The Classification of Jets by a Convolutional Neural Network

PHAS0056 Project, Department of Physics and Astronomy, University College London

Louis Choron

Abstract: An attempt to distinguish between signal and background jets using machine learning techniques is presented. A summary of quark-jet and machine learning theory is given to introduce the creation, training and optimisation of a model. Using images depicting the transverse energy of jets, the model was trained and evaluated to produce an F1 accuracy score of 0.79 (2.s.f), which demonstrates the effectiveness of machine-based, image classification techniques.

Introduction

I. Particle Jets

In high-energy physics, the identification of particles is a common task when studying interactions. Many produced particles propagate individually, and can be detected as single distinct events, whereas particles known as quarks form indistinct jets when produced. Quarks are a class of six fundamental particles, and are responsible for combining to form hadrons, including protons and neutrons. They are also the only matter particle to interact with the strong force, one of the four fundamental forces of nature, with the gluon being the mediating particle, just as the photon is for electromagnetism. Interestingly, the strong force is the only fundamental interaction to increase in strength with distance, due to an anti-shielding effect when quarks scatter. Therefore, when quarks are produced and separate, the potential energy between them increases until there is sufficient energy for a new quark-antiquark pair to be created, which must occur to minimise energy. These produced quarks then combine to form hadrons (hadronization), which travel in the same direction as the original quarks, producing a jet of hadrons after a few repetitions. Quarks are never observed directly in nature, so measurements of these jets are necessary for providing insights into the original quarks and other particles produced in the interaction. In an interaction with many decay products, it is necessary to distinguish the jets of interest from other background jets, which may include those from gluons or different quarks. It should be noted that despite the heaviest "top" quark decaying too quickly to form a jet, it does decay to the lighter "bottom" quark which does hadronize, and so reconstruction is still possible, albeit more challenging.

The distinction of jets is known as "tagging" [1], and can be difficult since the potentially overlapping jets are only detected as a cone of roughly collimated particles. This is why physicists turn to complex algorithms, and machine learning (ML) classification techniques. These techniques are very effective, as an incredibly large amount of jet data is available to train models to high accuracy. Such techniques were useful in discovering the Higgs Boson through the $b\bar{b}$ channel. Despite being the most frequent decay channel for the Higgs (58% of decays [2]), the

analysis was difficult due to poor resolution of the $b\bar{b}$ jets. Through the creation of complex models, the $b\bar{b}$ jets were distinguished well enough to show conclusive evidence for creation via Higgs decay [3].

II. Machine Learning

ML refers to the development of software which can learn and improve by analysing data, without being explicitly programmed. It has many applications ranging from the prediction of cancer progression [4] to the development of self-driving cars [5], and is often divided into three main categories, depending on how the learning is performed. These are supervised, unsupervised and reinforcement learning, and refer to situations where the model either learns to match inputs to labels (E.G., for image classification), discover patterns within only input data (E.G., for clustering similar data together) or to explore a dataset using a reward-based system (E.G., for beating games).

It should be noted that 'learning' refers to the improvement of a model, by optimising the available parameters to perform better at a specific task. This optimisation is performed by algorithms known as gradient decent optimizers, such as Stochastic Gradient Descent (SGD), adam or RMSProp [6]. Gradient descent involves selecting a point (corresponding to certain parameters) on a function, calculating the gradient over all data, and using this value to inform in which step you should move, to minimise the function. This is very computationally expensive, so different optimizers have been developed, which can converge to the optimal parameters quicker and more effectively. SGD is the most basic of these optimizers, whereby the gradient is calculated using only a small batch of data sampled at random, as opposed to the full dataset. Adam and RMSProp are similar, however use 'momentum' features to adjust how far they step, depending on the how the gradient changes over time.

When coding, there are many different libraries which can be used to perform ML, one of which being TensorFlow. Included within TensorFlow is the library Keras, which is designed specifically for training models within Python.

There are also many different types of ML model, one example being the "neural network" (NN), named due to having a structure of connected neurons, similar to a brain. There are different types/methods of creating such networks when coding with Keras, one of which being "sequentially", where various layers of neurons are stacked one after the other. Each network requires input and output neurons but can have any number of layers of neurons between them, called "hidden layers". The outputs of neurons in one layer are connected to the inputs of the neurons in the next, with each connection having an associated weight which affects how data is passed along the network. Each neuron takes its weighted inputs, applies a non-linear function (and constant bias), and outputs its own weighted data. These weights, biases and functions can all be optimised, for data to pass through in a specific way, allowing for different tasks to be completed.

Keras includes many different types of hidden layers, which can each have different effects and connect to the neurons of other layers in different ways. Examples of hidden layers include the basic "densely connected layers",

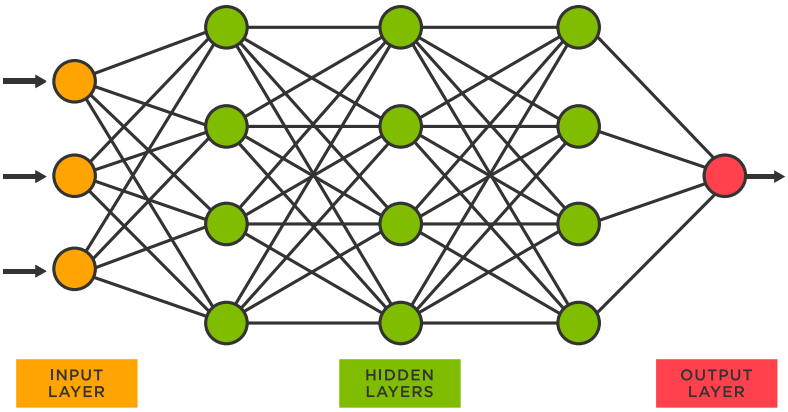

Figure 1: An example structure for a basic, densely connected NN.

where each neuron is simply connected with every output of the previous layer. These add many parameters to be optimised and can be useful for changing the dimensions of data. Other examples include the LSTM layer, which can add memory capabilities to a model, or dropout layers, which randomly sets some neuron weights to zero to force the network to learn using other neurons and remain unspecialised.

III. Project Description

In this project, a ML model was created to distinguish between specific signal and background jets, since they are often too similar to distinguish by eye. The jet data was given as a series of images, as shown in Fig. 2, which were used to create labelled training data, for which the NN was built and trained by supervised learning. Once trained, the model was optimised to improve its classification performance. The model's performance was then quantitatively evaluated, to demonstrate its classification ability and ensure that the project was completed.

