The Application of Neural Networks in Configuring the LHCb Vertex Detector

A thesis submitted in partial fulfilment of the requirements
of the University of Liverpool for the degree of Bachelor of Science
by

Daniel Glover
(201409569)

Under the supervision of
Dr David Hutchcroft

Department of Physics
The University of Liverpool
May 2022

Abstract

The upgraded LHCb VELO vertex detector is a pixel detector containing 41 million 55µm × 55µm silicon sensors. The analogue pixel signal is read out by the custom developed VeloPix ASIC containing many components including ADCs. As a result of the variation between pixels, equal charges dumped into two pixel sensors are usually not measured as equal by their respective ADCs, which is corrected for by the ADC trim. This report investigates the use of neural networks to predict the appropriate trim level for a pixel based on its measured ADC output at minimum and maximum trim. The trained neural networks make predictions accurate to ± 1 level in 100% of pixels and predict the exact trim in $\sim$ 97% of pixels. The impact of using trims predicted by neural networks on the particle track reconstruction efficiency was investigated using LHCb simulation software, and it was concluded that providing noisy pixels are masked, the efficiency remains acceptable (<1% below perfect).

Contents

Declaration

I hereby declare that this thesis is my own work and that it has not been submitted elsewhere for any award. Where other sources of information have been used, they are acknowledged.

Signature:

Date: 13/05/2022

1 Introduction

1.1 The LHCb Vertex Detector (VELO)

The purpose of a vertex detector is to reconstruct the angle and direction of particle tracks originating from an interaction or decay vertex. VELO (VErtex LOcator), the current vertex detector at the LHCb project is composed of many layers of silicon strip sensors surrounding the main beam. The new VELO upgrade is a pixel detector comprised of 52 modules, each of which contains 4 rectangular tiles, arranged in two opposite-facing L-shaped pairs such that they enclose the beam (the grey rectangles visible in Figure 1). Each tile consists of 3 adjacent square sensors (256x256 grids of 55µm silicon pixels) which are bump bonded onto 3 VeloPix readout chips[1][7].

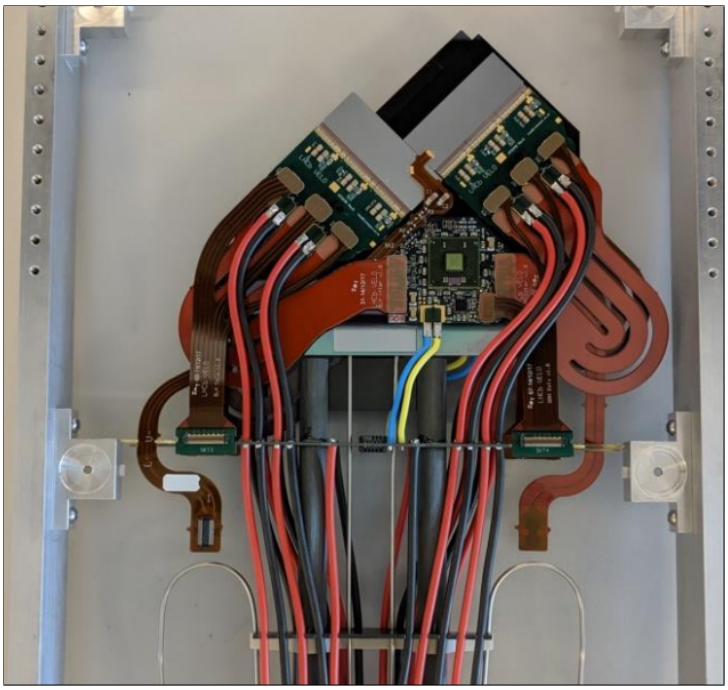

Figure 1: Photograph of an assembled VELO module.[7]

When a particle passes through one of the sensor pixels, a current pulse is induced (see Section 2.1) which is fed into an amplification circuit. An ADC (analogue to digital converter) constantly measures the output of the amplifier, converting it to an integer between 0 to 2047. The silicon sensor response varies from pixel to pixel, causing differences in the gain depending on which specific pixel a particle passes through. To combat this, each ADC has an adjustable trim value (set by a value from 0 to 15), which adjusts its output up or down. By assigning each pixel an appropriate trim level the variation can be suppressed, allowing reliable particle track detection by comparison of the trimmed output to a defined threshold.

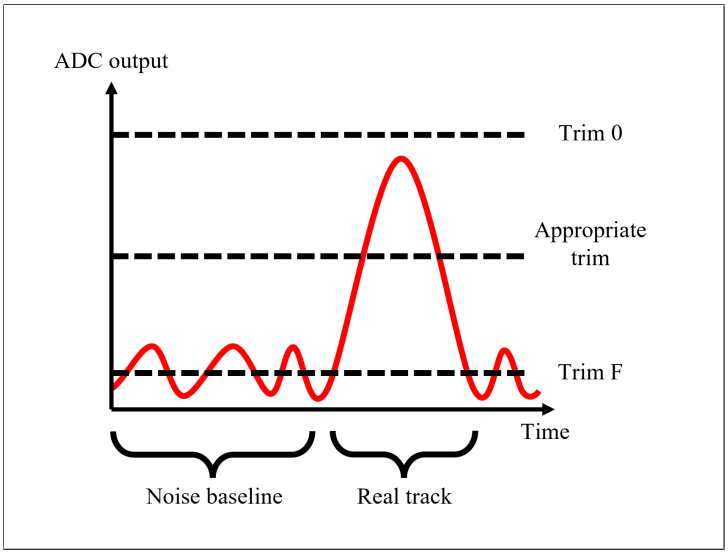

Figure 2: Simplified depiction of how the trim adjusts the ADC output relative to the detection threshold, demonstrating why an appropriate trim is necessary.

