Deep Learning on Imaging Calorimetry

August 2016

Author:
Jayesh Mahapatra

Guided by:
Maurizio Pierini
Jean-Roch Vlimant
Maria Spiropulu

Caltech CMS Group

CERN openlab Summer Student Report 2016

Abstract

Particle reconstruction is a significant task in analysis of CMS data. Currently physics based algorithms in conjunction with CMS readouts are used. In this project we explored deep learning as a technique for photon identification and reconstruction on the public LCD CaloImage dataset. Several network architectures were tested for performing classification (photon vs pion discrimination) and regression analysis of energy of particle hits. The performance of the topologies were plotted and compared.

Table of Contents

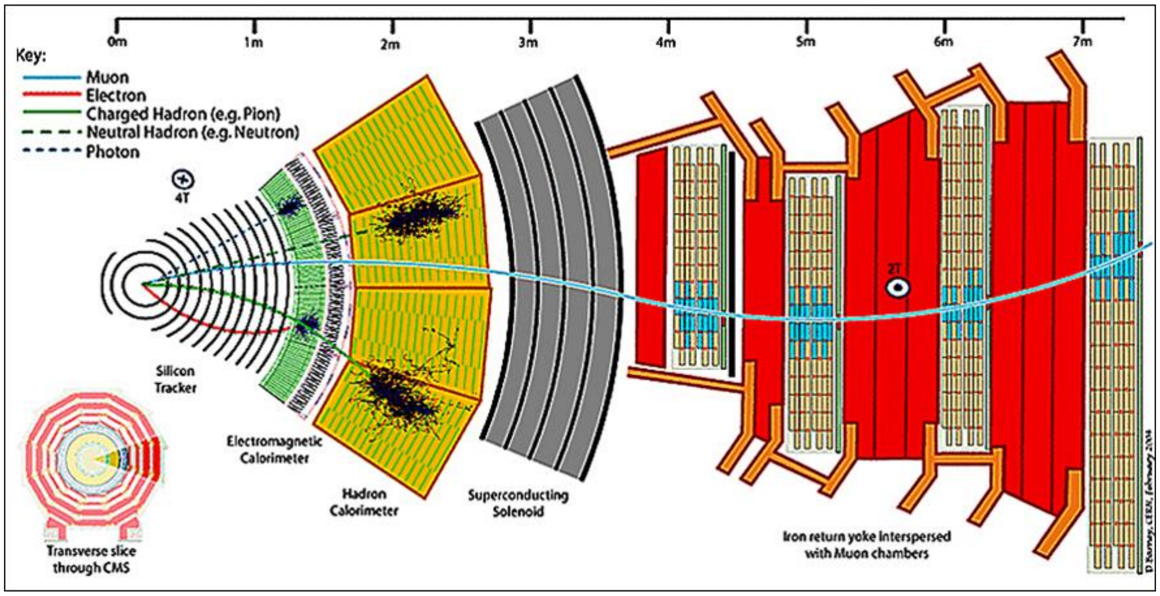
1. Introduction

The Large hadron collider accelerates particles close to the speed of light and smashes them together 40 million times a second. Some of these collisions are very energetic and head on. When this happens, some of the energy of the collision is turned into mass and previously unobserved, short-lived particles. [1]

CMS (Compact Muon Solenoid) is a particle detector which captures data about these collisions using various layers of detectors. The various layers of CMS detector are:-

1. A high quality central tracking system
2. Electromagnetic Calorimeter
3. Hadron Calorimeter
4. A high performance system to detect and measure muons

A representation of a slice of the CMS detector


0m
1m
2m
3m
4m
5m
6m
7m
Key:
Muon
Electron
Charged Hadron (e.g. Pion)
Neutral Hadron (e.g. Neutron)
Photon
4T
2T
Silicon Tracker
Electromagnetic Calorimeter
Hadron Calorimeter
Superconducting Solenoid
Iron return yoke interspersed with Muon chambers
Transverse slice through CMS

Source: - CMS Document 4172-v2

The data from all of the above mentioned detectors is filtered and only the data from interesting physics events are kept for further processing. Analysis is then carried out on this set of filtered data. One of the most significant tasks of this analysis is particle identification and reconstruction i.e. identifying which types of particles were formed and their energy

The data generated by CMS is around 600 megabytes/second and only set to increase in the coming years. This provides a challenge for both the trigger systems as well as particle reconstruction algorithms. One approach to tackle this is automated self-learning algorithms that can detect patterns and perform analysis independently.

We approach this problem from a machine learning perspective.

2. Machine Learning

Machine Learning is a field grown out of field of AI. One of the formal definition of machine learning was given by Arthur Samuel (1959): "Field of study that gives computers the ability to learn without being explicitly programmed".

A more recent definition of a well posed learning problem is given by Tom Mitchell (1998), Well posed learning problem : A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.[2]

Machine learning basically is the study and development of algorithms that learn from data and make a mathematical model, then use that model to make predictions on new unseen data.

Learning problems have been broadly classified into 3 main categories:

1. Supervised Learning: - The machine is presented with a set of inputs and their corresponding outputs, and the goal of the algorithm is to learn a general mapping between inputs and their outputs. This mapping is then applied on unseen inputs to predict their corresponding outputs.
2. Unsupervised Learning: - The machine is not given any outputs/labels. It is just given input data and is expected to find some existing structure in the data.
3. Reinforcement Learning: - The machine interacts with a dynamic environment to perform a certain task (example: - driving a car). It receives feedback from the environment in the form of a reward function and aims to maximize/minimize the reward.

