# Tagging jets produced in the LHC using a Convoluted Neural Network

Mohammed Safeer Zaheer

22/03/2023

## Abstract

In this machine-learning project, I used a convoluted neural network (CNN) to accurately tag jets produced in proton-proton collisions from the LHC. I trained it on a portion of the dataset and then tested it on the rest of it. The goal was to have it, with good accuracy, classify the images based on the jet-features in the image. The best loss value recorded for my neural network was 0.43, and the best accuracy value recorded was 0.81. I also analyzed the Receiver Operating Characteristic (ROC) and the Precision-Recall (PR) curves, with the respective areas under the curves being 0.86 and 0.84. Finally, I analyzed the F1-score of my CNN, which came out to 0.80. Although there is slight overfitting in our model, the result is a pretty accurate neural network that doesn't involve a lot of computational costs and is efficient in its code.

# Contents

# 1 Introduction

## 1.1 The LHC

The Large Hadron Collider is by far the most potent particle accelerators in the world. Built by the European Organization for Nuclear Research (CERN), it operates in the Franco-Swiss border near Geneva, Switzerland. 'Large' is an apt descriptor for it, as its main body consists of a massive circular tunnel with a circumference of 27km. This tunnel is then further connected to other sub-tunnels in the LHC-complex.[1] Although the structure is quite large, it is also quite empty when operational. In fact, the tubes are actually kept at an 'ultra-high vacuum' of $10^{-13}$ atmospheres, a value even lower than the interstellar void. This is to avoid the collision of the colliding hadrons with stray particles in the tubes. The accelerator also has superconducting electromagnets which are kept cool to a temperature of 1.9 K. At this low temperature, the material in electromagnet becomes a superconductor and looses any resistance to electrical current. It is cooled via liquid helium.[1]

The main purpose of the LHC is to 'collide Hadrons'; that is, smash together protons or ions to study the ensuing explosion. This is accomplished by the introduction of two Hadrons in the accelerator. A Hadron is any particle that experiences the strong nuclear force. This requires a constitution of quarks. In our case, two protons were used, as protons are made of 2 up-quarks and 1 down-quark, making them hadrons. Once protons are introduced into the accelerator, a changing electric field is applied to the protons, with the electric field switching from positive to negative at a pre-determined frequency. [2] While accelerating, the protons are kept in their circular paths via the magnetic field of the superconducting electromagnets. As they accelerate, the magnetic field of the electromagnet needs to be increased to maintain the same path. Once the protons are accelerated to sufficient (relativistic) speeds, they are made to collide at any one of 8 particle detectors in the LHC complex. This collision generates an explosion, with many different particles (e.g., quarks, bosons, gluons etc.) being released as a result. Along with this cacophony of particles, jets are emitted as well. This is the main area of importance in this project.

## 1.2 Jets

The high-energy collision of the protons causes the constituent quarks and gluons in the protons to interact. These then hadronized, producing a spray of hadrons. The term 'jet' is used to describe these sprays as they project out in a narrow cone formation. Initially, these jets contain W-Bosons and top-quarks, but these heavy particles go on to decay into smaller particles.

The W-boson decay results in a quark and anti-quark pair. Since these can be up or down, the decay-products hadronized, resulting in 2 jets.

The Top-quark first decays to form a W-boson and a Bottom-quark. The W-boson then further decays into (up or down) quark and anti-quark pair. This results in 3 jets being detected due to the 3 species formed.