Clearly, the variation between pixels means that each must be analysed to determine the best trim value for that specific pixel. To aid this, a capacitor is built into each pixel, which can be charged to the maximum chip voltage before dumping all of its charge into the sensor (this simulates the current induced when a particle passes through the pixel). The same amount of charge is dumped in each pixel, since the capacitors are all identical (ignoring negligible manufacturing variations). However, the aforementioned variations in the sensor/readout chip mean that the value measured by ADC will differ from pixel to pixel. By testing all 16 trim levels in turn for each pixel, an appropriate trim is selected for each. This process was carried out following the manufacture of the sensors, and some pixels flagged as problematic were masked. (Note: trim level 15 may later be referred to in hexadecimal as trim 'F'.)

1.2 Neural network investigation

The issue arises in the fact that during operation, radiation damage decreases the signal amplitude of some pixels, while simultaneously increasing their baseline noise level. For this reason, the trims must be periodically checked in-situ to ensure they are still appropriate. However, although the process of scanning all 16 levels was effective during manufacture, it is too time-consuming to be regularly performed during operation, and the purpose of this work is to investigate a faster alternative using neural networks. If a neural network was able to accurately determine the best trim levels using only the test-charge ADC output at minimum and maximum trim (0 and F, rather than all 16), then only two trim levels would need testing periodically, potentially speeding up the re-trim process eightfold.

The test-charge process described in Section 1.1 was carried out several times with all pixels set at trim 0, then again at trim F. For a given pixel, the result of this process was a range of ADC output values in both configurations (0 and F). To investigate the potential of the proposed method, the mean and width of these ranges (and the mask variable) were used to train neural networks, with the pre-calculated trim values as the "truth" data (the correct values, that the neural networks learned to reproduce). The accuracy metric of a neural network was defined as the fraction of pixels that were successfully assigned their "true" trim value. The neural networks were then tuned for maximum accuracy via optimization of their hyperparameters (see Section 2.1).

1.3 Data overview

(Note: 'sensor' refers to a 256x256 pixel sensor grid, not an individual pixel)

Each pixel is described by 5 variables, which are stored in 5 separate 256x256 CSV files, matching the dimensions of the sensor pixel grid. The test-charge ADC output of each pixel was measured several times at minimum and maximum trim, producing two distributions specific to that pixel. This provides 4 of the 5 variables: the mean and width of the trim 0 distribution, and the mean and width of the trim F distribution. In addition to these, a mask variable is used to identify several categories of pixel identified as problematic during the testing process (e.g., pixels too noisy to be rectified via trim). These 5 variables were used as the input data when training neural networks, which aim to reproduce the pre-calculated trim levels of each pixel (also stored in a 256x256 CSV file).

1.3.1 Input data

The 5 variables describing the VP0-0 sensor pixels (used as the neural network input data) are summarised below:

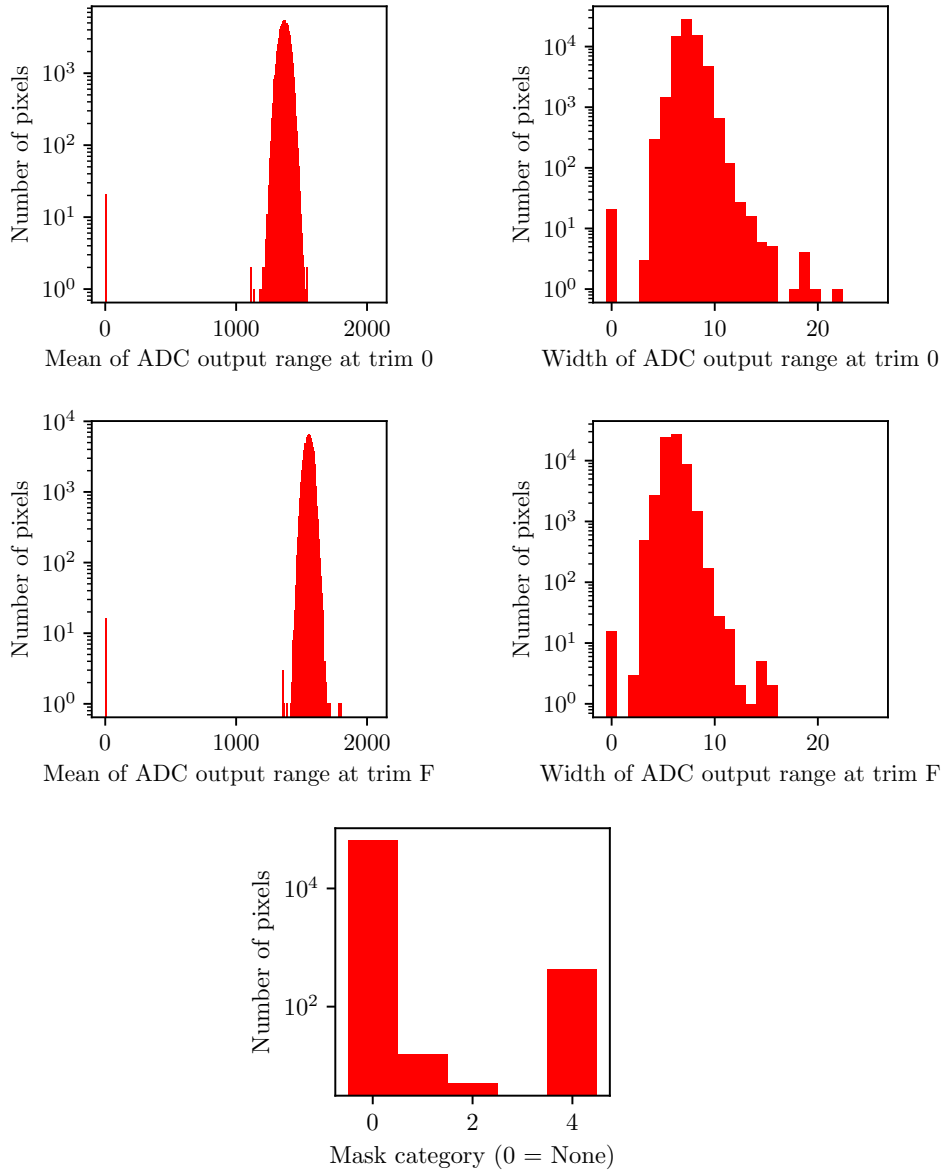

Figure 3: Histograms of the 5 neural network input variables describing the VP0-0 sensor: mean and width of the ADC output ranges at minimum and maximum trim, and pixel masks pre-applied following manufacture (before any radiation damage).

1.3.2 Truth data

The pre-calculated trim values (used as the neural network truth data) of the VP0-0 sensor can be seen below:

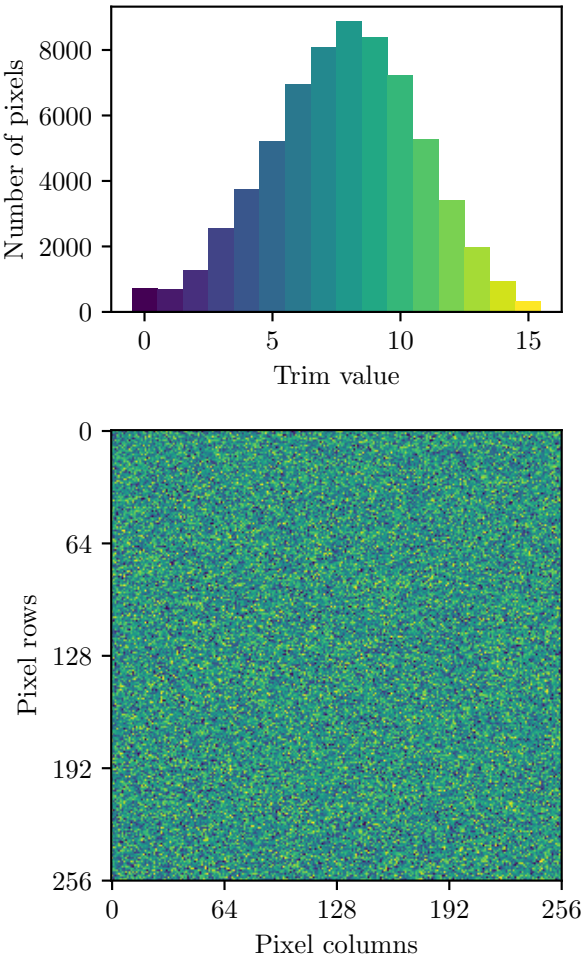

Figure 4: Example of "true" trim levels calculated during the testing process (Module 0 VP0-0 sensor). The histogram shows the number of pixels with each trim level, and the grid shows their positions on the sensor.

1.3.3 Correlations between input and truth data

Each of the 5 input variables were plotted against the output variable (the true trim) in 2D histograms to indicate (if any) the correlation between them. Patterns in the data (such as correlations) are fundamental to the neural network training process, as it learns to use these to arrive at its prediction.

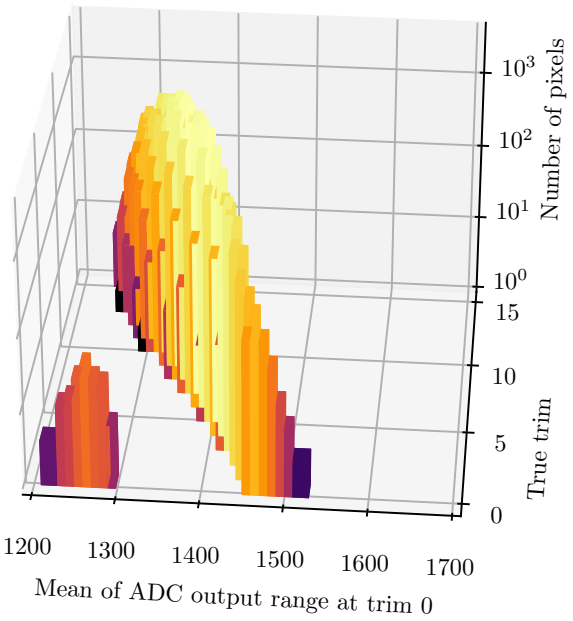

Figure 5: Histogram showing the relationship between the ADC output range mean at trim 0 (input) variable and the true trim (output) variable (Module 0 VP0-0 sensor).