Data and Setup

I. Project Data

For this classification project, the signal jets came from two sources. Firstly, the decay of W bosons to a quark-antiquark pair, which hadronizes, forming a collimated cone of detectable particles. Alongside this, the jet data from the decay of top quarks was given. As explained, the top quark decays quickly to a bottom quark and W boson, which itself decays to lighter quarks, forming jets. These jets allow for reconstruction of the top quark's properties, however, can be challenging to distinguish from each other. For this project, an attempt was made to separate these signals from the background jets produced from the hadronization of other light quarks and gluons created in interactions. Since modern day research often deals with high-energy, massive particles that can decay to many final-state particles, there are often many jets produced as background, highlighting the need for ML classification models. It should be noted that these three jets will hereinafter be referred to as W, top and Quantum Chromodynamic (QCD) jets, denoting the main particles involved or theory (QCD) describing them.

The jet information was given as four .npz files, related to four different energy ranges of measurement. Each dataset contained 30 subsections, with 10 per W, top and QCD jet type. The 10 categories contained different detection information for the jets, including data about different "Lund plane" measurements, which are theoretical representations of all the data in a system. More importantly, for this project, were the categories

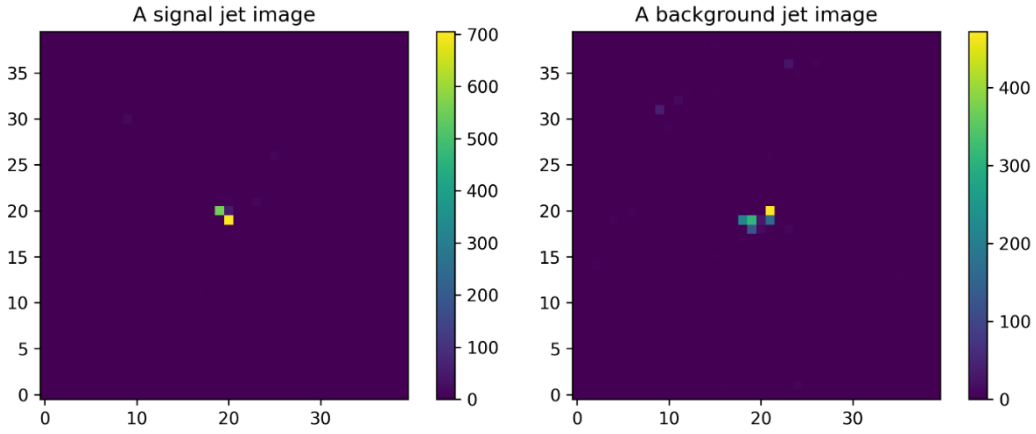

Figure 2: Input image examples for the different jet types, namely signal (left) and background (right).

containing the general jet images, which would be classified. Each category had 10,000 entries, with the jet images each being 40x40 pixels. The images themselves show the transverse energy of the jets and are centred on the jet's axis. The transverse energy is the energy perpendicular to the jet's axis and was used as a defining feature between the two jet types, since background jets generally have most transverse energy in the centre, whereas signal jets present a more spread, two-prong structure.

II. Notebook Setup

Before building any models, the coding environment had to be chosen and set up, with the local Jupyter Notebook application and server-based Google Colab being the two potential software options. Google Colab was chosen, for two main reasons. Firstly, since Google owns TensorFlow, Colab would not require any package installations to import the TensorFlow and Keras libraries. The second main draw to Colab was that by using Google's servers, small-scale access to their powerful computing hardware, featuring GPU acceleration, was available, which was used to reduce network training time drastically. It would later be seen that there were drawbacks to using Colab, including RAM limits and GPU usage constraints, however it is still believed to have been the superior choice.

Having created a notebook with Colab, the next step was to import the necessary libraries. Since this project required complex ML techniques, the list of imports was extensive, but featured primarily the TensorFlow, Keras and sklearn ML libraries. The list of imports was continuously updated throughout the project, as more modules were required. After this, the notebook was linked to a Google Drive, such that the datasets could be loaded.

III. Loading Data

Once the Drive was linked, the data itself needed to be loaded, which involved making a list of the paths to each .npz file, before using the ***numpy.load()*** function. The challenge came when selecting the necessary images from these loaded datasets and manipulating them into a form suitable for training a model. This was achieved by a function, which performed the following:

- Used a single filename and signal selection string as inputs.
- Loaded the image data for all W, top and QCD jets.
- Selected which signal to use based off the input string, before defining the signal and background image arrays (X_S and X_BG).

- Created labels for each image array (y_S, y_BG).
- Combined image and label arrays together, with reshaping.
- Split the data into training and testing datasets (X_train, X_test, y_train, y_test).
- Returned these training and testing pairs, alongside the pure images (X_S, X_BG).

Once created, this function was used to produce an initial training/testing dataset from the first, lowest energy-range .npz file. It should be noted that the returned X_S and X_BG image data was only used to plot random images of each type, to visualise the signal and background jets.

Creating a Model

I. Model Structure

For this binary classification project, a sequential convolutional NN was used, referring to a sequential model featuring convolutional layers. These layers are used to extract features from data, by using filters to recognise a particular pattern, regardless of its location in the data. This can be used to classify images, which aren't perfectly aligned or identical, as demonstrated in Fig. 3.

This process increases the amount of data however, as the original data set is reproduced by each filter. This could quickly result in uncontrollable amounts of data, and so convolutional layers are often used in conjunction with pooling layers, which summarise areas of data, reducing the overall size. One example is max pooling, which condenses areas (E.g., 2x2 pixels) by selecting only their maximum values (1 brightest pixel).

These layers formed the basis of our model, with the first convolutional layer taking the input image. However additional layers were still needed for the network to be usable for classification. The output of the model also needed to be structured, as two output neurons were desired, representing the likelihoods of the input image representing either a signal or background jet. A single input image would be transformed to multiple smaller images by the conv-pooling layers, which then needed to be condensed to the two output neurons. Therefore, a 'flatten' layer was used to smoothen the 2D image data to a 1D array, which was then passed to a dense layer, which would allow for reduction to the desired output. Thus, a combination of conv-pooling-flatten-dense layers represented the initial structure of our model, as shown in Fig. 4, which still required compilation, training and optimisation.

The compilation of a model is the final step before training, and involves defining features about the model, most important of which being the loss function, optimiser and evaluation metrics. The loss function is a mathematical function which evaluates how well the model's outputs match the true values. The aim of training a NN is to minimise this loss, which occurs through use of an optimiser. As described earlier, this is the algorithm used to update the model's parameters. Finally, the metrics are used to evaluate the model's performance at a task. Unlike the loss, which is used to update the model's parameters, the metrics are only included to assess the performance and ability for a model to complete a task.