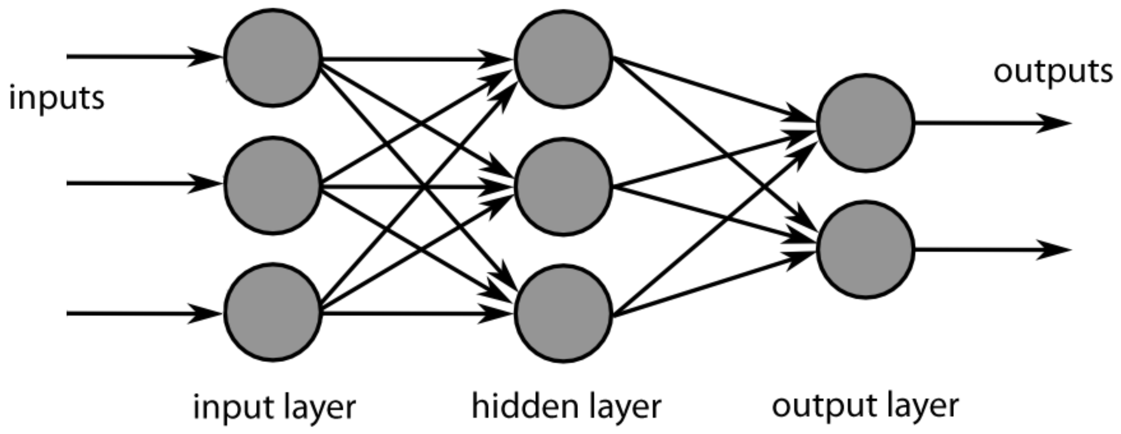
Some common machine learning algorithms are decision trees, support vector machines, artificial neural networks etc.

2.1. Artificial Neural Networks

Artificial Neural Networks are biologically inspired networks that are used to estimate or learn an approximate functions that map a large number of inputs to specified outputs. These are mostly used in the field of supervised learning where the outputs/labels are known.

An artificial neural network is made up of basic units called perceptron. A perceptron takes in multiple inputs and linearly combines the input after multiplying them with their respective weights and then outputs the result. In neural networks, perceptron might have an optional non linearity, which is applied to the result of the linear combination of product of weights and inputs and the final result obtained is the output of that unit.

Depending upon the problem being tackled (regression/classification) and also the depth of the networks, the activation functions are varied to obtain the best possible performance.



Source: - commons.wikimedia.org

2.2. Deep Learning

Deep learning (also known as deep structured learning, hierarchical learning or deep machine learning) is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using a deep graph with multiple processing layers, composed of multiple linear and non-linear transformations. [3]

The quintessential example of deep learning model is Multilayer perceptron (MLP). A multilayer perceptron is just a mathematical function mapping some set of input values to output values. The function is formed by composing many simpler functions. We can think of each application of a different mathematical function as providing a new representation of the input. [3]

Deep learning uses deep neural network to learn hierarchical abstraction in data.

2.3. Convolutional Neural Networks

Normal Neural networks do not scale well to images. The number of parameters increases drastically with increase in size due to full connectivity of neurons in each layer. This makes it computationally expensive and requires a great deal of memory. Further a lot of these connections are wasteful and a huge number of parameters might lead to overfitting.

Convolutional Neural Networks unlike normal neural networks have their neurons arranged in 3 dimensions (width, height and depth). The depth here is used to refer to the depth of activation volume rather than the whole network.

There are 3 main types of layers in a convolutional neural network.

1. Convolutional layer: - This layer consist of a set of learnable filters. Each filter only covers a fraction of the whole image in width and height but extends through the full depth of input volume. During the forward pass we convolve each filter along the input volume and compute the dot product of input volume and filters at any given position. The weights of the filters are shared and hence the layer is also called locally connected layer. There are three hyperparameters for this layer: depth of the output filter, the stride and padding.

2. Pooling Layer: - pooling layers are inserted between convolutional layers to progressively reduce the spatial size of representation to reduce computational time and prevent overfitting. Some common type of pooling are max, average and L2 norm pooling.

3. Fully-Connected Layer: - These are normal dense fully connected layers which are generally situated at the end of a convolutional neural network.

3. Dataset Used

Two sets of data were used for this project:-

1. Old LCD dataset
2. New LCD dataset

These were simulated calorimeter readouts of particle hits (photons and pions). The old LCD dataset contained simulated readouts of Electromagnetic calorimeter (ECAL). The initial data was a calorimeter readout of the whole barrel of calorimeter. A python script was used to find barycenter and extract readings of 20x20x25 cells around it and store it as an 3d array under ‘images’ key in an h5 file. The energy of the particle along with particle type was stored under ‘target’ keys. There were a total of 20 .h5 files for each energy containing 500 events each. These were concatenated and shuffled to form 100 files with shuffled data.

The New LCD dataset had readouts of both the Electromagnetic calorimeter (ECAL) and Hadron calorimeter (HCAL). The ECAL reading script was modified, to find the HCAL cells corresponding to the ECAL shower, using the absolute coordinates of the ECAL barycenter and store their information as well. The ECAL data was stored as 24x24x25 3d array and HCAL data was stored as 4x4x60 3d array in single h5 file with ’ECAL’ and ‘HCAL’ keys respectively. The particle energy, type and momentum information was stored under ‘target’ key in the same h5 file. There were 50 of these files for photons and 50 for pions each containing 10,000 events each. These were concatenated and shuffled to form 100 files with shuffled data.

4. Project workflow

The Project workflow: -

1. Generate simulated calorimeter readouts of particle hits. (Photons and Pions).
2. Process the readouts to find the barycenter of energy deposits at ECAL and take a window of reading around it. Find the coordinates of the corresponding point at HCAL (using absolute coordinates of ECAL barycenter) and take a window of reading of HCAL cells around that point. Store these window of readings as 3d arrays in h5 files along with the target energy and particle type.
3. Merge and shuffle the h5 files.

4. Feed the ECAL and HCAL readings as ‘images’ to convolutional neural networks and train them. The target provided depended upon the type of model we are training. We used three different types of model setups:-
    1. Regression only models: - Tested out various network topologies to perform only regression analysis of the energy of particle hits.
    2. Classification only models: - Tested out various network topologies to perform only classification on the particle type. (Photon or Pion).
    3. Regression and Classification models: - Tested out various network topologies to carry out both regression analysis of the energy of particle hits and perform classification of the particle type. (Photon or Pion).
    4. After training test them out on unseen data and plot the results.

5. Experimental Setup

The initial testing and training of our models was performed on Nvidia GeForce 900 series GTX TitanX graphics processing unit (GPU), built on 28 nm Maxwell architecture, and coupled with the Intel Core i7-5960 X CPU.

After the initial testing specific model topologies were selected and then trained on the CSCS (Swiss National Supercomputing Centre) Piz Daint supercomputer.

