Neural Networks for $f_0(980)$ detection in pp collisions at $\sqrt{s} = 5.02$ TeV

Submitted By

Abhishek Anil Deshmukh

School of Physical Sciences

National Institute of Science, Education and Research (NISER), Bhubaneswar

Under the Guidance of

Prof. Bedangadas Mohanty

Professor

School of Physical Sciences

National Institute of Science, Education and Research (NISER), Bhubaneswar

Acknowledgement

I would like to express my deep and sincere gratitude to my supervisor Prof. Bedangadas Mohanty for giving me the opportunity to undertake this work. I am extremely grateful to him for his invaluable guidance and support throughout the course of this project. His expertise and guidance have been instrumental in shaping this report.

I would like to express my sincere gratitude to Dr. Ranbir Singh, Dr. Varchasvi Kashyap, and Dr. Ganesh Tambave for their insightful suggestions and feedback. I am also grateful to Dr. Dukhishyam Mallick, Prottay Das, Mouli Chaudhuri, Ashish Pandav, Dr. Debashish Mallick, Swati Saha, and Sudipta Das for their valuable inputs, guidance, and support. I would also like to thank my lab mates, Rajesh Achari and Souradeep Satpathy, for their encouragement and support.

Finally, I would like to express my heartfelt gratitude to my family and friends for their constant encouragement and support. Their unwavering belief in my abilities has been a great source of motivation, and I owe a significant part of my success to them. I am grateful for their love, care, and understanding, which have helped me navigate through the challenges of this project.

Abhishek Anil Deshmukh

5th Year Integrated M.Sc.

School of Physical Sciences

NISER, Bhubaneswar

Abstract

This thesis investigates the application of neural networks to reduce background interference when measuring the production of $f_0(980)$ resonance in pp collisions at $\sqrt{s} = 5.02\text{TeV}$. The study focuses on improving the signal-to-background (S/B) ratio in the $f_0(980) \to \pi^+\pi^-$ hadronic decay channel using the ALICE detector. Traditional methods for calculating the p_T differential yield of $f_0(980)$ through invariant mass analysis are discussed. The theoretical basis and practical implementation of neural networks for background reduction are presented. A neural network-based approach is applied to a subset of the ALICE dataset, and the outcomes are compared with those obtained using invariant mass analysis. The study demonstrates the potential of neural networks to enhance the signal-to-background ratio, leading to improved measurements.

Contents

List of Figures

Chapter 1

Introduction

Resonances are particles with a very short lifespan, around 10^{-23} seconds, and they decay through strong interactions. In the field of particle physics, resonances play a crucial role in studying the fundamental building blocks of matter and how they interact. Understanding resonances is particularly important in collision experiments. In collisions between proton and proton (pp collisions), the decay of resonances contributes significantly to the final state of particles, providing valuable insights into hadron production and serving as a reference for event generators inspired by quantum chromodynamics (QCD).

Resonances are characterized by a sharp peak in the energy distribution of their decay products, which corresponds to the mass of the resonance itself. The $f_0(980)$ resonance was first observed in experiments studying the scattering of $\pi\pi$ particles in the 1970s [1, 2]. The structure of the $f_0(980)$ resonance remains an active area of research. Precisely measuring its production allows for comparisons with production in different configurations predicted by models like coalescence [3]. However, accurately measuring the production of the $f_0(980)$ resonance is challenging due to the significant background noise present in experimental data.

In recent years, neural networks have emerged as a promising approach to address the issue of background interference in particle physics experiments. Neural networks have the remarkable ability to distinguish between signal and background events, resulting in a substantial improvement in the signal-to-background ratio. By harnessing the power of neural networks to reduce background noise, the statistical significance of resonance measurements can be enhanced, enabling more precise determination of their properties.

1.1 Aim

This study aims to investigate the use of neural networks to reduce background noise in measuring the production of the $f_0(980)$ resonance during proton-proton collisions at a center-of-mass energy of $\sqrt{s} = 5.02$ TeV. The main goal is to demonstrate a significant improvement in the signal-to-background ratio specifically in the hadronic decay channel $f_0(980) \to \pi^+\pi^-$. We will utilize data collected by the ALICE detector for this purpose. By employing neural networks, our objective is to enhance the precision of resonance measurements and gain deeper insights into the dynamics of the strong force involved.

1.2 Terminology

Rapidity (y): Rapidity is a measure of the particle's velocity in the direction of its momentum. It is related to the particle's energy and longitudinal momentum. The formula for rapidity is given by the equation:

$$y = \frac{1}{2} \ln \left(\frac{E + p_z}{E - p_z} \right) \tag{1.1}$$

Pseudo-rapidity (η): Pseudo-rapidity is another measure of the particle's velocity, defined in terms of the angle at which the particle is emitted relative to the beam axis. It is often used instead of rapidity as it is more convenient for experimental measurements. The formula for pseudo-rapidity is given by:

$$\eta = -\ln \left(\tan \left(\frac{\theta}{2} \right) \right) \tag{1.2}$$

where θ is the polar angle of the particle with respect to the beam axis.

Transverse Momentum (p_T): Transverse momentum is the component of a particle's momentum perpendicular to the beam axis. It is a fundamental quantity used to study particle interactions. The transverse momentum is calculated using the formula:

$$p_T = \sqrt{p_x^2 + p_y^2} \tag{1.3}$$

Invariant Mass ($M_{\mathbf{inv}}$): Invariant mass is a property of a system of particles that remains constant under Lorentz transformations. It is calculated by combining the energies and momenta of the particles in the system. The formula for invariant mass is given by:

$$M_{\text{inv}} = \sqrt{E^2 - \mathbf{p}^2} \tag{1.4}$$

where E is the total energy of the system and $\mathbf{p}$ is the total momentum vector.

Z Coordinate of the Collision Vertex ($z_{\mathbf{vtx}}$): The collision vertex is the point where the colliding particles interact. z_{vtx} refers to the z-coordinate of this vertex along the beam axis.

Azimuthal Angle (ϕ): Azimuthal angle refers to the angle in the transverse plane measured from a reference direction. In particle physics, it is often measured relative to the beam axis. The range of ϕ is 0 to 2π.

1.3 Detectors

The Time-Projection Chamber (TPC) is the primary tracking detector of ALICE. It is a cylindrical detector with a large volume ($88\,\text{m}^3$). The detector has a length of 5.1 m and inner and outer radii of 0.85 m and 2.47 m, respectively. It covers the pseudorapidity range $|\eta| < 0.9$ with full azimuthal acceptance. The drift volume is filled with Ne:CO_2:N_2 (90:10:5) [4, 5]. The detector is divided into two drift regions by the central electrode located at its axial center. The central electrode acts as the cathode, while the anode is situated on the two end caps. Charged particles passing through the TPC volume ionize the gas along their path, releasing electrons that drift toward the end plates of the cylinder (anode). The maximum electron drift time is 94 μs. A total of 72 multiwire proportional chambers with cathode pad readouts are employed in the end plates. The ionized electrons drift for up to 2.5 m and are detected on 159 pad rows (rows in the radial direction).

The ALICE Time-Of-Flight (TOF) is a large detector that covers a cylindrical surface of 141 m^2 with an inner radius of 3.7 m. It provides pseudorapidity coverage of $|\eta| < 0.9$ and complete azimuthal coverage, except for the region $260^\circ < \phi < 320^\circ$ at $|\eta| < 0.14$, to minimize the material in front of the Photon Spectrometer (PHOS) [4]. The TOF utilizes Multigap Resistive Plate Chambers (MRPCs), which have an intrinsic time resolution better than 50 ps with an efficiency close to 100% [6]. The active region of the TOF barrel has a length of 741 cm. It consists of approximately 153,000 readout channels and has an average thickness of 25-30% of X_0, depending on the detector zone. The gas mixture used in the MRPCs is $C_2H_2F_4$ (93%) and SF_6 (7%). The MRPCs for the ALICE TOF are designed as double-stack strips [6]. Each stack has five gas gaps (250 μm wide), and both stacks are positioned on either side of a central anode. A high voltage is applied to the external voltage of the stack, and pickup electrodes are located further out. When a charged particle traverses the gas in the MRPCs, the high electric field amplifies the ionization, resulting in an avalanche. The resistive plates halt the avalanche development in each gap but allow the fast signals induced on the pickup electrodes by the movement of electrons to pass through. Therefore, the total signal is the sum of signals from all gaps. The excellent time resolution is achieved due to the narrow gaps between different layers of RPC.

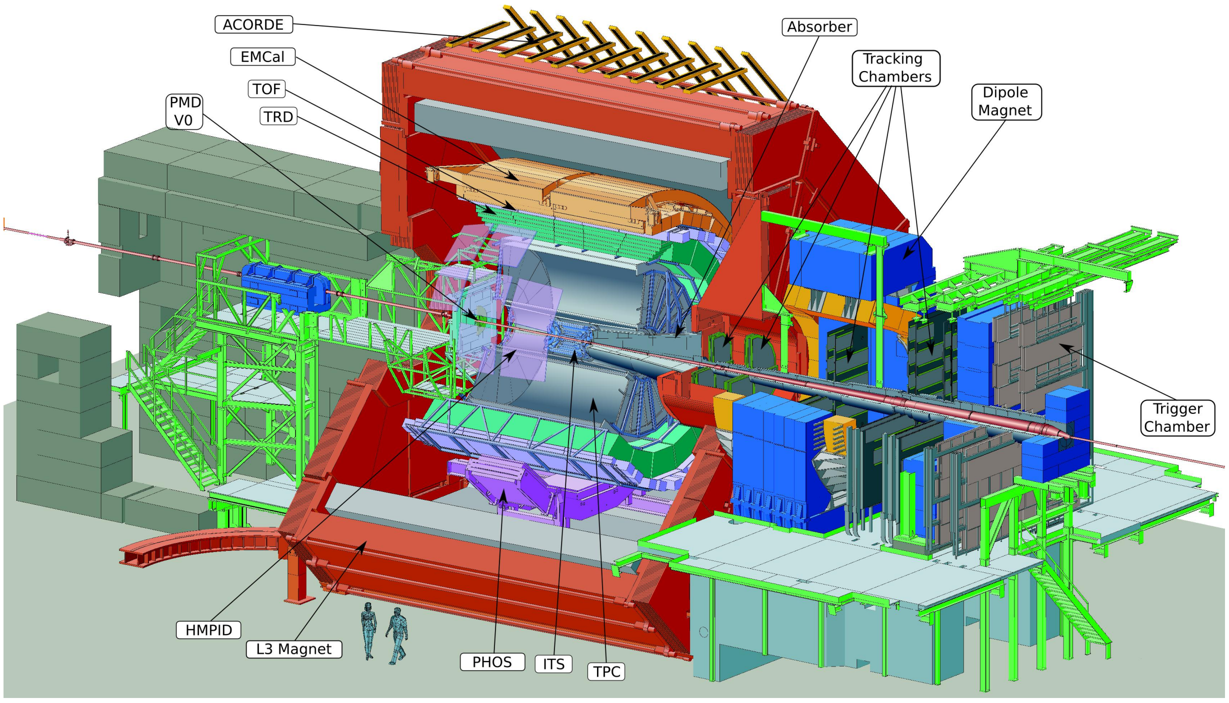


Fig. 1.1: Schematic diagram of the ALICE detector. The diagram illustrates the various components and subsystems of the ALICE detector, including the Time-Projection Chamber (TPC), Time-Of-Flight (TOF) detector, Photon Spectrometer (PHOS), and other essential elements

The TOF provides time measurements that, along with the momentum and track length measured by the tracking detectors, are used to calculate the particle mass. This enables the particle identification capabilities of the TOF detector.

In the next chapter, we will discuss the traditional method for calculating the p_T differential yield.

Chapter 2

Invariant Mass Analysis

2.1 Dataset

For this analysis, we use the data collected in 2015 and 2017 during the Run 2 period at the Large Hadron Collider. These are minimum-bias pp collisions at $\sqrt{s} = 5.02$ TeV.

Data samples used:

LHC15n pass4:
Run list (25 Hadron PID runs - recommended by DPG): 244628, 244627, 244626, 244619, 244618, 244617, 244542, 244540, 244531, 244484, 244483, 244482, 244481, 244480, 244456, 244453, 244421, 244416, 244377, 244364, 244359, 244355, 244351, 244343, 244340.

LHC17p pass1 (cent woSDD and fast):
Run list (41 Hadron PID runs - recommended by DPG): 282343, 282342, 282341, 282340, 282314, 282313, 282312, 282309, 282307, 282306, 282305, 282304, 282303, 282302, 282247, 282230, 282229, 282227, 282224, 282206, 282189, 282147, 282146, 282127, 282126, 282125, 282123, 282122, 282120, 282119, 282118, 282099, 282098, 282078, 282051, 282050, 282031, 282025, 282021, 282016, 282008.

Simulations: LHC18c8a (anchored to LHC15n) and LHC18c8b (anchored to LHC17p) with the same run numbers.

The AliRoot version ‘v5-09-59b_ROOT6-1‘ and AliPhysics version ‘vAN-20220722_ROOT6-1‘ are used for this analysis.

2.2 Event and Track selection

The goal of the event selection in this analysis is to select the minimum bias events:

- kINT7 trigger
- Standard Physics Selection
- Pileup rejection
- INEL > 0 selection
- $|z_{vtx}| < 10$ cm

- SPD vertex z resolution < 0.02 cm
- z-position difference between track and SPD vertex < 0.5 cm

After applying the event selection, we are left with 897 million events.

The $f_0(980)$ resonance is studied by reconstructing its hadronic decay into oppositely charged pions. In order to use the geometrical region in which ALICE performs full tracking, only tracks with pseudo-rapidity in $|\eta| < 0.8$ and $p_T > 0.15$ GeV/c have been accepted. To ensure a good quality of the reconstruction, StandardITSTPCTrackCuts2011 has been applied. Only resonance candidates produced at mid-rapidity are selected with a reconstructed pair rapidity cut $|y| < 0.5$.

2.3 Particle Identification

PID information is used to identify candidate $f_0(980)$ daughters. It is performed by combining the information of the specific energy loss (dE/dx) in the TPC and the time of flight measurement from the TOF. Aiming at rejecting as much of them as possible background while preserving good efficiency, the particle identification strategy chosen for the analysis is based on the following: if the TOF information is available, it is required that $|n\sigma_{TOF}| < 3$ and $|n\sigma_{TPC}| < 2$, otherwise identification is performed only using the TPC, with a cut of $|n\sigma_{TPC}| < 2$. This selection corresponds to the cut set named as AliRsnCutSetDaughterParticle::kTPCpidTOFveto3s in the resonance analysis package of AliPhysics.

2.4 Analysis description

When a decay occurs the invariant mass of the decays products is the same as that of the resonance it decayed from. Invariant Mass of $f_0(980)$ is 990 MeV/c^2 [7], the formula for calculating the invariant mass is here:

$$M_{inv} = (E_1 + E_2)^2 - |\vec{p_1} + \vec{p_2}|^2$$

The $f_0(980)$ signal is reconstructed by calculating the invariant mass of the decay products $\pi^+\pi^-$.

