Using Machine Learning Techniques for Data Quality Monitoring at CMS Experiment

by

Guillermo A. Fidalgo Rodríguez

A thesis presented for the degree of

BACHELLOR'S OF SCIENCE??

in

Physics

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS
2018

Approved by:

Sudhir Malik, Ph.D. President, Graduate Committee	Date
Héctor Méndez, Ph.D. Member, Graduate Committee	Date
Samuel Santana Colón, Ph.D. Member, Graduate Committee	Date
Rafael A. Rámos, Ph.D. Chairperson of the Department	Date

Abstract

The Data Quality Monitoring (DQM) of CMS is a key asset to deliver high-quality data for physics analysis and it is used both in the online and offline environment. The current paradigm of the quality assessment is labor intensive and it is based on the scrutiny of a large number of histograms by detector experts comparing them with a reference. This project aims at applying recent progress in Machine Learning techniques to the automation of the DQM scrutiny. In particular the use of convolutional neural networks to spot problems in the acquired data is presented with particular attention to semi-supervised models (e.g. autoencoders) to define a classification strategy that doesn't assume previous knowledge of failure modes. Real data from the hadron calorimeter of CMS are used to demonstrate the effectiveness of the proposed approach.

Keywords: [DQM, online, offline, Machine Learning]

Acknowledgments

I wish to thank United States State Department and University of Michigan for providing the opportunity to work abroad at CERN during the 2018 Winter Semester. I also wish to thank CERN staff, CMS Experiment , Texas Tech University, and the University of Puerto Rico at Mayagüez, with special thanks to Dr. Federico de Guio for his local mentorship and Dr. Nural Akchurin, and Dr. Sudhir Malik for their guidance. A very special thanks to Dr. Jean Krisch for accepting me for this great research opportunity and Dr. Steven Goldfarb for being a wonderful host and overall local guidance at CERN.

List of Figures

Contents

Chapter 1

Introduction

The work for this thesis was performed at CERN on CMS Experiment. CERN stands for European Organization for Nuclear Research. It was founded in 1954 and is located at the Franco-Swiss border near Geneva. At CERN, physicists and engineers are probing the fundamental structure of the universe. They use the world's largest and most complex scientific instruments to study the basic constituents of matter – the fundamental particles. The instruments used at CERN are purpose-built particle accelerators and detectors. Accelerators boost beams of particles to high energies before the beams are made to collide with each other or with stationary targets. Detectors observe and record the results of these collisions. The accelerator at CERN is called the Large Hadron Collider (LHC), the largest machine ever built by humans and it collides particles (protons) at close to the speed of light. The process gives the physicists clues about how the particles interact, and provides insights into the fundamental laws of nature. Seven experiments at the LHC use detectors to analyze particles produced by proton-proton collisions. The biggest of these experiments, ATLAS and CMS, use general-purpose detectors designed to study the fundamental nature of matter and fundamental forces and to look for new physics or evidence of particles that are beyond the Standard Model. Having two independently designed detectors is vital for cross-confirmation of any new discoveries made. The other two major detectors ALICE and LHCb, respectively, study a state of matter that was present just moments after the Big Bang and preponderance of matter than antimatter. Each experi-

ment does important research that is key to understanding the universe that surrounds and makes us.

Chapter 2 presents a basic description of the Large Hadron Collider and CMS Detector

Chapter 3 gives a brief description of what is Data Quality Monitoring and what it's importance for this experiment.

Chapter 4 is dedicated to describing the idea of Machine Learning and how to implement this tool for this project.

Chapter 5 Shows the results of this project.

Chapter 2

The CMS Experiment

The Compact Muon Solenoid (CMS) detector is a general purpose particle detector designed to investigate various physical phenomena concerning the SM and beyond it, such as Supersymmetry, Extra Dimensions and Dark Matter. As its name implies, the detector is a solenoid which is constructed around a superconducting magnet capable of producing a magnetic field of 3.8 T. The magnetic coil is 13m long with an inner diameter of 6m, making it the largest superconducting magnet ever constructed. The CMS detector itself is 21m long with a diameter of 15m and it has a weight of approximately 14,000 tons. The CMS experiment is one of the largest scientific collaborations in the history of mankind with over 4,000 participants from 42 countries and 182 institutions. CMS is located at one of these points and it essentially acts as a giant super highspeed camera that makes 3D images of the collisions that are produced at a rate of 40 MHz (40 million times per second). The detector has an onion-like structure to capture all the particles that are produced in these high energy collisions most of them being unstable and decaying further to stable particles that are detected. CMS detector was designed with the following features (as shown in Figure 2.1) :

1. A **magnet** with large bending power and high performance muon detector for good muon identification and momentum resolution over a wide range of momenta and angles.

