# Chapter 8

# Image recognition for BSM searches in the hadronic final states with $b$-jets

The study presented in this chapter is not yet published, instead, it serves as a concept for future research paper once additional research is finished. I am the primary author of the work presented here.

## 8.1 Introduction

Since its inception, machine learning (ML) has evolved and been applied to countless problems, including those in particle physics. The applications of ML are numerous, ranging from enhancing jet tagging efficiencies [96, 171, 172], and designing alternative jet clustering techniques [173] to exploring parameter space for BSM searches [174, 175]. In the case of LHC searches, more advanced ML techniques have been used, in particular, mapping the final state of the detector into an image.

Many of these jet physics studies, such as jet tagging and identification, are substantially independent of the specific physics from which they originated and can therefore be applied to other specialised LHC searches. ML's exceptional flexibility to adapt to different situations is a special benefit. Without much alteration, an ML model can be easily trained on other datasets from various physics processes.

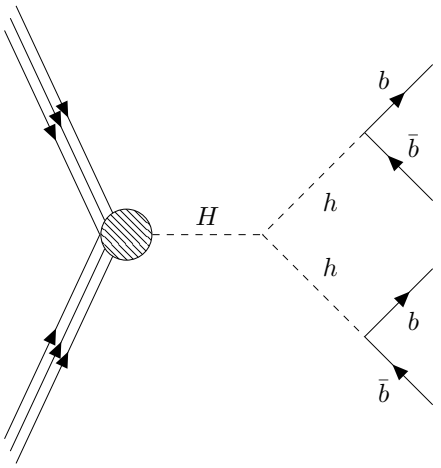


FIGURE 8.1: The 2HDM process of interest for this work.

In order to attempt to create a classification algorithm using ML, we need to organise the remnants of an event into a dataset that can be trained on. There are two main routes one takes in doing this, firstly using numerical information about the event, such as the jet $\eta$, $\phi$, $p_T$, $m$, etc., as in [176]. Secondly one can build so-called jet images, where the constituents of final state jets found in a detector are mapped into the $(\eta, \phi)$ plane such that we can unravel the barrel-shaped detector into an image. A neural network can then be trained on this jet image data to perform the desired classification.

There is an extensive catalog of studies investigating jet image recognition [177, 178, 179, 180, 181]. A general feature of such studies is for the algorithm to learn on a jet substructure. Image recognition allows this to be achieved without the need for calculations and analysis of jet variables such as $N$-subjettinness.

In this study, we seek to utilize ML techniques to build a classifier for finding signs of new physics at the LHC using 2HDM Type-II. In particular, we will incorporate the advanced technique of image recognition by designing a CNN and visualizing what is 'seen' in a detector in particle physics experiments. Of course, there are well-established LHC searches using traditional techniques and alternative clustering algorithms for evaluating 2HDM final states. Here, we will deploy an image recognition-informed jet tagger's exceptional ability to map the jet-level information to an image and distinguish signals from relevant backgrounds.

We will be dealing with multijet events, it might therefore be advantageous to include as much of the jet information as possible in order to maximise our learning capability. Therefore, we will employ jet-level image recognition studies performed

in the context of boosted decays such that the entire signal event can be clustered into a single fat jet using a large cone size, and the big discriminatory feature for signal and background is the presence of a two-prong jet substructure, as done in [178] in the context of $H_{SM} \to b\bar{b}$ decay. For our study, we focus on $b$-jet final states from 2HDM Type-II with decay chains of the form $gg \to H \to hh \to b\bar{b}b\bar{b}$, see Fig. 8.1.

The layout of the chapter is as follows. In the next section, we will briefly provide an overview of ML techniques. We then outline the event generation toolbox and MC analysis details. Following that, we will present the pre-processing steps for image generation and the CNN model used for training purposes. Finally, we will present our results and draw our conclusions for further investigation.

## 8.2 Overview of ML techniques in High Energy Physics

In this section, we will review the principles of ML techniques and their uses in tackling real-world problems. Tom M. Mitchell provided a fundamentally operational definition of machine learning and widely quoted [182]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

Generally speaking, ML is an umbrella term for solving prohibitively expensive problems by assisting machines in developing their own algorithms without any explicit human interference. ML techniques have been applied to text filtering, large language models in agriculture, computer vision, speech recognition, and medicine. However, in this context, we emphasize its application in high-energy physics relevant to our study.

### 8.2.1 ML categories

Modern ML techniques are broadly classified into three groups based on the nature of the problem to be solved, which we will quickly review here.

#### 8.2.1.1 Supervised Learning

Supervised learning algorithms work on a dataset with features, where each sample is also tagged with a target or label. The term supervised learning comes from the idea that target "T" informed by the user, instructs the ML system on what to do. For example, a supervised learning algorithm can analyse the melon dataset and learn to classify melons into different species based on measurements.

There are two types of supervised problems:

**Classification**: Classification requires the prediction of class labels in order to determine which of $x$ categories some input belongs to. For example, object recognition is a classification problem where an image is the input and a numeric code is an output for identifying the object in the image.

A well-established jet physics classification problem is jet tagging, where we attempt to categorise two target classes, i.e., a $b-$jet or not a $b-$jet.

**Regression**: Regression asks the machine to predict a numerical value for some given input. The regression task is similar to the classification method, however, the output format differs. An example of a regression task is the measurement of physical quantities such as prices of financial assets.

Going back to the jet tagging problem, a regression task can be used to correctly assign a $b$-tag to the jets we are certain about, resulting in the reduction of $b$-tag jets that were incorrectly allocated.