The Unlike-Sign Pair (USP) distribution is made by exhaustively pairing opposite-charged pions from the same events. $f_0(980)$ decays into $\pi^+\pi^-$, so it is in this distribution along with uncorrelated opposite-charged pion pairs from the same event.

The Like-Sign Background (LSB) is made by exhaustively pairing like-charged pions from the same events. This gives us two backgrounds; one made up of $\pi^+\pi^+$ pairs and the other from $\pi^-\pi^-$ pairs. We want to use this background to remove a lot of combinatorial background from the USP due to the uncorrelated opposite-charges pion pairs.

The formula for combining the two LSBs from $\pi^+\pi^+$ and $\pi^-\pi^-$ into a geometric mean and corresponding error propagation is as follows.

$$\text{LSB} = 2\sqrt{y_{++}y_{--}} \tag{2.1}$$

$$\delta_{LSB} = \sqrt{\frac{y_{++}^2(\delta y_{--})^2 + y_{--}^2(\delta y_{++})^2}{y_{++}y_{--}}} \tag{2.2}$$

The idea here is USP has the signal, and the LSB does not, as combining all the possible like-signed pairs completely excludes the resonance under study. Hence subtracting it should remove a lot of uncorrelated pion pairs. Figure 2.1 shows the distributions and the extracted signal. The figure shows the invariant mass distribution with integrated transverse momentum.

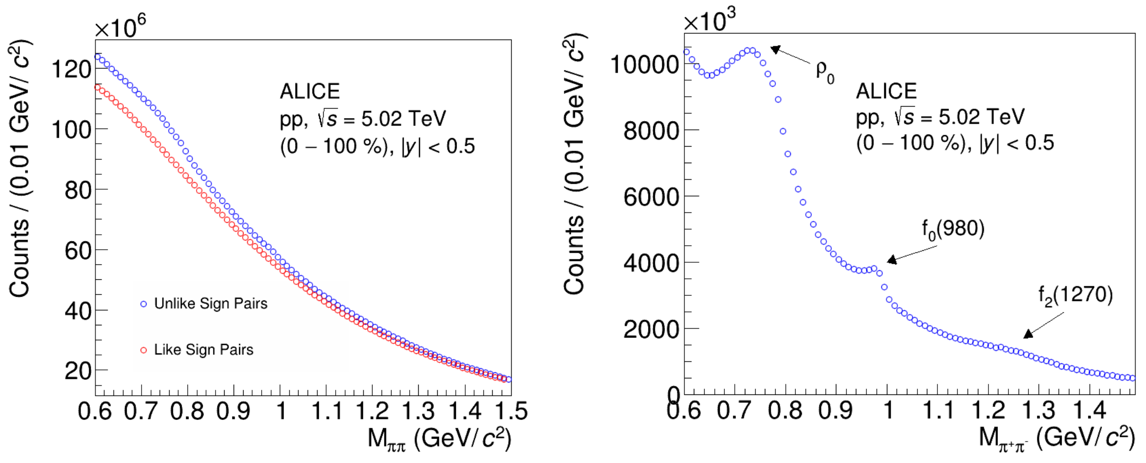


Fig. 2.1: Left image shows the Unlike pair signal (blue) and combined Like pair background (red) on the same axes for comparison. The right image shows the output from the subtraction of the two curves; we see the $f_0(980)$ peak.

The next section describes the fit model and the fitting strategy.

2.4.1 Fitting

The p_T range has been divided into 22 intervals and each interval is fit individually.

$$p_T(\mathrm{GeV/c}) = [0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4, 1.6, 1.8, 2, 2.5, 3, 3.5, 4, 4.5, 5, 6, 7, 8, 10, 12, 16] \tag{2.3}$$

There are way more particles in the lower p_T region than in the higher p_T region. Hence, the division scheme is more aggressive in the lower p_T region and keeps on getting meeker as we move to the higher p_T region allowing for each interval to have enough statistics to conduct the analysis.

$f_0(980)$ has a mass of 0.990 GeV/c^2 [7] hence we are working in the invariant mass range 0.8 to 1.4 GeV/c^2, other particles which decay in this mass range are ρ_0 and $f_2(1270)$, our fit model needs to account for them. The relativistic Breit-Wigner distribution is used to fit the signal from decays. For the background, we use a second-order polynomial.

Relativistic Breit-Wigner function [?] is as follows (in natural units):

$$BW(E, M, \Gamma) = \frac{k}{(E^2 - M^2)^2 + M^2\Gamma^2} \tag{2.4}$$

$$k = \frac{2\sqrt{2}M\Gamma\gamma}{\pi\sqrt{M^2+\gamma}} \tag{2.5}$$

$$\gamma = \sqrt{M^2(M^2+\Gamma^2)} \tag{2.6}$$

Here, E is the energy of the particle, M is the mass of the resonance, Γ is the width of the resonance, and $gamma$ is the relativistic mass correction factor.

The total fit function is the sum of three relativistic Breit-Wigner functions and a second-order polynomial:

$$f(x) = p_0 BW_{\rho_0}(x, p_1, p_2) + p_3 BW_{f_0(980)}(x.p_4.p_5) + p_6 BW_{f_2 1270}(x, p_7, p_8) + p_9 + p_{10}x + p_{11}x^2 \tag{2.7}$$

Here, the p_i s denote the parameters to fit. This model has a total of 12 parameters, but we will be fixing a few values to the ones from PDG [7]:

- Mass of ρ_0 $p_1 = M_{\rho_0} = 0.775\text{GeV}/c^2$
- Width of rho_0 $p_2 = \Gamma_{\rho_0} = 0.1491\text{GeV}/c^2$
- Width of $f_0(980)$ $p_5 = \Gamma_{f_0(980)} = 0.055\text{GeV}/c^2$
- Width of $f_2(1270)$ $p_8 = \Gamma_{f_2(1270)} = 0.1867\text{GeV}/c^2$

Figure 2.2 contains the invariant mass distribution along with the fit on it for four p_T bins. The right plot of Figure 2.3 shows the mass values along with PDG value [7]. We can integrate the area under the curve of the Breit-Wigner distribution of $f_0(980)$, which would give the number of $f_0(980)$s found for every p_T bin which is shown on the left plot of Figure 2.3.

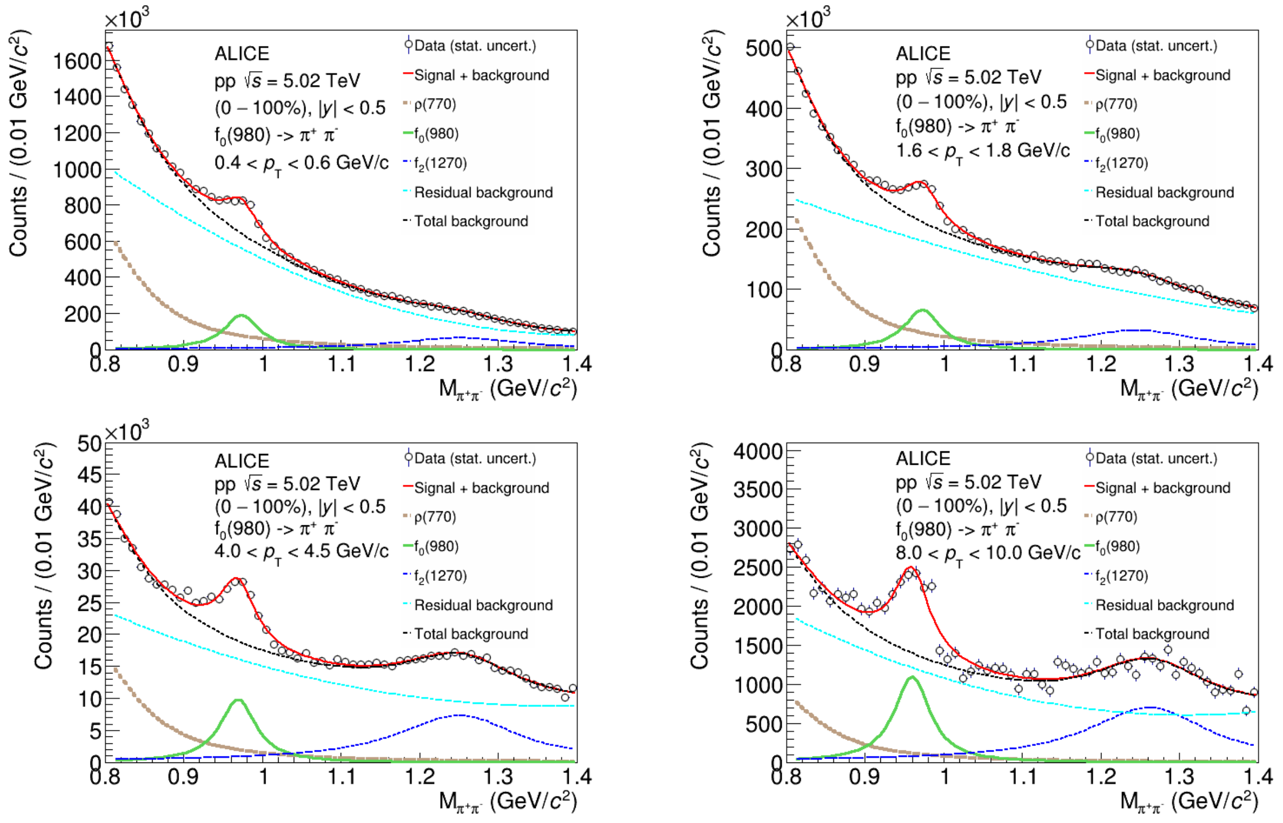


Fig. 2.2: Invariant mass distribution of $\pi^+\pi^-$ pairs after like-sign background subtraction shown here for four transverse momentum bins.

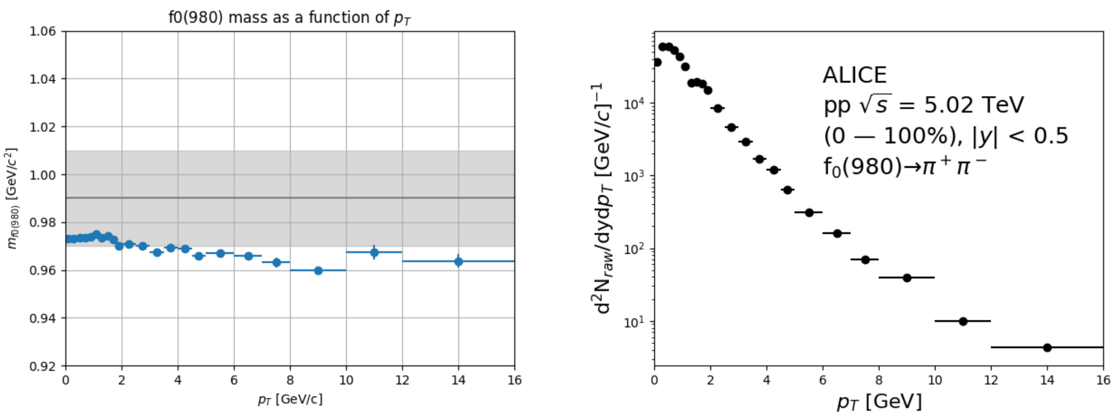


Fig. 2.3: The mass of the $f_0(980)$ signal from the fits along with PDG is shown (left). Raw p_T spectra: the height of the signal function (right).

2.4.2 Correction and Normalization

Monte-Carlo production simulation of pp samples at 5.02TeV with injected $f_0(980)$, $f_2(1270)$ and $\Lambda(1520)$ signals is used as the event generator. Geant3 [8] was used to simulate the particle interaction with the ALICE detector, and the event generator used for creating the simulation dataset is PYTHIA8 Monash tune [9]. The MC sample consists of 151 × 106 accepted events after the same event selection criteria as applied to the data.

The efficiency $\times$ acceptance can be defined as:

$$(Acc \times \epsilon)(p_T) = \frac{N_{rec}(p_T^{rec})}{N_{gen}(p_T^{gen})}$$

Where, p_T^{rec} is the transverse momentum measured by the tracking algorithm and p_T^{gen} is the transverse momentum generated by the event generator. The same track selection cuts as the data are applied on the tracks of the accepted events. The ratio of N_{rec} and N_{gen} expected to provide the $(Acc \times \epsilon)$ for every p_T bin. Figure 2.4 shows the Acceptance $\times$ Efficiency.

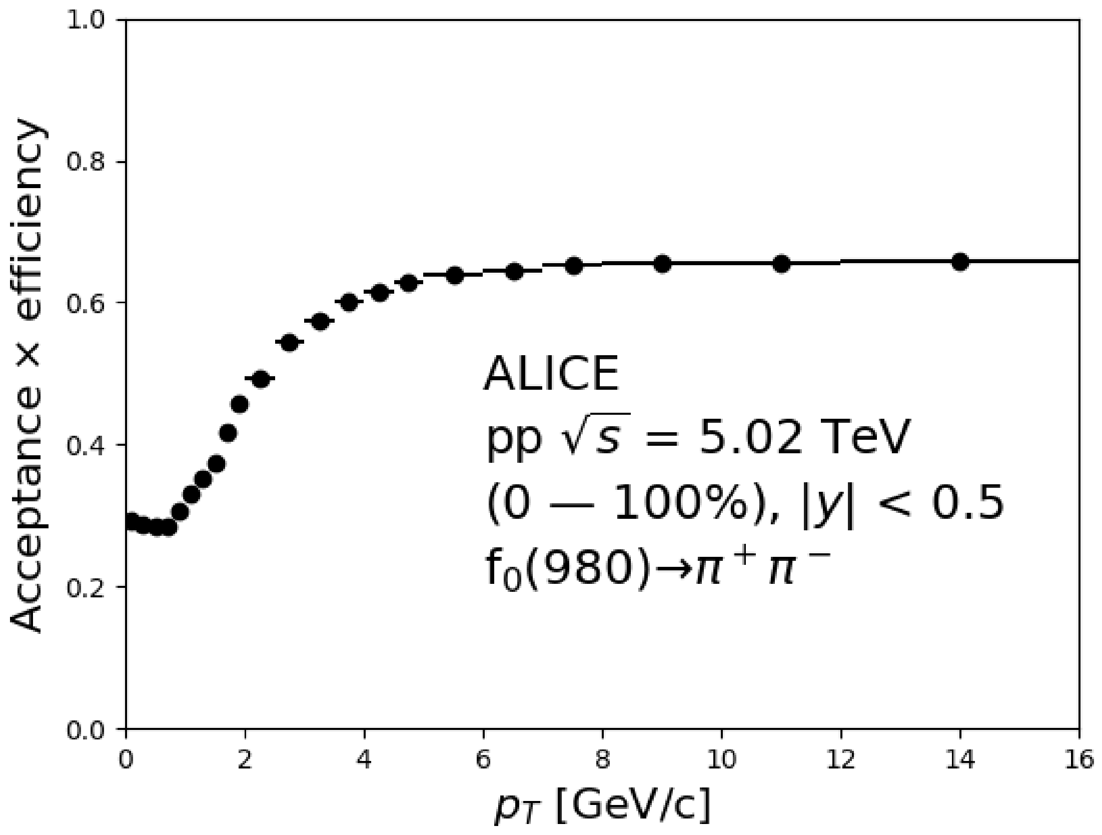


Fig. 2.4: Acceptance $\times$ efficiency

Monte-Carlo provided us with the correction to include all the events which satisfied the kINT7 trigger condition. Still, we want to include all the inelastic events, not just the ones that satisfy the kINT7 trigger. This factor could not be calculated due to a lack of general-purpose simulations that have defined $f_0(980)$, so the factor from ϕ meson analysis for the same collision system was used. This is acceptable because the signal loss correction for resonance decaying into two charged particles is not significantly affected by the mass of the resonance [10].