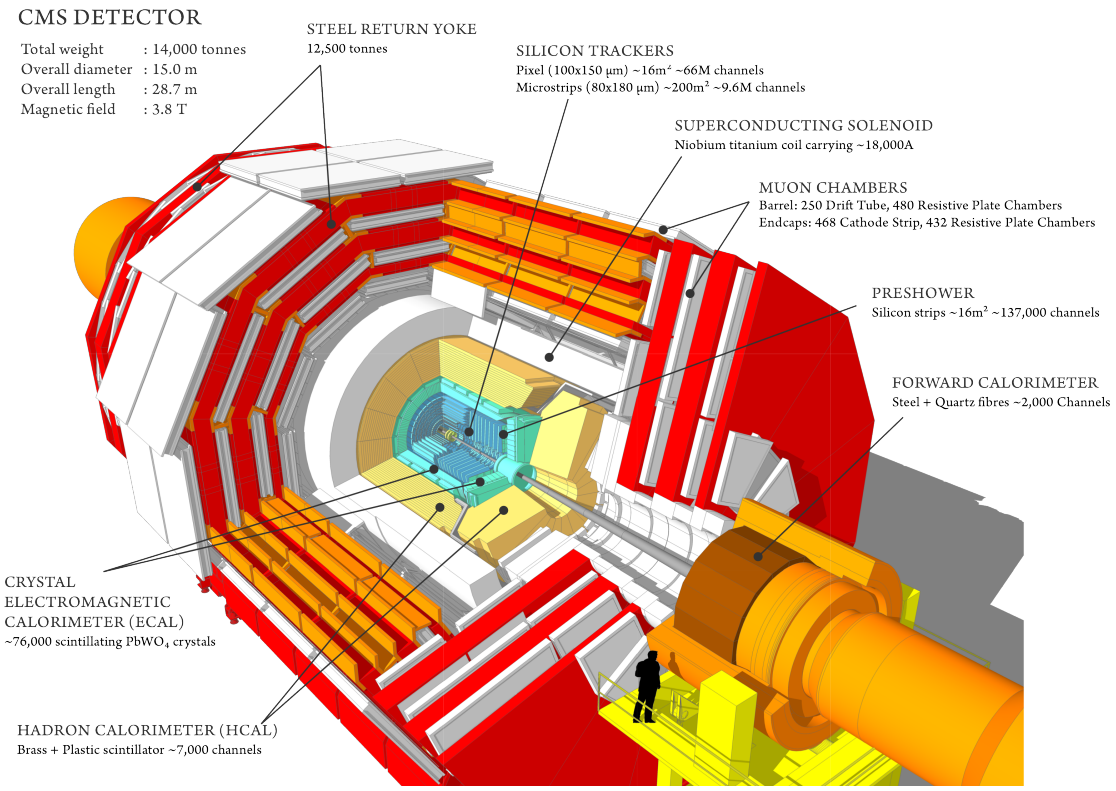


Figure 2.1: CMS Detector

2. An **inner tracking system** capable of high reconstruction efficiency and momentum resolution requiring **pixel detectors** close to the interaction region.

3. An **electromagnetic calorimeter** able to provide good electromagnetic energy resolution and a high isolation efficiency for photons and leptons.

4. A **hadron calorimeter** capable of providing precise missing-transverse-energy and dijet-mass resolution.

A property from these particles that is exploited is their charge. Normally, particles produced in collisions travel in a straight line, but in the presence of a magnetic field, their paths are skewed and curved. Except the muon system, the rest of the subdetectors lie inside a 3.8 Tesla magnetic field . Due to the magnetic field the trajectory of charged particle produced in the collisions gets curved (as shown in Figure 2.2) and one can calculate the particle's momentum and know the type of charge on the particle. The Tracking devices are responsible for drawing the trajectory of the particles by using a computer program that reconstructs the path by using electrical signals that are left by

the particle as they move. The Calorimeters measure the energy of particles that pass through them by absorbing their energy with the intent of stopping them. The particle identification detectors work by detecting radiation emitted by charged particles and using this information they can measure the speed, momentum, and mass of a particle. After the information is put together to make the "snapshot" of the collision one looks for results that do not fit the current theories or models in order to look for new physics.

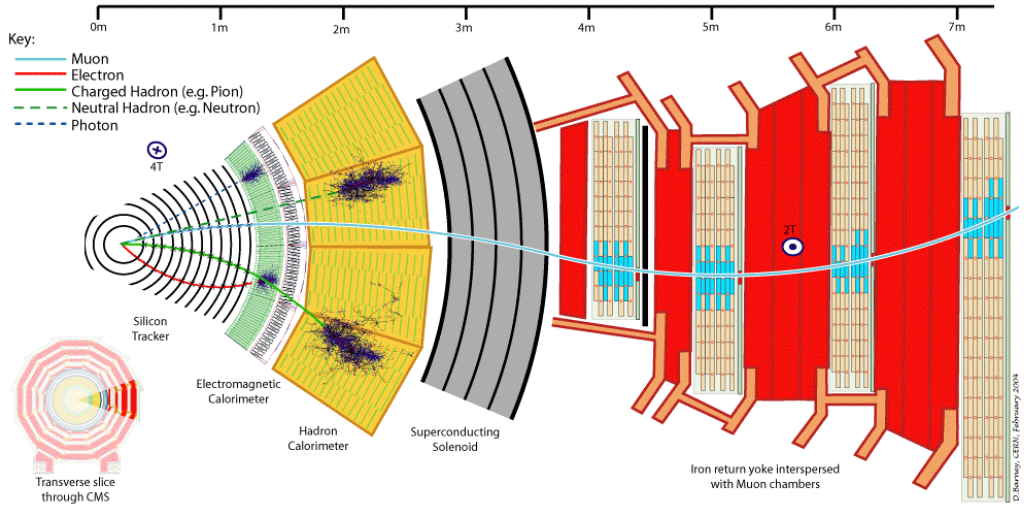


Figure 2.2: The trajectory of a particle traveling through the layers of the detector leaving behind it's signature footprint

The project focusses specifically on data collected from one of the Calorimeters, - the Hadron Calorimeter (HCAL). The HCAL, as its name indicates, is designed to detect and measure the energy of hadrons or, particles that are composed of quarks and gluons, like protons and neutrons. Additionally, it provides an indirect measurement of the presence of non-interacting, uncharged particles such as neutrinos (missing energy) . Measuring these particles is important as they can tell us if new particles such as the Higgs boson or supersymmetric particles (much heavier versions of the standard particles we know) have been formed. The layers of the HCAL are structured in a staggered fashion to prevent any gaps that a particle might pass through undetected. There are two main parts: the barrel and the end caps. There are 36 barrel wedges that form the last layer of the detector inside the magnet coil, there is another layer outside this, and on the endcaps, there are another 36 wedges to detect particles that come out at shallow angles with respect to the beam line.

Chapter 3

Data Collection and Data Quality Monitoring

3.1 What is Data Collection for CMS?

During data taking there are millions of collisions occurring in the center of the detector every second. The data per event is around one million bytes (1 MB), that is produced at a rate of about 600 million events per second [1], that's about 600 MB/s. Keeping in mind that only certain events are considered "interesting" for analysis, the task of deciding what events to consider out of all the data collected is a two-stage process. First, the events are filtered down to 100 thousand events per second for digital reconstruction and then more specialized algorithms filter the data even more to around 100 200 events per second that are found interesting. For CMS there is a Data Acquisition System that records the raw data to what's called a High-Level Trigger farm which is a room full of servers that are dedicated to processing and classify this raw data quickly. The data then gets sent to what's known as the Tier-0 farm where the full processing and the first reconstruction of the data are done. [2]

3.2 What is Data Quality Monitoring?

To operate a sophisticated and complex apparatus as CMS, a quick online feedback on the quality of the data recorded is needed to avoid taking low quality data and to guarantee a good baseline for the offline analysis. Collecting a good data sets from the collisions is an important step towards search for new physics as deluge of new data poses an extra challenge of processing and storage. This all makes it all the more important to design algorithms and special software to control the quality of the data. This is where the Data Quality Monitoring (DQM) plays a critical in the maintainability of the experiment, the operation efficiency and performs a reliable data certification. The high-level goal of the system is to discover and pinpoint errors, problems occurring in detector hardware or reconstruction software, early, with sufficient accuracy and clarity to maintain good detector and operation efficiency. The DQM workflow consists of 2 types: **Online** and **Offline**.