The Piz Daint supercomputer is a 28 cabinet Cray XC30 system with a total of 5'272 compute nodes. The compute nodes are equipped with an 8-core 64-bit Intel SandyBridge CPU (Intel® Xeon® E5-2670), an NVIDIA® Tesla® K20X with 6 GB GDDR5 memory, and 32 GB of host memory. The nodes are connected by the "Aries" proprietary interconnect from Cray, with a dragonfly network topology. [4]

For the actual training and testing of neural networks, we used Keras, a minimalist, highly modular neural networks library, written in python and capable of running on top of either TensorFlow or Theano. Keras was used for this project because of its modular nature, easy and fast prototyping and community support. We used Theano as the backend for Keras to train our models.

The datasets used were too large to fit in memory all at once and there was a need to split these datasets into test, train and validation set. To overcome this problem, we used a custom made data generator. Some features of the data generator used: -

1. The data generator is a python class which goes through all of the files and continues yielding batches of specified size indefinitely.

2. It keeps a record of where it left off reading after reading from a file and resumes reading from there for the next batch.

3. The generator also takes care of splitting the data into training, testing and validation set by using different methods in the class, train (), test () and validation () respectively.

4. Keras fit_generator() function is used to train model using this generator.

5. The generator feeds in batches of a specified size until the requirement of samples per epoch is satisfied after which a new epoch begins.

After the training is completed the models are tested on unseen data using the test () method of the generator class. Their performance is plotted using matplotlib library in python.

6. Results

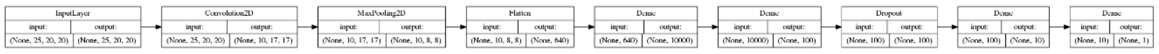
6.1. Energy Regression

Here we tested various topologies to perform regression analysis on the energy of particle hits. The target energy in GeV was provided. The input data was just the ECAL readouts for the old LCD dataset and ECAL+HCAL readouts for the new dataset.

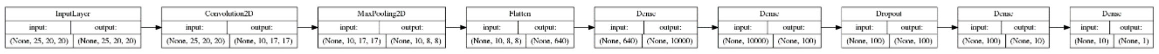
6.1.1. Network architectures

For the old LCD dataset containing just the ECAL readouts we used the following architectures. Since we just had ECAL data in the shape of 20x20x25 3d array, the input dimension of the first layer was same. Most of the topologies were convolutional 2d models with varying levels of dense layer after them. We also varied the activation function, experimenting with Leaky ReLU and Sigmoid. We were facing the dying ReLU problem with ReLU and hence avoided using ReLU for these models.

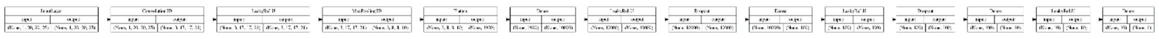
CNN2D_1


InputLayer input: (None, 25, 20, 20) output: (None, 25, 20, 20)
Convolution2D input: (None, 25, 20, 20) output: (None, 10, 17, 17)
MaxPooling2D input: (None, 10, 17, 17) output: (None, 10, 8, 8)
Flatten input: (None, 10, 8, 8) output: (None, 640)
Dense input: (None, 640) output: (None, 10000)
Dense input: (None, 10000) output: (None, 100)
Dropout input: (None, 100) output: (None, 100)
Dense input: (None, 100) output: (None, 10)
Dense input: (None, 10) output: (None, 1)

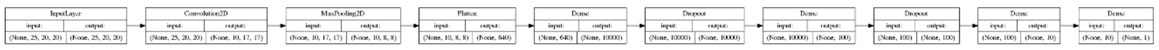
CNN2D_2


InputLayer input: (None, 25, 20, 20) output: (None, 25, 20, 20)
Convolution2D input: (None, 25, 20, 20) output: (None, 10, 17, 17)
MaxPooling2D input: (None, 10, 17, 17) output: (None, 10, 8, 8)
Flatten input: (None, 10, 8, 8) output: (None, 640)
Dense input: (None, 640) output: (None, 10000)
Dense input: (None, 10000) output: (None, 100)
Dropout input: (None, 100) output: (None, 100)
Dense input: (None, 100) output: (None, 10)
Dense input: (None, 10) output: (None, 1)

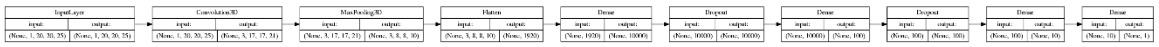
CNN2D_3


InputLayer input: (None, 1, 20, 20, 25) output: (None, 1, 20, 20, 25)
Convolution3D input: (None, 1, 20, 20, 25) output: (None, 3, 17, 17, 21)
LeakyReLU input: (None, 3, 17, 17, 21) output: (None, 3, 17, 17, 21)
MaxPooling3D input: (None, 3, 17, 17, 21) output: (None, 3, 8, 8, 10)
Flatten input: (None, 3, 8, 8, 10) output: (None, 1920)
Dense input: (None, 1920) output: (None, 10000)
LeakyReLU input: (None, 10000) output: (None, 10000)
Dropout input: (None, 10000) output: (None, 10000)
Dense input: (None, 10000) output: (None, 100)
LeakyReLU input: (None, 100) output: (None, 100)
Dropout input: (None, 100) output: (None, 100)
Dense input: (None, 100) output: (None, 10)
LeakyReLU input: (None, 10) output: (None, 10)
Dense input: (None, 10) output: (None, 1)

CNN2D_4

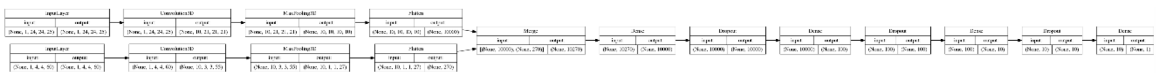

InputLayer input: (None, 25, 20, 20) output: (None, 25, 20, 20)
Convolution2D input: (None, 25, 20, 20) output: (None, 10, 17, 17)
MaxPooling2D input: (None, 10, 17, 17) output: (None, 10, 8, 8)
Flatten input: (None, 10, 8, 8) output: (None, 640)
Dense input: (None, 640) output: (None, 10000)
Dropout input: (None, 10000) output: (None, 10000)
Dense input: (None, 10000) output: (None, 100)
Dropout input: (None, 100) output: (None, 100)
Dense input: (None, 100) output: (None, 10)
Dense input: (None, 10) output: (None, 1)