2.4.3 Normalisation

$$\frac{d^2N}{dp_T dy} = \frac{\text{raw counts}}{dp_T dy} \frac{r(p_T)}{(Acc \times \epsilon)(p_T)} \left(\frac{\epsilon_{SL}}{\epsilon_{rec}}\right)(p_T) \frac{f_{norm} \times f_{vtx}}{N_{evt} BR} \tag{2.8}$$

Where,

- N_{evt} is the number of accepted events = 897 million events.
- $r(p_T)$ is the factor from reweighing from the last section.
- $(Acc \times \epsilon)(p_T)$ is the acceptance × efficiency.
- $(\epsilon_{SL}/\epsilon_{rec})(p_T)$ is the signal loss correction.
- BR is the branching ration of the $f_0(980)$ in the decay channel $\pi^+\pi^- = 46\%$ [7].
- f_{norm} is the factor for converting particle yield normalized to the number of trigger events to a yield normalized to the number of inelastic events, which for this collision system is found to be $= 0.757 \pm 0.019$ [11].
- f_{vtx} is the factor that corrects for vertex quality cuts but without the cut on the z-position of the vertex. For this collision system, it is estimated to be 0.958 [12].

2.5 Final p_T Spectra

The Efficiency corrected and normalized $f_0(980)$ p_T spectrum is shown in figure 2.5. The spectra fit the Levy-Tsallis function [13], the integral of this function is the total yield, and the mean is the mean p_T of the resonance.

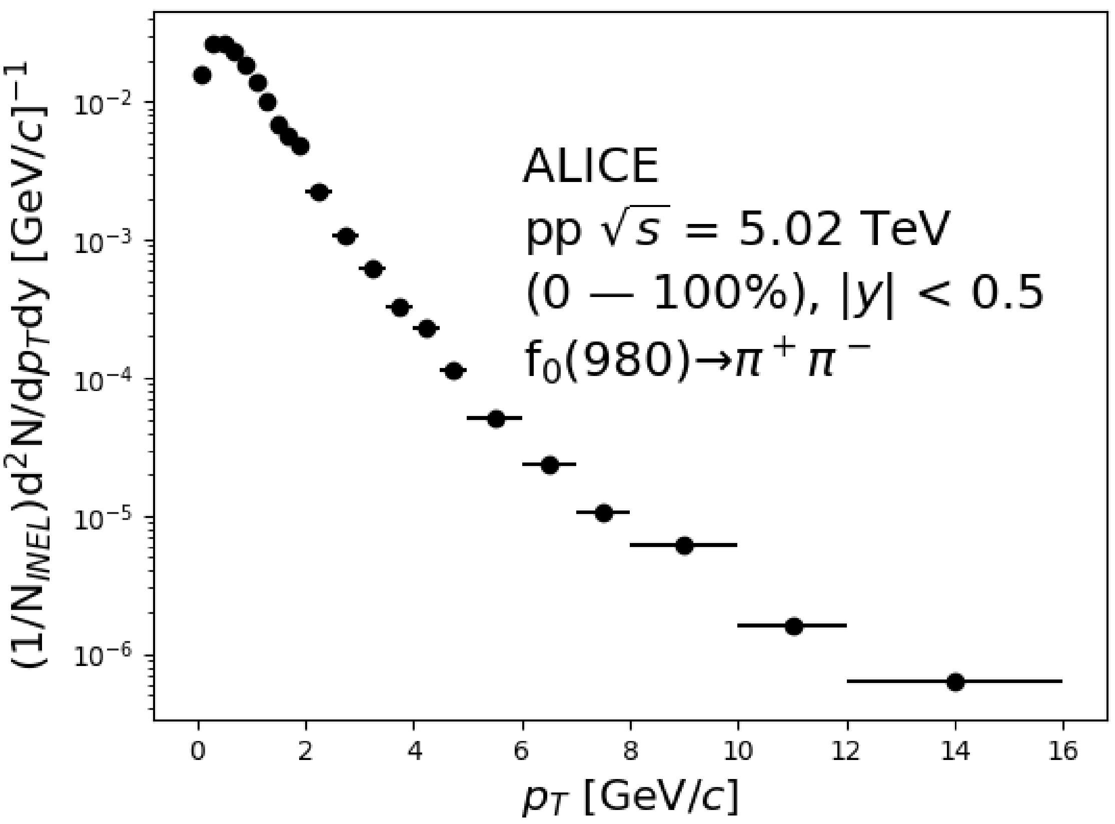
$$\frac{dN}{dy} = 0.0327 \pm 0.0001(stat.)$$

$$\langle p_T \rangle = 0.905 \pm 0.0024(stat.)\ \text{GeV}/c$$

In Table 2.1, results from this analysis are compared with ALICE results.

	This analysis	ALICE result [14]
dN/dy	0.0327 ± 0.0001(stat.)	0.03850 ± 0.0001(stat.) ± 0.0047(syst.)
$\langle p_T \rangle$	0.905 ± 0.0024(stat.) GeV/c	0.9624 ± 0.0014(stat.) ± 0.0357(syst.) GeV/c

Table 2.1: Comparing my results with ALICE



+ring

Fig. 2.5: The p_T differential yield of $f_0(980)$ for the collision system pp $\sqrt{s} = 5.02$ TeV. Statistical error to too small to see. Systematic error is not included.

2.5.1 Sources of error

The statistical error on the yield is calculated by propagating the error on the fit parameters through the integral of the relativistic Breit-Wigner function. The yield is calculated by integrating the relativistic Breit-Wigner function, which is a function of the fit parameters. Some of the sources of systematic error are listed below:

- Raw yield extraction
- Particle identification
- Event selection
- Track quality selection
- Material budget
- Global tracking (ITS-TPC Matching efficiency)
- Hadronic interaction cross section in the material
- Signal loss correction

Chapter 3

Machine Learning and Tensorflow

3.1 Machine Learning

Machine learning is a subset of artificial intelligence that involves designing and developing algorithms and models that can learn patterns and relationships from data. The primary goal of machine learning is to build predictive models that can generalize well to unseen data. In other words, given a dataset with input variables and a corresponding output variable, machine learning algorithms aim to learn a function that can accurately predict the output for new inputs. This is achieved by iteratively adjusting the model parameters using an optimization algorithm until the model minimizes a cost function that measures the difference between the predicted and actual output. In our context, we will be using labelled data from Monte-Carlo simulations to train a model to discriminate between background and signal and then use the model to reduce the background and extract signal with greater significance.

3.1.1 Terminologies

Features are the variables which are the input. They are the observables by which the data point is defined. We may also refer to them as discriminating variables, as the classifier makes predictions based on the discriminating power of these features.

Let $x_1, x_2, ...x_n$ be n number of features where each feature is a measure of some attribute of the data. We define the feature vector X to be of the form

$$X = [x_1, x_2, ..., x_n] \tag{3.1}$$

The feature vector is a fundamental component of machine learning algorithms, as it represents the input data that is used to train and test the model. In addition to the feature vector, machine learning algorithms require a set of corresponding output labels representing the target variable the algorithm is trying to predict. Together, the feature vector and output labels form the training data set used to train the machine learning model.

It is important to carefully select the features in the feature vector, as this can greatly impact the performance of the machine learning algorithm. Features should be relevant to the problem being solved and should have good discriminatory power to distinguish between different classes or categories. In some cases, it may be necessary to preprocess or transform the features in order to make them more informative or to reduce their dimensionality.

Overall, the choice and quality of features is a crucial aspect of machine learning, as it directly impacts the accuracy and effectiveness of the resulting models.

A **classifier** machine learning model is a type of algorithm that takes a set of input data, typically represented as a feature vector, and predicts the category or class to which the data belongs. The classifier model is trained using a labelled dataset, where each data point is associated with a known class label. The training process aims to learn a function that maps the input features to the corresponding output labels.

Many types of classifier models exist, including logistic regression, decision trees, support vector machines (SVMs), and neural networks. Each of these models has its own strengths and weaknesses, and the choice of model will depend on the specific problem being solved and the characteristics of the input data.

Once the classifier model has been trained, it can be used to make predictions on new, unlabelled data. The model takes the feature vector representing the input data and applies the learned function to predict the most likely class label. The accuracy of the classifier model is typically evaluated on a separate test dataset, which was not used during training, to ensure that the model can generalize to new data.

The **training dataset** is used to train the machine learning model. It consists of a set of labelled data points, where a feature vector and an associated class label or output value represent each data point. During the training process, the machine learning algorithm uses the training data to learn a mapping from the input features to the output labels by adjusting the parameters of the model based on the training examples.

On the other hand, the **testing dataset** is used to evaluate the performance of the trained machine learning model. It consists of a separate set of labelled data points, which were not used during training. The testing data is fed into the trained model, and the predicted output labels are compared to the true labels in the testing dataset. The accuracy and performance of the model can then be evaluated based on how well it predicts the labels for the testing data.

It is important to use separate training and testing datasets in order to avoid overfitting, which occurs when a machine learning model becomes too closely tuned to the training data and performs poorly on new, unseen data. By evaluating the model's performance on a separate testing dataset, we can ensure that the model can generalize well to new data and make accurate predictions in real-world applications.

3.2 Neural Networks

Neural Networks are a type of machine learning algorithm modeled after the structure and function of the human brain. We will be using one of the most popular types of neural networks, Multilayer Perceptron (MLP). MLP is a feedforward network consisting of an input layer, one or more hidden layers, and an output layer. Each neuron in the input layer receives input from the data and sends it to the neurons in the next layer. A presentation of the model is shown in the figure 3.1. Each neuron in the hidden layer takes the input and applies a nonlinear activation function, such as the sigmoid or ReLU function (Shown in figure 3.2), to produce an output that is then sent to the next layer. The output layer takes the final output of the last hidden layer and produces a final output that can be used for classification or regression tasks.

$$\text{ReLU}(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{3.2}$$

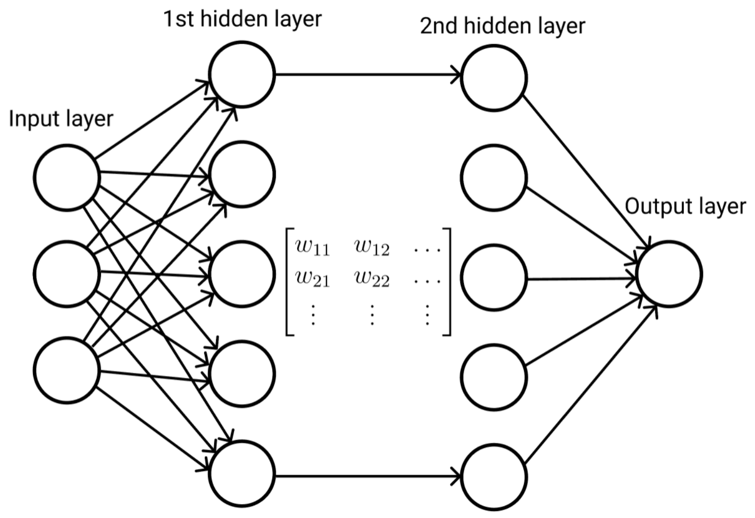
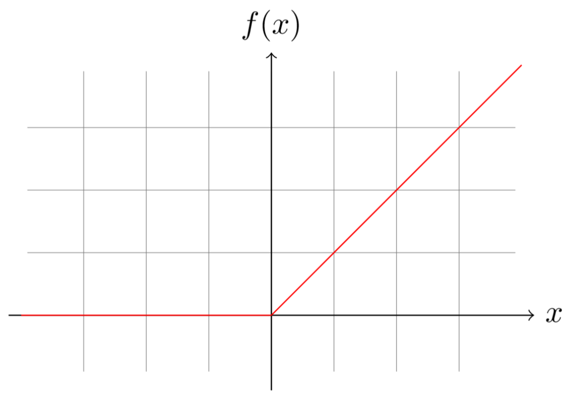

1st hidden layer
2nd hidden layer
Input layer
Output layer
$\begin{bmatrix} w_{11} & w_{12} & \cdots \\ w_{21} & w_{22} & \cdots \\ \vdots & \vdots & \vdots \end{bmatrix}$

Fig. 3.1: Representation of a Multilayer Perceptron with an input layer, 2 hidden layers, and an output layer.



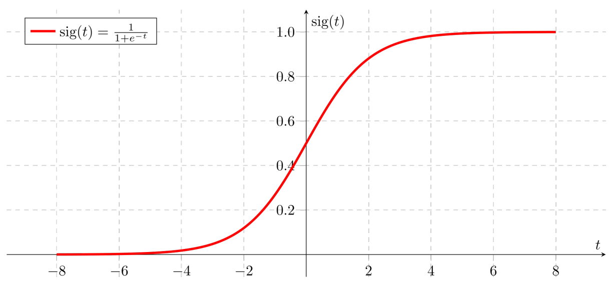


Fig. 3.2: Rectified Linear Unit (ReLU) on the left and Sigmoid on the right. These are common activation functions. In our case, ReLU is used in all the layers except for the last one, a sigmoid.

$$\mathrm{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

Figure 3.3 shows a visualization of the weights and biases used in a Multi-Layer Perceptron (MLP). The colours in the figure represent values that correspond to the colour map shown beside each heatmap. Each heatmap on the top represents a matrix, and the ones on the bottom represent a vector. The values in the matrix represent the weights used in the network, and the values in the vector represent the biases used in the network.

To perform prediction on a new data point, the input parameters are first organized in a vector. This vector is then multiplied by the weight matrix and the biases are added to the result. This process is repeated for every layer in the network. Before moving to the next matrix multiplication, the values in the resulting vector are passed through an activation function, which helps to introduce non-linearity into the network. The output of the last layer is the final prediction for the input data point.