The **Online** DQM consists of receiving data taken from the event and trigger histograms to produce results in the form of monitoring elements like histogram references and quality reports. This live monitoring of each detector's status during data taking gives the online crew the possibility to identify problems with extremely low latency, minimizing the amount of data that would otherwise be unsuitable for physics analysis. The scrutiny of the Online DQM is a 24/7 job that consists of people or shifters that work at the CMS control center constantly monitoring the hundreds of different plots and histograms produced by the DQM software. This consumes a lot of manpower and is strenuous work.

The **Offline** DQM is more focused on the full statistics over the entire run of the experiment and works more on the data certification. In the offline environment, the system is used to review the results of the final data reconstruction on a run-by-run basis, serving as the basis for certified data used across the CMS collaboration in all physics analyses. In addition, the DQM framework is an integral part of the prompt calibration loop. This is a specialized workflow run before the data are reconstructed to compute and validate the most up-to-date set of conditions and calibrations subsequently used during

the prompt reconstruction.

This project aims to minimize the DQM scrutiny by eye and automate the process so that there is a more efficient process to monitor the detector and the quality of the data by implementing Machine Learning techniques.

Chapter 4

What is Machine Learning?

Machine Learning (ML) can be defined as an application of Artificial Intelligence that permits the computer system to learn without being told explicitly. In ML 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 [3]. ML has made tremendous strides in the past decades and has become very popular recently due to its multifaceted applications. It is being used on social media, marketing, and in the scientific community as well. Some examples of ML applications are: the algorithms used on application in smartphones to detect human faces, self-driving cars, computer games, stock prediction, and voice recognition. An interesting characteristic of ML algorithms is that the more data one inputs the better is the performance. The ML application has a very wide spectrum covering almost every aspect of human endeavor that involves a lot of data. Scientific analysis today generates enormous data and is a hence is a perfect used case to apply ML techniques. In this work we use enhanced ML techniques based on progress in the recent past.

In general, there are two main categories to classify machine learning problems: **Supervised Learning** (SL) and **Unsupervised Learning** (UL). SL is the most used ML approach and has proven to be very effective for a wide variety of problems. Examples of common SL problems are: spam filters, predicting housing prices, identifying a malignant or benign tumor, etc. These types of problems are characterized by providing a

"right answer" as a reference. For example, spam filter algorithms identify emails that are spams by training on a dataset that has examples of such emails. In case of predicting house prices, the algorithm is trained on a dataset of houses involving features like the area of the house, number of rooms, and the selling price of the house.

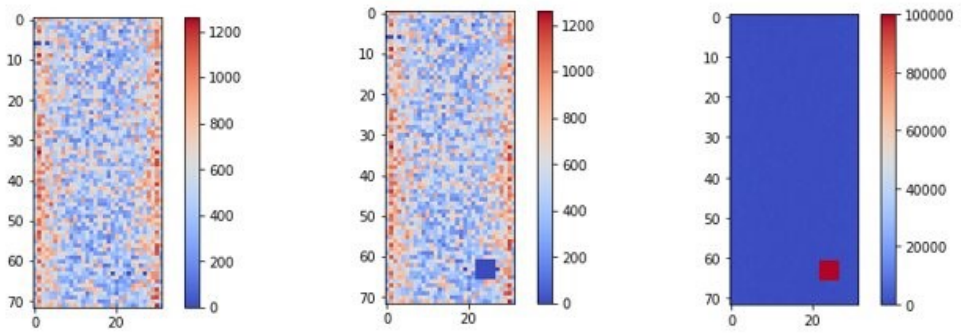
UL algorithms are different in the sense that they do not have the "right answers" given to the machine. Instead, UL algorithms are used for finding patterns and make clusters from the given data. That is what also forms the basis of a search engine (e.g. Google news). Clicking on a link to a news article, one gets many different stories of different journals that have some correlation with the article searched. This happens because the ML algorithm is capable of learning features and shared patterns from a bunch of data without being given any specifics. Another interesting UL problem is the so-called "cocktail party" that involves distinguishing the voice of two people recording on two microphones located at different places. The ML algorithm is able to separate the sources of the voices in the recordings by learning the voice features that correspond to each person, showing the power of the UL algorithm.

In this study, I have focused on an SL approach and a variant of the UL approach, called the **Semi-Supervised Learning** approach (SSL). The SSL is named so because the data involves looking at images that are already known to be "Good" but one doesn't necessarily know every possible situation that produces a "Bad" image. The purpose is to define a metric for a "good" image and subsequently decide if an image is "bad" in case it deviates too much from an acceptable value.

4.1 Developing the Algorithm

To develop an ML algorithm the following are taken into consideration, what is the task? and what is the method to approach the task? In our case, we are looking into images that have information about the activity that the channels in the HCAL are detecting. These images are called "occupancy maps" and they are a visual way of monitoring the health of the detector itself (see Figure 4.1). There are two common problems that can be

identified by viewing occupancy maps which are called ”dead channels” and ”hot towers”. They are referred to as “**dead**” and “**hot**” respectively in the rest of this document. Dead channels mean that on a certain place in the occupancy map there is not any readout from the channels on the HCAL and hot channels mean that there are channels that are being triggered by noise or are damaged in a way that makes them readout too much activity.



(a) Good Image **(b)** Dead Image **(c)** Hot Image

Figure 4.1: Occupancy maps with 5x5 affected regions

The problem is the following, to create a model that can detect and classify what type of scenario is occurring on each occupancy map. For this, we want to go with a SL approach which means that we will give the model the images as the input and it will train on these images by learning to identify patterns or features in the image and try to do a “fit” from the images to their corresponding labels. After the training, the algorithm will be given a testing set for us to evaluate the model’s ability to correctly detect if there is a problem with the image and what type of problem is being detected. The output of the model will be the predicted class of the test image. The predictions are based on the labels and their corresponding images that were given to the model during training. This means that if the model was trained with 3 different types of images with their corresponding label the model will only work well for images that present similar patterns or characteristics to those presented in the training. For example, if we only train the model to distinguish between “good” and “hot” then when the model encounters images that aren’t either of these two, like an image labeled “dead”, then the model will not know what to do with this image and will give it an incorrect label. After the SL model has been tested the next step is trying an SSL model. The term semi-supervised

simply means that there isn't a ground truth label that is being given to the model during training because either there isn't necessarily a ground truth, or we don't know what the ground truth is. What we do know, is what is considered as a "good" image and what this approach hopes to accomplish is to use the error in the reconstruction of the input image and use that information to discriminate between the "good" vs the "bad" images.