CNN3D_1

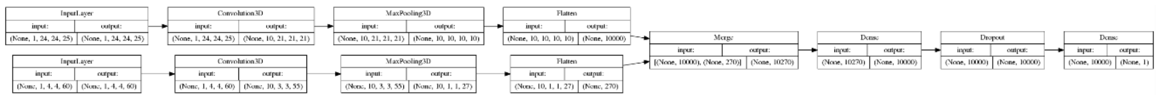

InputLayer input: (None, 1, 20, 20, 25) output: (None, 1, 20, 20, 25)
Convolution3D input: (None, 1, 20, 20, 25) output: (None, 3, 17, 17, 21)
MaxPooling3D input: (None, 3, 17, 17, 21) output: (None, 3, 8, 8, 10)
Flatten input: (None, 3, 8, 8, 10) output: (None, 1920)
Dense input: (None, 1920) output: (None, 10000)
Dropout input: (None, 10000) output: (None, 10000)
Dense input: (None, 10000) output: (None, 100)
Dropout input: (None, 100) output: (None, 100)
Dense input: (None, 100) output: (None, 10)
Dense input: (None, 10) output: (None, 1)

For the new LCD dataset, we had readouts of both the ECAL and HCAL. The ECAL readouts were of the shape 24x24x25 and HCAL were of the shape 4x4x60. Since the shape of both the readouts were different, we decided to use branched convolutional neural networks. The ECAL and HCAL data were fed into different input branches of the same neural network. The number of dense layers and convolutional layers were varied to get different topologies. We also tested one dense network to benchmark the performance of the convolutional networks. We used linear as the activation function for the output neuron in the last layer.

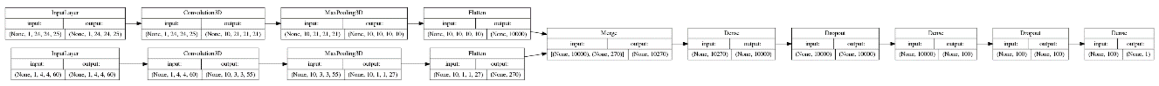
BCNN1_REG



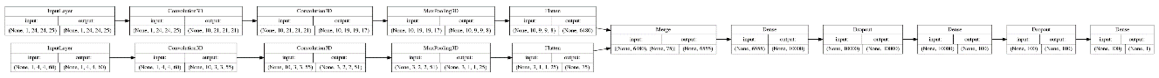
BCNN2_REG


InputLayer
input: | output:
(None, 1, 24, 24, 25) | (None, 1, 24, 24, 25)
Convolution3D
input: | output:
(None, 1, 24, 24, 25) | (None, 10, 21, 21, 21)
MaxPooling3D
input: | output:
(None, 10, 21, 21, 21) | (None, 10, 10, 10, 10)
Flatten
input: | output:
(None, 10, 10, 10, 10) | (None, 10000)
InputLayer
input: | output:
(None, 1, 4, 4, 60) | (None, 1, 4, 4, 60)
Convolution3D
input: | output:
(None, 1, 4, 4, 60) | (None, 10, 3, 3, 55)
MaxPooling3D
input: | output:
(None, 10, 3, 3, 55) | (None, 10, 1, 1, 27)
Flatten
input: | output:
(None, 10, 1, 1, 27) | (None, 270)
Merge
input: | output:
[(None, 10000), (None, 270)] | (None, 10270)
Dense
input: | output:
(None, 10270) | (None, 10000)
Dropout
input: | output:
(None, 10000) | (None, 10000)
Dense
input: | output:
(None, 10000) | (None, 1)

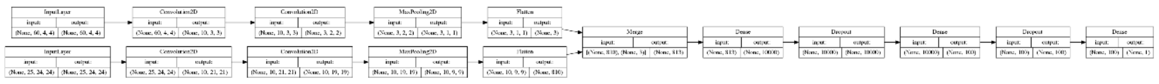
BCNN3_REG



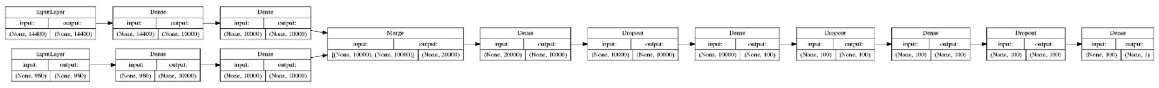
BCNN4_REG



BCNN5_REG



DENSE1_REG



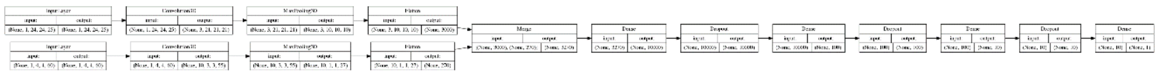
6.2. Particle Classification

Here we tested various topologies to perform regression analysis on the energy of particle hits. The target was the particle type. This was a binary classification problem. We performed classification on the new LCD dataset with both ECAL and HCAL readouts.

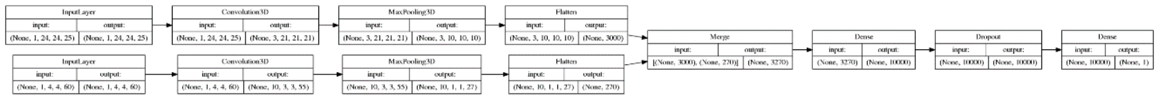
6.2.1. Network architectures

For the new LCD dataset, we had readouts of both the ECAL and HCAL. The ECAL readouts were of the shape 24x24x25 and HCAL were of the shape 4x4x60. Since the shape of both the readouts were different, we decided to use branched convolutional neural networks. The ECAL and HCAL data were fed into different input branches of the same neural network. The number of dense layers and convolutional layers were varied to get different topologies. We also tested one dense network to benchmark the performance of the convolutional networks. We used sigmoid as the activation function for the output neuron in the last layer.