#### 8.2.1.2 Unsupervised Learning

Unsupervised learning learns important information about the structure of the dataset containing input data without labels. An excellent application of unsupervised learning in physics is jet clustering techniques. In this situation, we have a dataset with relevant information, but we do not know how to split them into jets. Unsupervised learning can be used to cluster the particles into jets instead of traditional methods.

Spectral clustering [173] is one such example of unsupervised clustering learning techniques.

#### 8.2.1.3 Reinforcement Learning

Reinforcement Learning educates the machine through trial and error to take the best action by developing a feedback loop between its environment and learning system. Reinforcement Learning can be used to drive autonomous vehicles by informing the machine when it made the correct judgments or train models to play games, allowing it to learn what actions should be performed over time.

### 8.2.2 Importance of Data in ML

As we have seen, the input data is a crucial component of ML. Together with the hyperparameters of the ML model, input data determines how useful the outcome will be. ML algorithms use data to understand the correlations and patterns between input variables and target labels that can be used for classification or prediction tasks. Data can be numerical, time series, or categorical and can come from a variety of sources. Numerical data consists of values that can be measured and ordered, such as income, house price, or age. Categorical data consists of the values of categories, such as tree type or gender. Typically, data can be divided into two branches

- **Labelled Data**: consists of label or target variables for which the model is attempting to predict.
- **Unlabelled Data**: does not consist of label or target variables for model predictions.

A basic notion for developing an ML model is dividing the input data into training and testing sets. The reason behind this is that models are developed by using a minimising function that symbolises the error in the model's predictions, and using the input data to train the model itself to assess its performance is unreliable.

Instead, we utilise a training subset to train the model and a testing subset to assess the model's performance, while keeping in mind that the input data is split between representative and random manner.

### 8.2.3 ML Models

In this section, we will briefly review some of the ML models, with a particular emphasis on deep learning models relevant to our study.

#### 8.2.3.1 Supervised Learning Models

In literature, there are many supervised learning models; here, we briefly outline some of the most commonly used models:

- **Logistic Regression**: It is used to determine whether an input belongs to a specific group and to estimate the probabilities for each output class [183]. It is used to represent a binary dependent variable, which has two values, 0 and 1, to represent outcomes.
- **Linear Regression**: The most basic sort of regression is linear regression [184]. This model is used to identify correlations between two continuous variables.
- **Decision Trees**: These are flow-chart-like classifiers that are used to determine a branching approach to illustrate every possible consequence of a decision [185]. Each node in the tree demonstrates a test on a variable, with each branch indicating the result of that test.
- **Support Vector Machine (SVM)**: SVM [186] simply filters data into classes by giving a set of training samples, with each set flagged as falling into either of the two classes. SVM then constructs a model that allocates new values to one of the two classes.
- **K Nearest Neighbours (KNN)**: The KNN algorithm [187] groups the nearest objects in a dataset and determines the most average or frequent attributes among the objects.

Other supervised learning models include Naive Bayes [188], Random Forest [189], and Boosting algorithms [190].

#### 8.2.3.2 Unsupervised Learning Models

In this subsection, we briefly outline some of the most commonly used unsupervised learning models:

- **K-Means Clustering Algorithm**: The K-Means algorithm [191] is used to classify unlabelled data. The algorithm works to detect similarities between objects and categorise them into K clusters.
- **Hierarchical Algorithm**: Hierarchical clustering [192] creates a tree of nested clusters without specifying the exact number of clusters.

### 8.2.4 Deep Learning Models

In this section, we summarize modern deep learning models used to solve practical problems, with a special emphasis on CNN.

Deep learning neural networks (NN) are a set of algorithms modeled after biological neurons. Despite their very simple core function, neurons can send and receive electrical impulses and can be organised in large arrays to solve complex problems. NN recognises numerical patterns stored in vectors such that all the real-world data (such as images, text, time series, or sound) must be translated into the relevant form. We can use NN to cluster the unlabeled data based on similarities between given inputs and also classify the labeled data for model predictions.

There are several types of NN that exist in the literature. Here, we discuss some of them that are relevant to this study.

#### 8.2.4.1 Perceptron

A neural network can be built using a sequence of neurons, in which each neuron is fed an input $y_i$ and associated weight $w_i$, which are then coupled with a static bias value. This information is fed to subsequent neurons until reaching an output $z$, which is given by:

$$z = \sum_{i}^{n} w_i y_i = \vec{y}^T \vec{w}. \tag{8.1}$$

This intermediate output is then subsequently delivered to an appropriate activation function, which determines the neuron's final output value given by:

$$h_w(\vec{y}) = \mathcal{H}(z). \tag{8.2}$$

This is an example of a single-layer perceptron (SLP) which can be generalised to build a multi-layer perceptron for tackling complex problems.

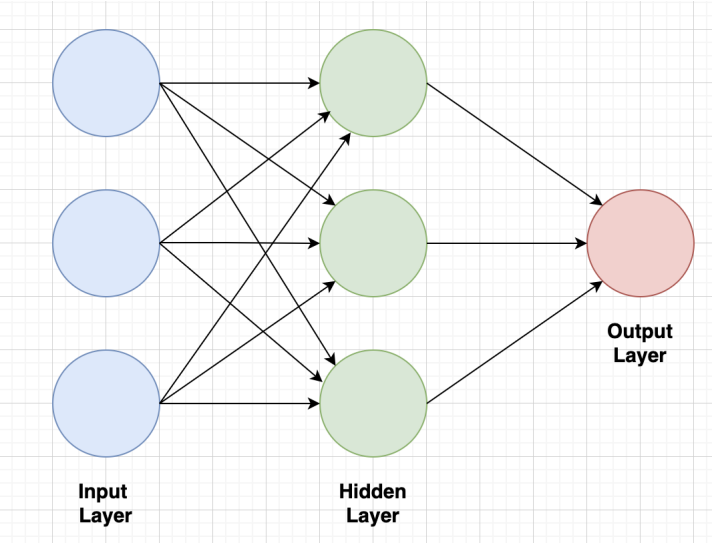


FIGURE 8.2: Simple MLP visual diagram with three characteristics, including one hidden layer.