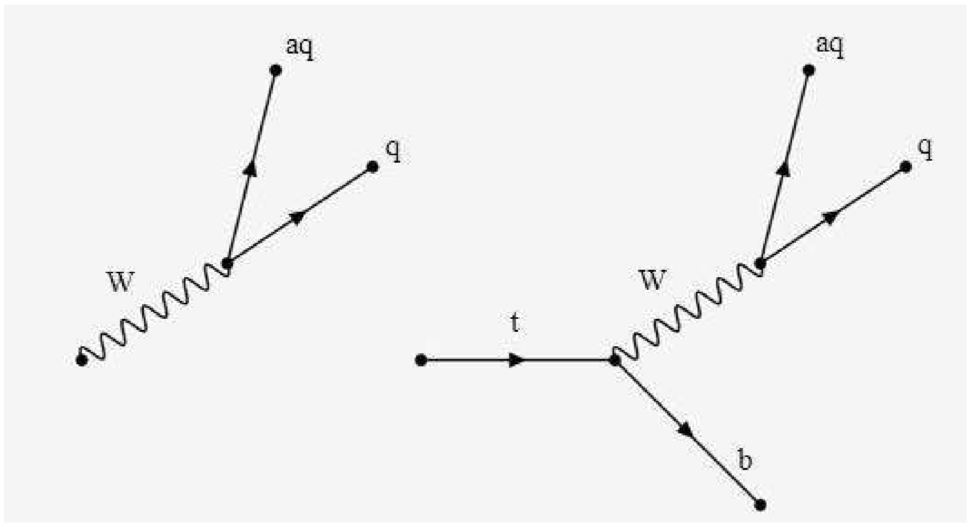

Fig 1: Feynman diagrams depicting W-boson and Top-quark decay.[3]

The QCD-jet, where QCD stands for Quantum-Chromo Dynamics, is formed due to the quarks and gluons freed in the collision interact with the dense medium of particles around them, resulting in a single beam of quarks and gluons hitting the detector, which we classify as QCD-jets. Thus, this type of jet produces only 1 jet. The energy values of these jets are then recorded via a calorimeter at LHC.

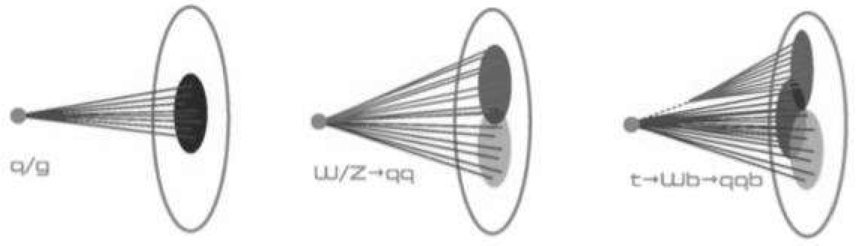

Fig 3: Illustration of how the QCD-jet (left-most), the W-jet (middle) and the Top-jet interact with the calorimeter to produce the images we are using.[4]

Looking at figure 3, we see how the three different jet-types distribute themselves. Notice how in the W-jet detection cross-sections, there is overlap between the two particles. This means that while the beams detected in general will be two separate beams, they will not always be completely separate. But, even when overlapping, the beams detected in general will still be more delocalized than QCD-jets. This is even more true for the Top-jet.

## 1.3 Convoluted Neural Networks

In this project, I will be utilizing a CNN to tag these jet-images. A Convoluted Neural Network is machine-learning tool that can be trained on images of the jets to be able to classify the jets. This is done via multiple layer of neurons in the CNN that modify the weights and biases applied to any data running through them to get better classification of targets. We can then test it with an untrained dataset to see how well it does at recognizing key-features and tagging the jets correctly [5].

The dataset given to us contains images of all three types of jets. But in this project, although all three need to be unpacked to access the data, I am actually only interested in the Signal (W) jets and Background (QCD) jets. Thus, the project seems to be a binary classification problem of teaching a CNN how to differentiate between a QCD and W-jet to be able to classify them quickly and more accurately. Looking at the physics of the two jets, we can expect a more disperse, and even two-pronged, structure in the brightest parts of the image for the signal images, as opposed to the more localized and single-pronged structure of the background jet. Both these jet-types can be seen below.