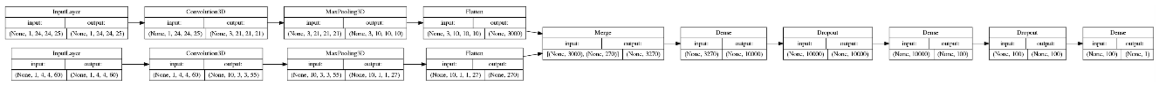
BCNN1_CL



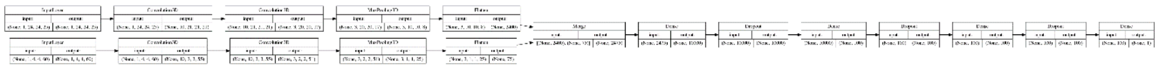
BCNN2_CL


InputLayer
input: (None, 1, 24, 24, 25)
output: (None, 1, 24, 24, 25)
Convolution3D
input: (None, 1, 24, 24, 25)
output: (None, 3, 21, 21, 21)
MaxPooling3D
input: (None, 3, 21, 21, 21)
output: (None, 3, 10, 10, 10)
Flatten
input: (None, 3, 10, 10, 10)
output: (None, 3000)
InputLayer
input: (None, 1, 4, 4, 60)
output: (None, 1, 4, 4, 60)
Convolution3D
input: (None, 1, 4, 4, 60)
output: (None, 10, 3, 3, 55)
MaxPooling3D
input: (None, 10, 3, 3, 55)
output: (None, 10, 1, 1, 27)
Flatten
input: (None, 10, 1, 1, 27)
output: (None, 270)
Merge
input: [(None, 3000), (None, 270)]
output: (None, 3270)
Dense
input: (None, 3270)
output: (None, 10000)
Dropout
input: (None, 10000)
output: (None, 10000)
Dense
input: (None, 10000)
output: (None, 1)

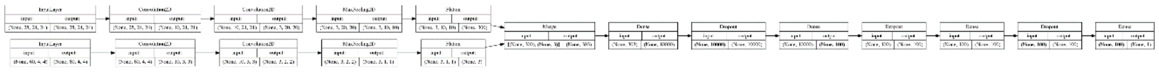
BCNN3_CL



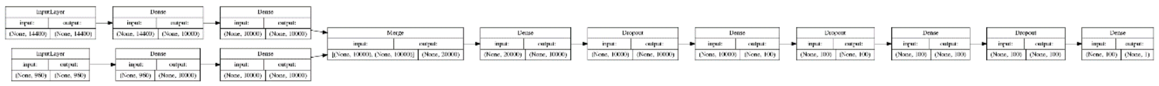
BCNN4_CL



BCNN5_CL



DENSE1_CL



6.3. Classification and Regression

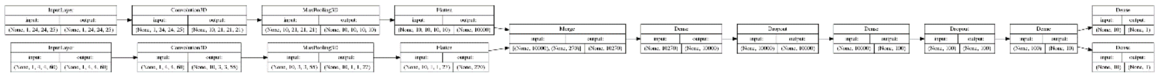
Here we tested various topologies to perform both regression analysis on the energy of particle hits and classification of particle into photon or pion. The target was the particle type and energy of particle hit. We used the new LCD dataset with both ECAL and HCAL readouts for this.

6.3.1. Network architectures

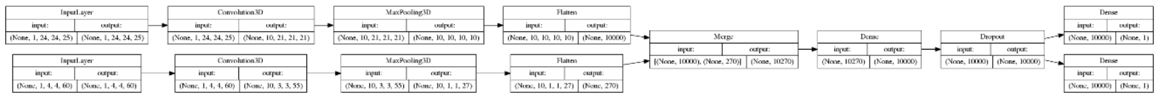
For the new LCD dataset, we had readouts of both the ECAL and HCAL. The ECAL readouts were of the shape 24x24x25 and HCAL were of the shape 4x4x60. Since the shape of both the readouts were different, we decided to use branched convolutional neural networks. The ECAL and HCAL data were fed into different input branches of the same neural network. The number of dense layers and convolutional layers were varied to get different topologies. We also tested one dense network to benchmark the performance of the convolutional networks.

Further since the networks performed both classification and regression they had two output neurons at the last layer, one for classification and one for regression. The activation functions were linear for the neuron performing regression and sigmoid for the neuron performing classification.

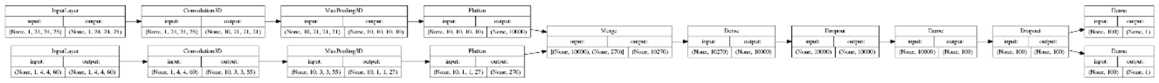
BCNN1_REGCL



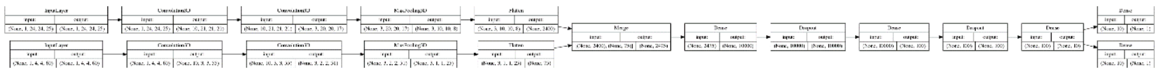
BCNN2_REGCL


InputLayer
input: | output:
(None, 1, 24, 24, 25) | (None, 1, 24, 24, 25)
Convolution3D
input: | output:
(None, 1, 24, 24, 25) | (None, 10, 21, 21, 21)
MaxPooling3D
input: | output:
(None, 10, 21, 21, 21) | (None, 10, 10, 10, 10)
Flatten
input: | output:
(None, 10, 10, 10, 10) | (None, 10000)
InputLayer
input: | output:
(None, 1, 4, 4, 60) | (None, 1, 4, 4, 60)
Convolution3D
input: | output:
(None, 1, 4, 4, 60) | (None, 10, 3, 3, 55)
MaxPooling3D
input: | output:
(None, 10, 3, 3, 55) | (None, 10, 1, 1, 27)
Flatten
input: | output:
(None, 10, 1, 1, 27) | (None, 270)
Merge
input: | output:
[(None, 10000), (None, 270)] | (None, 10270)
Dense
input: | output:
(None, 10270) | (None, 10000)
Dropout
input: | output:
(None, 10000) | (None, 10000)
Dense
input: | output:
(None, 10000) | (None, 1)
Dense
input: | output:
(None, 10000) | (None, 1)