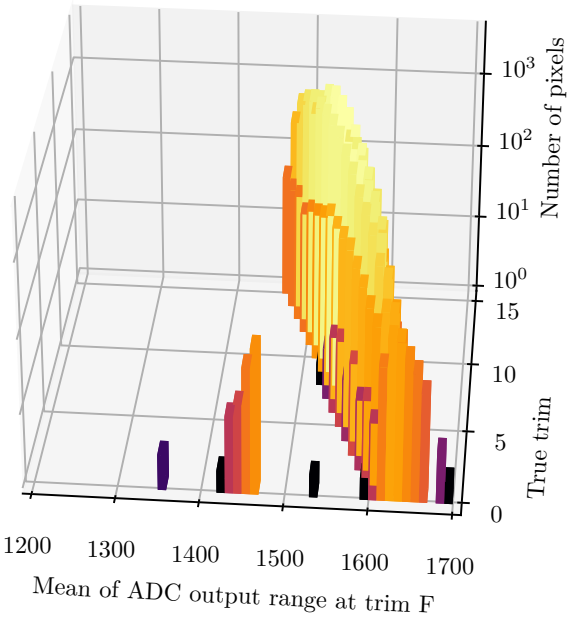

Figure 6: Histogram showing the relationship between the ADC output range mean at trim F (input) variable and the true trim (output) variable (Module 0 VP0-0 sensor).

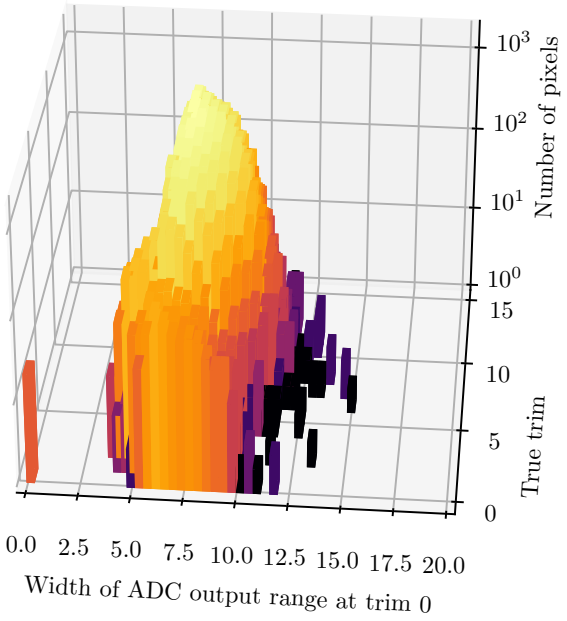

Figure 7: Histogram showing the relationship between the ADC output range width at trim 0 (input) variable and the true trim (output) variable (Module 0 VP0-0 sensor).

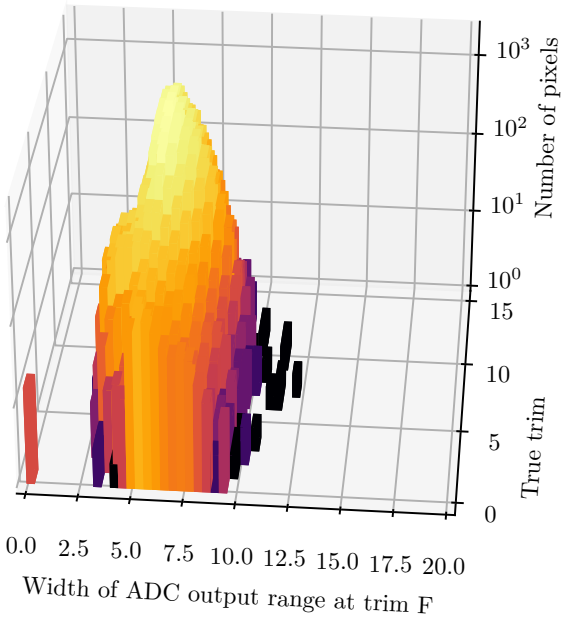

Figure 8: Histogram showing the relationship between the ADC output range width at trim F (input) variable and the true trim (output) variable (Module 0 VP0-0 sensor).

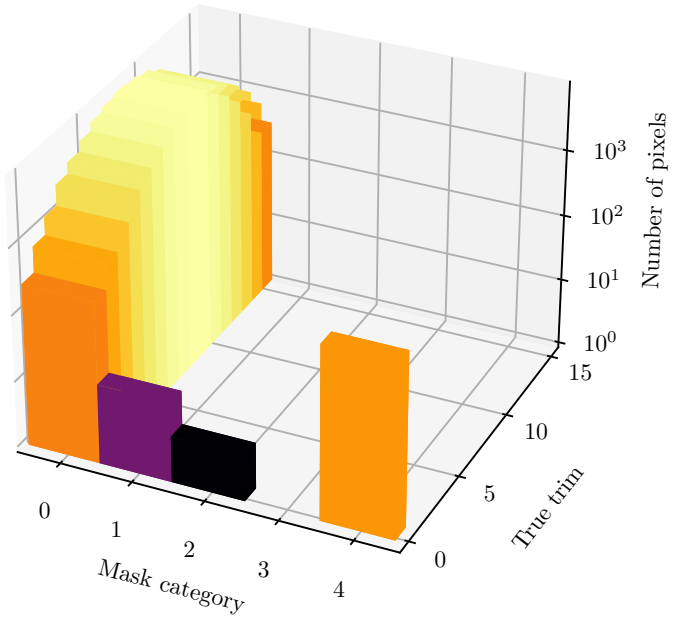

Figure 9: Histogram showing the relationship between the masking level (input) variable and the true trim (output) variable (Module 0 VP0-0 sensor).