#### 8.2.4.2 Multi-Layer Perceptron (MLP)

MLPs are probably the most prevalent type of NN, consisting of the input layer, an output layer, and additional hidden layers missing in SLPs. The visual representation of MLP is shown in Fig. 8.2. The three layers' functions are as follows:

- `Input layer`- This is the first layer of nodes in the NN and defines the dimensions of the input vector.
- `Hidden layer`- Hidden layers are intermediary nodes that partition the input space into soft boundary regions. They are fed weighted inputs and generate outputs using an activation function.
- `Output layer`- This layer provides the output of the NN.

Each neuron in a particular layer is connected to every neuron in the previous and next layers, making MLPs a fully connected network.

To train an MLP NN, in addition to the method outlined in the previous section, one must compute the loss function and use a backpropagation algorithm to adjust the weight to minimise the loss. The mathematical nuances of the loss function and backpropagation are beyond the scope of this chapter.

Another distinction between the two perceptrons is the choice of activation function. To model non-linear data, activation functions like sigmoid, rectified linear

unit (ReLU) and tan are used with softmax acting as an output layer activation function.

MLPs have an advantage over SLPs in that they can be used for deep learning but are complex to design and may take longer to train depending on the number of hidden layers.

#### 8.2.4.3 CNN

In this section, we will discuss our final deep-learning model, CNN [193], which we will use later on in the study. CNNs are a type of neural network that have a grid-like structure for processing data and are highly effective for time-series data and image data recognition tasks.

Images can be used as inputs by translating each pixel to a numerical scale expressing color values. Traditionally, NN layers would have to employ matrix multiplication of parameters, having a separate parameter characterising the interaction between each input and output unit, resulting in every output unit interacting with every input unit. However, it has been found that, in certain circumstances, the performance of more complex images is highly limited.

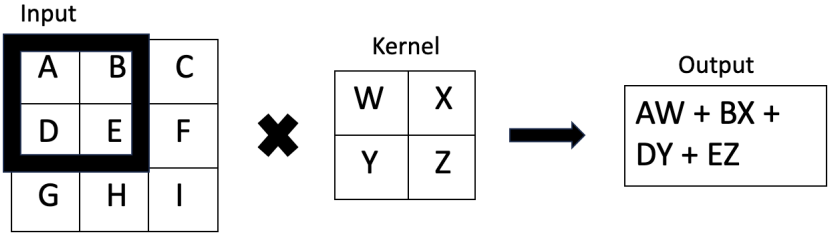


FIGURE 8.3: An example of a subset of an image being reduced to an element of the output tensor by the convolutional kernel layer. The black box illustrates the appropriate upper-left region of the input being used by the kernel layer to create the element of the output tensor.

CNN, on the other hand, uses a convolutional layer, which applies a small filter (also known as the kernel) to a subset of the input image and performs element-wise multiplication representing them as a single element of the output tensor (see Fig. 8.3). This procedure assists the network in learning the edges, texture, local patterns, and high-level visual features from the data. This method also reduces the amount of data being used for model training, resulting in better computational performance.

After convolutional layers, CNNs frequently employ pooling layers to downsample the dimensionality of the feature maps and reduce computational complexity but in a more elementary fashion. Pooling methods that are commonly used include max pooling and average pooling, shown in Fig. 8.4.

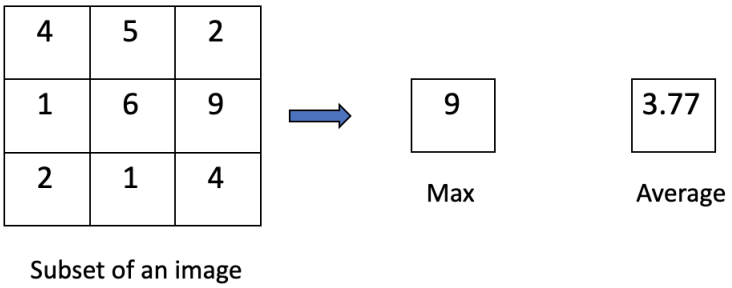


FIGURE 8.4: An example of a subset of an image being reduced to a single element using max pooling and average pooling.

CNNs have shown exceptional performance in various computer vision tasks and have been developed and adapted for use in other domains. These architectures differ in depth, layer arrangement, and network design choices to optimise performance for various applications and processing resources.

In the next sections, we will use CNNs to build a classifier for detecting signs of new physics beyond the standard model (BSM) at the LHC and visualising what is 'seen' in particle physics detectors.

## 8.3 Methodology

The goal of this research is threefold: select a suitable benchmark and generate events for training purposes, pre-process the relevant data and map them into a sample of images, and adapt these methods to produce a suitable classifier for BSM final states from 2HDM Type-II.