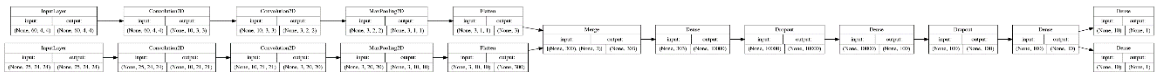
BCNN3_REGCL



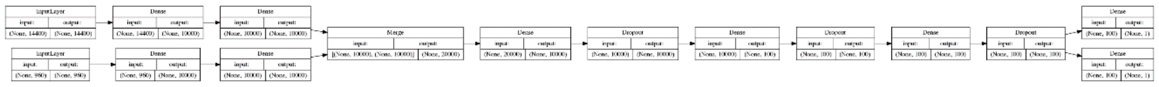
BCNN4_REGCL



BCNN5_REGCL



DENSE1_REGCL

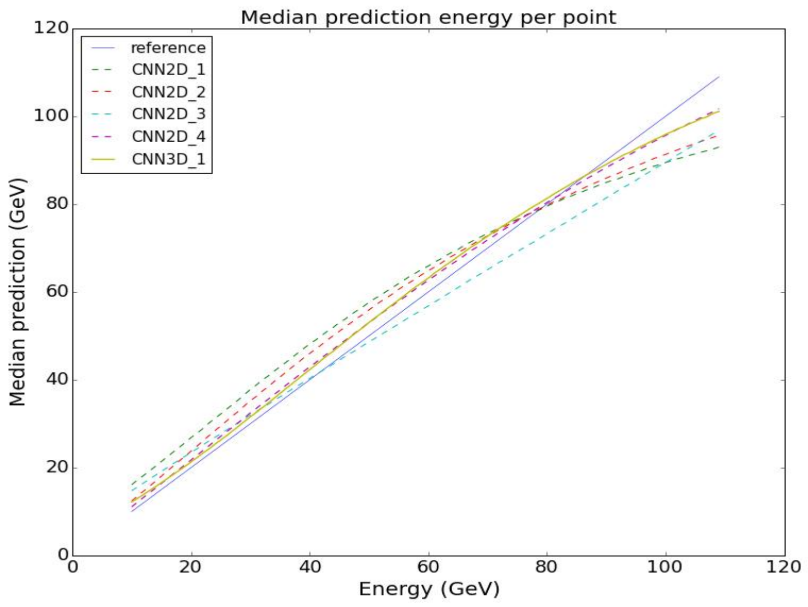


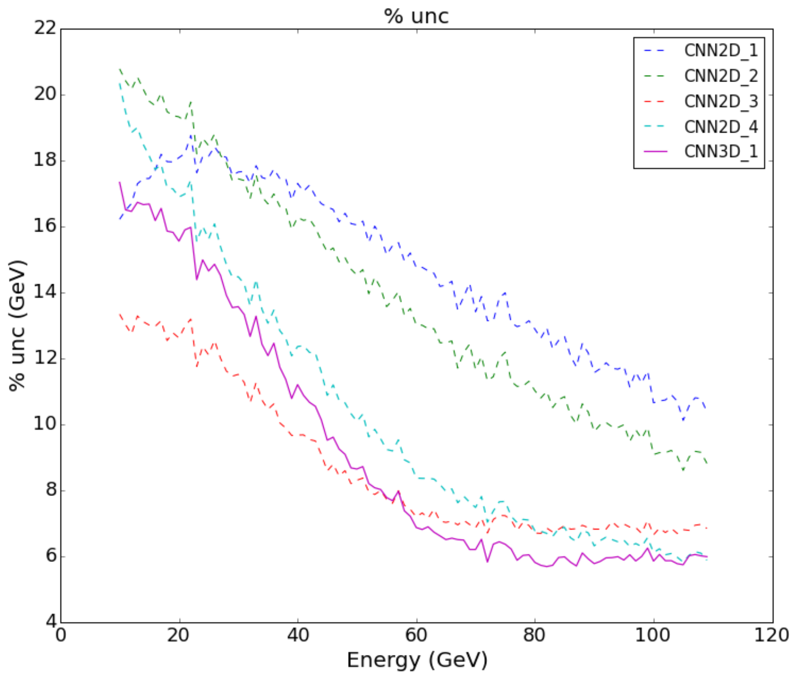
6.4. Results

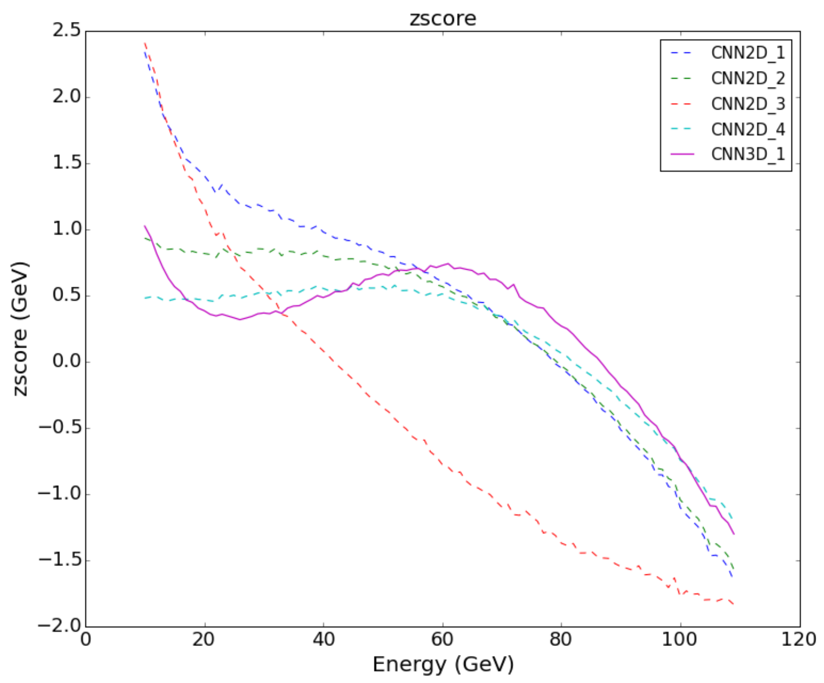
6.4.1. Energy Regression

The performance of trained neural networks was tested on unseen test data.

For the old LCD dataset (containing only ECAL data):-

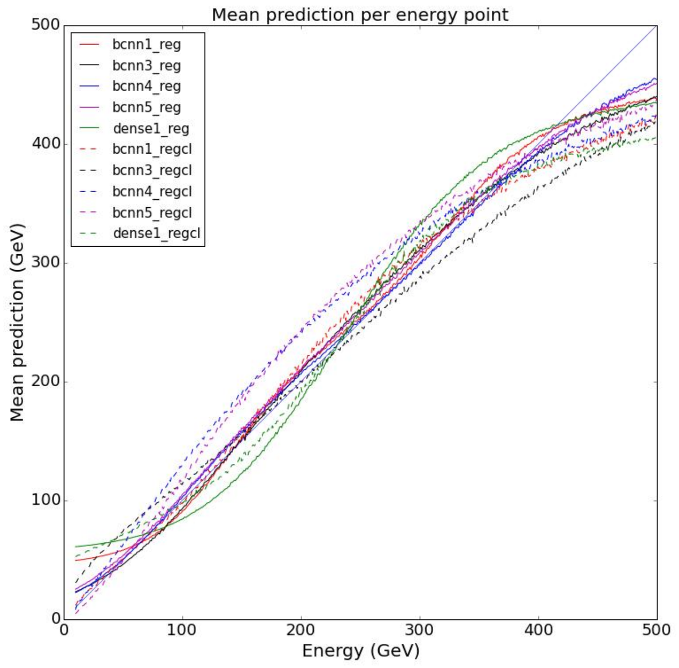

Median prediction energy per point
Median prediction (GeV)
Energy (GeV)
reference
CNN2D_1
CNN2D_2
CNN2D_3
CNN2D_4
CNN3D_1
0
20
40
60
80
100
120