It can be seen in Figures 5 and 6 that there is a negative correlation between the noise distribution means (at both trim 0 and F) and the corresponding true trim value. It is expected that lower trim values are assigned to noisier pixels, which this correlation confirms. Since the majority of pixels appear to follow this correlation at both minimum and maximum trim, it is likely that the neural networks will learn to use this to predict the trim value. In both figures, there are a number of pixels at various noise mean values that do not follow the same linear correlation as the majority, all of which have been assigned a trim value of 0. It is likely that these are the pixels found to be problematic, and that a significant number of them are already masked.

The noise distribution widths, however, do not appear to be correlated to the true trim variable in any way (Figures 7 and 8). At first, it may be assumed that this means the distribution width variable is not useful in predicting the trim, however neural networks are often able to extract patterns in data that are less obvious to the human eye. For example, the sum of the widths at trim 0 and F may correlate in some way to the trim, or perhaps a more complex combination. On the other hand, these variables may in fact be of no use to the neural network, in which case the training process will learn to ignore them. For this reason, it is sensible to allow them to remain as inputs regardless.

Figure 9 shows that the majority of pixels are in mask category 0 (unmasked). The remaining pixels in the 4 masked categories are all set at trim 0, but this is not important as the trim of masked pixels is irrelevant since their output is effectively ignored. Given this data, it is likely that the neural networks learn to assign any masked pixel a trim of 0, and ignore the mask variable entirely for unmasked pixels.

1.4 VELO simulation

The usefulness and implementation of neural networks for this purpose will depend on the level of accuracy that they are able achieve. If the neural networks were able to determine a perfect trim level for every (un-masked) pixel, leaving no problematic pixels, then they could be used to rapidly re-trim the entire detector as often as required. On the other hand, if neural networks were only able to correctly classify the *majority* of pixels, then they could, for example, be used to monitor sensor degradation over time and determine when a full, slower re-trim was necessary.

Clearly, if the trim values calculated using neural networks were to be used in practice, the performance of the vertex detector must not be significantly degraded as a result. Through simulation, it was observed that noisy (low trim) pixels are much more problematic in this regard than dead (high trim) pixels. Since even a simulated perfect detector (a detector with no noisy or dead pixels) cannot reconstruct 100% of simulated tracks, "significant" degradation was defined to be when the track reconstruction efficiency was more than 1% below that of the perfect model.

The simulations were performed using parts (Boole and DaVinci) of the full LHCb simulation software. These applications were used to work backwards and find the threshold fraction of noisy pixels that caused the track reconstruction efficiency to fall below the acceptable 1% level. Considering the worst-case scenario, where all problematic (low-trim) pixels are assumed to be noisy, this threshold gives an indication of how inaccurate the neural networks can afford to be, if their calculated trims were to be used in the real detector. Should the accuracy of the neural networks mean that they are unable to meet this threshold requirement, they may (as suggested earlier) instead find use as part of a sensor monitoring chain.

2 Theory

2.1 Silicon detectors

Silicon detectors are a type of semiconductor detector, which take advantage of their relatively small band gap by allowing the promotion of electrons into the conduction band (and creation of holes in the valence band) when an energetic particle is incident. An applied reverse-bias voltage separates the electrons and holes, carrying them to the detector electrodes, where the resulting current pulse is measured[3]. (Each pixel of a given VELO sensor is in fact a microscopic silicon detector, and the current pulse is then fed through an amplifier and into the readout chain.) The large number of pixels per sensor allow an accurate measurement of the position of the incident particle to be made. If a particle passes through several sensors, its trajectory can be deduced by a track reconstruction algorithm (similar to a line fitting algorithm).

2.2 Neural networks

2.2.1 Neural network structure

A well-designed and successfully trained neural network appears to behave like a black box, taking input data and rapidly returning an intelligent output. However, the inner-workings and structure must be understood in order to design a network with optimal results. A neural network consists of several layers: an input layer, any number of hidden layers, and an output layer. A layer contains a number of nodes, each of which is (for dense layers) connected to every node in the layer immediately above and below it. When a node receives a value from a connection to a node in the previous layer, it simply applies a mathematical function to it and passes the result on to the next layer. The number of nodes in the input and output layers correspond to the desired number of input and output variables, while hidden layers can have any number of nodes. The adjustable aspects of a neural network are referred to as hyperparameters. These include: the number of hidden layers, the number of nodes per hidden layer, and the choice of a node's mathematical function (known as its activation function), among others. The optimal hyperparameters for a specific application are usually unknown and must be determined through experimentation.

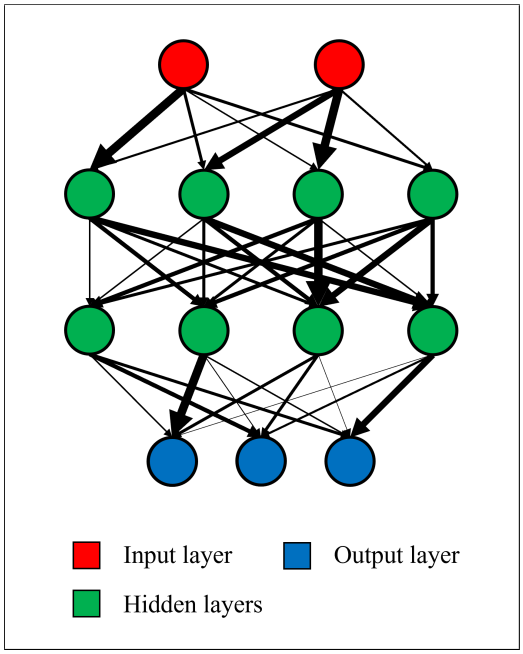

Figure 10: Structure diagram of a neural network with two input variables and three output variables (categories).