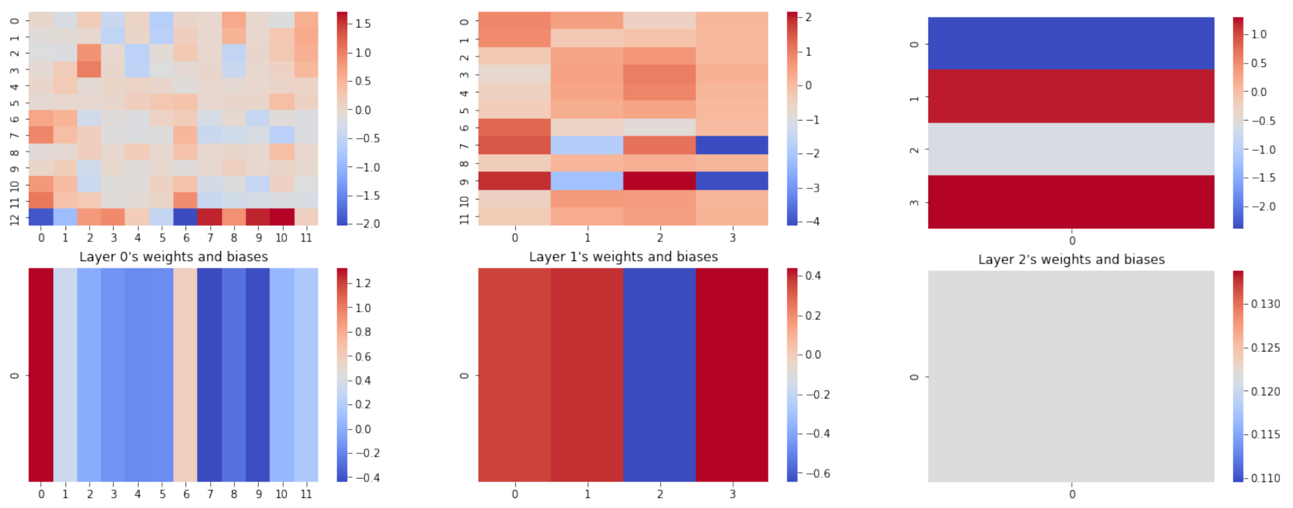


Fig. 3.3: Heatmap representation of an MLP

The claim is that a big enough neural network can be used to imitate any function if the weights and biases are set correctly.

The Universal Approximation Theorem: The theorem states that a feedforward neural network with a single hidden layer, containing a finite number of neurons, can approximate any continuous function on a compact set to an arbitrary degree of accuracy, provided that the activation function used in the neurons is non-constant, bounded, and monotonically-increasing. Simply said, an MLP with a single hidden layer is capable of approximating any complex functions given enough nodes, and if the values of the matrix and vector are set correctly. These values can be found out it just fitting these parameters to data. In the next subsection, we look at gradient descent, one of the most commonly used methods to set these values.

3.2.1 Gradient Descent - Training Algorithm

Gradient descent is a popular choice because it is computationally efficient and can be applied to many models. The goal of gradient descent is to find parameters that give a small value for the loss function for a given dataset. We start with a random set of weights and biases, at each iteration, the gradient of the cost function with respect to the weights and biases is computed and used to update the values in the direction of the negative gradient. This process is repeated until convergence. Mathematically we can write it like this.

$$\Theta = \Theta - \alpha \Delta J(\Theta) \tag{3.4}$$

Here, Θ are the weights and biases of the model, α is the learning rate that controls the step size, and $\Delta J(\Theta)$ is the gradient of the loss function.

$$\Delta J(\Theta) = \left[\frac{\delta J(\Theta)}{\delta \Theta_1}, \frac{\delta J(\Theta)}{\delta \Theta_2}, \frac{\delta J(\Theta)}{\delta \Theta_3}, \cdots, \frac{\delta J(\Theta)}{\delta \Theta_n} \right] \tag{3.5}$$

Where n is the number of parameters in the model. In each iteration of gradient descent, the algorithm updates the parameter values by subtracting the gradient of the cost function with respect to the parameters multiplied by the learning rate α. This updates the parameter values in the direction of the steepest descent of the cost function until a good minimum is found.

3.2.2 Tensorflow

TensorFlow is a powerful machine learning library that is widely used in the development of multilayer perceptron (MLP) models. It provides a flexible and scalable platform for creating, training, and deploying machine learning models, including MLPs. TensorFlow is particularly well-suited for deep learning applications, which require the use of complex neural network architectures with many layers. With TensorFlow, developers can easily design and implement custom MLP models, optimize their performance through hyperparameter tuning and regularization, and train them using advanced optimization algorithms such as gradient descent. TensorFlow also provides a variety of tools for visualizing and interpreting model outputs, making it a valuable tool for machine learning researchers and practitioners. In the broader context of machine learning, TensorFlow is just one of many powerful libraries and tools that are available to developers, and it is used in conjunction with other popular libraries such as Scikit-learn, Keras, and PyTorch.

Chapter 4

Neural Network Analysis

4.1 Data set and selection

For this analysis, we used the data collected in 2015 during the Run2 period at the Large Hadron Collider. These are minimum-bias pp collisions at $\sqrt{s} = 5.02$ TeV.

Data: LHC15n Run list (25 Hadron PID runs - recommended by DPG): 244628, 244627, 244626, 244619, 244618, 244617, 244542, 244540, 244531, 244484, 244483, 244482, 244481, 244480, 244456, 244453, 244421, 244416, 244377, 244364, 244359, 244355, 244351, 244343, 244340.

Simulation: LHC18c8a (anchored to LHC15n).

4.1.1 Track Selection Cuts

We have used the following track cuts. These are some standard cuts used in the ALICE experiment to select primary tracks.

1. $p_T > 0.15$ GeV/c^2
2. $|y| < 0.5$
3. $|\eta| < 0.8$

4.2 Analysis description

We reconstruct $f_0(980)$ using the $f_0(980) \rightarrow \pi^+\pi^-$ hadronic decay channel with a branching ratio of 46%. We identified charged pions using the Time Projection Chamber ($|\sigma_{\mathrm{TPC}}| < 2.0$) and Time Of Flight ($|\sigma_{\mathrm{TOF}}| < 3.0$) detectors. Simulations of the same collision system were used to generate data for training our model. Pions were identified using the PID provided, and a table was generated containing all possible pairs of unlike charged pions for every event individually with kinematic observables, namely 2 momentum vectors, 2 angular coordinates, and the rapidity for both pions. The table also included the invariant mass of the $f_0(980)$ assuming the pions were decay daughters, as well as a boolean value indicating if they were actually decay daughters of the same $f_0(980)$, labelledl 'is_f0_pair'. Our goal was to set the parameters of the MLP such that given the kinematic variables as input, it would output either 1 or 0 depending on the value of 'is_f0_pair'. This MLP can then be used on real data to label weather a pion pair is from the background or is coming from a $f_0(980)$.

px1	px2	py1	py2	pz1	pz2	eta1	eta2	phi1	phi2	theta1	theta2	invmass
0.235	1.60	-0.04	0.82	-0.07	-0.88	-0.31	-0.47	3.349	0.474	1.884	2.029	1.377
2.852	-7.83	-0.95	-2.34	-0.20	-1.85	-0.06	-0.22	3.463	3.432	1.638	1.794	0.860
0.103	-0.07	0.700	-0.27	-0.50	-0.15	-0.66	-0.53	1.717	4.461	2.191	2.079	0.933
.2885	0.686	-0.36	-0.63	0.004	0.739	0.009	0.724	5.387	5.535	1.561	0.902	0.579

Table 4.1: Representation of the unlike charged pion pairs table (The decimal places have been decreased to show all columns in one go). Out goal is to classify which row corresponds to a $f_0(980)$ (signal), and which doesn't (background).

4.2.1 Loss Function

In general, when fitting a model, a loss function such as Mean Squared Error (MSE) is used. However, for binary classification problems, a well-known loss function that performs better is Binary Cross-Entropy. This loss function can be calculated for a given set of inputs $\vec{X}$ and corresponding true labels (from simulation) $\vec{Y}$ using the following formula:

$$J = -[\vec{Y} \cdot \log(\text{MLP}(\vec{X})) + (\vec{1} - \vec{Y}) \cdot \log(\vec{1} - \text{MLP}(\vec{X}))] \tag{4.1}$$

Here, J represents the loss, $\vec{Y}$ is a vector of labels consisting only of 1s and 0s, and the output from the MLP is a vector of real numbers between 0 and 1, each corresponding to an output for $\vec{X}$. As can be seen, when the predictions are equal to the labels, the loss reaches 0.

4.2.2 Preparation of data for training and testing

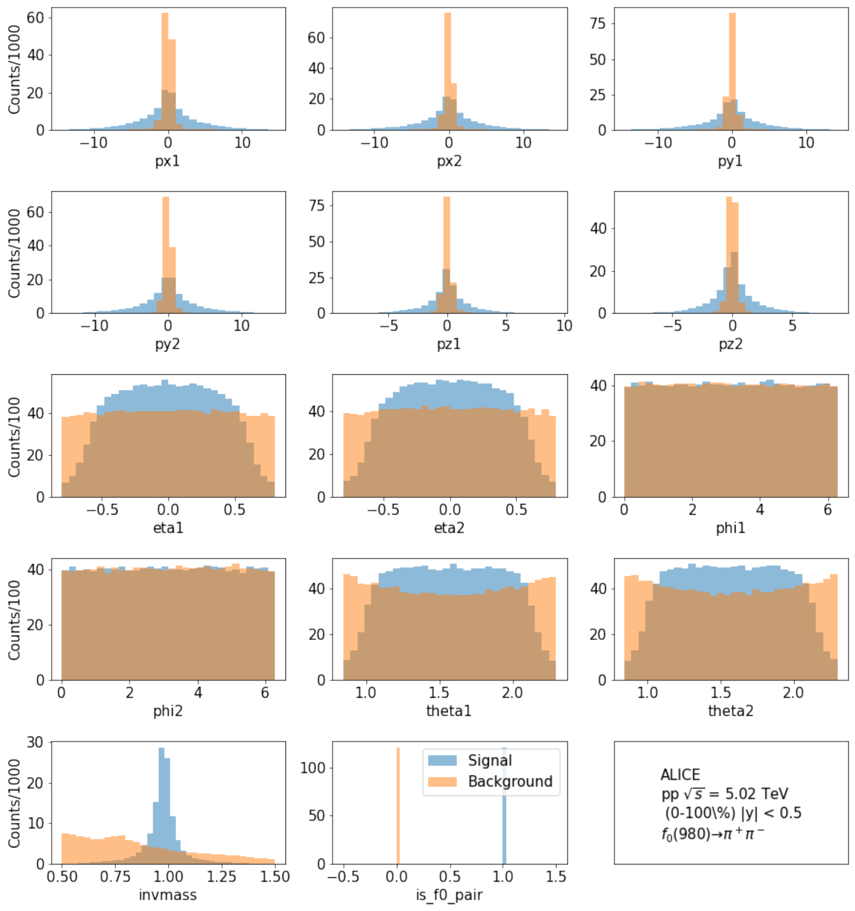


Fig. 4.1: Distributions of input features.

Once the labelledl data table was prepared, it was observed that the number of rows for the background was several orders of magnitude higher than that for signal ($f_0(980)$). This was expected. To ensure that the model does not bias towards labelling everything as background, it was necessary to balance the data by having an equal number of signal and background data points. After balancing the classes, the input distributions were compared. Figure 4.1 shows the distributions corresponding to the columns in the table after balancing.

The data was then split into training (80%) and testing (20%) datasets. The reason for this split and its significance is explained in the Terminologies section. In the next section, we will train and optimize an MLP.

4.3 Model Training and Optimization

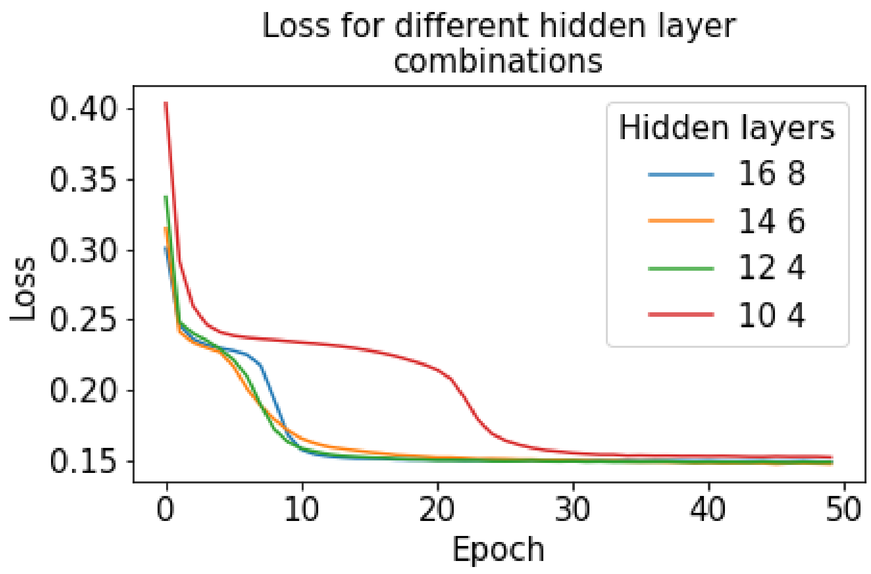


Fig. 4.2: Loss as a function of training for models with different number of nodes in the 2 hidden layers. One can observe that for a smaller model it takes more iterations to decrease the loss.

We decided to use 2 layers in our model. To determine the number of nodes in a layer, we started by trying out a few sizes. Figure 4.2 shows how the model's loss decreases as we train it. We can see that larger models learn quicker. However, if we use an arbitrarily large model, the computational costs for training and using the model increase exponentially. Hence, we try to get a smaller model and train it for a longer time. To decide on the number of nodes we can calculate the importance of the k^{th} node in the l^{th} layer using the following formula:

$$\text{Importance}(l, k) = \sum_{i=0}^{i=m} |w_{ki}^{l}|, \tag{4.2}$$

where w_{ki}^{l} is the weight at (k, i) index from the l^{th}-layer matrix, and m is the number of columns in this matrix. We have taken the sum of absolute values of the weight matrix that a particular node is getting multiplied to. The weights for the 12-4 model are shown in heatmap representation in Figure 4.3. The importance of each feature and node is shown in a bar plot in Figure 4.4. We normalized the values with the largest value in each table because the importance must be seen relative to nodes in the same layer. We found that z-momenta and ϕ are not very important, and only 4 nodes from the 1st hidden layer are significant, as only 4 of them have a relative importance greater than 0.1. We then tried a 4-4 model and trained it for double the number of epochs. The heatmap representation of the 4-4 model is shown in Figure 4.5. All the nodes have significant importance values, so we chose this model for our classification.

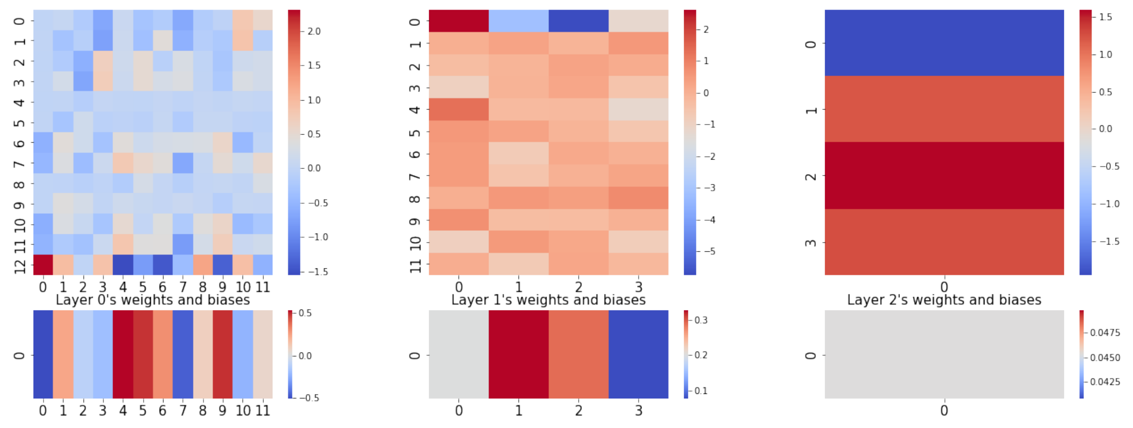


Fig. 4.3: Heatmap representation of the model with hidden layers containing 12 and 4 nodes respectively. Layer 0 is the operation on the input layer and so on.