There are many different optimisers, loss functions and metrics; each with varying applications, however, the following were used initially:

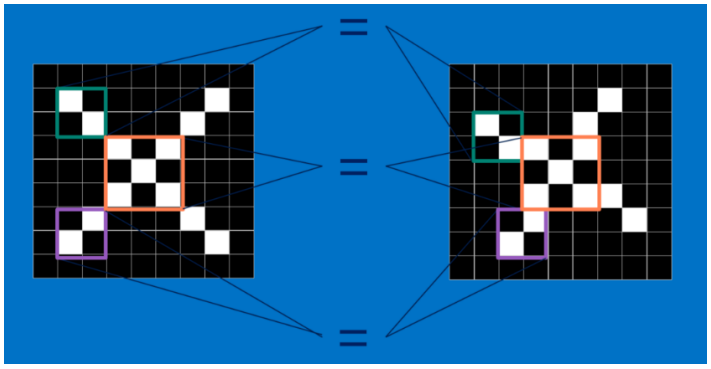

Figure 3: A visualisation of how filters are used to recognise patterns in convolutional layers.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 38, 38, 64)        640

 max_pooling2d (MaxPooling2D  (None, 19, 19, 64)       0
 )

 dropout (Dropout)           (None, 19, 19, 64)        0

 flatten (Flatten)           (None, 23104)             0

 dense (Dense)               (None, 500)               11552500

 dense_1 (Dense)             (None, 2)                 1002

=================================================================
```

Figure 4: The initial layer structure of the model. Notice the additional dropout layer, which was included as a standard method of improving generalisation when training.

- Optimiser – Adam – A common SGD algorithm which is widely applicable and uses adaptive momentum features to minimise loss quickly.
- Loss function – Sparse Categorical Cross-entropy – Cross-entropy is a standard measure of performance for classification models which output probabilities. Categorical cross-entropy is used as there are two jet classes, which are sparse as two output probabilities are requested, despite using a single input label format.
- Metrics – Accuracy – A common indicator of performance, describing the fraction of model classifications which are correct.

II. Training and Validation

To minimise a model's loss and improve its performance at completing a task, the model's parameters must be optimised through training. In this project, supervised training was used, which as explained earlier, involves a NN learning to match input data with associated labels. For this project, the training images and labels were created for each .npz file individually, using the ***import_data()*** function defined earlier. This would occur in a loop, with each loop featuring the creation of training/validation data from a .npz file, and then the training of the model on this data.

The ***import_data()*** function created labels, such that the model could be trained to 'learn' which images correspond to signals (1) and which to background jets (0). Not all images taken from the four .npz files were used to train the model however, as some must be left 'unseen' by the model, for performance evaluation without bias. If the model was trained and tested on the same data, then the model's ability to generalise to different data would be unknown. Therefore, a proportion of the total set of images (25%) had already been split-off and used to 'validate' the model's ability to generalise during each step of training, but not affect the optimisation of parameters whatsoever.

The function used to train the model was Keras' ***model.fit()***, which took input training images/labels, a number of epochs (training loops), a batch size and a validation set of images/labels. As will be seen later, the training history of a model can be shown, and can provide valuable insights into how the metrics and loss progress per

epoch. These can help identify areas for optimisation, as well as highlight potential issues with the model, as will be explained below.

Three key points should also be noted, starting with how the signal images were only selected to be from W jets for this project. Due to time constraints, it was decided that only one model would be trained, meaning that only one signal type would be used. Secondly, it should be noted that the model was initially trained on each of the 4 image sets in sequence, starting from the lowest energy range. This sequential training would later be seen to be suboptimal. Finally, this model produced two outputs, namely the probabilities of an image corresponding to a signal or background jet. It should be noted that these probabilities are independent, and do not sum to 1.

III. Optimisation

The optimisation of a NN is essential to improving its performance, and can involve adding new layers, changing the quantity or ordering of current layers and the adjustment of layer features. These optimisations may improve different aspects of the network, including its ability to generalise, predict or achieve very high accuracies. The challenge with optimisation, however, is that there is always an element of randomness with training NN's, due to the random nature of gradient descent, and so the result of changing a model's features may not be unclear or may differ when retrained. When the computationally expensive and time-consuming nature of training a complex model is also considered, then the challenge of optimisation becomes more apparent. Since this project only involves training a network and evaluating its performance, creating a reasonably successful model was prioritised, and optimisation occupied most of the time.