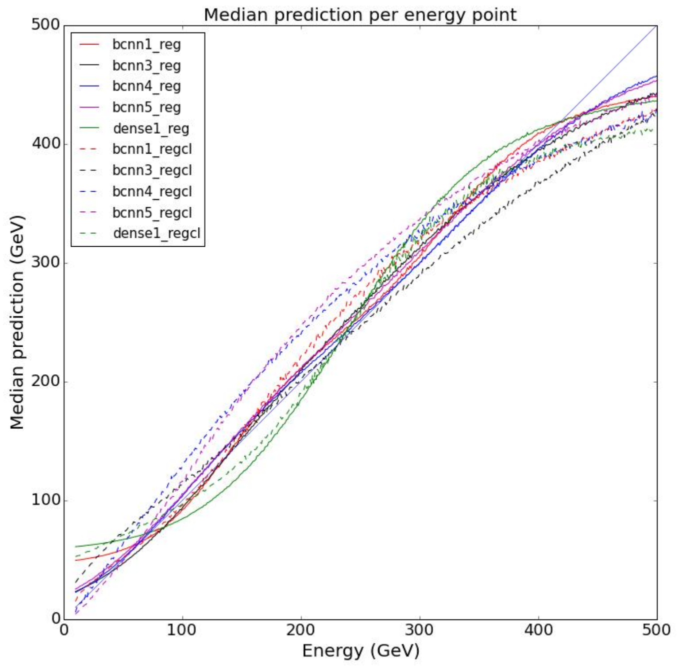

% unc
% unc (GeV)
Energy (GeV)
CNN2D_1
CNN2D_2
CNN2D_3
CNN2D_4
CNN3D_1
4
6
8
10
12
14
16
18
20
22
0
20
40
60
80
100
120

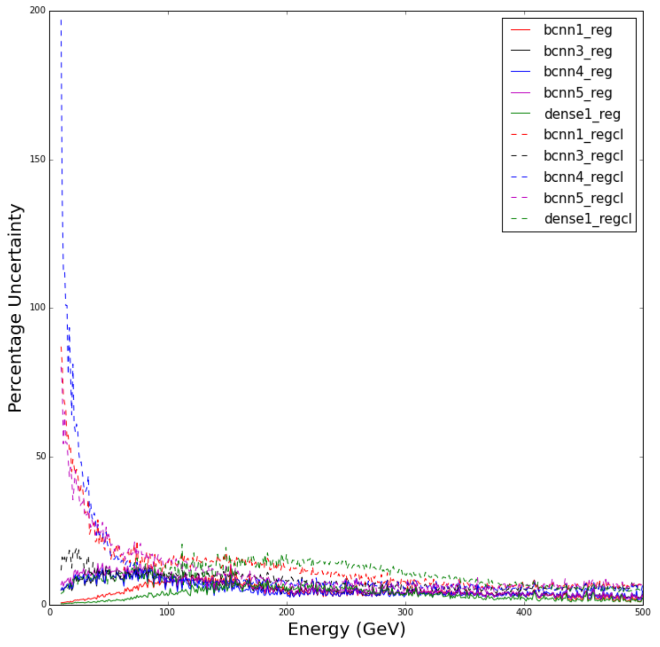


We can clearly see that in case of the old dataset cnn3d which is a convolutional 3d neural network outperforms most of the 2d convolutional networks.

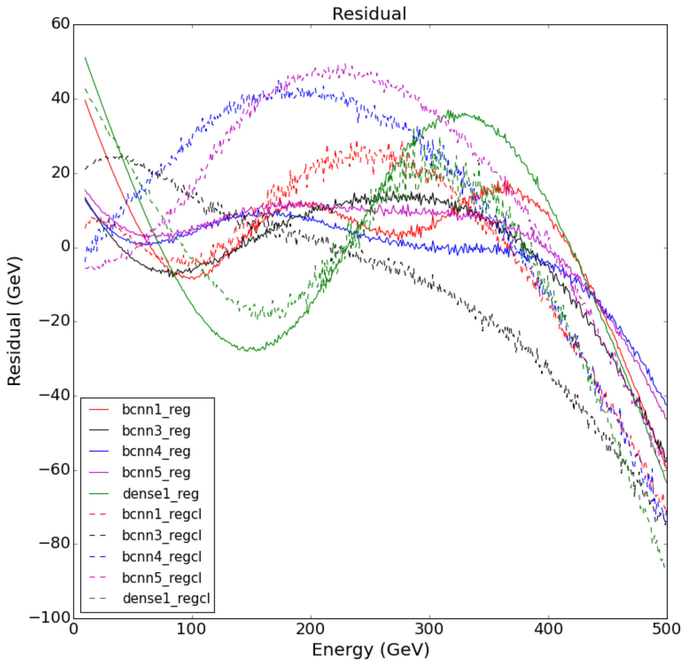
For the new LCD dataset containing both ECAL and HCAL:-


Mean prediction per energy point
bcnn1_reg
bcnn3_reg
bcnn4_reg
bcnn5_reg
dense1_reg
bcnn1_regcl
bcnn3_regcl
bcnn4_regcl
bcnn5_regcl
dense1_regcl
Mean prediction (GeV)
Energy (GeV)
0
100
200
300
400
500