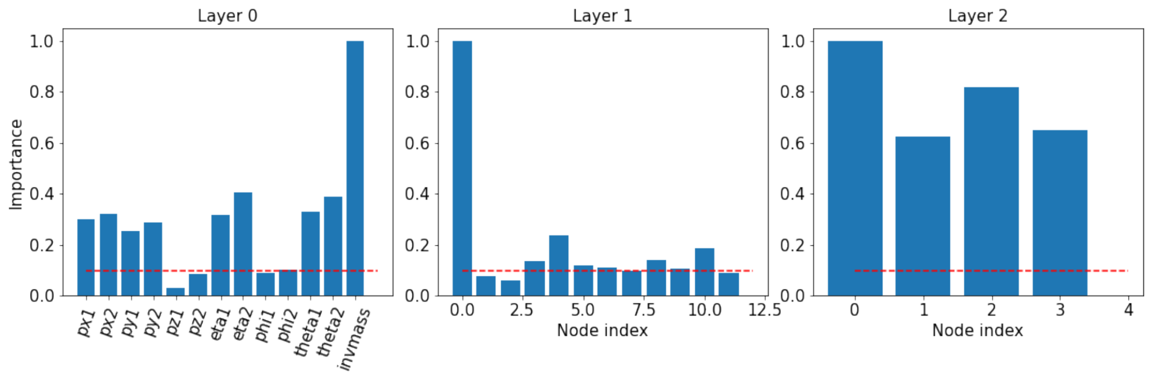


Fig. 4.4: Relative importance of each node for every layer. Layer 0 corresponds to the input layer. The red dotted line represents 0.1. In the input, we see that z-momenta and ϕ are not very important. In the 1st hidden layer, hardly 4 nodes are actually important. In the last hidden layer, all the nodes are important.

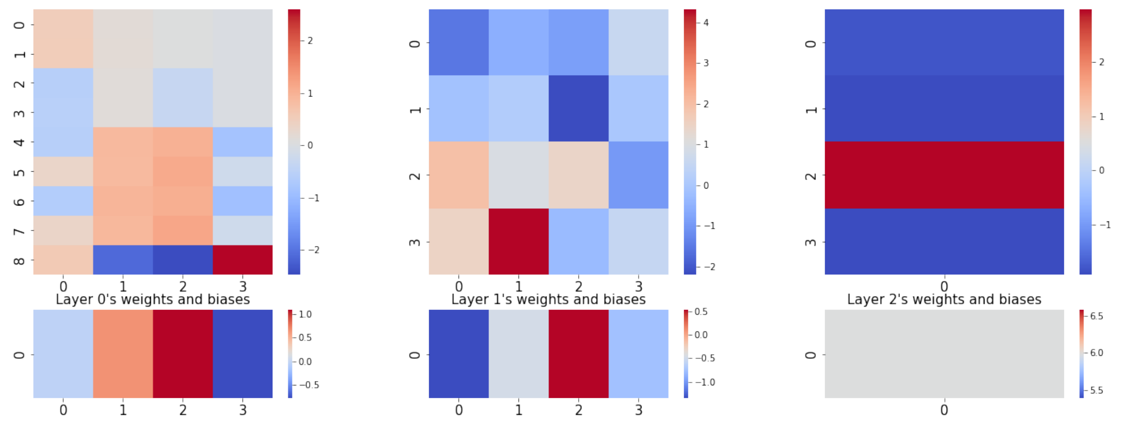


Fig. 4.5: Heatmap representation of optimized and trained MLP with 2 hidden layers having 4 nodes each. z-momenta and ϕ have been removed from the input.

Next, we evaluated the model's performance on testing datasets. The output is shown in Figure 4.6. We now need to decide on a threshold such that the rows corresponding to the prediction value higher than the threshold are predicted as a signal, and the rows with a prediction value less than the threshold are predicted as background. We calculated the accuracy as the number of correct predictions divided by the total number of data points. The y-axis on the right-hand side shows the accuracy. The red line shows the accuracy as a function of the threshold. We see that maximum accuracy is achieved when the threshold value is 0.5. Now that the model is trained and optimized, we will use it to classify the signal and background on real data and make measurements.

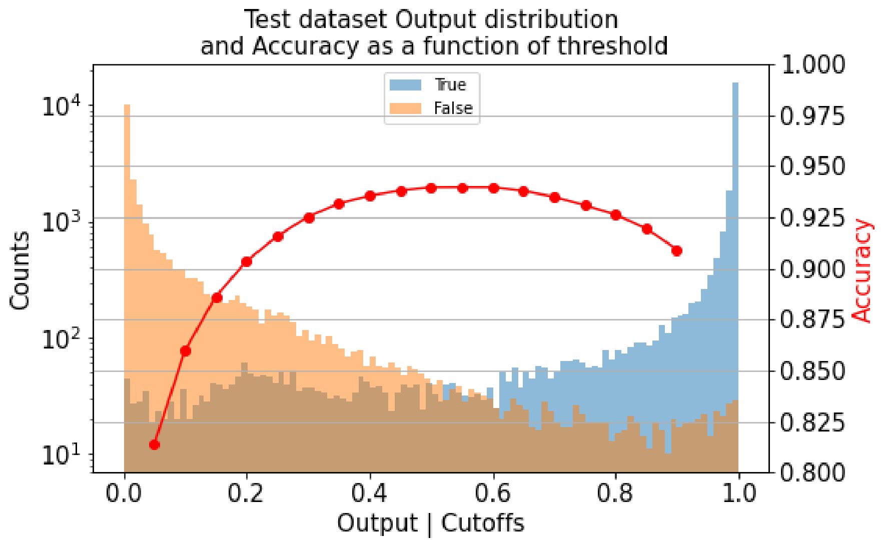


Fig. 4.6: Distribution of neural network output on test data signal and background. The red line represents the accuracy as a function of the threshold, we see that accuracy peaks when 0.5 is taken as the threshold.

4.4 Application

One by one each row is passed through the 4-4 model and the predictions are stored in another column. We now apply cuts on the value of prediction from 0.1 to 0.9 and look at the invariant mass distributions. The invariant mass distributions are shown in figure 4.7 as red hollow dots. We see that the shape of the peak is distorted when we take 0.2 as the threshold and gets better as we increase the threshold. We fit the invariant mass distributions with a model consisting of the relativistic Breit-Wigner distribution with mass-dependent width and a poly2 for any remaining background. The Breit-Wigner can be written as follows.

$$BW(m) = \frac{\sqrt{m^2\Gamma^2(m)}}{(m^2 - M^2)^2 + m^2\Gamma^2(m)} \tag{4.3}$$

where m is the invariant mass of the particle, M is the mass of the particle, and $\Gamma(m)$ is the mass-dependent width of the particle, given by:

$$\Gamma(m) = \Gamma_0 \frac{q}{q_0} \left(\frac{M}{m}\right)^{2J+1} \frac{\sqrt{m^2 - (m_1 + m_2)^2}\sqrt{m^2 - (m_1 - m_2)^2}}{\sqrt{M^2 - (m_1 + m_2)^2}\sqrt{M^2 - (m_1 - m_2)^2}} \tag{4.4}$$

where Γ_0 is the width at the resonance peak, q and q_0 are the momentum of the particle at m and M, respectively, m_1 and m_2 are the masses of the decay products, and J is the spin of the particle.

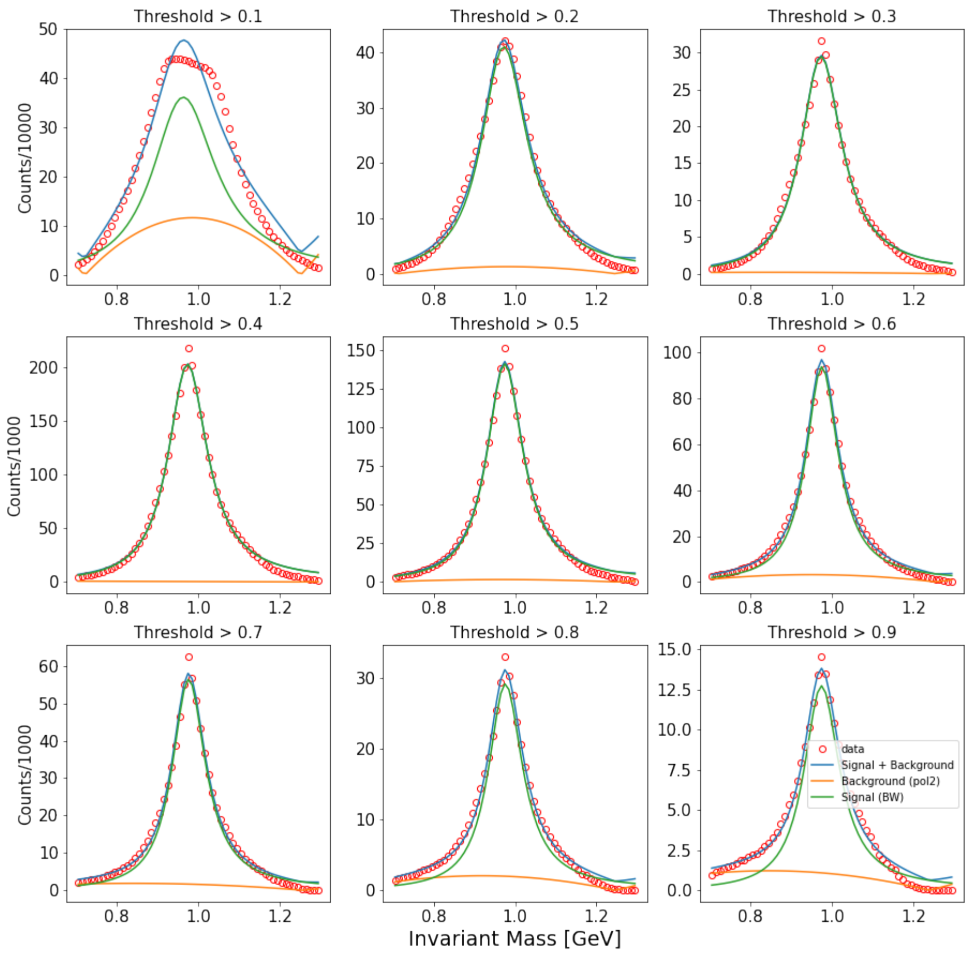


Fig. 4.7: Invariant mass distributions of the signal as predicted by the 4-4 model for a given threshold. Data points are in red, the blue line is the total model, the green line is the signal peak, and the orange line is the background.

We do not need to worry about other resonances or decays as we have to with the traditional method, this is because the model has been trained to label everything that is not $f_0(980)$ as background. The figure 4.7 shows fits of the model for different thresholds.

In figure 4.8 we see the distribution of the fit parameters as a function of threshold. It is seen that the parameters which describe the shape of the peak stabilise after 0.5. This is an indication that enough background has been removed. We also see that the width has stabilised to around 100 GeV/c^2 which is in agreement with PDG accepted range of 10 to 100GeV/c^2, although using the invariant mass method the width comes around 50 GeV/c^2 [14], this could be because we are using a completely different analysis method. In the next section, we will see which p_T range is resulting in a good signal.

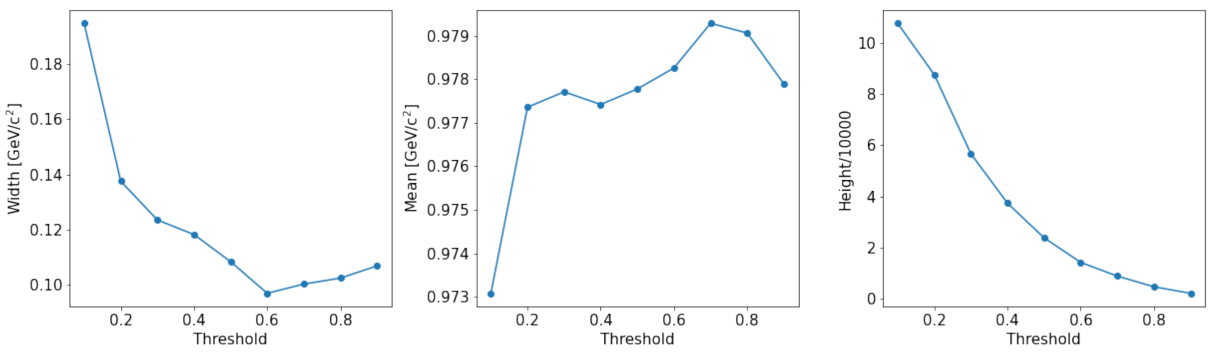


Fig. 4.8: Distribution of fit parameters of the signal as a function of threshold. We see that the shape parameters stabilise after 0.5.

4.4.1 p_T cuts

We analyzed the invariant mass distributions of the signal for different p_T ranges using a threshold of 0.5, as shown in Figure 4.9. Our findings indicate that a good signal peak is only observed in the transverse momentum range of $1.0 < p_T < 5.0$ GeV. The thin peak in the low p_T region is expected due to a large amount of background and the inability of detectors to measure below $p_T < 0.15$ GeV. However, the high p_T region shows poor distributions, indicating an issue with the training data.

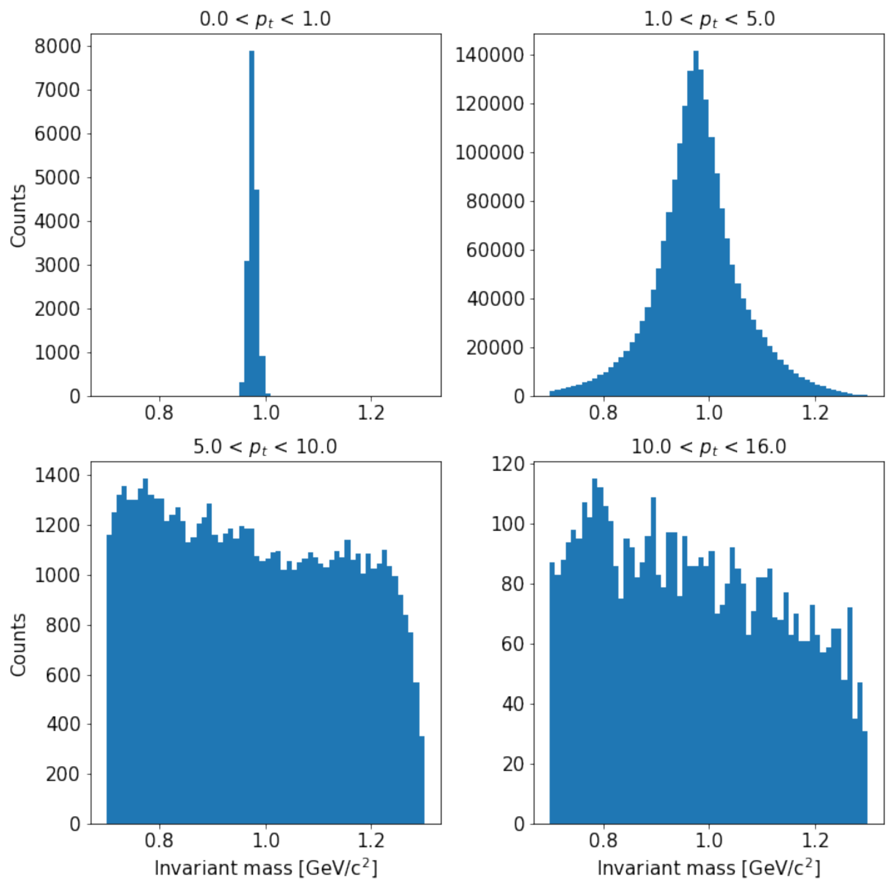


Fig. 4.9: Invariant mass distribution of signal for different transverse momentum ranges.

