Machine Learning Models for Data Quality Monitoring

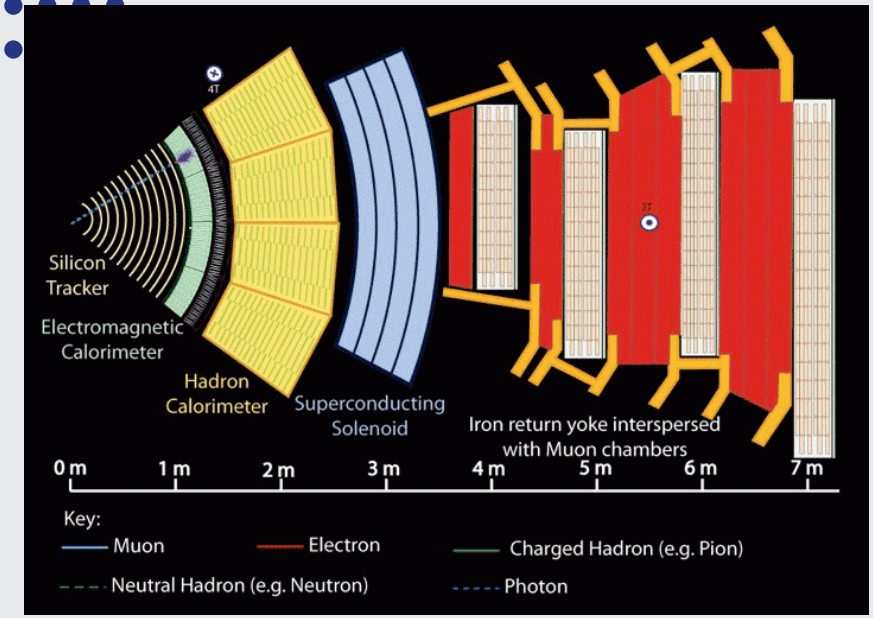
Presented by: Sarah Al Khudari

Supervisor: Roberto Seidita
Co-Supervisor: Antonio Vagnerini

Introduction

WHAT IS THE CMS DETECTOR?

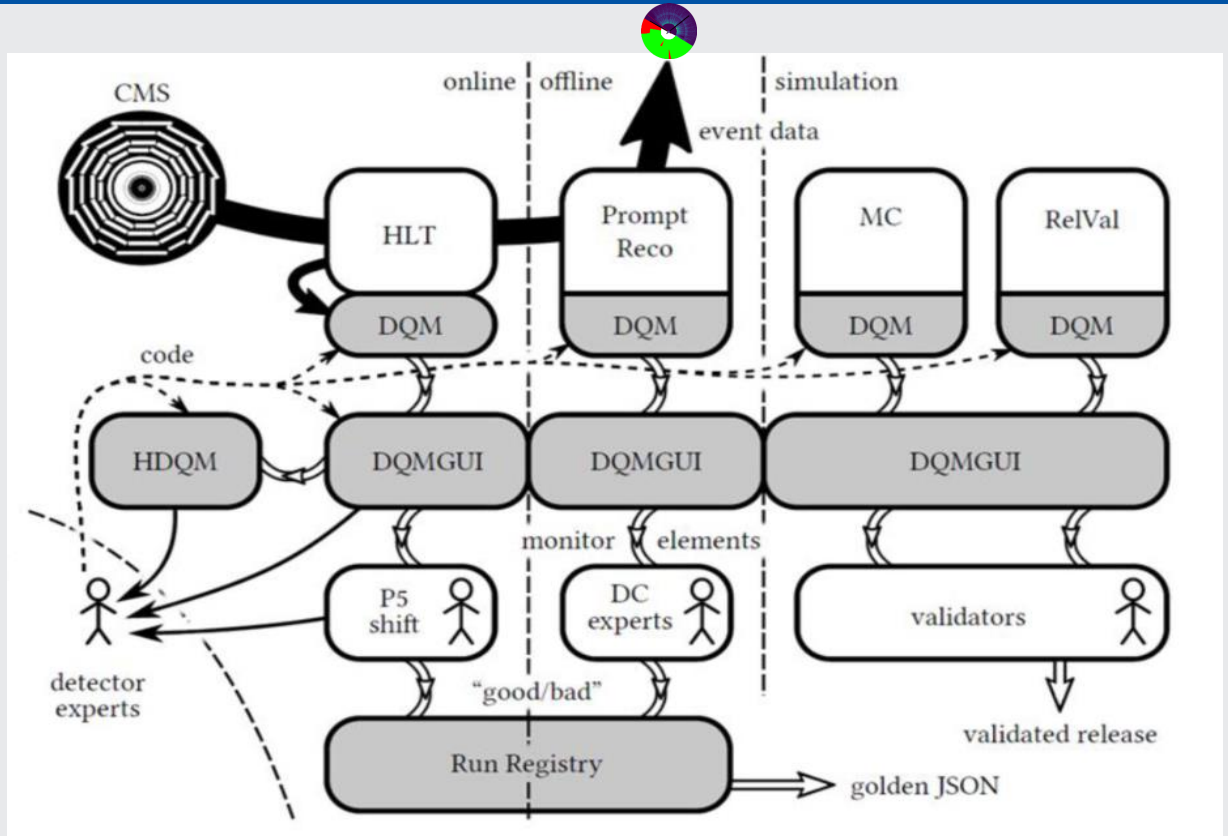
- The CMS detector isa general-purpose detector designed to identify different particles from proton-proton collisions