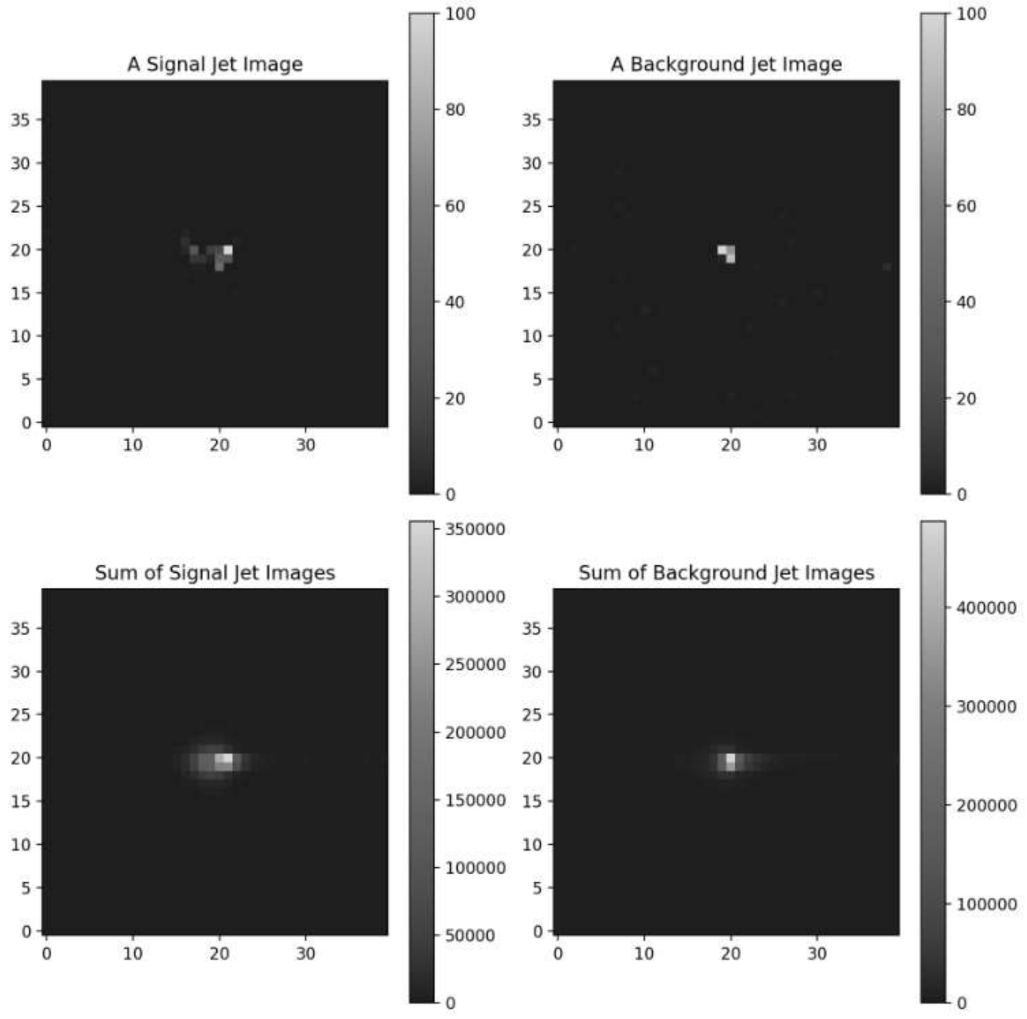

Fig 3: Randomly selected Signal and Background samples along with images of the sum of their respective datasets.

Looking at the images above, we can see that the individual and summed up images seem to appear as expected. The signal images, in both cases, have a much more dispersed shape than the background images. This is indicative of the particle physics explained above.

# 2 Methodology

## 2.1 Data processing

The data was given to me in the form of 4 '.npz' files. Each file was made up of 30,000 images, each composed of 40×40 pixels. The images were of the three jets that were detected by the LHC after a controlled Proton-Proton collision. Each image came labelled with what exactly it was to help in creating testing and training datasets. Thus, each file had a shape of (30000, 40,40,1), where '1' tells us that this data is monochromatic, and only deals with intensity variation of a single channel.

The first step was to separate our signal and background images from the Top-decay images. Since each of the 30000 images came labeled, I was able to separate the three jets into their own datasets using the tags 'top_jet_images', 'W_jet_images' and 'QCD_jet_images'. Each one of these jet-specific datasets had 10000 images in it.

The next step was to process the data so that each and every image in the signal (W-jet) and background (QCD-jet) datasets was rescaled to the same range of value. Although I could have gone with a much simpler rescaling method via setting the np.max value of the whole dataset as the rescaling parameter of the images, I instead opted for a more case-by-case rescaling. This was implemented by running a for-loop that calculated the np.max value of each image individually and then rescaled each image based on its maximum value. I chose a range of 0-100 as the scale for better visual clarity when reading numbers from plots. This resulted in every image in the dataset being rescaled to the same range of 0-100. This method of scaling was appropriate for this data as we need to tackle a binary classification problem, which would be more accurate if the jets detected (which are usually responsible for the np.max value) all had the same max-value, making the recognition of jet-structures in the image easier for the CNN.