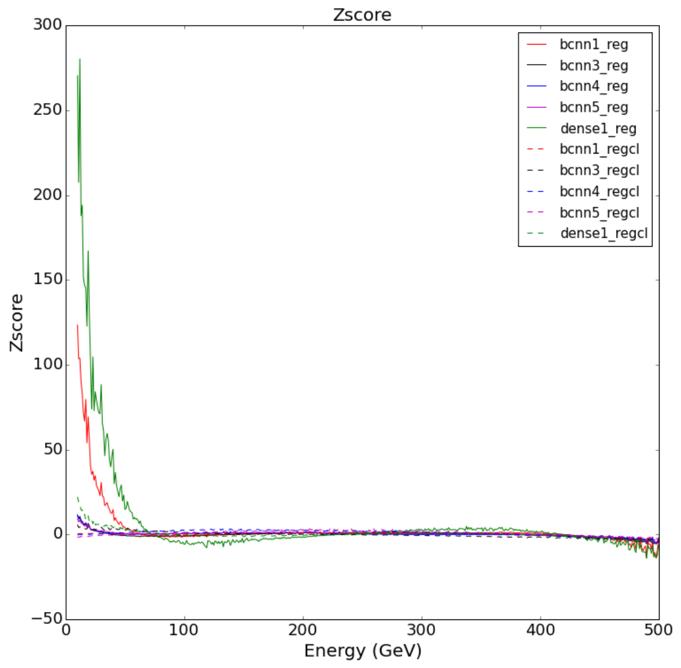

Median prediction per energy point
bcnn1_reg
bcnn3_reg
bcnn4_reg
bcnn5_reg
dense1_reg
bcnn1_regcl
bcnn3_regcl
bcnn4_regcl
bcnn5_regcl
dense1_regcl
Median prediction (GeV)
Energy (GeV)
0
100
200
300
400
500



Percentage Uncertainty
Energy (GeV)
bcnn1_reg
bcnn3_reg
bcnn4_reg
bcnn5_reg
dense1_reg
bcnn1_regcl
bcnn3_regcl
bcnn4_regcl
bcnn5_regcl
dense1_regcl

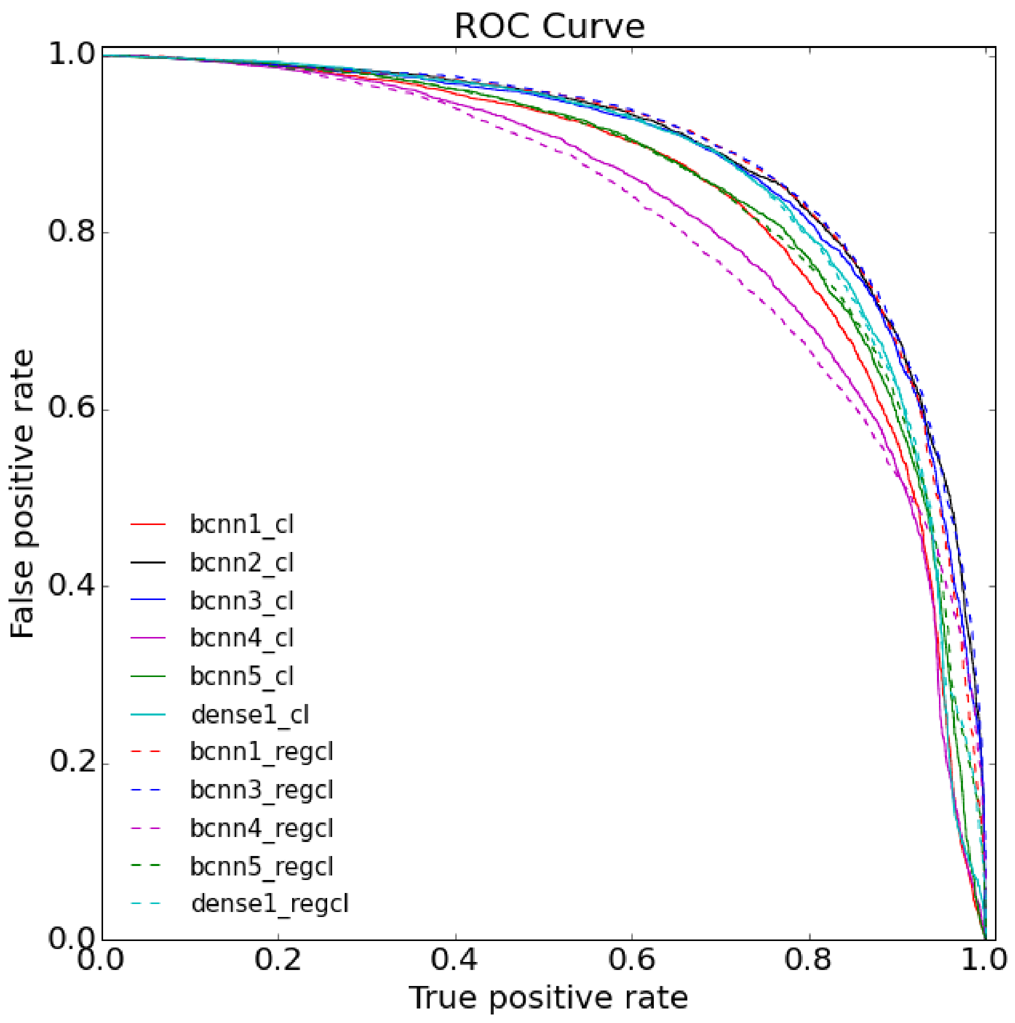


Residual
Residual (GeV)
Energy (GeV)
bcnn1_reg
bcnn3_reg
bcnn4_reg
bcnn5_reg
dense1_reg
bcnn1_regcl
bcnn3_regcl
bcnn4_regcl
bcnn5_regcl
dense1_regcl



The convolutional 3d networks are performing better than the 2d convolutional networks and simple dense neural networks. The performance of neural networks degrades below 100 GeV and while approaching 500 GeV.

6.4.2. Classification



The highest signal to noise ratio we obtained across all energies was 80:20. The performance might vary along different energies.

7. Future Work

The future scope of this project is as follows: -

1. Trying out new type of branched networks using up- sampling/down-sampling to enable branches to see the whole of ECAL and HCAL data at once instead of at separate branches.
2. Hyper-parameter optimization for improved networks which perform well in a wider range of energies.

3. Publication: -
    a. The dataset will be released to public under the opendata initiative
    b. Publication of our results on applying machine learning to this dataset
4. Classification of charged particles and jets.

References

1. http://cms.web.cern.ch/news/what-cms
2. Machine Learning, Tom Mitchell, McGraw Hill, 1997
3. Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016). Deep Learning. MIT Press
4. http://www.cscs.ch/computers/piz_daint_piz_dora/index.html