The original p_T distribution of the background to signal is presented in the left plot in figure

4.10. We observed that the signal p_T distribution is flat since the p_T distribution of $f_0(980)$ was not well-studied during the simulations, and a flat p_T distribution was injected. However, the challenge arises when balancing the signal and the background, as seen in the right plot of Figure 4.10. In the high p_T region, there is essentially no background due to a considerable amount of background being in the low p_T region. Consequently, the model learned that if p_T is greater than 5 GeV, it is always a signal. We attempted to sample from both classes in small p_T bins to balance in p_T. However, this resulted in other issues since other items in the background ($f_2(1270)$) were also inserted with a flat p_T distribution for the same reasons. Therefore, we decided to abandon the efforts to balance p_T and proceeded with making a measurement in the $1.0 < p_T < 5.0$ range.

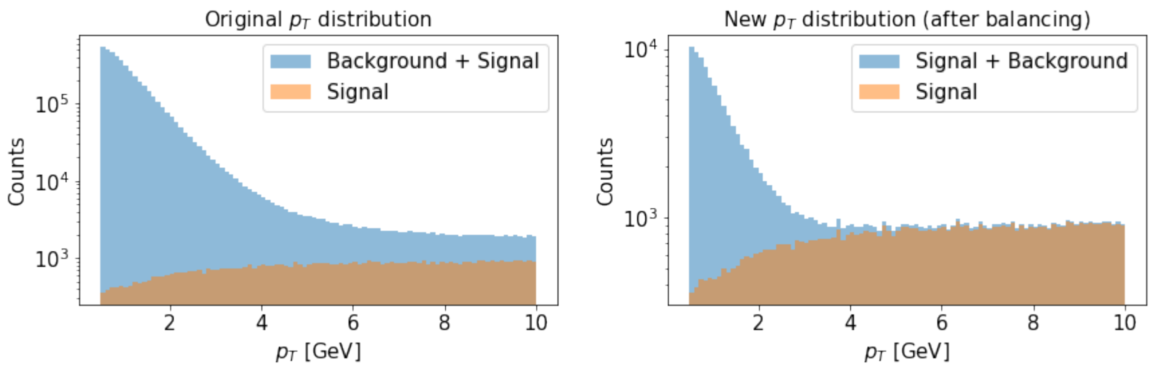


Fig. 4.10: The effect of balancing the classes on p_T distribution of signal and background.

The invariant mass distributions for different thresholds in the transverse momentum range of 1 to 5 GeV are shown in Figure 4.11. We observed that the relevant fit parameters stabilize after 0.5. Moreover, we observe that the fluctuations have decreased, indicating good classification. Before making the measurement and comparing it with the results from the Invariant Mass Analysis, we would like to explore the possibility of using a weighted loss function when the background is even larger.

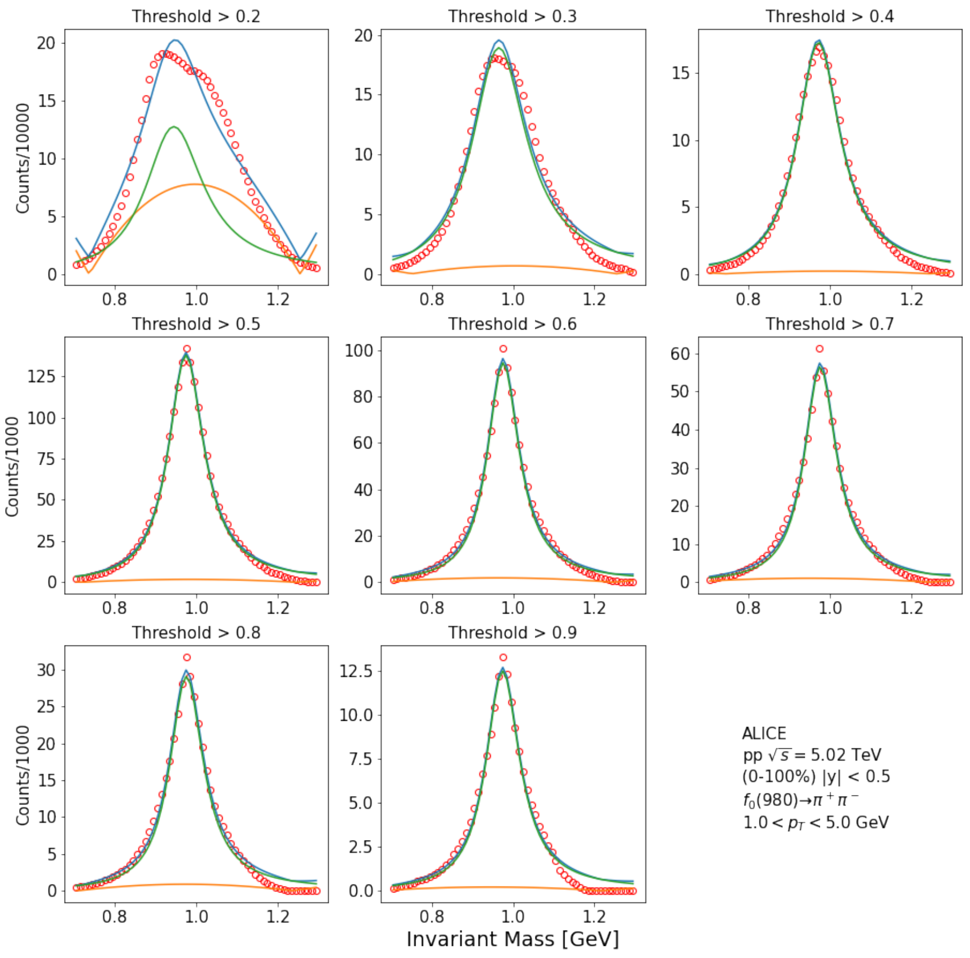


Fig. 4.11: Invariant mass distributions of the signal as predicted by the 4-4 model for a given threshold in the transverse momentum range of $1 < p_T < 5$ GeV. Data points are in red, the blue line is the total model, the green line is the signal peak, and the orange line is the background.

4.4.2 Weighted Loss Function

When the background is high, as in heavy-ion collisions, it is necessary to remove it more aggressively. To achieve this, we propose using a weighted loss function. One way to implement this is to multiply a number to the term that calculates the loss due to background misclassification. This will encourage the model to learn to remove the background more effectively.

The loss function now looks like this:

$$J = -[w\vec{Y} \cdot \log(\text{MLP}(\vec{X})) + (\vec{1} - \vec{Y}) \cdot \log(\vec{1} - \text{MLP}(\vec{X}))] \tag{4.5}$$

Here, w is a positive real number we call the weight. We trained three models with the same architecture using loss functions with weights of 1, 4, and 10. Figure 4.12 shows the output distributions on test data for signal and background. As can be seen, the background distribution is pushed further toward 0 for higher weight values. The signal distribution is also pushed slightly towards 0 because there are limits to the information present in the input variables. We observed the invariant mass distribution for different thresholds using the three models on experimentally obtained data from pp collisions. However, the distributions are unsatisfactory, and more work is required to achieve the desired results.

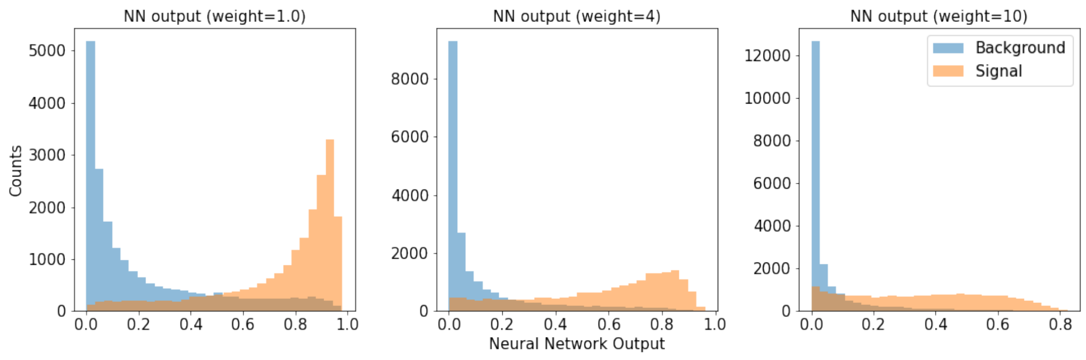


Fig. 4.12: Output distribution of 3 MLPs with custom loss function with the weights 1, 4, and 10. We see that more weight leads to more aggressive background removal. This happens at the cost of some signal being marked as background.

4.4.3 Results

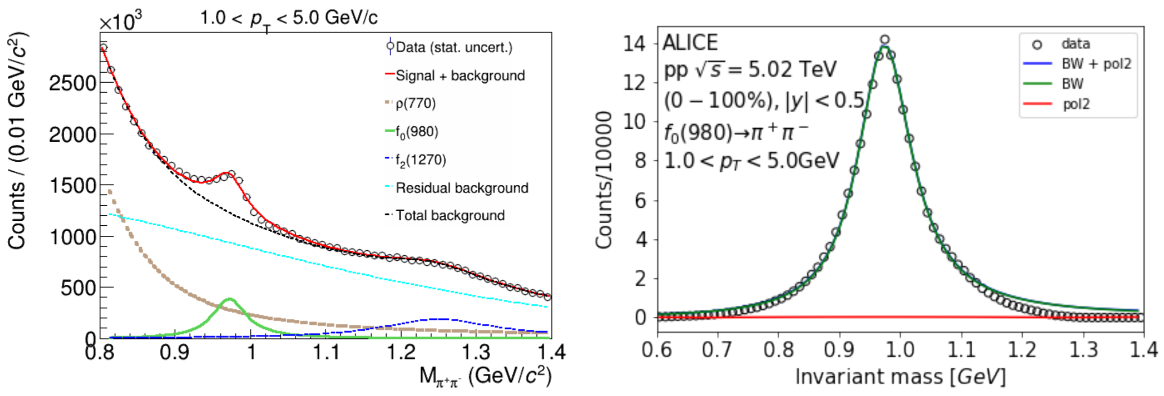


Fig. 4.13: Right side is the signal measured using a Neural network, while the one on the left is using Invariant Mass analysis.

The left plot in figure 4.13 shows the invariant mass distribution used to extract the signal using Invariant Mass analysis, on the right is the invariant mass distribution after removing the background using our neural network. The width comes out to be 106 GeV/c^2 which, with the systematic error of 10%, would lie within the accepted range from PDG which is 10 to 100 GeV/c^2. The counts of signal and background calculated by integrating the fit functions using the two methods are shown in table 4.2. We see that the signal-to-background ratio has increased by a factor of ∼1,000, and the significance is improved by a factor of ∼3.

Analysis method	**# of events**	**Signal**	**Background**	**S/B**	**Significance**
IMD ($\pm 5\sigma$)	8.79×10^8	3.90×10^5	1.04×10^7	3.74×10^{-2}	36.05
IMD ($\pm 3\sigma$)	8.79×10^8	3.72×10^5	5.31×10^6	7.02×10^{-2}	47.51
Neural Network ($\pm 5\sigma$)	8.3×10^7	2.05×10^4	120	170.55	142.64

Table 4.2: Comparison of Neural Networks with standard Invariant Mass analysis (Like background subtraction). In $1 < p_T < 5$ GeV we see that the signal-to-background ratio has increased by a factor of ∼1,000, the significance is also improved by a factor of ∼3. The significance is calculated using the formula $S/\sqrt{S+B}$. The signal for the IMD is scaled to match the number of events in this analysis for calculating the significance.

Chapter 5

Summary, Conclusions, and Outlook

In this 10th-semester project, the effectiveness of neural networks in conjunction with invariant mass analysis was explored. The findings suggest that this approach has the potential to significantly enhance the signal efficiency (S/B) by up to 1,000 times and improve the significance three-fold within the transverse momentum range of 1 to 5 GeV. It is important to note that the successful removal of background resulted in a loss of approximately 40% of the signal, highlighting the tradeoff inherent in the methodology. The measurement of the width of the $f_0(980)$ resonance was determined to be 106 MeV/c^2, which may differ from alternative analysis techniques.

Addressing p_T artifacts in the analysis proved challenging, but a proposed solution involves assuming a p_T distribution for injected particles, such as $f_0(980)$ and $f_2(1270)$, and sampling data points from this distribution. Further investigation is required to confirm the effectiveness of this approach.

Although the utilization of weighted loss functions did not yield favorable results, the findings demonstrate the promising potential of neural network analysis in enhancing the sensitivity of invariant mass analysis. Future research should focus on refining and optimizing this technique to achieve even more accurate and precise measurements of high-energy particles and their properties.

The application of this technique holds particular relevance in the measurement of resonances during heavy-ion collisions, where backgrounds are often substantial and pose challenges to traditional methods. By effectively differentiating between signal and background events, neural networks can significantly improve the accuracy and precision of such measurements, thereby enhancing our understanding of high-energy particle properties. This approach also has broader implications for other areas of particle physics research, where accurate identification of signal events from large background sources is vital for meaningful measurements.