2.2.2 Neural network training

Once a neural network has been designed with a specific set of hyperparameters, it is trained with a dataset typical of the data it will encounter when in use. The training dataset consists of samples (sets of input variables, and their corresponding output variables). These samples are then fed into the neural network in batches, and after each batch, the network updates the strengths of connections between nodes (depicted as the arrow thickness in Figure 10), forming pathways as it learns how the output variables relate to the input variables. One pass through the whole training dataset is referred to as an epoch, and a neural network is typically trained for many epochs. The number of samples in a batch (referred to as the batchsize) is another hyperparameter, which has an optimal value in a given application. A small batchsize would require many batches to complete a full epoch, or in contrast, the entire dataset can be used in one large batch, once per epoch.

In an ideal training procedure, the accuracy of the neural network would improve after each epoch. In reality, this is sometimes not the case, as the accuracy can either stop increasing or even begin to decrease. A useful analogy is the concept of local minima in a mathematical function: the neural network can reach a dead end, where it has not reached its potential maximum accuracy, yet cannot progress. In such situations, early stopping conditions are useful. These are rules which dictate how many successive non-improving epochs the neural network is allowed complete before training is stopped.

2.2.3 Activation functions

Depending on the purpose of a neural network, different activation functions may be chosen for the network's hidden layers. The three most commonly used functions are: the rectified linear unit (ReLU), sigmoid (sometimes called 'logistic'), and the hyperbolic tangent (tanh)[2]. These are all single-variable functions that return an output based on their input.

$$relu(x) = \begin{cases} 0, & \text{x} < 0 \\ \text{x}, & \text{x} \geq 0 \end{cases} \tag{1}$$

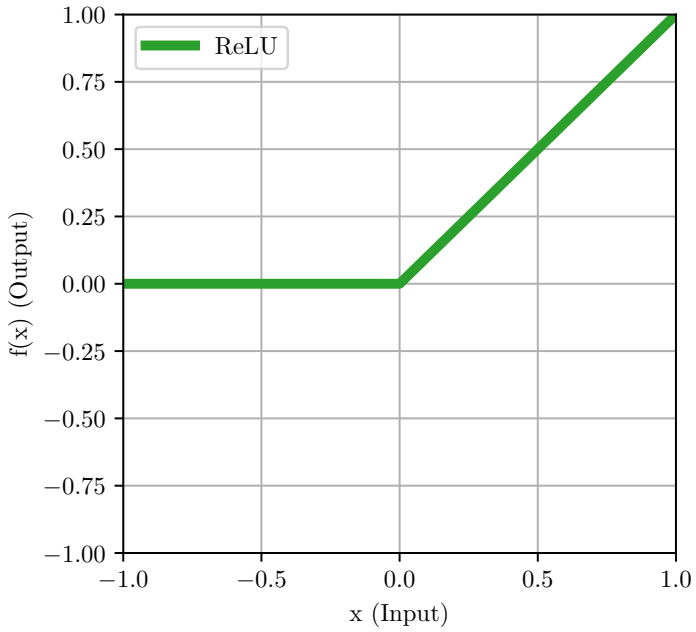

Figure 11: The rectified linear unit function, used as a node activation function.

The ReLU function is generally regarded as the most effective activation function for the majority of neural network applications. This function is non-linear, returning 0 for any negative input which allows certain pathways to be "switched off" in the event that previous nodes return a negative value. When given a negative input, sigmoid or tanh would return a negative value, so do not offer this functionality. Since this function always returns either 0 or x (its input) it is also very fast to compute, which can improve training times.

$$sigmoid\,(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

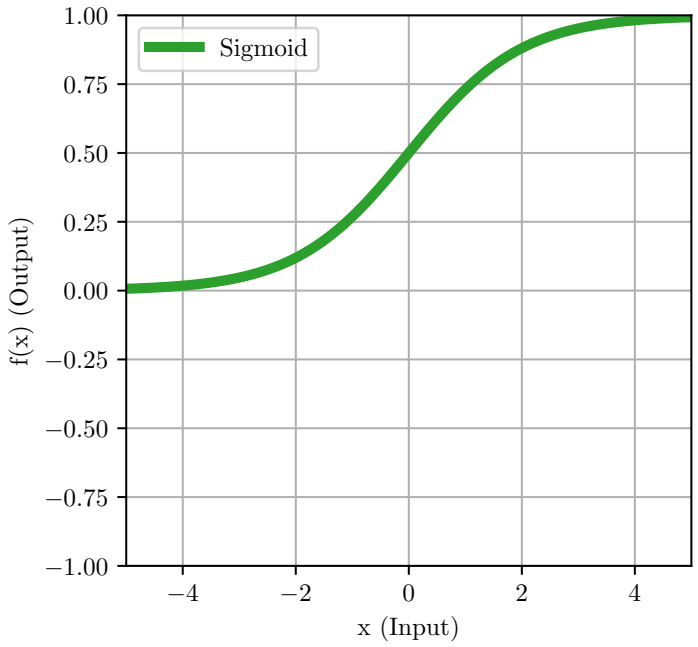

Figure 12: The sigmoid function, used as a node activation function.