4.2 Teaching the Algorithm

The way an ML algorithm learns is by an iterative process called an optimization algorithm in which the predicted output value of the model is compared to the desired output (See Figure 4.2) and the weights and biases of the model are adjusted such that the predicted output is closer to the desired output.

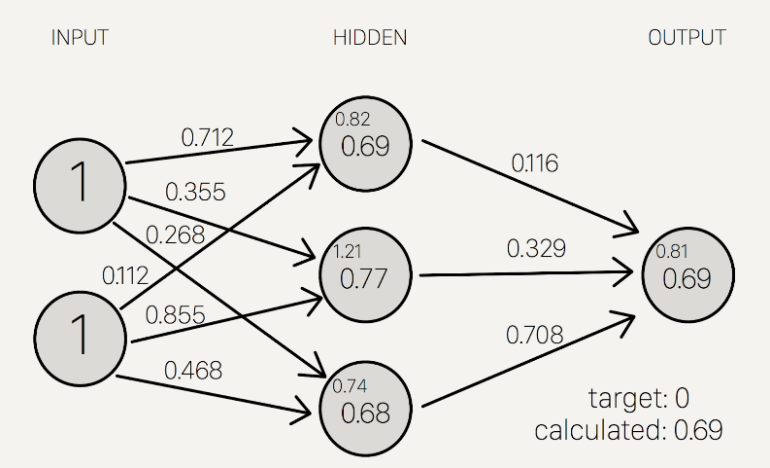


Figure 4.2: Weights and Biases

"Optimization algorithms helps us to **minimize** (or **maximize**) an **Objective** function *(another name for **Error** function)* **E(x)** which is simply a mathematical function dependent on the Model's internal **learnable parameters** which are used in computing the target values(**Y**) from the set of *predictors*(**X**) used in the model. For example - we call the **Weights(W)** and the **Bias(b)** values of the neural network as its internal learnable *parameters* which are used in computing the output values and are learned and updated in the direction of optimal solution i.e. minimizing the **Loss** by the network's training process and also play a major role in the ***training*** process of the Neural Network Model." [5].

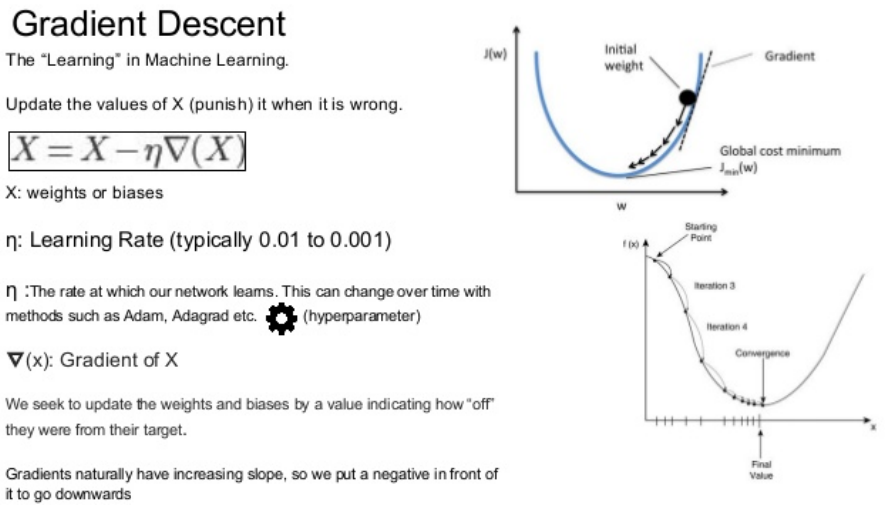


Figure 4.3: Gradient Descent algorithm

The most basic and probably the most used optimizer is called Gradient Descent (GD). GD is based on the concept of using the gradient of a loss or cost function and moving the weights and biases of the ML model so that the predicted value is taking a step in the decreasing direction of this error function (See Figure 4.3). In general, the "terrain" of the loss function is not a smooth bowl-shaped surface like the one present in the image. The most general form of the surface is more similar to a rocky mountain (See Figure 4.4), which presents a problem when using simple optimizers like GD.

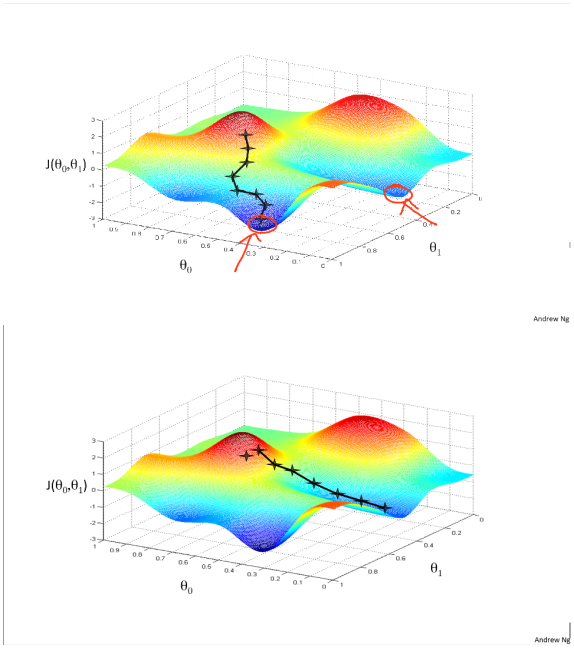


Figure 4.4: Loss Function surface

Chapter 5

Results

Here first the limitations of Scikit-learn predefined ML models - Logistic Regression(LR) and Multi-Layer-Perceptron(MLP), are described. The Logistic Regression Model seems to work almost perfectly with all 3 classes when the bad region size is 5×5 (as in Figure 4.1) with either the same or randomized location. When the bad region size is 1x1 like in Figure 5.1 the LR Model performs poorly with an accuracy of approximately 20%. The MLP does not seem to work in any of the used cases that are studied as it always performs poorly with an accuracy of $\approx 40\%$.