### 8.3.1 Simulation Details and Cutflow

We first select the phenomenologically preferred 2HDM Type-II benchmark point where the discovered 125 GeV Higgs is the lighter of the two scalars with a heavier

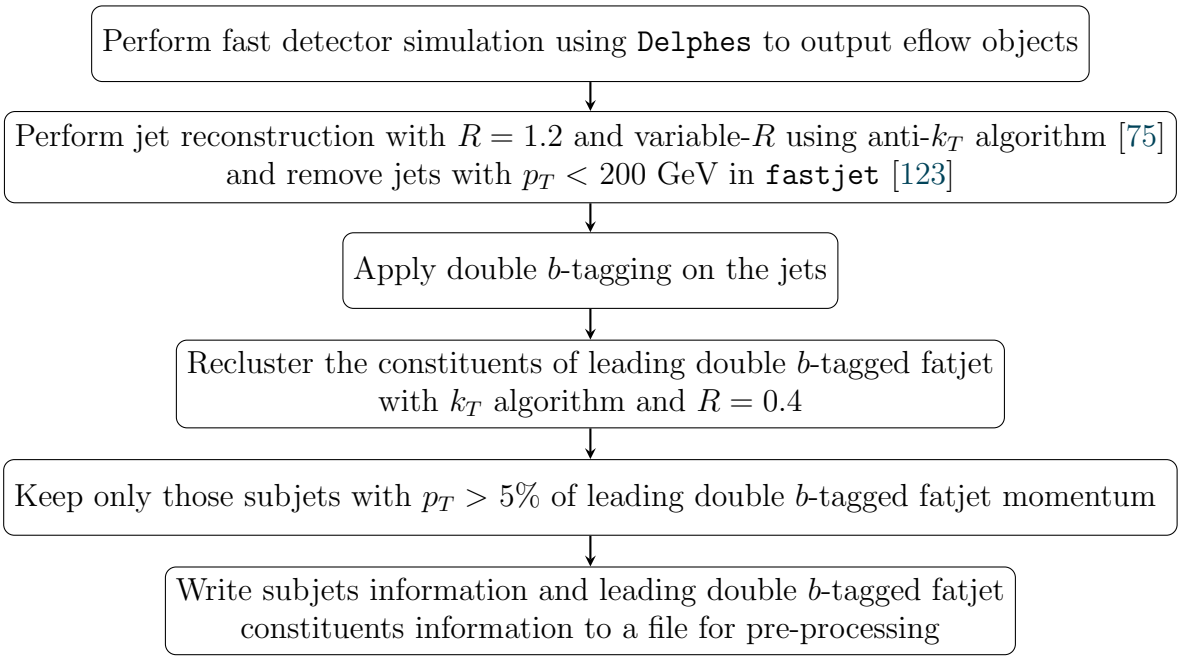


FIGURE 8.5: Description of the procedure used to analyse generated events for ML training.

CP-even Higgs boson mass set as $m_H = 700$ GeV. We test the benchmark against 2HDMC [40], HiggsBounds [113], HiggsSignals [114] as well as checking flavor constraints with SuperISO [115]. The 2HDM parameters and selected benchmark point information are given in Tab. 3.2 (see Point1 for the reference).

We generate samples of $\mathcal{O}(10^5)$ at $\sqrt{s} = 13$ TeV for the process $gg \to H \to hh \to b\bar{b}b\bar{b}$ using Madgraph5 [117]. We also consider leading SM backgrounds, such as:

$$gg, q\bar{q} \to t\bar{t} : \; p_T^{\text{gen}}(t) > 250 \text{ GeV},$$

$$gg, q\bar{q} \to b\bar{b}b\bar{b} : \; p_T^{\text{gen}}(b) > 100 \text{ GeV},$$

$$gg, q\bar{q} \to Zb\bar{b} : \; p_T^{\text{gen}}(Z) > 250 \text{ GeV}, \; p_T^{\text{gen}}(b) > 200 \text{ GeV}.$$

Here, we apply the generation level cuts within Madgraph5 to improve the selection efficiency and ensure a sensible signal to background analysis. We shower and hadronise the generated events using Pythia8 [118] with MPIs and ISR/FSR switched on. To perform a realistic MC analysis we use Delphes [119] to apply detector simulations and perform jet reconstruction and analysis using MadAnalysis5 [120, 121]. A full description of the cutflow is given in Fig. 8.5. For variable-$R$, we use $\rho = 300$ with $R_{min} = 0.4$ and $R_{max} = 2.0$. These values are influenced by the

$p_T$ scale of the $b$-jets as well as the scan on $\rho$. For fixed-$R$, we have used a wider cone size of $R = 1.2$ to capture the particles into fat $b$-jets.

We also implement a simplified (MC truth-informed) double $b$-tagger. For events clustered with a fixed-$R$ cone size, we look for jets that contain two $b$-quarks present within the angular distance $\Delta R$ and tag them as double $b$-tagged fat jets. For the variable-$R$ approach, we take the size of the tagging cone as the effective size $R_{eff}$ of the jet.

### 8.3.2 Construction of Jet Images

In this section, we describe in detail the generation of jet images as well as the reasoning for certain preprocessing steps. Our procedure substantially resembles that reported in [181].

#### 8.3.2.1 Input Data

Following event generation using `Pythia8`, jets are reconstructed with the anti-$k_T$ algorithm [75, 137] using EFlow objects information sourced from photons, neutral, and charged hadrons in the detector simulation (refer to Fig. 8.5 for full detail). Subsequently, we select the leading double $b$-tagged fatjet and recluster the constituents into subjets with the $k_T$ algorithm using the fixed size of $R = 0.4$. Only subjets with more than 5% of the leading double $b$-tagged fatjet momentum are kept for image preprocessing. In particular, we store:

- Subjets information:
  $n^{event},\ n^{subjet},\ p_T^{subjet},\ m^{subjet},\ \eta^{subjet},\ \phi^{subjet}$
- Leading double $b$-tagged fatjet information:
  $n^{event},\ n^{jetcons},\ p_T^{jetcons},\ m^{jetcons},\ \eta^{jetcons},\ \phi^{jetcons}$

This data is in raw form and to successfully convert it into jet images for visualisation, we apply the preprocessing steps of translation, pixelisation, rotation, reflection, cropping, and normalisation. These steps are designed to eliminate spatial symmetries of sorts so that CNN can readily pick out the subjets and learn the substructure, thereby distinguishing Higgs jets from the relevant backgrounds.