In the past, the use of sigmoid activation functions was common practice, but they are now less favourable due to the general preference for and speed of the ReLU function.

$$
\begin{aligned}
tanh\,(x) &= \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \\
&= 2 \cdot sigmoid\,(2x) - 1
\end{aligned}
\tag{3}
$$

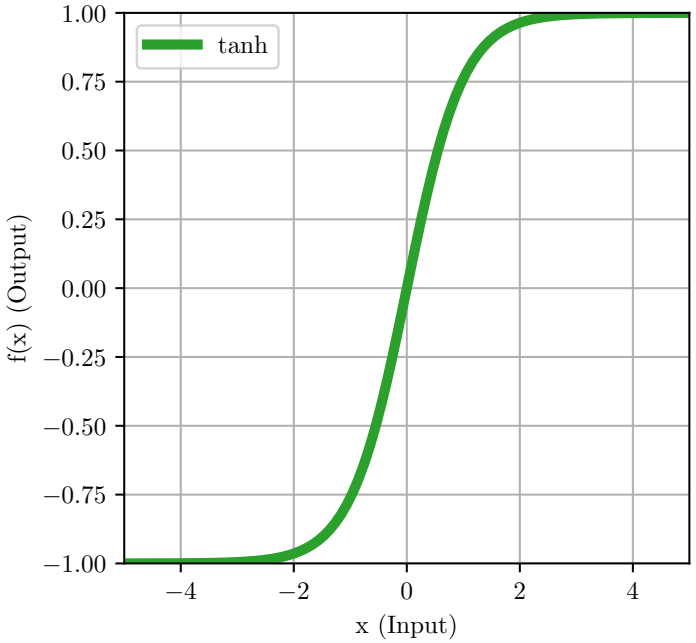

Figure 13: The tanh function, used as a node activation function.

The tanh function has a similar shape to sigmoid, and is in fact a slight modification of the same formula. The two main differences are: the output range (0 to 1 for sigmoid, -1 to 1 for tanh) and the gradient (the gradient of tanh is four times that of sigmoid, causing more extreme weight updates during training). Despite the widespread preference for ReLU, it is important to try several activation functions for a given application, since certain situations favour less-common network structures.

3 Methodology

3.1 Neural network investigation

This section covers the experimentation into using neural networks with VELO data, and the evolution of the final optimised neural network.

3.1.1 Implementation

The neural networks used in this investigation were implemented via the Keras framework, which provides a Python interface to the Tensorflow machine learning library. The input layer (as mentioned earlier, remaining unchanged across all networks) consisted of 5 nodes, configured to receive and normalise the 5 input variables. The output layer was also constant throughout: 16 nodes using the softmax activation function, corresponding to the 16 possible trim values. The softmax function[5] scales the outputs to ensure that they sum to 1, allowing them to be interpreted as a probability distribution. A confident trim categorization by the neural network would result in one output node returning a value of almost 1, and the rest near 0.

Initially, neural networks were trained for each of the 12 sensors belonging to module 0, with the assumption that they should provide a sufficient representation of the variation between all sensors as a result of the manufacturing process. For each 256x256 sensor (65536 pixels), the input data was arranged into samples (5-element arrays, containing the values of the 5 input variables corresponding to a single pixel). Half of the sample pixels (32768) were used as the training data, and the other half for post-training evaluation. Evaluating the neural network with separate data ensures that it has learned to categorize samples generally, rather than just those it was trained on.

3.1.2 Hyperparameter optimisation

Tuning the hyperparameters constituted the majority of the neural network investigation, requiring many different networks to be trained with slight variations between each iteration. The effects of varying four different hyperparameters were investigated: the number of hidden layers, the number of nodes per hidden layer, the activation function of hidden layer nodes, and the training batch size. The different configurations tested are summarised as follows:

Hyperparameter	Values tested
Number of (5-node) hidden layers	1, 2, 3, 4, 5
Number of nodes (in a single hidden layer)	5, 50, 100, 250, 500
Node activation function (in a single 5-node hidden layer)	ReLU, sigmoid, tanh
Training batchsize (for a single 5-node hidden layer)	256, 512, 1024, 2048, 4096, 8192, 16384, 32768

Table 1: Different configurations tested in the hyperparameter optimization process for the pixel-trim classification neural network.

In order to isolate the effects of changing one hyperparameter at a time, the others were held constant while each was being investigated. 3 isolated repeats with random initial conditions were performed for each configuration, to gauge the stability of the final accuracy. The training duration was also monitored throughout, and any time comparisons made here are between neural networks trained at a similar time, on the same computer (Intel i5-7200U CPU, no GPU used by Keras) to ensure their validity.

3.1.3 Widening the accuracy metric

Previously, the accuracy metric for a trained neural network had been defined as the fraction of pixels which were assigned exactly the same trim as their true value. However, it may be the case that there are several trim levels that lie in the appropriate range, any of which would allow sufficient discrimination of tracks from noise. To investigate this, two new accuracy metrics were proposed: the fraction of pixels that are assigned their true trim ± 1 level, and the fraction of pixels that are assigned their true trim ± 2 levels. Although it is unknown how wide the acceptable trim range is in reality (this range may also vary between pixels), the new accuracy metrics aimed to demonstrate *how* inaccurate the inaccurate trim classifications were.