Appendix A

Code

In this appendix, we present the important code snippets used in the implementation of Invariant Mass Analysis. These code snippets provide additional insights into the implementation details. The code has been formatted and syntax-highlighted for clarity using the `listings` package in LaTeX.

```cpp
double background_model(double x,double b,double n,double c) {
    // double E = x-0.28;
    if (x < 0.28) {
        return 0;
    }
    double E = TMath::Power(x-0.28, n);
    return b*TMath::Sqrt(E)*TMath::Power(c, 1.5)*TMath::Exp(-c*E);
}

// double poly2(double x,double a,double b,double c) {
//     return a + b*x + c*x*x;
// }
// double poly2_helper(double *x, double *par) {
//     return poly2(x[0], par[0], par[1], par[2]);
// }

// double poly3(double x,double a,double b,double c,double d) {
//     return a + b*x + c*x*x + d*x*x*x;
// }
// double poly3_helper(double *x, double *par) {
//     return poly3(x[0], par[0], par[1], par[2], par[3]);
// }

double background_model_helper(double *x, double *params) {
    return background_model(x[0], params[0], params[1], params[2]);
}

// double breit_wigner_relativistic(double x, double median, double
   gamma)
// {
//   double mm = median*median;
//   double gg = gamma*gamma;
//   double mg = median*gamma;
//   double xxMinusmm = x*x - mm;

//   double y = sqrt(mm * (mm + gg));
//   double k = (0.9003163161571060 6*mg*y)/(sqrt(mm+y)); //2*sqrt(2)
   /pi = 0.9003163161571060 6

//   double bw = k/(xxMinusmm*xxMinusmm + mg*mg);
//   return bw;
// }

// double breit_wigner_relativistic_helper(double *x, double *params
   ) {
//     return params[0]*breit_wigner_relativistic(x[0], params[1],
   params[2]);
// }

double breit_wigner_relativistic_width_dep_mass(double x, double
   median, double gamma)
{

  double mm = median*median;
  double xx = x*x;
  double xxMinusmm = xx - mm;
  double j = 0;
  double m_pi_sq = 0.13957018*0.13957018;

  gamma = gamma * pow((xx - 4*m_pi_sq)/(mm - 4*m_pi_sq) , (2*j + 1)
     /2) * x/median;

  double mg = median*gamma;
  double gg = gamma*gamma;

  double y = sqrt(mm * (mm + gg));
  double k = (0.9003163161571060 6*mg*y)/(sqrt(mm+y)); //2*sqrt(2)/pi
     = 0.9003163161571060 6

  double bw = k/(xxMinusmm*xxMinusmm + mg*mg);
  return bw;
}

double breit_wigner_relativistic_width_dep_mass_helper(double *x,
   double *params) {
    return params[0]*breit_wigner_relativistic_width_dep_mass(x[0],
       params[1], params[2]);
}

double fit_function(double *x, double *params) {
    return
    // rho0
    params[0] * breit_wigner_relativistic_width_dep_mass(x[0],
       params[1], params[2]) +
    // f0 980
    params[3] * breit_wigner_relativistic_width_dep_mass(x[0],
       params[4], params[5]) +
    // f0 1270
    params[6] * breit_wigner_relativistic_width_dep_mass(x[0],
       params[7], params[8]) +
    background_model(x[0], params[9], params[10], params[11]);
    // poly2(x[0], params[9], params[10], params[11]);
}



double fit_function_without_980(double *x, double *params) {
    return
    // rho0
    params[0] * breit_wigner_relativistic_width_dep_mass(x[0],
       params[1], params[2]) +
    // f0 980
    // params[3] * breit_wigner_relativistic_width_dep_mass(x[0],
       params[4], params[5]) +
    // f0 1270
    params[6] * breit_wigner_relativistic_width_dep_mass(x[0],
       params[7], params[8]) +
    background_model(x[0], params[9], params[10], params[11]);
    // poly2(x[0], params[9], params[10], params[11]);
}
```

Listing A.1: Functions used for fitting

```
int make_InvMass_proj_in_pt_range(THnSparseF* PM, THnSparseF* PP,
   THnSparseF* MM, double pt_low, double pt_high, bool verbose=false)
    {
    double val_pos, val_neg, err_pos, err_neg, combined_value,
       combined_error;
    int pt_range_index;

    auto *out_file = new TFile("fit_analysis.root", "UPDATE");
    TH1D *hist_pm, *hist_pp, *hist_mm, *hist_like, *hist_signal;

    // saving the total histogram. The whole pt-range
    if (verbose) cout << "Working on pt range: " << pt_low << " - "
       << pt_high << endl;

    int pt_low_bin_index = PM->GetAxis(1)->FindBin(pt_low);  //
       getting bin index for pt_low and pt_high
    int pt_high_bin_index = PM->GetAxis(1)->FindBin(pt_high);

    PM->GetAxis(1)->SetRange(pt_low_bin_index, pt_high_bin_index);
    PP->GetAxis(1)->SetRange(pt_low_bin_index, pt_high_bin_index);
    MM->GetAxis(1)->SetRange(pt_low_bin_index, pt_high_bin_index);
    hist_pm = (TH1D*)PM->Projection(0, "E");
    hist_pp = (TH1D*)PP->Projection(0, "E");
    hist_mm = (TH1D*)MM->Projection(0, "E");

    hist_like = (TH1D*)hist_pp->Clone();
    hist_signal = (TH1D*)hist_pp->Clone();

```

```cpp

    for (int i=1; i< hist_pp->GetNbinsX(); i++) {
        val_pos = hist_pp->GetBinContent(i);
        val_neg = hist_mm->GetBinContent(i);
        err_pos = hist_pp->GetBinError(i) / val_pos;
        err_neg = hist_mm->GetBinError(i) / val_neg;

        combined_value = 2*TMath::Sqrt(val_pos*val_neg);
        hist_like->SetBinContent(i, combined_value);

        combined_error = combined_value*TMath::Sqrt(
            TMath::Power(err_pos/val_pos, 2) + TMath::Power(err_neg/
               val_neg, 2)
            );
        hist_like->SetBinError(i, combined_error);

        hist_signal->SetBinContent(i, hist_pm->GetBinContent(i) -
           combined_value);
        hist_signal->SetBinError(i, hist_pm->GetBinError(i) +
           combined_error);
    }

    // making the name with bin edge values
    std::ostringstream pt_low_string;
    pt_low_string << std::fixed << std::setprecision(1) << pt_low;
    std::ostringstream pt_high_string;
    pt_high_string << std::fixed << std::setprecision(1) << pt_high;
    TString hist_name = "Signal_" + pt_low_string.str() + "_" +
       pt_high_string.str();

    if (verbose) cout << "Signal calculation done " << hist_name <<
       endl;
    out_file->WriteObject(hist_signal, hist_name);
    out_file->Close();
    return 1;
}

// driving function
int do_pt_binning(bool verbose=true) {
    auto *file = new TFile("../all_data.root");
    auto *key = (TKey*)file->GetListOfKeys()->At(1);
    auto *list = (TList*)file->Get(key->GetName());

    THnSparseF * PM = (THnSparseF *) list->FindObject("
       F0ppData_F0_UnlikePM");
    THnSparseF * PP = (THnSparseF *) list->FindObject("
       F0ppData_F0_LikePP");
    THnSparseF * MM = (THnSparseF *) list->FindObject("
       F0ppData_F0_LikeMM");

    // pt binning
    // double pt_binning[] =
       {0,0.2,0.4,0.6,0.8,1,1.2,1.45,1.6,1.8,2,2.5,3,3.5,4,4.5,5,
        6,7,8,10,12,16};
    double pt_binning[] = {1.0, 5.0};
    int pt_bin_count = (int)sizeof(pt_binning)/sizeof(pt_binning[0])
       ;

    make_InvMass_proj_in_pt_range(PM, PP, MM, pt_binning[0],
       pt_binning[pt_bin_count-1], verbose);

    // making the histograms for individual pt_ranges
    for (int pt_range_index=0; pt_range_index < pt_bin_count-1;
       pt_range_index++) {

        double pt_low = pt_binning[pt_range_index];
        double pt_high = pt_binning[pt_range_index+1];
        make_InvMass_proj_in_pt_range(PM, PP, MM, pt_low, pt_high,
           verbose);

    }
    if (verbose) cout << "pt binning done" << endl;
    return 1;
}
```

Listing A.2: Projecting ND-histograms from AliPhysics into 1D-histograms for fitting

```
#include "model_functions.C"
#include "helper_functions.C"

int write_to_file(TFitResultPtr r, double pt_low, double pt_high,
   TString file_name, double chi2_ndf, double yield, double yield_err
   , double pair_mass_min, double pair_mass_max) {
    ofstream file;
    file.open(file_name, ios::app);

    file << pt_low << "," << pt_high << ",";

    for (int i=0; i<12; i ++) {
        file << r->Parameter(i) << "," << r->ParError(i) << ",";
    }
    file << chi2_ndf << "," << yield << "," << yield_err << "," <<
       pair_mass_min << "," << pair_mass_max << endl;
    file.close();
    return 1;
}

void fit_analysis(bool verbose=true) {
    TString output_filename("fit_result.csv");
    ofstream output_file;
    output_file.open(output_filename);
    output_file << "pt_low,pt_high,p0,p0_err,p1,p1_err,p2,p2_err,p3,
       p3_err,p4,p4_err,p5,p5_err,p6,p6_err,p7,p7_err,p8,p8_err,p9,
       p9_err,p10,p10_err,p11,p11_err,chi2_ndf,yield,yield_err,
       pair_mass_min,pair_mass_max" << endl;

    double pt_binning[] =
       {0,0.2,0.4,0.6,0.8,1,1.2,1.4,1.6,1.8,2,2.5,3,3.5,4,4.5,5,
       6,7,8,10,12,16};
    int pt_bin_count = (int)sizeof(pt_binning)/sizeof(pt_binning[0])
       ;
    cout << pt_bin_count << " pt bins" << endl;

    auto *file = new TFile("fit_analysis.root");

    double initial_parameters[12][23] =
{//         total           1          2               3
                 4            5                6              7
                8                 9              10             11
                 12               13             14              15
                16             17             18               19
                20           21          22
/*0*/    { 100.0,        100.0,       49804.746,    498.746,
   498.746,       178.,             49.,             49.,
   49.746,       49804.746,      49804.746,      44384.000,
   44384.000,      21024.000,      7464.863,       2864.863,
   2864.863,        2162.143,        877.343,        877.343,
   877.343,     100,         100       },
/*1*/    { 0.8,          0.8,         0.8,          0.8,
   0.8,            0.8,             0.8,            0.775,
   0.775,           0.775,           0.775,          0.775,
   0.775,          0.775,           0.764,          0.764,
   0.764,           0.764,            0.764,          0.764,
   0.78,        0.78,        0.764     },
/*2*/    { 0.149,        0.149,       0.149,        0.149,
   0.149,          0.149,           0.149,          0.149,
   0.149,           0.149,           0.149,          0.149,
   0.149,          0.149,           0.149,          0.149,
   0.149,           0.149,            0.149,          0.149,
   0.149,       0.149,       0.149     },
/*3*/    { 6212.475,     6782.295,    6212.475,     15000.0,
   6212.475,       8401.475,        6212.475,       6212.475,
   6212.475,        6212.475,        6212.475,       1168.000,
   1168.000,       876.000,         810.300,        310.300,
   310.300,         31.536,           14.600,         14.600,
   14.600,      5.840,       5.840     },
/*4*/    { 0.965,        0.973,       0.965,        0.974,
   0.965,          0.965,           0.965,          0.965,
   0.965,           0.965,           0.965,          0.965,
   0.965,          0.965,           0.964,          0.964,
   0.964,           0.964,            0.964,          0.964,
   0.964,       0.964,       0.964     },
/*5*/    { 0.055,        0.055,       0.055,        0.055,
   0.055,          0.055,           0.055,          0.055,
   0.055,           0.055,           0.055,          0.055,
   0.055,          0.055,           0.055,          0.055,
   0.055,           0.055,            0.055,          0.055,
   0.055,       0.055,       0.055     },
/*6*/    { 9193.854,     3153.43,     9193.854,     11000.0,
   9193.854,       7893.854,        9193.854,       9193.854,
   9193.854,        9193.854,       9193.854,       7592.000,
   7592.000,       3504.000,       1657.316,       657.316,
   657.316,         50.224,          17.520,          17.520,
   17.520,       9.344,       9.344    },
/*7*/   { 1.257,         1.257,        1.257,         1.257,
   1.257,          1.257,          1.257,          1.257,
   1.257,           1.257,          1.257,          1.257,
   1.257,          1.257,          1.257,          1.257,
   1.257,           1.257,           1.257,           1.257,
   1.257,       1.257,       1.257    },
/*8*/   { 0.187,         0.1867,       0.187,         0.187,
   0.187,          0.187,          0.187,          0.187,
   0.187,           0.187,          0.187,          0.187,
   0.187,          0.187,          0.187,          0.187,
   0.187,           0.187,           0.187,           0.187,
   0.187,       0.187,       0.187    },
// /*9*/   { 143521.504,   3212640,     143521.504,   1430521.0,
   143521.504,     143521.504,     143521.504,     143521.504,
   143521.504,      143521.504,     143521.504,     233600.000,
   400000,         46720.000,      23441.935,      15006,
   15006,          6601.244,        3097.244,       2097.244,
   3097.244,    1345.244,   1345.244 },
// /*10*/   { 30000.543,   70000.322,   300000.543,   -2000000.0,
   3.543,          4.043,          3.543,          3.543,
   3.543,           3.543,          3.543,          3.543,
   -400000,          3.543,          1.839,          1.9,
   1.9,             1.839,           1.839,           1.839,
   1.839,       1.3,        1.67     },
// /*11*/   { 1000000,     1000000,     1000000,      700000,
   -3.543,          -4.043,         -3.543,         -3.543,
   -3.543,          -3.543,        -3.543,        -3.543,
   70000,          -3.543,        -1.839,        -1.9,
   -1.9,             -1.839,         -1.839,          -1.839,
   -1.839,      -1.3,         -1.67      },
/*9*/   { 1e+6,          2e+5,         1.4e+5,       1.4e+5,
   1.4e+5,         1.3e+5,         1.3e+5,         1.3e+5,
   143521.504,      143521.504,      143521.504,      233600.000,
   2e+5,           46720.000,      23441.935,      15006,
   15006,          6601.244,        3097.244,       2097.244,
   3097.244,    1345.244,   1345.244 },
/*10*/   {  1,            1,            1,            1,
   1,              1,              1,                1,
   1,               1,              1,              1,             1,
                 1,              1,              1,              1,
                1,              1,              1,              1,
             1,        1       },
/*11*/   { 1,             1,            1,            1,
   1,              1,              1,                1,
   1,               1,              1,              1,             5,
                 1,              1,              1,              1,
                1,              1,              1,              1,
             1,        1       },
    };
```

```cpp



    const char *parameter_names[12] = {
        "rho_0_height", "rho_0_mass", "rho_0_width",
        "f0980_height", "f0980_mass", "f0980_width",
        "f01270_height", "f01270_mass", "f01270_width",
        "bg_b", "bg_n", "bg_c"
    };


    int low_delta_state=0, high_delta_state=0; // -1 to subtract; 0 to not do anything; 1 to add
    /*v*************************************************************/
    // CHECKING STABILITY OF THE FIT (Part start)
    // int options[3] = {-1, 0, 1};
    // for (int a = 0; a<3; a++) {
    //     low_delta_state = options[a];
    //     for (int b = 0; b<3; b++) {
    //         high_delta_state = options[b];
    /*^*************************************************************/

    double low_range_delta = 0.03, high_range_delta = 0.03;
    // int i=2;
    // for (int pt_range_index=i; pt_range_index < i+2; pt_range_index++) {
    // for (int pt_range_index=0; pt_range_index < 10; pt_range_index++) {
    for (int pt_range_index=0; pt_range_index < pt_bin_count-1; pt_range_index++) {
        // binning and name tstring
        cout << endl << endl << "RI:" << pt_range_index << endl;
        cout << "fit_index:" << pt_range_index + 1 << endl;

        double pair_mass_min = 0.81, pair_mass_max = 1.4;

        // Ignore if not checking stability
        if (low_delta_state == 1) {
            pair_mass_min += low_range_delta;
        } else if (low_delta_state == -1) {
            pair_mass_min -= low_range_delta;
        }
        if (high_delta_state == 1) {
            pair_mass_max += high_range_delta;
        } else if (high_delta_state == -1) {
            pair_mass_max -= high_range_delta;
        }
        //

        auto *func = new TF1("model", fit_function, pair_mass_min, pair_mass_max, 12);
        double pt_low = pt_binning[pt_range_index];
        double pt_high = pt_binning[pt_range_index + 1];
        std::ostringstream pt_low_string, pt_high_string;
        pt_low_string << std::fixed << std::setprecision(1) << pt_low;
        pt_high_string << std::fixed << std::setprecision(1) << pt_high;
        TString pt_low_tstr = pt_low_string.str();
        TString pt_high_tstr = pt_high_string.str();
        TString hist_name = "Signal_" + pt_low_tstr + "_" + pt_high_tstr;
        auto *hist = (TH1D*)file->Get(hist_name);
        // hist->Rebin(2);
        if (verbose) cout << hist_name << endl;

        // setting parameters
        for (int i=0; i<12; i++) {
            func->SetParameter(i, initial_parameters[i][pt_range_index+1]);
            func->SetParName(i, parameter_names[i]);
        }
        func->SetParLimits(0, 0, hist->Integral(0, 4)); // keeping height of rho0 positive (might not always work)
        cout << "Max rho0 height:" << hist->Integral(0, 2) << endl;
        // func->SetParLimits(1, 0.47, 0.8); // rho0 mass (2\sigma of PDG)
        // func->SetParLimits(1, 0.675, 0.8); // rho0 mass
        func->FixParameter(1, 0.775); // rho0 mass
        func->FixParameter(2, 0.1491); // rho0 width
        func->SetParLimits(3, 0, hist->Integral(0, 2)); // keeping height of f0(980) positive (might not always work)
        func->SetParLimits(4, 0.93, 1.05); // f0 mass (3\sigma of PDG)
        // func->FixParameter(5, 0.055); // f0 width (Try freeing the width a little)
        func->SetParLimits(6, 0, hist->Integral(0, 2)); // keeping height of f2 positive (might not always work)
        func->SetParLimits(7, 1.2, 1.35); // f2 mass  (3\sigma of PDG + some)
        // func->SetParLimits(7, 1.2731, 1.2779); // f2 mass  (3\sigma of PDG + some)
        // func->FixParameter(7, 1.2755);
        func->FixParameter(8, 0.1867); // f2 width
        func->SetParLimits(10, 0.1, 100); // `n` the bg parameters
        func->SetParLimits(11, 0, 10); // `c` the bg parameters (function is only positive definite for 0 < c)

        /// specific configuration for specific fits are here

        // linear interpolation of mass
        // if (pt_range_index == 7) {
        //     func->FixParameter(1, 0.7326);
        // }
        // if (pt_range_index == 8) {
        //     func->FixParameter(1, 0.7342);
        // }
        // if (pt_range_index == 9) {
        //     func->FixParameter(1, 0.736);
        // }
        // if (pt_range_index == 10) {
        //     func->FixParameter(1, 0.739);
        // }
        // if (pt_range_index == 11) {
        //     func->FixParameter(1, 0.742);
        // }
        // if (pt_range_index == 0) {
        //     func->FixParameter(7, 1.2755);
        // }

        // printing initial parameters
        if (verbose) {
            cout << "Initial parameters:" << endl;
            for (int i=0; i<12; i++) {
                cout << func->GetParameter(i) << endl;
            }
        }

        // hist->SetAxisRange(0, (1.2 + 0.6*(pt_range_index>5) + 0.5*(pt_range_index > 11) + 1*(pt_range_index>14))*hist->Integral(0, 1), "Y");
        hist->SetMinimum(0.);
        hist->SetAxisRange(pair_mass_min, pair_mass_max, "X");
        hist->SetXTitle("m_{#pi^{+}#pi^{-}} (GeV/c^2)");
        hist->SetYTitle("Counts");
        TF1 *hist_func = hist->GetFunction("model");
        TFitResultPtr r = hist->Fit("model", "S");
        TF1* BW_f0_980 = make_composite_plot(r, pt_low_tstr, pt_high_tstr, hist, pair_mass_min, pair_mass_max);
        double chi2_ndf = r->Chi2()/r->Ndf();
        cout << "Chi2/ndf: " << chi2_ndf << " (" << r->Chi2() << "/" << r->Ndf() << ")" << endl;
        cout << "yield of f0(980): ";
        // get bin width of hist
        double invmass_bin_width = hist->GetBinWidth(1);
        double pt_bin_width = pt_high - pt_low;
        double pt_center = (pt_high + pt_low)/2;
        cout << "Mass Bin width: " << invmass_bin_width << endl;
        cout << "Pt bin width: " << pt_bin_width << endl;
        cout << "Pt center: " << pt_center << endl;
        // double factor = invmass_bin_width*pt_bin_width*pt_center;
        double factor = invmass_bin_width*pt_bin_width*2;
        double f0_980_yield = BW_f0_980->Integral(0.990-5*0.055, 0.990+5*0.055)/factor;

        TMatrixDSym cov = r->GetCovarianceMatrix(), cov1;
        cov.GetSub(3, 5, cov1);
        double f0_980_yield_err = BW_f0_980->IntegralError(0.990-5*0.055, 0.990+5*0.055, BW_f0_980->GetParameters(),
            cov1.GetMatrixArray())/factor;
        cout << f0_980_yield << "+-" << f0_980_yield_err << endl;

        // write_to_file(r, pt_low, pt_high, "fit_result_1_to_5.csv
           ", chi2_ndf, f0_980_yield, f0_980_yield_err, pair_mass_min
           , pair_mass_max);
    }

    /*v*********************************************************/
    // CHECKING STABILITY OF THE FIT (Part end)
    //      }
    // }
    /*^*********************************************************/


    return 1;
}
```