- ***Silicon Tracker:*** It measures the paths of charged particles (like electrons and protons)
- ***Electromagnetic Calorimeter (ECAL):*** Incoming electrons and photons produce a shower in the calorimeter
- ***Hadron Calorimeter (HCAL):*** Creates showers to measure energy of hadrons(quarks)
- ***Superconducting Solenoid:*** Magnet that bends the paths of charged particles. To measure momentum
- ***Muon Chambers:*** detect these muons and track their paths.

Data Quality Monitoring (DQM)

CMS Processing Data



What are we Monitoring?

DQM monitors different elements coming from each run of the CMS

- ***Each run is made of of several lumisections; 23 seconds of data-taking***
- ***Lumisections have elements studied such as:***
  - ***METSig***
  - *METPhi*
  - *MET_2*
  - *SumET*

GOOD data VS BAD data

All calibrations make sense	Imbalance and anomalies spiked
Data can be understood and studied	Calibrations off
Events of physics interest	Faulty components in detector

$$\vec{E}_{\mathrm{T}}^{\mathrm{miss}} = -\sum_{\mathrm{observable}} \vec{p}_{\mathrm{T},i}$$

$$\mathcal{S} = \frac{E_{\mathrm{T}}^{\mathrm{miss}}}{\sqrt{\sum E_{\mathrm{T}}}}$$

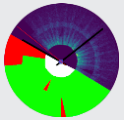
Data Quality Monitoring (DQM)

ONLINE DQM

- Main goal for online DQM is **check data quality in real time**
- Data monitoring done with **24/7 shifters**
- **Histogram subsets** focusing on subgroups performance real time

OFFLINE DQM

- *Main goal of offline DQM is **to certify if data can be used for physics**.*
- *Labeling "GOOD" or "BAD"*
- *Monitor **detector health** and particle **reconstruction quality***
- *Use of DIALs aids in the process:*

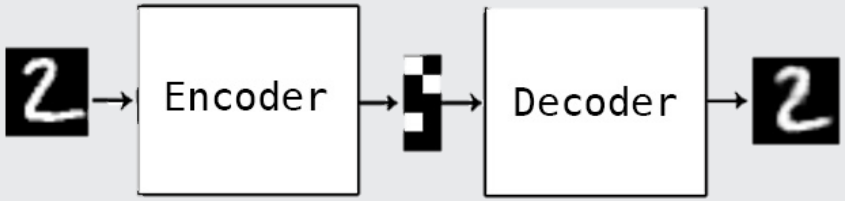


- ~3hrs Prompt reconstruction
- Full reconstruction after 24hrs

ML4DQM: Autoencoders

What are Autoencoders?