#### 8.3.2.2 Preprocessing Steps

The jet images are preprocessed to enable ML techniques to learn the distinguishing features between signal and background while avoiding learning symmetries of space-time. This approach can significantly enhance performance while also reducing the size of the sample used for testing.

In this sub-section, we describe the complete jet image production and preprocessing stages as follows:

- **Translation**:
  The first step is the translation where we translate all constituents of the leading double $b$-tagged fatjet in $(\eta, \phi)$ space such that the leading subjet is placed at the origin.

- **Pixelisation**:
  The next step is to construct a pixel grid of size $(0.1 \times 0.1)$ in $(\eta, \phi)$ space, and then the transverse momentum measured within each pixel is used to create a jet image.

- **Rotation**:
  The third step is to rotate the jet image such that the subleading subjet is placed directly beneath the leading subjet. In cases where no subjets are found, rotate the image to align the principal component along the vertical axis.

- **Reflection**:
  The jet image is then reflected to place the third leading subjet on the right-hand side. If only two subjets are present, the image is reflected to make sure that the total intensity of the image is highest on the right side. A cubic spline interpolation is utilised whenever the modified pixels do not line up with the original image pixels.

- **Cropping and normalisation**:
  The last step is to crop the jet image to a fixed size of $24 \times 24$ and then normalise the pixel intensities such that their squared sum evaluates to one.

After completing all the preprocessing steps, we obtain a jet image for each event based on the leading double-$b$ tagged fatjet constituent and subjet information. Next, we aim to classify the signal from backgrounds using the stack of jet images represented as an array of shape $(24 \times 24 \times 1)$.

### 8.3.3 Average Jet Images

After laying out the methodology for the jet image generation, we will proceed to examine the distinguishing features of the signal and background images that we intend to exploit for training our model. To avoid the impracticality of scrolling through a gallery containing a large number of images, we present a representative average jet image of $N$ events.

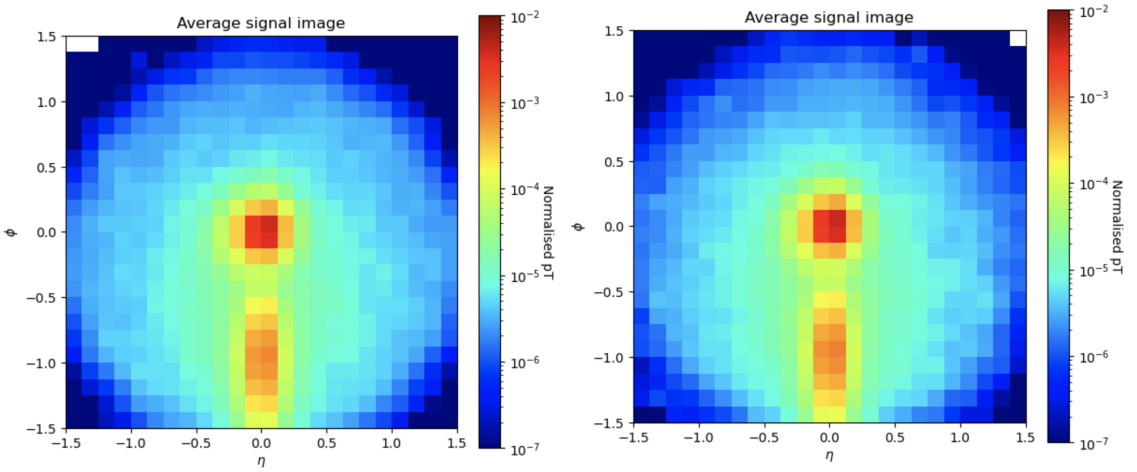


FIGURE 8.6: Left panel: The average signal image for leading double $b$-tagged jets coming from the process $gg \to H \to hh \to b\bar{b}b\bar{b}$ for fixed $R = 1.2$. Right panel: The average signal image using the variable-$R$ approach.