Listing A.3: Function used to fit the models onto data histogram. The commented sections can be used to nudge the initial fitting parameters to check stability of the fit. The fitted parameters are saved to a csv file and the yield is calculated by the method discussed in the analysis section.

```
#include "rapidcsv.h"
#include "mcefficiency_reweight.C"

int correction_and_normalization(bool verbose=true) {
    double pt_binning[] =
       {0,0.2,0.4,0.6,0.8,1,1.2,1.4,1.6,1.8,2,2.5,3,3.5,4,4.5,5,
       6,7,8,10,12,16};
    int pt_bin_count = (int)sizeof(pt_binning)/sizeof(pt_binning[0])
       ;
    cout << "pt_bin_count: " << pt_bin_count << endl;

    rapidcsv::Document fit_result_csv("fit_result.csv");
    std::vector<float> pt_low = fit_result_csv.GetColumn<float>("
       pt_low");
    std::vector<float> pt_high = fit_result_csv.GetColumn<float>("
       pt_high");
    std::vector<float> yield = fit_result_csv.GetColumn<float>("
       yield");
    std::vector<float> yield_err = fit_result_csv.GetColumn<float>("
       yield_err");

    rapidcsv::Document acceptance_efficiency_csv("
       acceptance_efficiency.csv");
    std::vector<float> acceptance_efficiency =
       acceptance_efficiency_csv.GetColumn<float>("
       acceptance_efficiency");
    std::vector<float> pt_mid = acceptance_efficiency_csv.GetColumn<
       float>("pt_low");

    if (verbose) {
        cout << "pt_low, pt_high, yield, yield_err,
            acceptance_efficiency" << endl;
        for (int i=0; i<pt_bin_count-1; i++) {
            cout << pt_low[i] << " " << pt_high[i] << " " << yield[i
               ] << " " << yield_err[i] << " " <<
               acceptance_efficiency[i] << endl;
        }
    }

    // Corrected spectra
    TH1D* corrected_spectra = new TH1D("corrected_spectra", "
       Corrected Spectra", 22, pt_binning);
    for (int i=0; i<pt_bin_count-1; i++) {
        double factor = acceptance_efficiency[i];
        if (verbose) cout << i << "acceptance: " << factor << "(" <<
            pt_mid[i] <<")" << endl;
        corrected_spectra->SetBinContent(i+1, yield[i]/factor);
        corrected_spectra->SetBinError(i+1, yield_err[i]/factor);
    }
    TFile* f = new TFile("corrected_spectra.root", "RECREATE");
    corrected_spectra->Write();
    f->Close();

    // reweighing and multiply the reweighing factor to the
       corrected spectra per pt bin < 1 only
    // mcefficiency_reweight();
    TFile *f_reweighting = new TFile("reweight.root", "READ");
    TH1D *reweights = (TH1D*)f_reweighting->Get("
       corrected_spectra_correction_i2");
    for (int i=0; i< 5; i++) {
        if (verbose) cout << i << "reweighting factor: " <<
           reweights->GetBinContent(i+1) << endl;
        corrected_spectra->SetBinContent(i+1, corrected_spectra->
           GetBinContent(i+1)*reweights->GetBinContent(i+1));
        corrected_spectra->SetBinError(i+1, corrected_spectra->
           GetBinError(i+1)*reweights->GetBinContent(i+1));
    }

    // signal loss correction
    rapidcsv::Document loss_correction_csv("loss_correction.csv");
    std::vector<float> loss_correction = loss_correction_csv.
       GetColumn<float>("loss_correction");
    pt_mid = loss_correction_csv.GetColumn<float>("pt_mid");
    // multiply the factors to the pt_spectra per pt bin
    for (int i=0; i<pt_bin_count-1; i++) {
        if (verbose) cout << i + 1 << "loss_correction: " <<
           loss_correction[i]  << "(" << pt_mid[i] << ")" << endl;
        corrected_spectra->SetBinContent(i+1, corrected_spectra->
           GetBinContent(i+1)*loss_correction[i]);
        corrected_spectra->SetBinError(i+1, corrected_spectra->
           GetBinError(i+1)*loss_correction[i]);
    }

    // Normalization
    double branching_ratio = 46.0/100.0; // pm 6%
    double f_norm = 0.7574; // pm 0.019
    double f_vtx = 0.958;
    int event_count = 897000000;
    double factor = f_norm*f_vtx/event_count/branching_ratio;
    if (verbose) cout << "Normalization factor: " << factor << endl;
    for (int i=0; i<pt_bin_count-1; i++) {
        corrected_spectra->SetBinContent(i+1, corrected_spectra->GetBinContent(i+1)*factor);
        corrected_spectra->SetBinError(i+1, corrected_spectra->GetBinError(i+1)*factor);
    }

    // open file to write to
    ofstream corrected_spectra_csv;
    corrected_spectra_csv.open("normalised_corrected_spectra.csv");


    // write the corrected_spectra to a csv file
    corrected_spectra_csv << "pt_low,pt_high,yield,yield_err" << endl;
    for (int i=0; i< pt_bin_count-1; i++) {
        corrected_spectra_csv << pt_binning[i] << "," << pt_binning[i+1] << "," << corrected_spectra->GetBinContent(i+1) << ", " << corrected_spectra->GetBinError(i+1) << endl;
    }

    TFile *f2 = new TFile("normalized_corrected_spectra.root", "RECREATE");
    corrected_spectra->Write();
    f2->Close();

    // sum up the corrected spectra*pt_bin_width
    double dNdy = 0;
    for (int i=0; i<pt_bin_count-1; i++) {
        cout << "bin " << i << ": " << corrected_spectra->GetBinContent(i+1) << " * " << pt_binning[i+1]-pt_binning[i] << " = " << corrected_spectra->GetBinContent(i+1)*(pt_binning[i+1]-pt_binning[i]) << endl;
        dNdy += corrected_spectra->GetBinContent(i+1)*(pt_binning[i+1]-pt_binning[i]);
    }
    cout << "dNdy: " << dNdy << endl;

    return 1;
}
```

Listing A.4: Correction and Normalization of p_T spectra

Bibliography

[1] S. D. Protopopescu, M. Alston-Garnjost, A. Barbaro-Galtieri, S. M. Flatté, J. H. Friedman, T. A. Lasinski, G. R. Lynch, M. S. Rabin, and F. T. Solmitz. $\pi\pi$ partial-wave analysis from reactions $\pi^+p \to \pi^+\pi^-\Delta^{++}$ and $\pi^+p \to K^+K^-\Delta^{++}$ at 7.1 gev/c. *Phys. Rev. D*, 7:1279–1309, Mar 1973.

[2] B. Hyams, C. Jones, P. Weilhammer, W. Blum, H. Dietl, G. Grayer, W. Koch, E. Lorenz, G. Lütjens, W. Männer, J. Meissburger, W. Ochs, U. Stierlin, and F. Wagner. phase-shift analysis from 600 to 1900 mev. *Nuclear Physics B*, 64:134–162, 1973.

[3] An Gu and Fuqiang Wang. Transverse momentum spectra of $f_0(980)$ from coalescence model. 6 2023.

[4] B. Abelev et al. Production of $k^*(892)^0$ and $\phi(1020)$ in pp collisions at $\sqrt{s}$=7 tev. *The European Physical Journal C*, 72(10), oct 2012.

[5] Subhash Singha. *Identified particle production in Pb-Pb and pp collisions at LHC energies*. PhD thesis, National Institute of Science Education and Research (NISER), 2014.

[6] A. Alici. The mrpc-based alice time-of-flight detector: Status andperformance. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 706:29–32, 2013. TRDs for the Third Millenium.

[7] Particle Data Group, P A Zyla, and Barnett et al. Review of Particle Physics. *Progress of Theoretical and Experimental Physics*, 2020(8), 08 2020. 083C01.

[8] R Brun, R Hagelberg, M Hansroul, and J C Lassalle. *Simulation program for particle physics experiments, GEANT: user guide and reference manual.* CERN, Geneva, 1978.

[9] P. Skands, S. Carrazza, and J. Rojo. Tuning pythia 8.1: The monash 2013 tune. *Eur. Phys. J. C*, 74(8):3024, 2014.

[10] S. Acharya et al. Evidence of rescattering effect in pb–pb collisions at the lhc through production of k(892)0∗ and (1020) mesons. *Physics Letters B*, 802:135225, 2020.

[11] ALICE luminosity determination for pp collisions at $\sqrt{s} = 5$ TeV. 2016.

[12] G. Breit and E. Wigner. Capture of slow neutrons. *Phys. Rev.*, 49:519–531, Apr 1936.

[13] Cheuk-Yin Wong and Grzegorz Wilk. Tsallis fits to pp collisions at the LHC. *Physical Review D*, 87(11), jun 2013.

[14] $f_0(980)$ production in inelastic pp collisions at $\sqrt{s} = 5.02$ TeV. Technical report, 2022. 16 pages, 5 captioned figures, 1 table, submitted to PLB, figures at http://alice-publications.web.cern.ch/node/8276.

[15] B. El-Bennich, O. Leitner, J.-P. Dedonder, and B. Loiseau. Scalar meson $f_0(980)$ in heavy-meson decays. *Phys. Rev. D*, 79:076004, Apr 2009.