Starting from the model seen in Fig. 4, the effects of multiple features were investigated, usually by running the model creation and training code within a loop, changing a single feature's value with every run. In some cases, the final metric values (accuracy, loss, validation accuracy and validation loss) were saved as a .csv file, such that plots could then be made.

The first feature to be investigated was the number of convolutional layers used, which produced the results in Table 1. These values suggested that fewer convolutional layers were better, however, from experience gained during the PHAS0056 Week 4 MNIST classification task, it was known that more convolutional layers usually improved performance, especially when considering such few layers. Adding layers could correspond to improved final performance, however due to the increased number of parameters, more training would be required to optimise them, so better results may not be seen when trained for the same time. As a compromise between the theory and results, two convolutional layers were used initially.

TABLE I. The effect of the number of convolutional layers on model accuracies.

Number of Convolutional Layers	Accuracy	Validation Accuracy
1	0.6679	0.6640
2	0.6282	0.6560
3	0.5208	0.5134

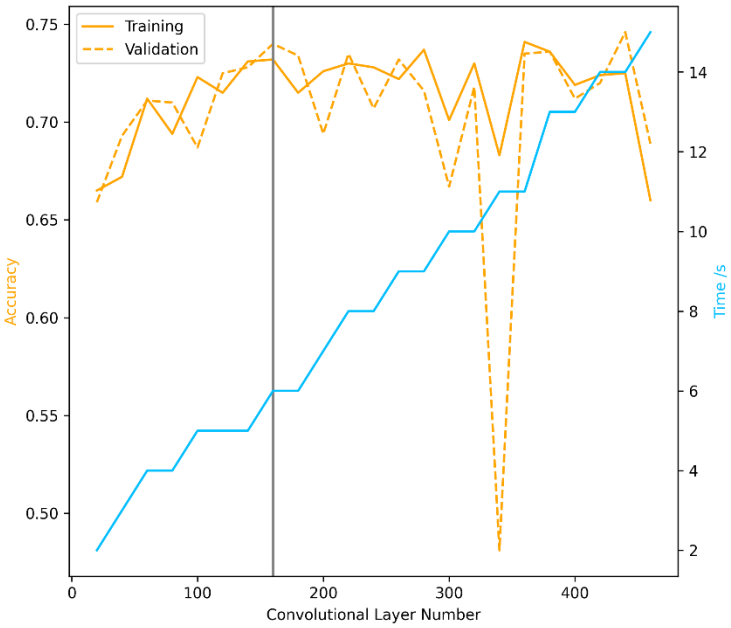

Figure 5: The recorded data for the investigation into the optimal number of convolutional layers to use. Their effect was measured against the training and validation accuracies, and training time.

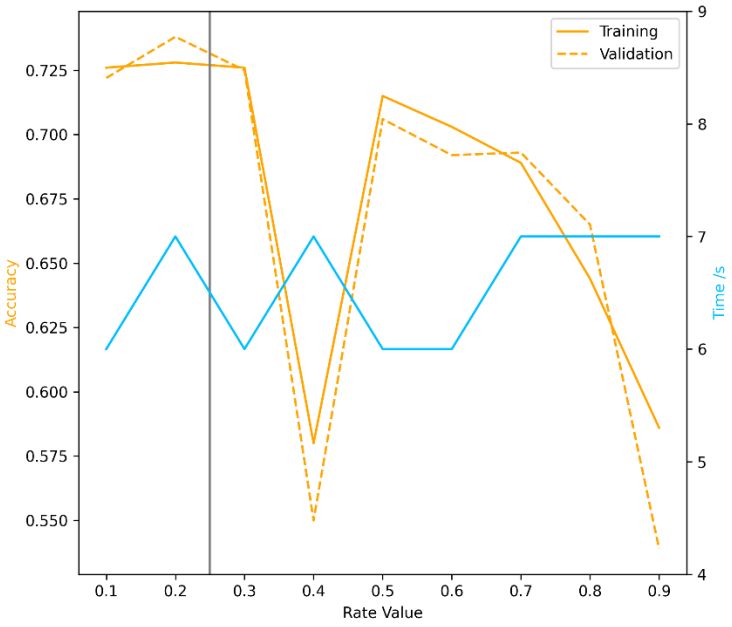

Figure 6: The effect of the rate value for the dropout layer on the training and validation accuracies, and training time.

Secondly, the number of filters used in the convolutional layer was investigated. A loop was used, to adjust the filter number over the range set by ***np.arange(20, 500, 20).*** This test produced Fig. 5, from which a value of 160 filters was decided as optimal.

After this, the dropout rate was investigated, to see whether different rates would allow for better model generalisation. A loop was run for rates in the range ***np.arange(0.1, 1, 0.1),*** and Fig. 6 was produced, although it was deemed unreliable. Unfortunately, due to GPU acceleration usage limits and the number of other features to be investigated, no further investigation was performed, but 0.25 was taken as the optimal compromise between time and performance, before being later increased to 0.3 through random testing.

The number of dense layers was then investigated, producing the results in Table. 2, which were seen to fluctuate almost randomly. It was noticed that the accuracy would decrease with every run after a kernel restart, despite the number of dense layers. Therefore, the results were deemed unreliable again. As with the convolutional layers, a higher number of layers could improve performance, however, would require more training to be seen. Furthermore, since dense layers are fully connected, increasing the number of layers would also drastically increase the run-time, as well as increase the chance of overfitting, a common issue which will be discussed later. For these reasons, two dense layers were included in the model (not including the output layer), as a compromise.

TABLE II. The effect of different numbers of dense layers on model training and testing accuracies.

Number of Dense Layers (excl. output)	First Run		Second Run	
	Accuracy	Validation Accuracy	Accuracy	Validation Accuracy
1	0.7107	0.7392	0.7459	0.7278
2	0.6711	0.7024	0.7056	0.7298
3	0.7239	0.7576	0.6620	0.6878

The batch size was investigated next, with results shown in Table. 3.

TABLE III. An investigation into the effect of training batch size on evaluation metrics

Batch size	Loss	Accuracy	Validation Loss	Validation Accuracy
32	0.3987	0.8179	0.7121	0.6762
64	0.4852	0.7686	0.4787	0.7722
128	0.5286	0.7283	0.5122	0.7444

Research about the batch size was performed, and it was discovered that decreasing the batch size could improve training accuracy [7], however, this could also lead to overfitting, as is explained below. Therefore, it was decided that a larger batch size would be used, with more training epochs.