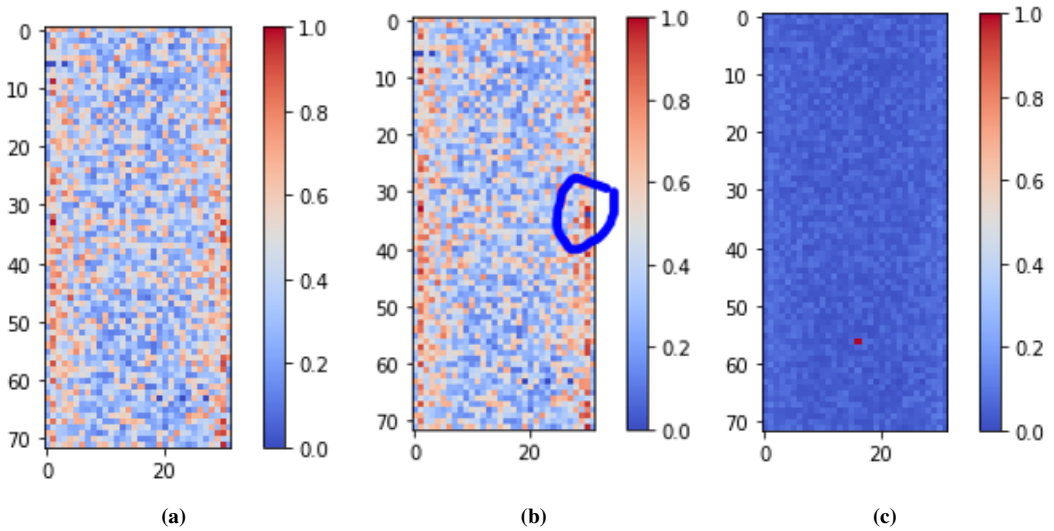


Figure 5.1: Occupancy Maps with 1x1 bad regions. A) Good image B) Dead image C) Hot image

Also, the use of Scikit-learn's library is limited in comparison with the Keras module since one cannot customize the structure of the ML model with detail. Moreover, Keras is

an ML library designed for developing deep neural networks. Hence it was decided to use Keras primarily for the creation of the model. With the Keras library, numerous models were designed with both, SL method and SSL learning method. Using SL method, we are interested in detecting anomalies and classifying what type of anomaly is seen. With SSL method, we are interested in looking at the error of the reconstruction of an image to give an idea that the image given can be considered good or that it might have some unseen anomalies

5.1 SL Models for known anomalies in the HCAL data for DQM

We considered three SL Models for classification of known anomalies in the HCAL data for DQM. These models are based on Convolutional Neural Networks and differ in the number of layers utilized, their ordering and number of units in each layer. The Models and the corresponding results are described below.

5.1.1 Two Convolutional Layers for binary classification

Several variations of the two Convolutional Layers Model were tested and optimized on the DQM data. This led to an optimal value of 8 units/neurons in the Convolutional layers. The detail of selecting the number of units per layer is of great importance to find a balance between efficiency and complexity of a model. More complex models (more layers and connections) are "heavy" to train in terms of computational cost, provide better results and are prone to "overfitting" to the training data. Simpler models (fewer layers and connections) are quicker to train, efficient and computationally economic. However, simpler models are more likely to "underfit" to the data. The Figure 5.2 below shows a code snippet with this model. Figure 5.3 below shows the learning curve for this model trained with Good and Hot images for fixed 5×5 location and the corresponding Confusion Matrix.

```
model = Sequential([
BatchNormalization(input_shape=input_shape))
Conv2D(8, kernel_size=(3, 3), strides=(2, 2), activation='relu')
Conv2D(8, kernel_size=(3, 3), strides=(2, 2), activation='relu')

Dropout(0.25)

Flatten()

Dense(2, activation='softmax')

])
```

Figure 5.2: Two Convolutional Layers Model

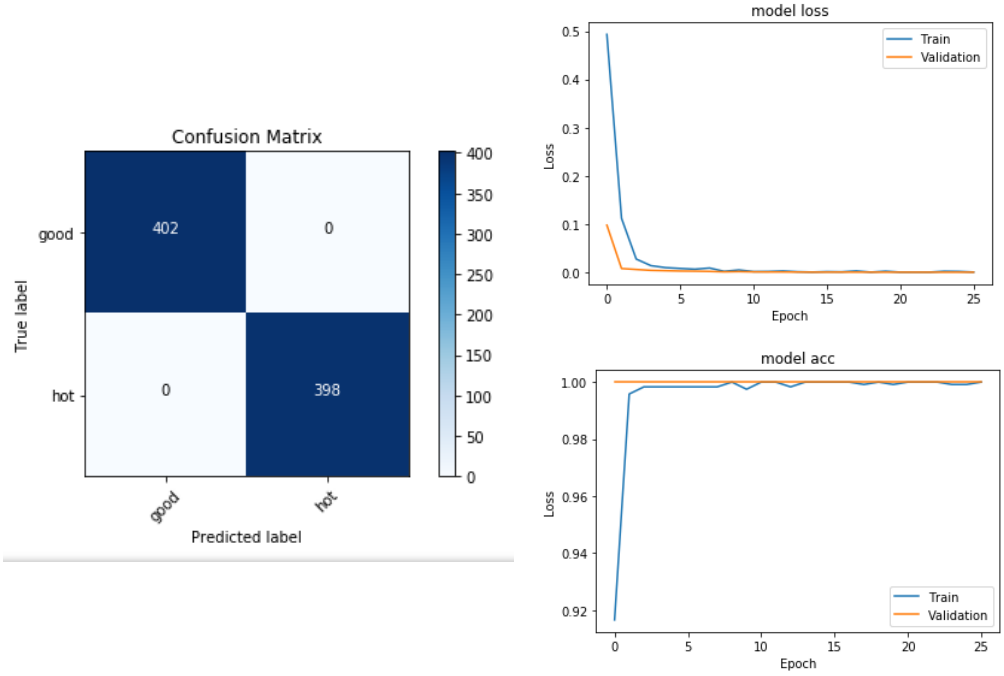


Figure 5.3: Confusion Matrix results and Learning curve for 5×5 damaged area with on the same location for all trials

Figure 5.4 shows the learning curve for this model trained with Good and Hot images for fixed 5×5 location and the corresponding Confusion Matrix.