- *Machine learning a subcategory of AI*
- *ML that uses encoding and decoding layer*
- *Compresses the input layers and then later build again when decoded*



Unsupervised learning

- *This model is trained by using unsupervised learning. This means that it's given a data set without a label.*
- *The model is trained on what **we know** as "GOOD" data.*
- *After reconstructing the data constantly, if there is a spike in the output histogram we can name is as "anomalies"*

Project Process

Run Registry

NumPy

TensorFlow

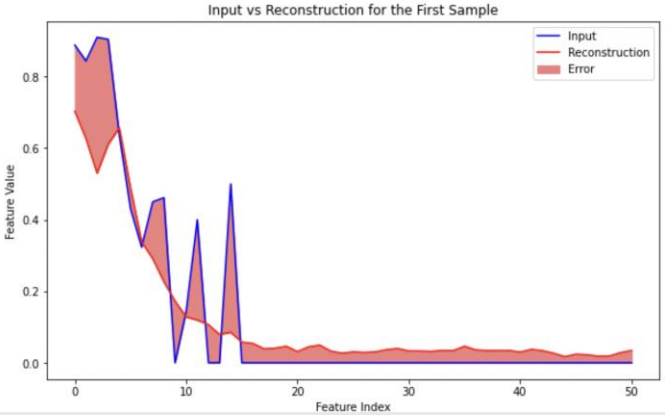
OPTUNA

Training Model with only good Data

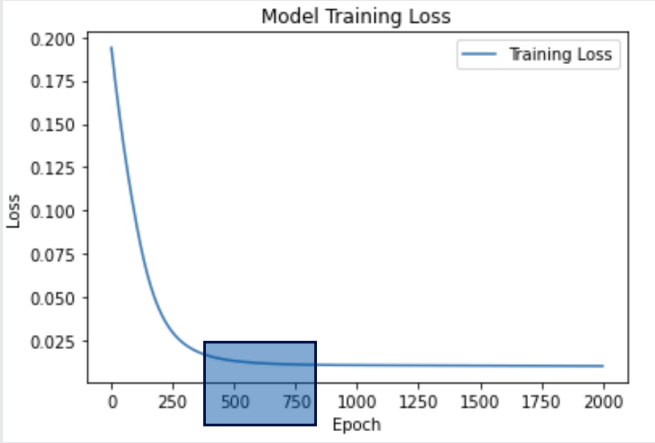
```
def good_training():
    # Ask for run numbers input (comma-separated if multiple)
    input_runs = input("Enter run number(s) seperated by ,: ")
    run_list = [item.strip() for item in input_runs.split(',')]

    # Print the resulting list
    print("Your run list is:", run_list)
    outputs =[]
    histories = []
    if len(run_list) > 1:
        print("Your runs will be concatenated")
        # Process each item in the list
        for run in run_list:
            output, _, _ = harvest_DIALS(run, ME='JetMET/MET/pfMETT1/Cleaned/METSig')
            outputs.append(output)
        print("Appended output list:", outputs)
        # Concatenate all individual outputs into a single array
        output_finals = np.concatenate(outputs, axis=0)
        print("Concatenated array:", output_finals)
        normalized_inputs_anomaly = preprocess(output_finals)
        normalized_inputs = normalized_inputs_anomaly
        model = train_autoencoder_sequential(normalized_inputs, best_params['encoding_dim'], best_params['encoding_dim_2'], 1e-4, best_params['batch_size'], epochs
        print("The best parameters for the models that was used are:\nEncoding Dimension:",best_params['encoding_dim'],"\nEncoding Dimension 2:", best_params['enc
    else:
        # Process the single input
        run = input_runs
        output, _, _ = harvest_DIALS(run, ME='JetMET/MET/pfMETT1/Cleaned/METSig')
        outputs.append(output)
        print("Single output:", outputs)
        normalized_inputs_anomaly = preprocess(np.array(outputs[0]))
        normalized_inputs = normalized_inputs_anomaly
        model = train_autoencoder_sequential(normalized_inputs, best_params['encoding_dim'], best_params['encoding_dim_2'], 1e-4, best_params['batch_size'], epochs
        print("The best parameters for the models that was used are:\nEncoding Dimension:",best_params['encoding_dim'],"\nEncoding Dimension 2:", best_params['enc

    # Evaluate the model
    mse = evaluate_reconstruction(model, normalized_inputs)

    # Print the best parameters used
    print("The best parameters for the model that were used are:")
    print("Encoding Dimension:", best_params['encoding_dim'])
    print("Encoding Dimension 2:", best_params['encoding_dim_2'])
    print("The batch size used:", best_params['batch_size'])



    return model
```