IV. Obstacles and Solutions

A. Overfitting

One of the most common issues faced by NN's is overfitting. This occurs when the model specialises to the training data set, such that its ability to generalise to unseen data is reduced. There are many reasons as to why a model would overfit the data, including not having sufficient training data. This would mean that there is insufficient data for the model to determine patterns, and so may attempt to fit the noise/anomalies instead. This is especially important for jet images, which can be very similar. Another cause of overfitting could be that the model has too many parameters relative to the amount of training data, such that each parameter could be optimised to exactly replicate a pixel of data, rather than find general patterns. When starting this project, it was presumed that the more complex the model the better, however this was a time-consuming and inaccurate assumption. Moreover, it was believed that training for more epochs would also improve the model's performance, however training for too long can also cause the model to become 'too specialised' at fitting the training set.

A common sign of overfitting was seen whilst training this model and is shown in Fig. 7. Here, the training accuracy continues to improve, but the validation accuracy remains constant, or decreases. This highlights how the model is only learning the training data and is struggling to generalise. This should have been expected, for a model with approx. 21 million parameters and 4 training sets of 10,000 images each.

To tackle the issue of overfitting and improve the validation accuracy, a few different techniques were used. Firstly, the number of densely connected neurons per layer was reduced from 400 to 200, and a second max pooling layer was added, to decrease the number of parameters to around 3 million. Secondly, two regularisation techniques, namely a second dropout layer and L1L2 regularisation, were implemented, which are used to help a model generalise. L1 and L2 are separate techniques which both rely on adding penalties proportional to the magnitudes of parameter weights. These often result in weights being set to 0, and so forces the model to use an ever-changing network structure, improving its ability to generalize. L1L2 regularisation is a function which makes use of both techniques. Unfortunately, these techniques did not prevent the validation accuracy from

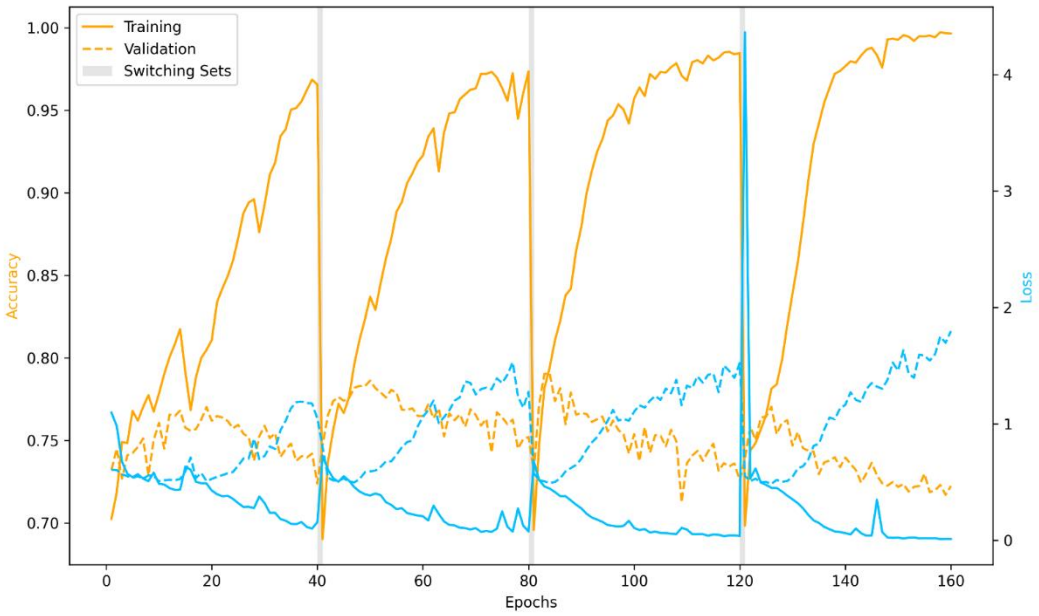

Figure 7: A plot of the model's training history, displaying clear signs of overfitting. Despite the large number of epochs and high training accuracy, the validation accuracy continues to decrease.

decreasing once peaked, so another technique known as 'early stopping' was tested. This involved setting a call-back, such that the model stops training once the metric peaks. A patience setting was included, such that the validation accuracy could fluctuate a little over epochs before training was stopped. This prevented the model's accuracy from decreasing, however, did not allow for much training, and so was deemed more of a way to hide the model's poor structure, than improve it. The model's training accuracy was vastly greater than the validation performance, and so the number of epochs was also reduced from 40 to 15.

It was then decided that the number of parameters may be too high still, so a third max pooling layer was added. This reduced the number of parameters further to around 780,000 and increased the validation accuracy to around 0.8. The number of densely connected neurons was then randomly tested again for values from 50 to 500, and counterintuitively, 400 was seen to produce the best validation accuracy. This increased the number of parameters to around 1.3 million and showed how random the optimisation of NN's can be. These adjustments produced the metric values shown in Fig. 8.

B. Sequential Training

Due to time constraints, the optimisation process was halted, and the evaluation of the model was attempted. This model's accuracy was evaluated on each of the four datasets, and a significant issue was observed. The model's performance was significantly worse on the datasets trained earliest in the sequence and improved towards the later datasets (Validation accuracy increase from 0.4 - 0.8). In retrospect, this was as to be expected, as over the course of training the parameters would be optimised for the current dataset and would move away from the optimal values for classifying previous image sets. Therefore, it should have been obvious that the model would perform better on the final dataset of the sequence, and worse on the first. This error of sequential training became very apparent when the order of training image sets was reversed as compared to Fig. 8, producing Fig. 9. This suggested that the datasets should be merged and shuffled before being used for training, such that the model could generalise to classify the jets for all energy ranges equally well.