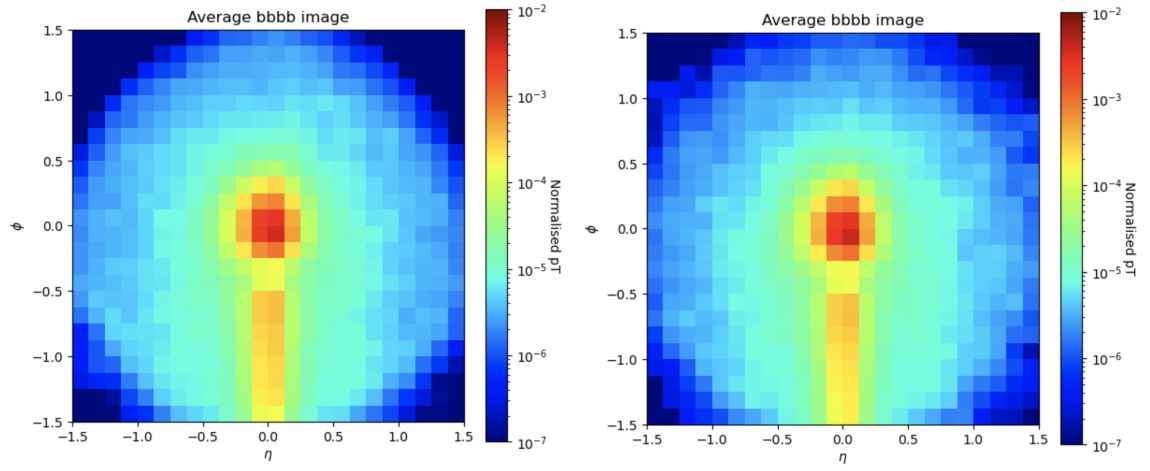


FIGURE 8.7: Left panel: The average background image for leading double $b$-tagged jets coming from the process $pp \to b\bar{b}b\bar{b}$ for fixed $R = 1.2$. Right panel: The average background using the variable-$R$ approach.

Figs. 8.6 – 8.9 contains the average images for the signal and relevant backgrounds for the jet reconstruction procedures. The figures depict the general substructure of the leading double-$b$ tagged wide cone jets for each process. In the signal image,

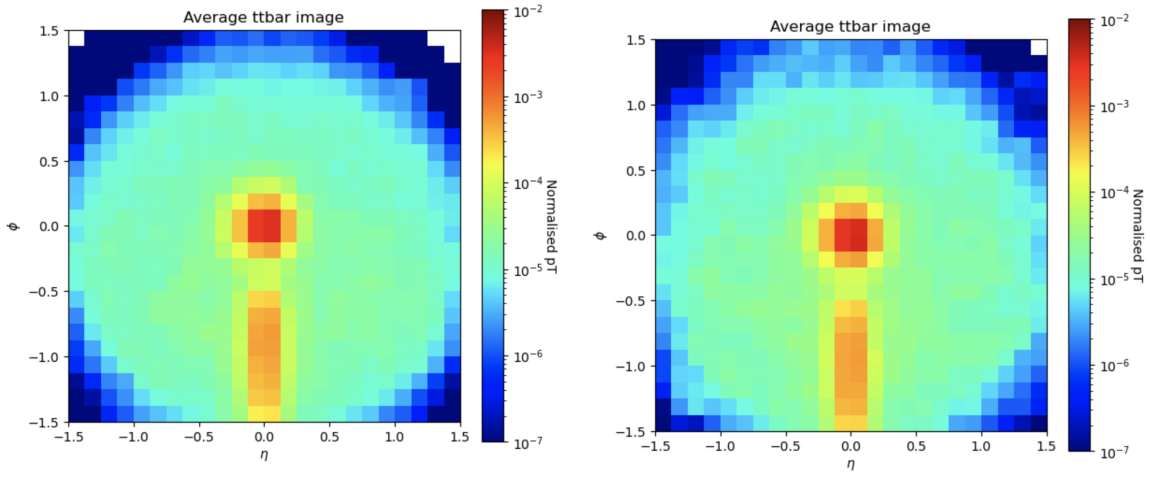


FIGURE 8.8: Left panel: The average background image for leading double $b$-tagged jets coming from the process $pp \to t\bar{t}$ for fixed $R = 1.2$. Right panel: The average background using the variable-$R$ approach.

we can see a prominent two-prong structure where the second subjet is visible and spatially defined as expected.

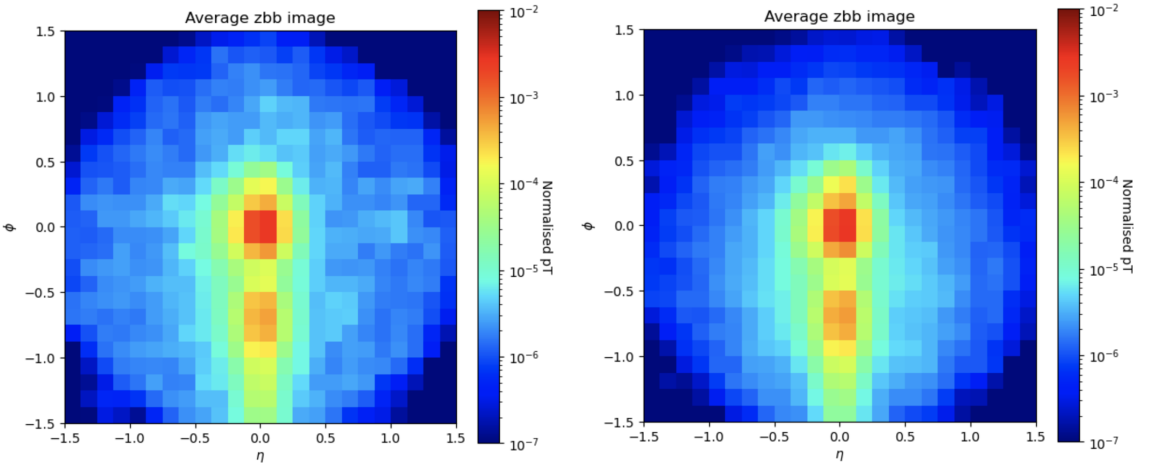


FIGURE 8.9: Left panel: The average background image for leading double $b$-tagged jets coming from the process $pp \to z b\bar{b}$ for fixed $R = 1.2$. Right panel: The average background using the variable-$R$ approach.