Figure 5.5 shows the learning curve for this model trained with Good, Hot and Dead images for random 5×5 location and the corresponding Confusion Matrix

Figure 5.6 shows the learning curve for this model trained with Good, Hot and Dead images for random 1×1 location and the corresponding Confusion Matrix. The corresponding learning curves and confusion matrix for a fixed location for 3-class (Good, Hot, Dead) configuration give the same behavior as 2-labels (Good, Hot) images

In a more realistic scenario, the problems with HCAL DQM would be more granular

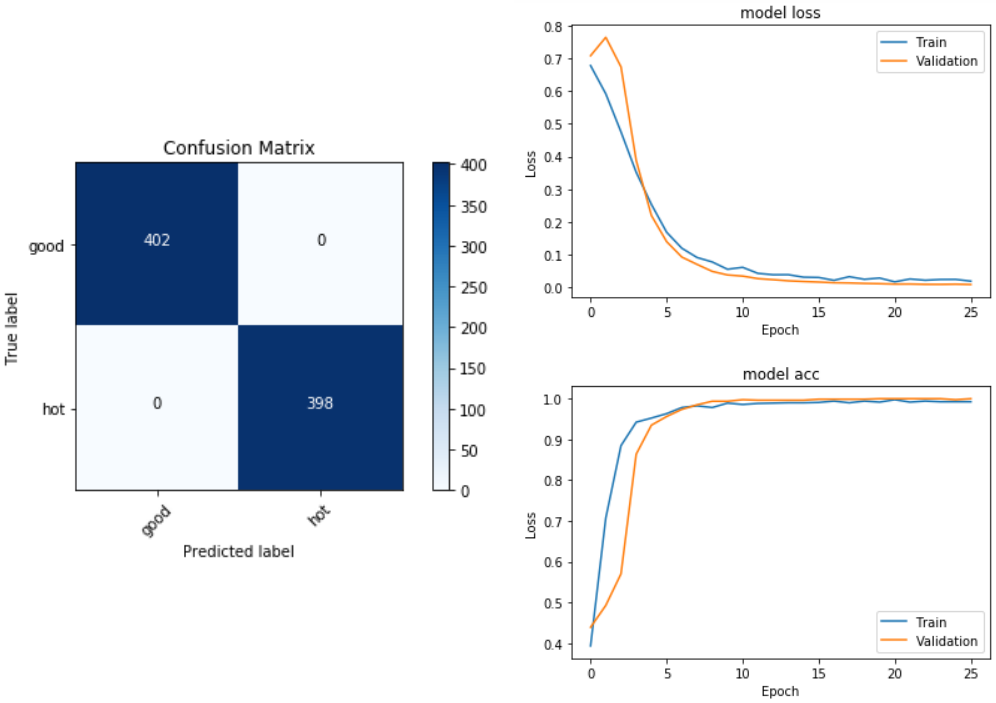


Figure 5.4: Confusion Matrix results and Learning curve for 5×5 damaged area with on the random location for all trials

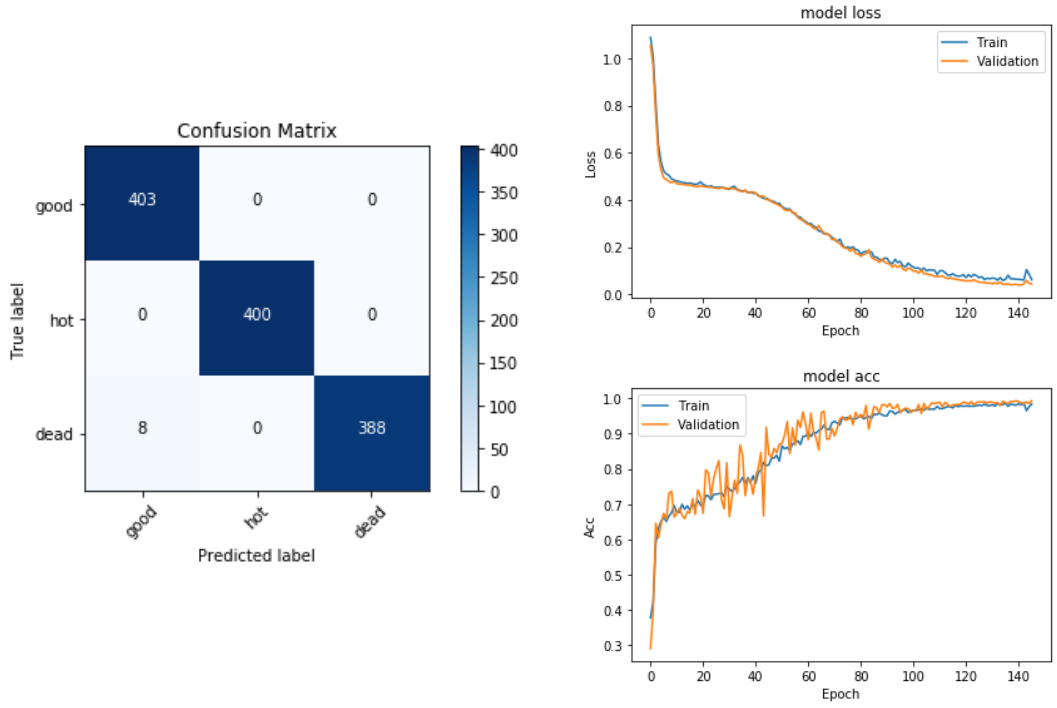


Figure 5.5: Confusion Matrix results and Learning curve for 5×5 damaged area with an extra class to identify with random location for all trials