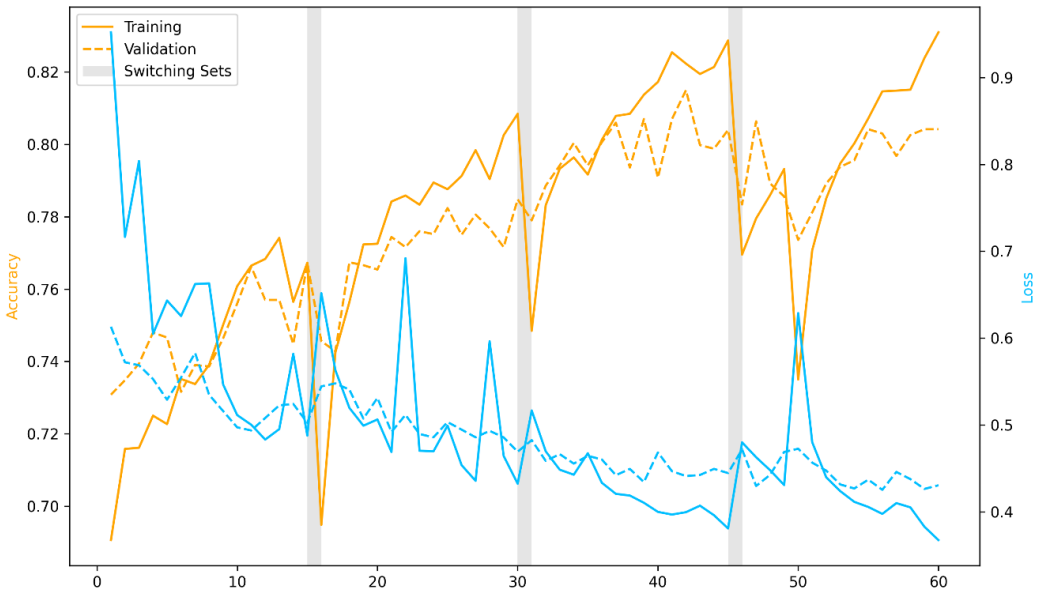

Figure 8: The training history for the current model structure, demonstrating how overfitting has been reduced. This reduction is especially obvious when the history is compared to that of Fig. 7.

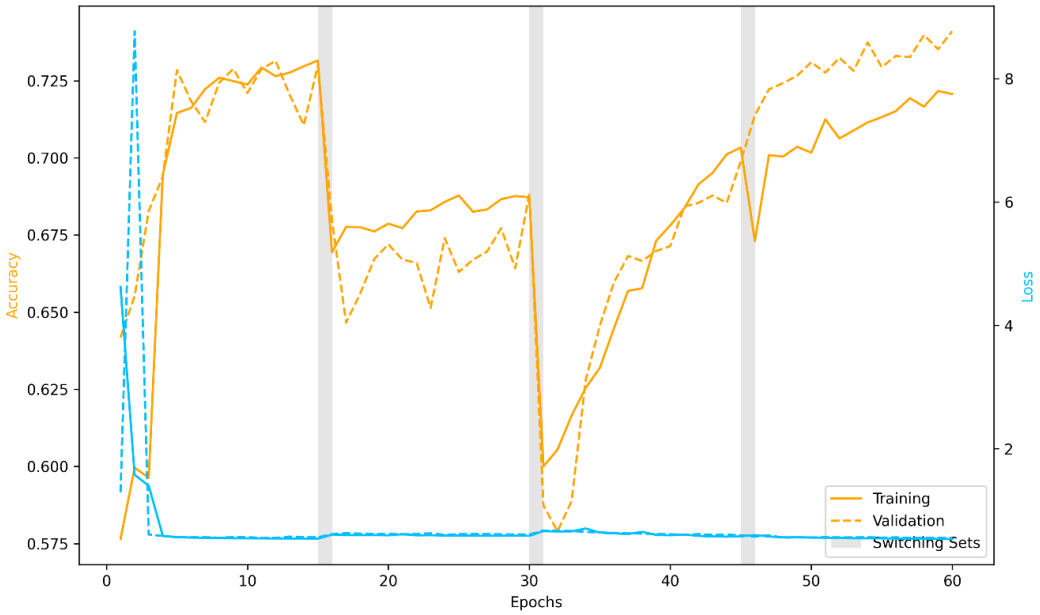

Figure 9: The history of the model, as trained on the image sets in reverse order; decreasing from the highest energy range.

Merged Data Training

I. Optimisation

The process of merging the datasets was more complicated than predicted, due mainly to RAM constraints within Colab. When attempting to load and append the 30,000 jet images per .npz file into a list, the program would crash. Appending only a certain number of each 10,000 images per jet type was then attempted, by using array slicing on the data loaded, when looping over each .npz file. This meant that less RAM would be used, and that merged training and validation datasets could be produced successfully. The ***import_data()*** function was updated, to take the full list of filenames as an input, and would loop through each when run, returning merged data sets.

Half of each 10,000-image set was loaded initially, however during training, the metrics showed an accuracy of 0.5 consistently, indicating that the model was guessing. The error was assumed to be due to a disparity between images and labels, however after rigorous checking and the removal of unnecessary code, these were shown to be consistent. Therefore, it was concluded that the images from the different energy ranges may not have been comparable, such that merging them together made it difficult for the model to find distinct patterns.

In an attempt to fix the 0.5 accuracy, the optimiser was switched to RMSProp. This is another commonly used optimiser, which was chosen due to its ability to often outperform adam for problems with similar features, such as how similar the signal and background images were. Using RMSProp raised the validation accuracy, which would now plateau at around 0.7.

More optimisation was then attempted, including using larger fractions of the image data. Interestingly, using the loading method created, the number of images could be increased gradually from 5,000, 7,000, 9,000, 9,500 to 10,000, and the program would no longer crash. The number of neurons per densely connected layer was also investigated again, since more training data could now be loaded. Having 400 neurons in the first dense layer was found to overtrain the data, as shown in Fig. 10, whereas a smaller model with only 200 neurons (780,000 parameters) was quicker to train and yielded slightly improved accuracy and loss results.

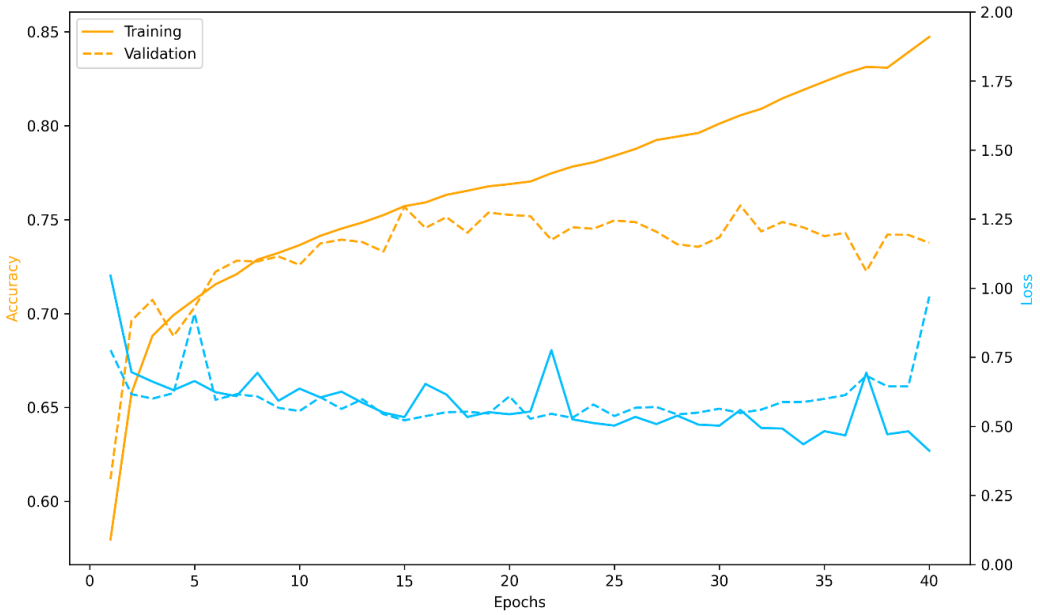

Figure 10: The training history of the current model, showing clear signs of overfitting, as the validation accuracy remains constant, despite the training accuracy increasing.

For similar reasons, the addition of another convolutional/max pooling layer combination was tested and improved the validation performance minimally. The batch size was also tested again, producing the results in Table IV, which gave no clear conclusion. As before, a larger batch size was decided upon.

TABLE IV. Data describing the effect of different training batch sizes on the evaluation metrics.

Batch size	Loss	Accuracy	Validation Loss	Validation Accuracy
32	0.6085	0.7228	0.6222	0.7365
64	0.6586	0.6493	0.6120	0.6941
128	0.6122	0.7566	0.5233	0.7526

II. Final Model

The final model had 1,015,522 parameters and is shown with its training history in Fig. 11 below. It was trained using 75% of a merged set of 40,000 signal and 40,000 background jet images, over 20 epochs, with a batch size of 128. The remaining 25% was used for validation. The model was compiled with the RMSProp optimiser, SparseCategoricalCrossentropy loss function and accuracy metric.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 38, 38, 160)	1600
conv2d_1 (Conv2D)	(None, 36, 36, 160)	230560
max_pooling2d (MaxPooling2D)	(None, 18, 18, 160)	0
conv2d_2 (Conv2D)	(None, 16, 16, 160)	230560