For $pp \to b\bar{b}b\bar{b}$ background, we observe that the leading fatjets do not have a well-defined substructure. This is due to the fact that leading fatjets are generated by QCD processes with no boosting, hence we rarely see the two-prong structure visible in the signal image. For the $pp \to t\bar{t}$ background, there is a hint of a second subjet but it is softer and somewhat more smeared in comparison to the signal. We can also see a lot of low $p_T$ activity spread across the image owing to the fact that there is a more intricate substructure associated with the $b$-jets and jet from $W^{\pm}$ boson originating from $t\bar{t}$ decays. Visually, the $pp \to z b\bar{b}$ background

closely resembles the signal substructure, with two different subjets. However, when compared to the signal, the average *zbb* image looks more compact.

Returning to the differentiation between the fixed-$R$ and variable-$R$ average images for signal and three backgrounds, we find little difference between the two sets of images. This is due to the fact that these images are based on the constituents and subjet information of the leading double-$b$ tagged jets, which may differ slightly due to the different reconstruction procedures but still preserve the true substructure of the particles involved in the processes.

### 8.3.4 CNN Model Architecture

After taking a look at the jet images we're training on, in this section, we will describe the CNN architecture utilised to train the classifier and present preliminary results. This CNN architecture can be thought of as a toy CNN prototype used to test the viability of distinguishing the signal from the relevant background images.

The CNN architecture comprises of following layers:

- 2D Convolutional layer, (3×3)
- 2D Convolutional layer, (3×3)
- Max Pooling layer, (2×2)
- 0.5 Dropout layer
- Flattening layer, length 128
- 0.25 Dropout layer
- 2 node output layer.

The convolutional and pooling layers aid in extracting meaningful information from the images. To avoid overfitting the model, we use the dropout layer to drop specific nodes, while a flattening layer is used to feed the data into a fully connected MLP-like structure. Finally, we design a two-node output layer defining the probability of distinguishing a jet image as signal (1) or relevant background (0). To train the toy model, we will use a simple 50/50 split between signal and background data and 20 epochs. Of course, a final CNN model based on

prospective improvements from the next section would be built by scanning over the tunable hyperparameters to improve the classifier's efficiency.

## 8.4 Results

In this section, we present the performance of the CNN model described in Section 8.3.4.

To assess the model's training, we plot the progress of both loss and accuracy on both the training and validation datasets. Next, we examine the CNN output score for each image in the validation dataset, which is a score ranging from 0 and 1. This can be interpreted as the probability of the production of a specific instance from the signal. When the output score is close to 1, the model predicts it is a signal, and when it's close to 0, the model predicts it is a background. Finally, a receiving output characteristic (ROC) is plotted to evaluate the area under the curve (AUC).

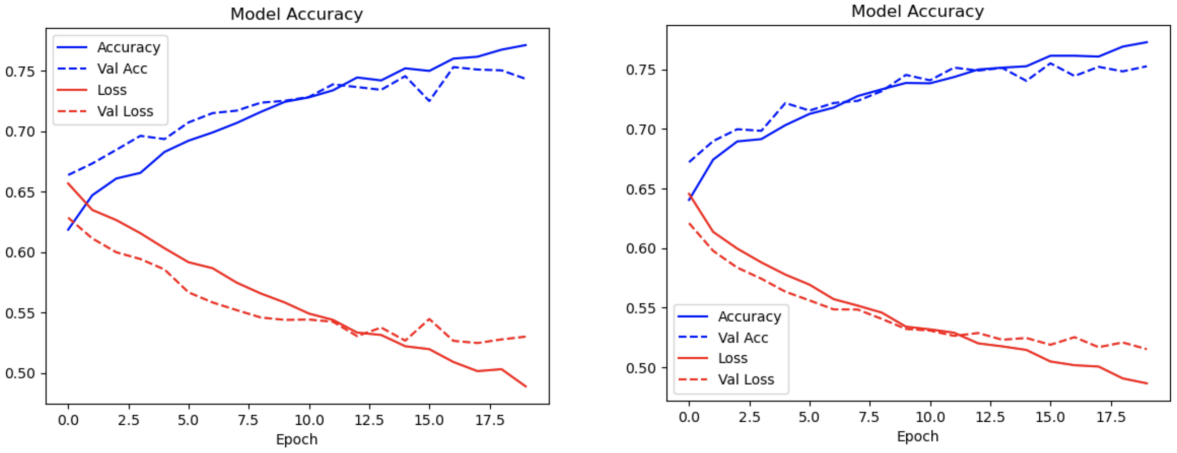


FIGURE 8.10: Left panel: The accuracy and loss progression training across 20 epochs for the fixed-$R$ case. Right panel: The accuracy and loss progression training across 20 epochs for the variable-$R$ case.

Fig. 8.10 shows the accuracy and loss progression training across 20 epochs for the reconstruction procedures. We can see that both the algorithm's validation accuracy and loss closely mirror the training sets. This is a good sign as it depicts that our model is not overfitting and can be generalised to the new set of images.

In Fig 8.11, we assess the performance of the model by splitting the signal and background datasets to examine whether the model's predictions match the truth information for both fixed-$R$ and variable-$R$ algorithms. Background events clearly