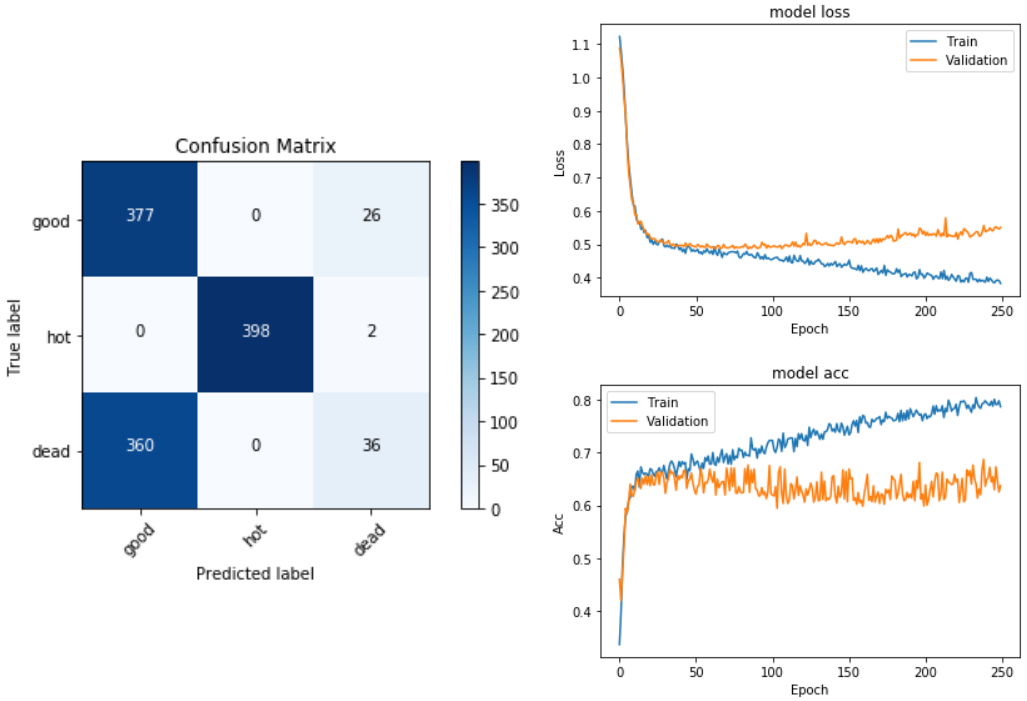


Figure 5.6: Confusion Matrix results and Learning curve for 1×1 damaged area with random location for all trials

i.e. 1×1 type. When this model is tested for problematic channels in 1×1 configuration the learning curves for the training (blue) and validation (orange) sets depart after few epochs as shown in Figure 12 (right part). From the left part of the figure, dividing the sum of numbers along the diagonal (377+398+36) by the sum of all the numbers in the matrix gives 1/3. This demonstrates that the model is "overfitting" to the training set and misclassifies images $\approx 33\%$ of times. Hence, we consider adding a Convolutional layer to gain more prediction accuracy as shown in the next section.

5.1.2 Three Convolutional Layer for multiclass classification

In this model we add another Convolutional layer to increase the prediction accuracy for a more realistic scenario in the HCAL, that is, 1×1 test cases and overcome the deficiency of the previous model. The code snippet in Figure 5.7 reflects the changes made with respect to the subsection 5.1.1.

Figure 5.8 shows the performance of this model. The learning curves on the right-side

```
model = Sequential([
Conv2D(10, kernel_size=(2, 2), activation='relu', strides=(1, 1),input_shape=input_shape),
MaxPooling2D(pool_size=(2,2)),
BatchNormalization(),
Conv2D(8, kernel_size=(3, 3), activation='relu', strides=(1, 1)),
MaxPooling2D(pool_size=(2,2)),
Conv2D(8,kernel_size=(1,1), activation='relu'),
Dropout(0.25),
Flatten(),

Dense(8,activation='relu'),
Dense(3, activation='softmax')
])
```

Figure 5.7: Code snippet of a ML model with three convolutional layers for better identification

show that the training (blue) and validation (orange) sets correlate well with each other. In other words, the model can successfully apply what it learns from the images. The **ROC** curve on the lower left shows the variation of the true positive rate (**TPR**) versus the false positive rate (**FPR**) for three classes 0, 1 and 2 corresponding to the labels Good, Hot and Dead, respectively. "Each point on the **ROC** curve represents a sensitivity/specificity pair corresponding to a decision threshold" [4]. The top left of the figure shows the Confusion Matrix (CM) for this model, which when compared to the CM of the previous model subsection 5.1.1 is more diagonal.

5.1.3 Three Convolutional Layers with a new architecture for multi-class classification

As can be seen in the CM of Figure 5.8 that there is still scope to improve the prediction accuracy and minimize FPR. To achieve this, we introduce the use of BatchNorm before every activation layer as can be seen in the code snippet in Figure 5.9. This model makes the CM more diagonal and brings the ROC AUC (Area Under the Curve) closer to 1 compared to previous model (previous section) as seen in Figure 5.10.