Project Process

```
8/8 [==============================] - 0s 9ms/step - loss: 0.0114 - val_loss: 0.0348
Epoch 994/2000
8/8 [==============================] - 0s 9ms/step - loss: 0.0114 - val_loss: 0.0348
Epoch 995/2000
8/8 [==============================] - 0s 9ms/step - loss: 0.0114 - val_loss: 0.0348
The best parameters for the models that was used are:
Encoding Dimension: 10
Encoding Dimension 2: 2
The batch size used: 128
Mean Squared Error for the first sample: 0.014494354134580853
```



- *Model shows that reconstruction error is low, MSE is close to initial input…model is training well, good performance*

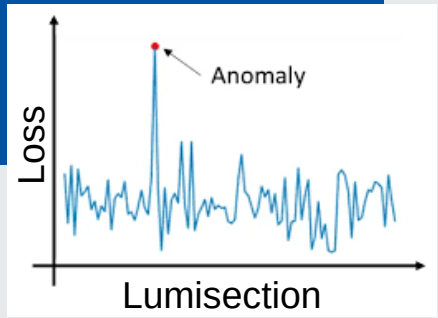



```
In [62]: import numpy as np
         from keras.models import load_model
         import matplotlib.pyplot as plt
         from sklearn.metrics import mean_squared_error

         run = input("Enter Run Number:")
         # Load your trained model
         autoencoder_final_model = load_model('optimized_autoencoder_model.h5')

         # Fetch test data using the harvest_DIALS function
         test_data_array, ls_numbers, sorted_results = harvest_DIALS(run, 'JetMET/MET/pfMETT1/Cleaned/METSig')
         normalized_input = preprocess(test_data_array)
         auto_output = autoencoder_final_model.predict(normalized_input)
         # print(normalized_input)
         # mse = tf.keras.losses.MSE(normalized_input, auto_output)
         # print('MSE on test set:', mse.numpy())

         mean_squared_error(normalized_input, auto_output)

         Enter Run Number:380310
Out[62]: 0.017742091577027223
```

Future Works and Goals

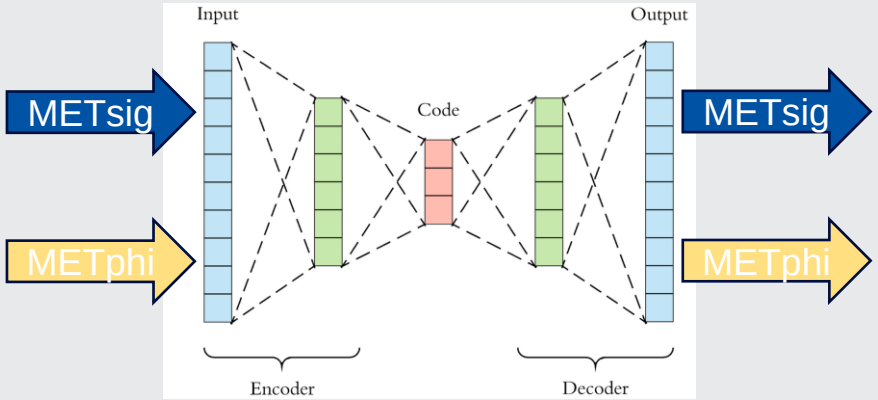
Future Works:

- *Evaluating the model on "Anomalies" with "Bad Data"*
- *When a "Bad Data" is run in model there is a "spike"*
- *Analyze the Model:*
  - *Compare Trained Data and Test Data*
  - *Overfitted or not (difficult with unsupervised)*
  - *Efficiency*
  - *And so on…*



Goals:

- *The current model takes in one element, the goal is taking in two monitory elements*
- *Anomalies in one run could be in other elements*



THANK YOU!

Any Questions?