After processing the data, we need to use it to generate training and testing datasets for the CNN. We first began by shuffling the data so that any sequential patterns in subsequent images is not utilized by the CNN to tag the images, as the CNN needs to be able to identify individual jet images without using data from other images. Next, I created label data for the background (BG) and signal (S) images, which consisted of 1's and 0's in a 1D-array, with '1' corresponding to BG and '0' corresponding to S. This was done to better represent the probabilties associated with tagging the images and the binary nature of this endeavor. For example, if an image gets tagged with a value of 0.45, it is more likely to be S than BG. Another way of looking at it is that the value tagged on the image represents how likely an image is BG. An image with a value of 1.0 would indicate a 100% chance of it being BG, but an image with a value of 0.0 would indicate a 0% chance of it being BG. But, since our options are binary, falsifying one option makes the other one true. Next, after combining the everything into an image-dataset and a label-dataset, I used the train_test_split function from sklearn to split the datsets up into training and testing datasets. 60% of the data was used for training and 40% was used for testing. Finally, I plotted a random image from the S and BG datasets, along with the image-sum of each dataset. This can be seen in figure 3.

## 2.2 Creating, Testing and Training the CNN

For the neural network, I implemented the use of 4 different types of neural leyrs: Convolutional, MaxPooling2D, Dense and Dropout. The convolutional layers are primarily used to extract features or patterns from the images, allowing the network to learn via pattern recognition. They also have a secondary function of reducing the data's dimensions, thus helping combat against overfitting and high computation cost. MaxPooling2D is used to reduce the dimensionality of data to a high degree while still retaining important features. This helps the CNN learn from characteristic parts of an image while ignoring irrelevant data. Once important features are extracted from the image and the image is flattened, Dense layers can be used. These layers are the point at which the data gets classified into its classes. This is done by adjusting the weights of the layers to minimize a loss function. Finally, Dropout layers are added-in sporadically to turn off a specified amount of neurons in a layer. These neurons are selected randomly. This is done to reduce overall computation cost and overfitting, while still maintaining some of the output expectations from a layer. These layers were used in the combination shown in figure 4. All convolutional and dense layers(except the dense-output layer) had 'ReLU' activation function. This was chosen due to ReLU being better at picking out features in the images. Also, it helps 'prevent exponential growth in the computation required to operate the neural network[6]. But the output layer utilizes a sigmoid activation as the result needs to be a probability value between 1 and 0, which is well represented by the sigmoid graph.

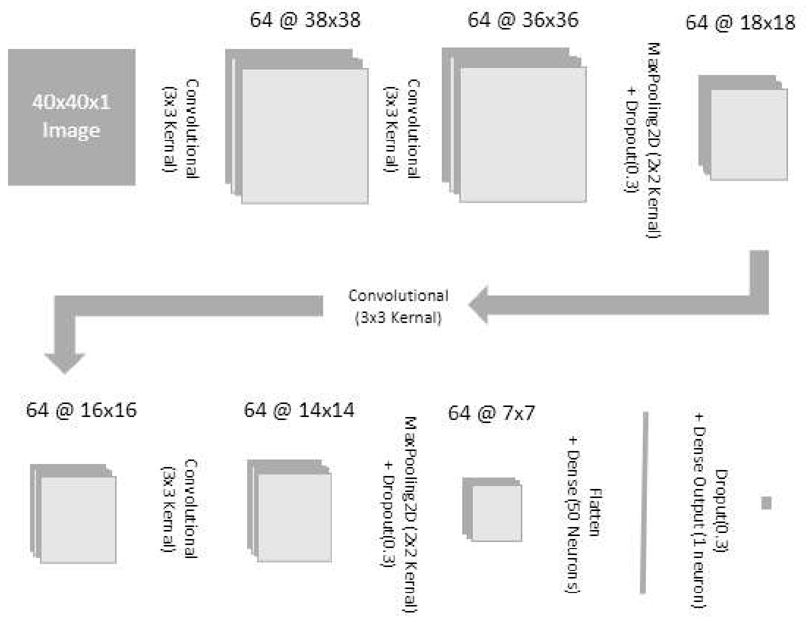
40x40x1 Image
Convolutional (3x3 Kernal)
64 @ 38x38
Convolutional (3x3 Kernal)
64 @ 36x36
MaxPooling2D (2x2 Kernal) + Dropout(0.3)
64 @ 18x18
Convolutional (3x3 Kernal)
64 @ 16x16
Convolutional (3x3 Kernal)
64 @ 14x14
MaxPooling2D (2x2 Kernal) + Dropout(0.3)
64 @ 7x7
Flatten + Dense (50 Neurons)
Dropout(0.3) + Dense Output (1 neuron)

Fig 4: Diagram depicting the architecture of the CNN

When compiling the model, the loss function used was 'binary_crossentropy', and the optimizer used was 'Adam'. Binary_crossentropy was a better option than Mean-squared error due to the binary nature of our data. Mean-squared error expects real values in the range of (-inf , +inf)[7]. 'Adam' stands for 'Adaptive Moment Optimization algorithms'. It was used here because of my previous experience with it. I attempted the use of RMSprop, but that resulted in a very jagged loss graph. Thus, I did not use it.