dropout (Dropout)	(None, 16, 16, 160)	0
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 160)	0
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 160)	0
flatten (Flatten)	(None, 2560)	0
dense (Dense)	(None, 200)	512200
dropout_1 (Dropout)	(None, 200)	0
dense_1 (Dense)	(None, 200)	40200
dense_2 (Dense)	(None, 2)	402

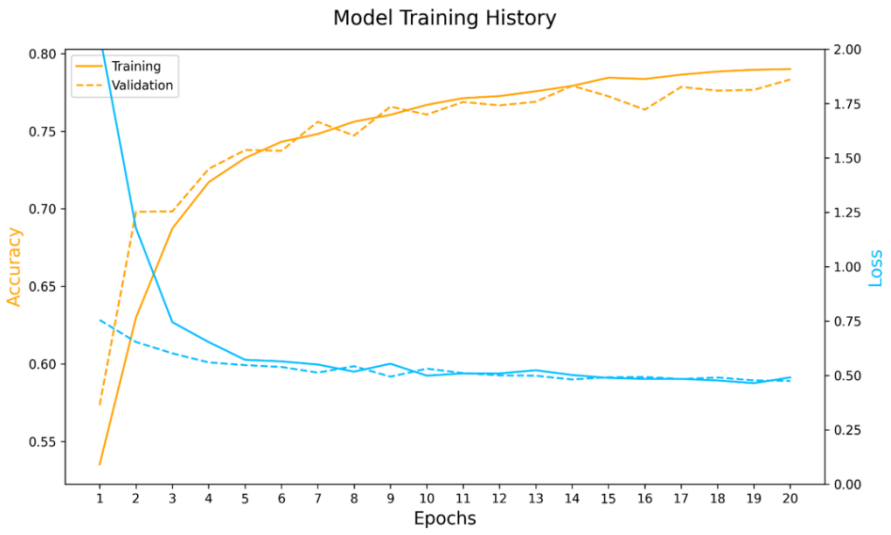

Figure 11: A description of the final model, featuring its layer structure (left) and training history (right).

III. Evaluation

To evaluate the model, different metrics were used, with the results shown in the first row of Table V. The first metric investigated was accuracy, which is simply the number of correct classifications out of the total number made. The model's accuracy was evaluated using the validation data of the merged image set, as well as the validation sets of each individual energy range.

The accuracy results show that the model learnt to generalise effectively, as it performed successfully when test ed on the merged and individual image sets. In fact, the model performed better on each individual data set than their combination, indicating that the jets at different energies may share similar features, such that a model trained across the full range is better suited to picking out subtle differences within each energy range. Furthermore, the accuracies show that the third energy range has the most easily distinguishable signal jet, as the accuracy is highest. Therefore, if this model were to be used to classify jets for an experiment, use of this energy range should be suggested for the best results.

Following the accuracy, the F1 score was calculated, which is a metric combining the precision and recall abilities of the model. The recall quantifies how accurate the model is when predicting positive results, whereas the precision quantifies how well the model identifies all positive results. The precision and recall values require the model to make predictions, so ***model.predict()*** was used with the total validation image set. The calculated results are shown in Table V, and since all three scores are on a scale of 0-1, then the results are seen to indicate a successful model, to a certain extent.

TABLE V. The metric data determined through evaluation of the model, which was trained on W jet signals. The two rows correspond to the signal jet images used for validation/evaluation, as the model's applicability to classifying top jet signals was also tested.

Signal jet evaluated	Accuracy					Precision	Recall	F1 Score	AUROC
	All images merged	1st Energy Range	2nd Energy Range	3rd Energy Range	4th Energy Range				
W	0.7833	0.7934	0.7992	0.8176	0.8096	0.7787	0.7940	0.7863	0.8263
Top	0.6672	0.7070	0.6548	0.6492	0.6708	0.7254	0.5389	0.6184	0.6774

The precision score indicates that when the model predicts a specific jet type, it is correct 78% of the time. The recall of approx. 0.79 however, suggests that out of all jets, 79% were correctly identified, and approximately 21% were misidentified. Overall, these scores combine to make the F1 Score, which indicates a reasonably good model.

The final metrics used to evaluate the model were the Receiver Operator Characteristic (ROC) curve and corresponding Area Under the ROC curve (AUROC). The ROC curve is a probability curve, used to evaluate the binary classification performance of the model, at different thresholds. Here, thresholds refer to different false positive (FPR) and true positive rates (TPR), which represent the two axes of the curve plot, as shown with the results in Fig. 11. The associated AUROC quantitatively summarises how well the model distinguishes between the two jet classes. An AUROC of 1 describes a model which correctly predicts signals as signals every time, whereas an AUROC of 0 would predict signals as background every time. Therefore, an AUROC of 0.5 (a y=x ROC) is the baseline, as this could be achieved by random classification. The AUROC for the ROC curve shown in Fig. 12 is given in Table V and is considered excellent [8], suggesting that the model has strong classification capabilities.

For clarity, histograms of the model's performance were also produced, to visualise how the model assigns probabilities to each jet type and see whether the larger 'winning' probability is always high, or just higher than the other (see Appendix A). Finally, despite not having the time to train and optimise a model on the top quark jet images, the W jet model could still be evaluated with the top quark images, to see how well it performs. This evaluation was performed for every metric described above, producing the data in the second row of Table V, with ROC shown in Fig. 12.