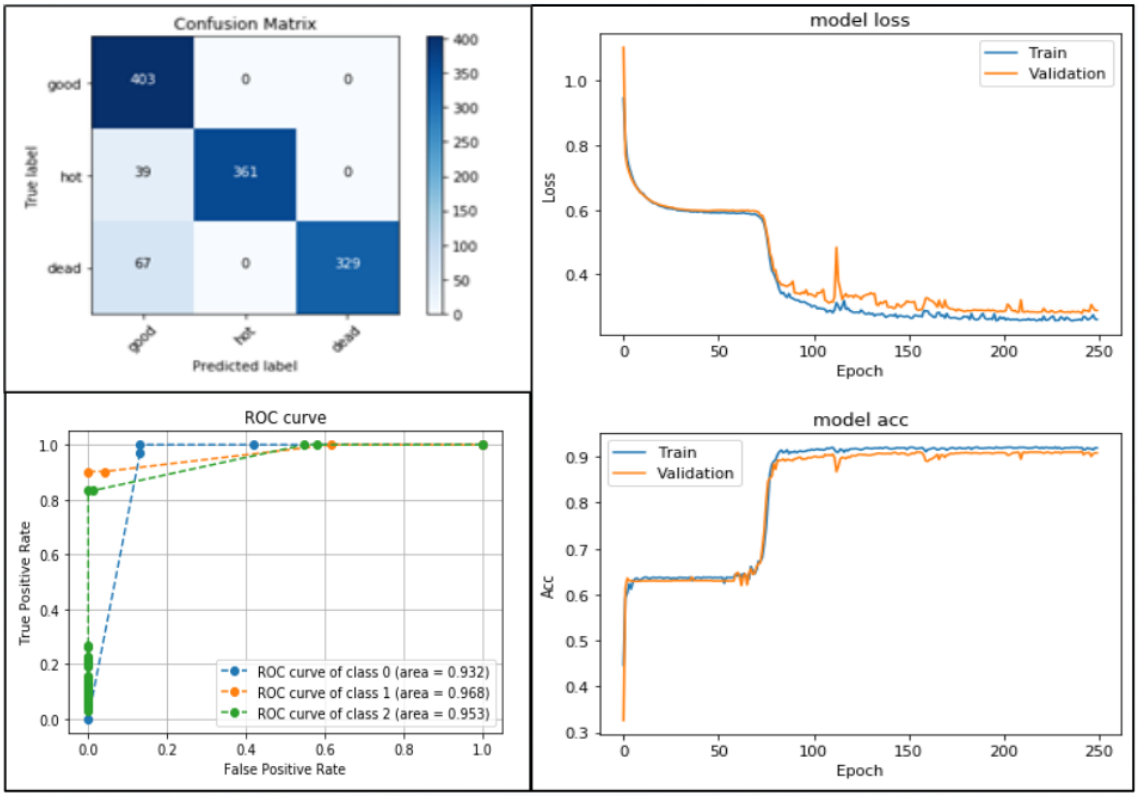


Figure 5.8: Three convolutional layers model results

```
model = Sequential()

model.add(Conv2D(10, kernel_size=(2, 2), strides=(1, 1),input_shape=input_shape))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(8, kernel_size=(3, 3),strides=(1, 1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(8,kernel_size=(1,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dropout(0.25))
model.add(Flatten())

model.add(Dense(8))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dense(3, activation='softmax'))


model.compile(loss='categorical crossentropy',
              optimizer='adam',#Adam(lr=1e-3),
              metrics=['accuracy'])
```

Figure 5.9

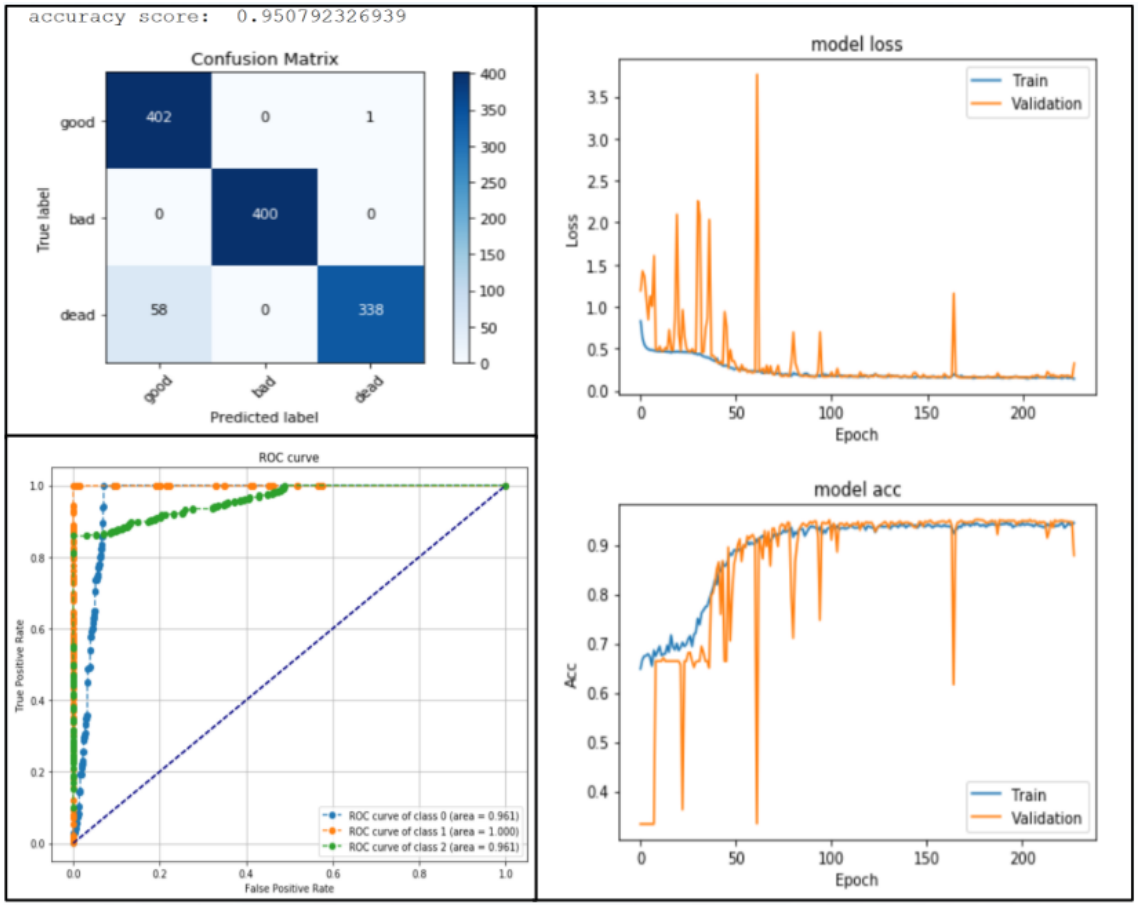

accuracy score: 0.950792326939
Confusion Matrix
good
bad
dead
True label
Predicted label
402
0
1
0
400
0
58
0
338
model loss
Train
Validation
Loss
Epoch
model acc
Acc
ROC curve
True Positive Rate
False Positive Rate
ROC curve of class 0 (area = 0.961)
ROC curve of class 1 (area = 1.000)
ROC curve of class 2 (area = 0.961)

Figure 5.10

```
input_img = Input(shape=(input_shape))  # adapt this if using `channels_first

x = Conv2D(86, (3, 3), padding='same')(input_img)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(64, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (4, 4, 8) i.e. 128-dimensional

x = Conv2D(32, (3, 3), padding='same')(encoded)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(64, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(86, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adadelta', loss='mse')
```

Figure 5.11

5.2 SSL Model for unknown anomalies in the HCAL data for DQM

In this model, we try to generalize the learning strategy using SSL to identify unfamiliar anomalies in the HCAL DQM data. We use the reconstruction error as the discriminating factor to identify deviations from the good images. Figure 5.11 shows the code snippet for the SSL model.

Chapter 6

References

[1] CERN. Processing what to record?, 2018.

[2] CMS. The CMS Computing Project. Technical report, CERN, 2005.

[3] Coursera. Machine learning, 2018.

[4] MedCalc. Roc curve analysis, 2018.

[5] Anish Singh Walia. Types of optimization algorithms used in neural networks and ways to optimize gradient descent, 2018.