I maintained a batch-size of 200 as it gave the best balance of accuracy and computational time. I ran 15 epochs as that was about the point at which there were diminishing returns experienced, while an accuracy of about 80% was reached. Thus, I reasoned that 15 epochs was a good stopping point to maintain efficient and well-balanced code.

# 3 Results and Analysis

## 3.1 Accuracy and Loss

The loss and accuracy had final values of 0.434 and 0.812 respectively. The change in loss and accuracy over epochs is imaged below.

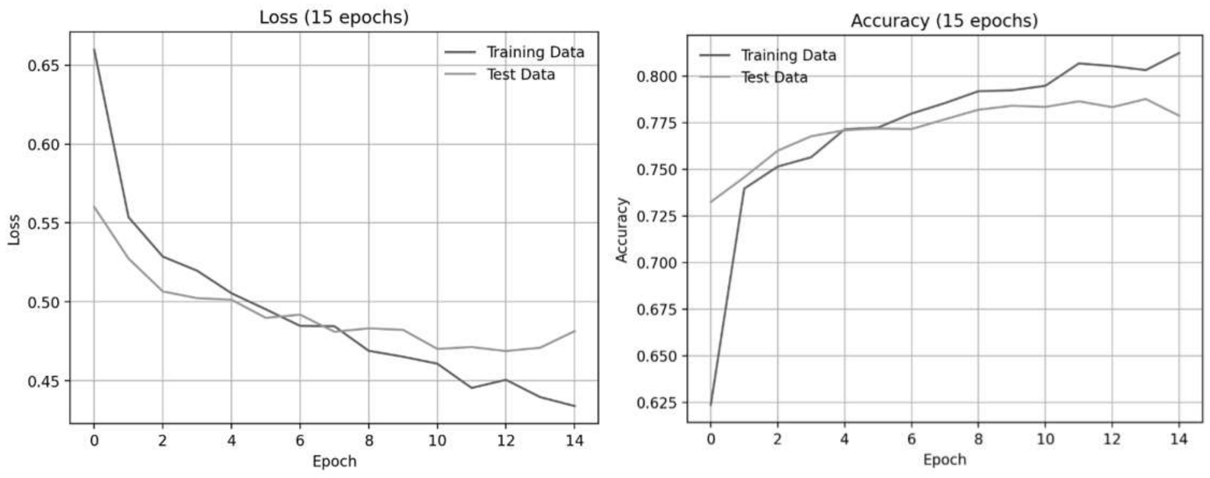

Fig 5: From left to right, graph of Loss and Accuracy as Epochs pass.

Looking at figure 5, the overfitting of loss begins at about the 6th epoch, and for the accuracy, it begins at about the 5th epoch. This is fine considering two factors:
1)The difference in the training and testing values are not massive. The overfitting is not very high.
2)Changing the neural network (for the most part) still results in overfitting, but the point where overfitting begins changes.
Thus, I focused on decreasing the degree of overfitting and increasing the final accuracy of the model while maintaining good code effeciency. Looking at the shape of the accuracy graph, we can see that it changes lesser and lesser as the epochs pass, giving further credence to having a lower number of epochs.

## 3.2 The ROC Curve

I further tested the performance by looking at the area under the curve (AUC) for the ROC and PR curves.

One way of showing the effectiveness of a binary classifier system is called a ROC (Receiver Operating Characteristic) curve. For different classifying criteria, it plots the true positive rate (TPR) vs the false positive rate (FPR).

The proportion of positive events (true positives) that the system correctly classifies is known as the TP (number of times QCD-jet was guessed correctly), whereas the proportion of negative events (false positives) that the system wrongly classifies as positive is known as the FP (number of tim es QCD-jet was guessed incorrectly).

On the other end, we have true negatives (guessing W-jet correct), TN, and false negatives (guessing W-jet wrong), FN.

