vertexY = int(k*x_arr_sorted[i] + b)
            break
        i += 1


    ########################
    # Polar Transformation #
    ########################


    ### Init im_pol And Scalings

    sw = 1  # 1 if 180, 2 if 360

    # If 180 is chosen, x will start running from the vertex point
    if sw == 1:
        x_start = vertexX
        x_arr = np.delete(x_arr, np.where(x_arr <= x_start))
    elif sw == 2:
        x_start = 0

    # Size of the output image
    width_pol = width
    height_pol = height

    # Define max length of a track to scale to width_rad
    maxradius = np.sqrt(width**2 + height**2)

    # Initialize a blank pic
    im_pol = np.zeros((height_pol, width_pol))

    # Find scalings
    rscale = float(width_pol) / maxradius
    tscale = float(height_pol) / (sw*180 + 1)


    ### Transformation

    for y in y_arr:
        dy = y - vertexY     # calculate from the vertex
        for x in x_arr:
            dx = x - vertexX     # calculate from the vertex
            t = ((np.arctan2(dy,dx)*180)/np.pi) + sw*90 # find angle
            r = np.sqrt(dx**2 + dy**2)     # find radius
            t_sc = int(np.floor(t*tscale))
            r_sc = int(np.floor(r*rscale))
            im_pol[t_sc, r_sc] = im[y, x]


    ### Return im_pol

    im_pol = im_pol.astype(np.uint8)
    return im_pol
```