These scores show that the model trained on W jets would still be better at classifying top jets from the background than random guessing, however with a large reduction in performance as opposed to when W jets are tested. This is as to be expected, as the model was trained using W jets, and therefore would obviously be able to classify those signals better. However, the fact that the AUC is still 0.68 signifies that the top jets must be reasonably similar to the W jets, and both notably distinct from the background jets.

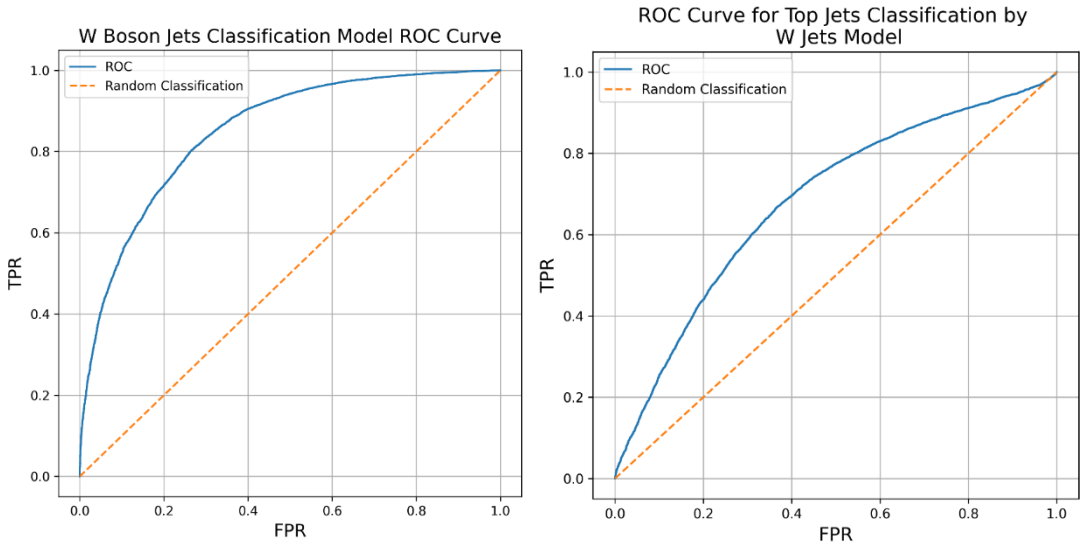

Figure 12: The ROC curves for the model trained on classifying W jets, as evaluated on W jets (left) and top jets (right).

Conclusion

The classification of w-boson jets from background QCD jets by a sequential convolutional NN has been performed. Through supervised training on jet images, an F1 Score of 0.79 and AUROC of 0.83 were achieved. These demonstrate that the model can successfully distinguish between the jet images, to respectable accuracy. The model was also tested on classifying top jet signal images, despite being untrained on them. An F1 Score of 0.62 and AUROC of 0.68 were achieved, showing reduced performance as expected, although still displaying an improvement above random classification, highlighting that signal jets are more similar to each other than to background jets.

Concerning improvements to the model, a more structured approach to the optimisation could have been performed. More optimisation data could have been recorded to avoid drawing unreliable conclusions and improve the model, however this was often due to limits in Colab, since the model was time-consuming to test without GPU acceleration. Furthermore, having more training images could also have improved the model's performance, however, this would have introduced more issues due to Colab RAM limits. Therefore, obtaining more local computing power such that different coding software could have been used, could have helped optimise the model better and improve validation performance. Given more time, a model could have been trained on the top jet images, to see which signal jet type is more easily distinguishable from the background.

References

[1] G. Kasieczka, "CERN Document Server," 14-16 November 2018. [Online]. Available: https://indico.cern.ch/event/745718/contributions/3205082/attachments/1753205/2841505/JetTagging_Overview.pdf. [Accessed 26 March 2023].

[2] C. Vittori, "CERN Document Server," 30 March 2019. [Online]. Available: https://cds.cern.ch/record/2669538/files/ATL-PHYS-PROC-2019-028.pdf. [Accessed 26 March 2023].

[3] A. Buzatu, "CERN Document Server," 27 November 2018. [Online]. Available: https://cds.cern.ch/record/2651122/files/ATL-PHYS-SLIDE-2018-1037.pdf?version=1. [Accessed 26 March 2023].

[4] K. Kourou, T. P. Exarchos, K. P. Exarchos, M. V. Karamouzis and D. I. Fotiadis, "Machine learning applications in cancer prognosis and prediction," *Computational and Structural Biotechnology Journal,* vol. 13, pp. 8-17, 2015.

[5] A. El Sallab, M. Abdou, E. Perot and S. Yogamani, "Deep Reinforcement Learning framework for Autonomous Driving," *IS&T Electronic Imaging, Autonomous Vehicles and Machines,* vol. 29, pp. 70-76, 2017.

[6] R. Zaheer and H. Shaziya, "A Study of the Optimization Algorithms in Deep Learning," in *2019 Third International Conference on Inventive Systems and Control (ICISC)*, Coimbatore, 2019.

[7] S. S. A. B. Aldin and N. B. Aldin , "Accuracy Comparison of Different Batch Size for a Supervised Machine Learning Task with Image Classification," in *2022 9th International Conference on Electrical and Electronics Engineering (ICEEE)*, Alanya, 2022.

[8] J. N. Mandrekar, "Receiver Operating Characteristic Curve in Diagnostic Test Assessment," *Journal of Thoracic Oncology,* vol. 5, no. 9, pp. 1315-1316, 2010.

Appendices

I. Appendix A

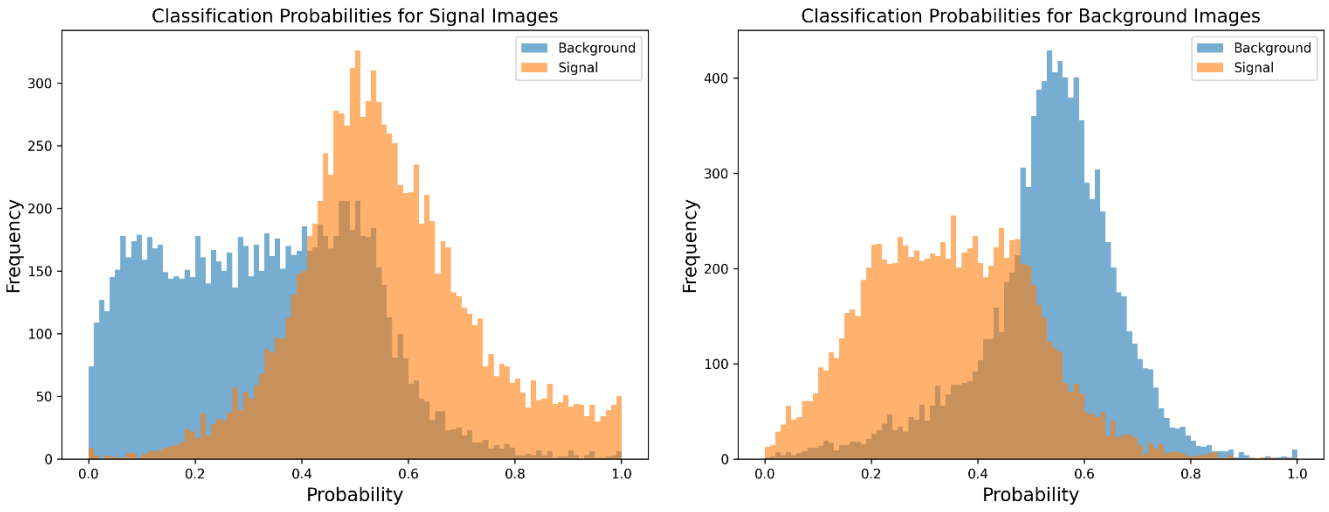

Figure 13: The output classification probabilities, representing the probabilities of an image being either a signal or background, for the actual signal (left) and background images (right).

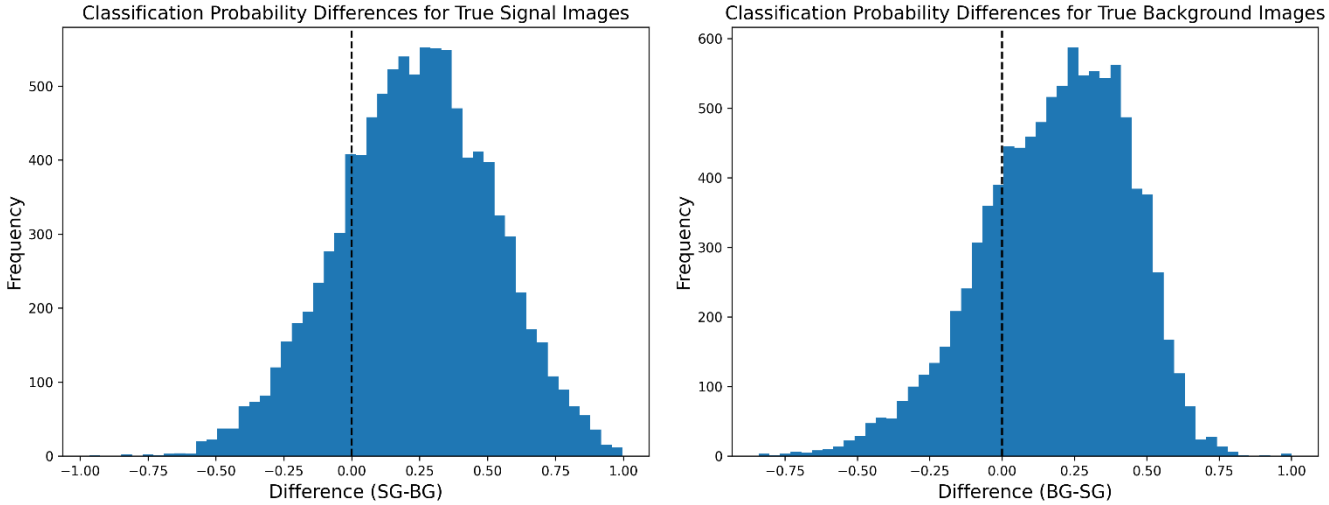

Figure 14: The difference between the output classification probabilities, for the actual signal (left) and background images (right). The subtraction was made from the correct jet type, to demonstrate the model's successful classification ability.