The TPR, true positive rate, is the ratio of TP events and all actually positive (Background Jet) ev ents. This can be calculated using the following equation:

$$TPR = \frac{TP}{TP + FN}$$

The FPR, false positive rate, is the ratio of FP events and all actually negative (Signal Jet) events. This can be calculated using the following equation:

$$FPR = \frac{FP}{FP + TN}$$

The function 'roc_curve' plots the TPR/FPR graph for varying thresholds. Here, 'threshold' means the probability at or over which the outcome will be categorized as positive

A common way of assessing the general performance of a classifier system is the area under the ROC curve. While an area-under-curve of 0.5 implies a classifier that is no better than random, an area-under-curve of 1 shows perfect categorization.

Basically, the higher the area-under-curve, the greater the predictive power. The AUC for ROC was found to be 0.86. The graph of the ROC curve can be seen below.

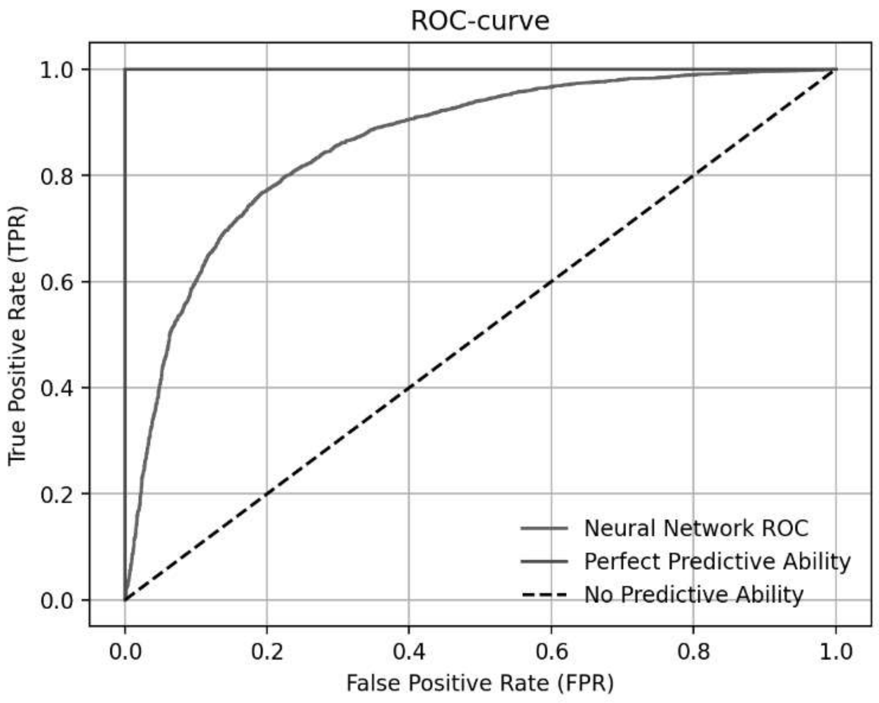

Fig 6: Graph of ROC curve for three different cases, including our model.

The area-under-curve value calculated above suggests a good predictive accuracy of about 86% for our CNN.

## 3.3 The PR Curve and the F1-Score

Precision is the fraction of all positive predictions that are correctly guessed.

$$\mathrm{Pr}ecision = \frac{TP}{TP+FP}$$

Recall is the same as TPR.

Similar to the ROC curve, here, the predictive ability of my CNN will judged based on the area under the curve generated by these parameters.