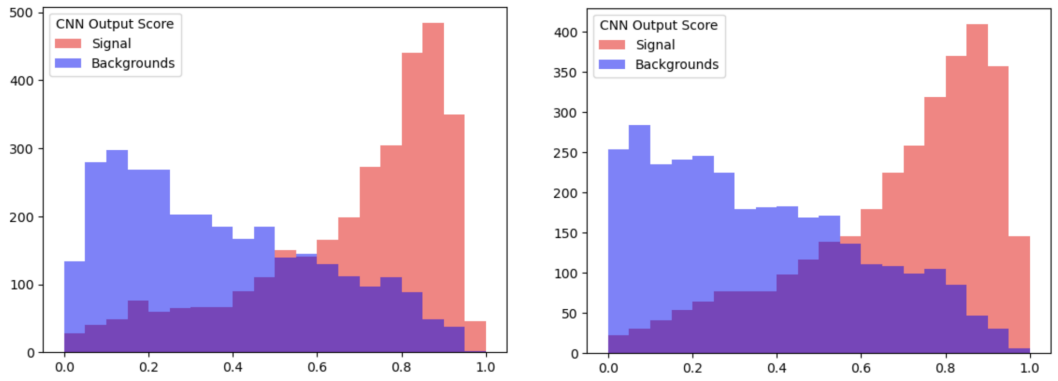


FIGURE 8.11: Left panel: The CNN model output score in the validation set for fixed $R$ case. Right panel: The CNN model output score in the validation set for the variable-$R$ case.

forecast values closer to zero, whereas the signal events peak around 1.0. However, we can see some overlap, more for variable-$R$ than fixed-$R$, indicating that the model is somewhat struggling to correctly identify the signal from backgrounds and requires additional refinement.

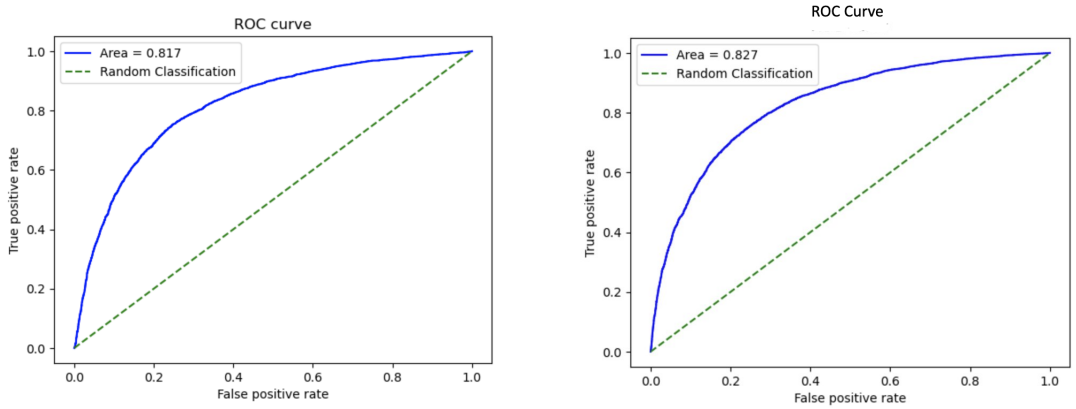


FIGURE 8.12: Left panel: The ROC curve plot to show the performance of the model's final iteration in training for the fixed $R$ case. Right panel: The ROC curve plot to show the performance of the model's final iteration in training for the variable-$R$ case.

Lastly, we plot the ROC to evaluate the AUC for the data in the validation set, see Fig 8.12. The ROC plot represents the true positive rate (TPR) and false positive rate (FPR) given by:

$$\text{TPR} = \frac{\text{TP}}{\text{P}} \quad ; \quad \text{FPR} = \frac{\text{FP}}{\text{N}}, \tag{8.3}$$

where TP represents true positives, P is the condition positive, FP denotes false positives, and N is the condition negative. Points on the AUC correspond to

a distinct decision boundary for the model. With an adequately set decision boundary, we can see that the model can reach a TPR of 0.81 for fixed-$R$ and 0.82 for variable-$R$ with a background rejection of less than 0.3 for both cases. Another key point is that once the variable-$R$ parameter $\rho$ is fine-tuned, we may expect considerably better results when compared to fixed-$R$, increasing the efficiency of distinguishing the signal from backgrounds.

## 8.5 Conclusions and Future Work

In this chapter, we presented a working toy model for jet visualisation using CNN for classifying the 2HDM Type-II signal $gg \to H \to hh \to b\bar{b}b\bar{b}$ from the relevant backgrounds $pp \to b\bar{b}b\bar{b}$, $pp \to t\bar{t}$ and $pp \to zb\bar{b}$ using fixed-$R$ and variable-$R$ jet reconstruction procedures. We showed that a simple ML CNN model can obtain an ROC area under the curve of up to 0.81 for fixed-$R$ and 0.82 for variable-$R$ procedures when trained for the signal jet images against a balanced mixture of the relevant backgrounds. These preliminary results further inspire us to create a more robust model to classify two sets of jet images.

The potential model improvement would be to incorporate more information, like in [178]. In our model, we solely train on leading double-$b$ tagged fatjet constituents and subjets information. To include more information, we can add a second stream to the CNN model that uses global event information covering more of the detector picture. Another adjustment would be to filter our event samples using a more sensible selection procedure guided by the experimental setup. The inclusion of $p_T$ window and masses of fatjets in the cutflow can help minimise the backgrounds to a level comparable to the signal, hence improving the model's performance.

To train our binary model, we employed a simple 50/50 split between signal and background data; if the data is not evenly split, the network will be biased towards the class with more samples. This is, however, not true for the underlying physics, where each process has different cross-sections and event rates. If the event rate for the signal is very rare when compared to the backgrounds, one might use a different ML method, such as anomaly detection, or continue to train the model using a 50/50 split. Another aspect to consider is the grouping of background data into a single class, despite knowing that each background has unique characteristics distinguishing it from the signal. Although this is a useful solution, more research should be done to examine the separation of power between each background separately.

These are some of the suggestions that can help address the input and physics problems. Of course, a final model based on prospective improvements would be constructed by scanning over the tunable hyperparameters to improve the classifier's efficiency.