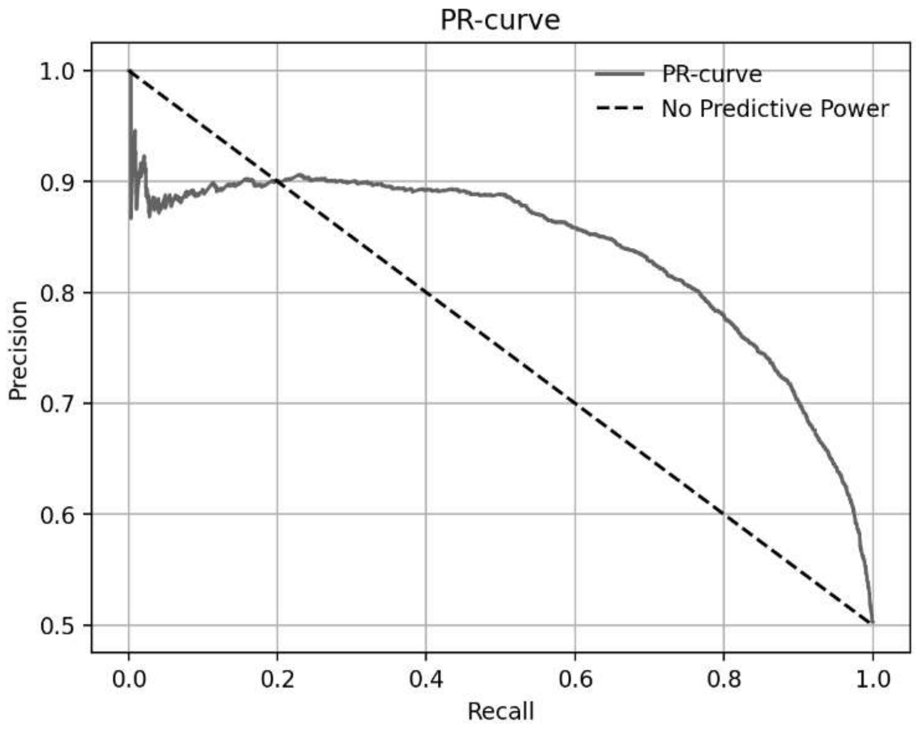

Fig 7: Graph of PR curve for our model and a model with no predictive power.

The area under the graph was found to 0.84.

We can also the precision and recall values calculated above to generate an F1-score for our neural network.
This value is a harmonic mean of recall and precision. It has a range of 0 to 1, with larger numbers indicating greater performance, and 1 corresponding to a perfect score. It is calculated using the following equation:

$$F1 = 2 \cdot \frac{precision \times recall}{precisio + recall}$$

Since this calculation takes into account the network's ability to correctly identify positive events and avoid falsely identifying negative events, it is a great way of evaluating our CNN's performance in binary classification. The final F1-score was found to be 0.80.

Looking at the graph, the curved part is as expected, with the sharp drop in the beginning being caused by a rise in false positives, resulting in fewer correct classifications. Overall, our NN seems to be working very well. The F1-score and the area-under-curve of PR-curve both have high values.

## 4 Conclusions

Given the comparably smaller dataset than the one we used in MNIST, we got pretty good performance from the model. I achieved final accuracy and loss values of 0.812 and 0.434 respectively. Evaluating the AUC of the ROC-curve gave us a value of 0.85. For the PR-curve, this was found to be 0.84. And the F1-score was found to be 0.80. I would have also liked to analyze Rejection-Efficiency curve, but that didn't gave any strong insight into the performance of the CNN due to it not being a percentage metric.

There is a lot of room for growth - I would have liked to achieve 90% accuracy – but the results are quite considering that I only trained the network for 15 epochs to keep down computational costs. Also, the CNN was structured such that the number of trainable parameters in it were less than 300,000. Thus, all things considered, this was a pretty good result, even though there is more room for improvement.

# 5 Bibliography

## References

[1] CERN. “Facts and Figures about the LHC | CERN.” Home.cern, home.cern/resources/faqs/facts-and-figures-about-lhc#:~:text=What%20is%20the%20LHC%3F.

[2] CERN. “How an Accelerator Works | CERN.” Home.cern, 8 Apr. 2019, home.cern/science/accelerators/how-accelerator-works.

[3] Aivazis, Alec. “Draw Feynman Diagram Online.” Feynman.aivazis.com, feynman.aivazis.com/. Accessed 23 Mar. 2023.

[4] Moreno, Eric A., et al. “Interaction Networks for the Identification of Boosted H→Bb¯ Decays.” Physical Review D, vol. 102, no. 1, July 2020, doi:https://doi.org/10.1103/physrevd.102.012010.

[5] Ramprasath, Muthukrishnan, et al. “Image Classification Using Convolutional Neural Networks.” *International Journal of Pure and Applied Mathematics*, vol. 119, no. 17, 2018.

[6] Baeldung. “How ReLU and Dropout Layers Work in CNNs | Baeldung on Computer Science.” *Www.baeldung.com*, 30 May 2020, www.baeldung.com/cs/ml-relu-dropout-layers#:~:text=As%20a%20consequence%2C%20the%20usage.

[7] Khan, Rafay. “Why Using Mean Squared Error(MSE) Cost Function for Binary Classification Is a Bad Idea?” *Medium*, 6 Dec. 2019, towardsdatascience.com/why-using-mean-squared-error-mse-cost-function-for-binary-classification-is-a-bad-idea-933089e90df7.