3.2 Vertex detector simulation

This section details how parts of the LHCb detector simulation were used to investigate the effect of realistic sensor imperfections on the track reconstruction efficiency. This was then compared to the imperfect pixel fractions identified by the neural networks.

3.2.1 Simulation overview

A file containing pre-generated event data originating from the earlier stages of the LCHb simulation was input to the Boole application, which simulated the detector response based on the chosen sensor conditions (noisy/dead pixel fractions). The number of events used from the input file was also configurable.

The output of Boole mimics data from the real detector, allowing it to be input to the DaVinci application, which attempts to reconstruct particle tracks from hits recorded in the detector. Since the entire process uses simulated, rather than real data, the true number of tracks is known. The detector efficiency can therefore be calculated as the fraction of real tracks that DaVinci was able to reconstruct.

3.2.2 Exploratory investigation

Initial simulations were carried out to gain an appreciation of the respective effects of varying the noisy and dead pixel fractions. These simulations were performed using the data from 10 events, equating to ~ 850 tracks. The first simulation performed modelled a theoretical "perfect" detector ("perfect" refers to a detector with no noisy or dead pixels in any of its sensors). The fraction of tracks reconstructed was subsequently considered to be the maximum possible ("perfect") efficiency.

The neural network trim results of the 12 sensors were then used to obtain noisy and dead pixel fractions as test data. Initially, it was planned that pixels classified as trim 0, 1 or 2 would be considered noisy, and D, E or F (plus all masked pixels) as dead. While this was suitable for the dead pixels, the noisy pixel fraction proved to be too large and caused the DaVinci application to crash. For this reason, and after trialling successively smaller noisy fractions, it was decided that only un-masked pixels which were categorized as trim 0 would be considered noisy. (The condition that a noisy pixel be un-masked was due to the fact that masked pixels have already been dealt with, so to speak, and are therefore no longer noisy.)

Two sets of initial simulations were performed using the above definitions for the noisy and dead pixel fractions: simulating only noisy pixels, and only dead pixels. Testing these conditions separately allowed the effects of noisy and dead pixels to be observed independently, since the final reconstructed track fraction gives no indication of the extent to which it was affected by each. The results of these simulations were plotted to visually demonstrate the effect of noisy and dead pixels on tracks reconstruction efficiency.

3.2.3 Noisy pixel limit investigation

Arguably the most crucial outcome of the VELO simulations was the determination of the threshold fraction of noisy pixels, beyond which the detector efficiency is significantly reduced. A significant reduction in efficiency was defined as more than 1% worse than a perfect detector, and so was calculated using the previous "perfect" detector simulation.

The purpose of the earlier initial simulations was to roughly indicate the rate at which detector efficiency decreased with increasing noisy pixels. Here, however, the aim was to map out the shape of this relationship as it crosses the 1% significance level. With knowledge of the maximum possible efficiency and the general rate at which efficiency falls with noise, an appropriate range of noisy fraction values was decided: 0.0 to 0.0001, in steps of 0.00001. A script was written to perform these simulations and collect the efficiency results such that they could be plotted to resolve the shape of the curve.

This result would show clearly the maximum fraction of pixels that could be noisy before the detector efficiency was negatively impacted to a significant degree. This threshold is ultimately the key factor in assessing the suitability of neural networks in this application.

3.2.4 Masking noisy pixels

The final simulations aimed to model the case where every pixel identified as noisy by the neural networks has been masked. In reality, it is likely that some noisy pixels may not be identified by the neural networks, however the detector efficiency can still be raised significantly by masking those that are.

The simulations were performed in the same way as in the previous sections, but with altered noisy and dead pixel fractions. Since all noisy pixels are to have been masked, the previous noisy fraction was added to the simulated dead fraction, and the simulated noisy fraction set to 0.

4 Results

4.1 Neural network investigation

4.1.1 Estimating the error on accuracy

During the hyperparameter investigation, each neural network configuration was tested with 3 repeats (3 networks were trained with the same hyperparameters, but randomised initial conditions). For each configuration, the mean of the 3 final accuracy values achieved by the separate networks was calculated. This gave a value for the expected accuracy (a) of the current neural network configuration, and the uncertainty on this value was calculated using the standard RMS formula with Bessel's correction (due to the small ($n = 3$) number of repeats):

$$\Delta a = \sqrt{\frac{n}{n-1} \times (\langle a^2 \rangle - \langle a \rangle^2)} \tag{4}$$

This was implemented via the stats.sem() function from the SciPy Python library[8], and used to calculate the error on the accuracy for hyperparameter configurations tested below.

4.1.2 Hyperparameter optimisation

Below are the results of the results of the hyperparameter investigation, demonstrating the effects of varying the number of nodes, number of layers, activation function and batchsize. While investigating each hyperparameter, the values of the others were kept constant to avoid interference between their effects. As such, the effects of changing one hyperparameter cannot necessarily be compared to those of another, as their constant conditions may differ (although this was avoided where possible). In other words, the results of the values tested should only be judged relative to other values of the same hyperparameter. (Note that throughout this section (4.1.2), "accuracy" refers to the original *un-widened* accuracy metric, where a trim classification is only considered accurate if it is equal to the true trim level.)

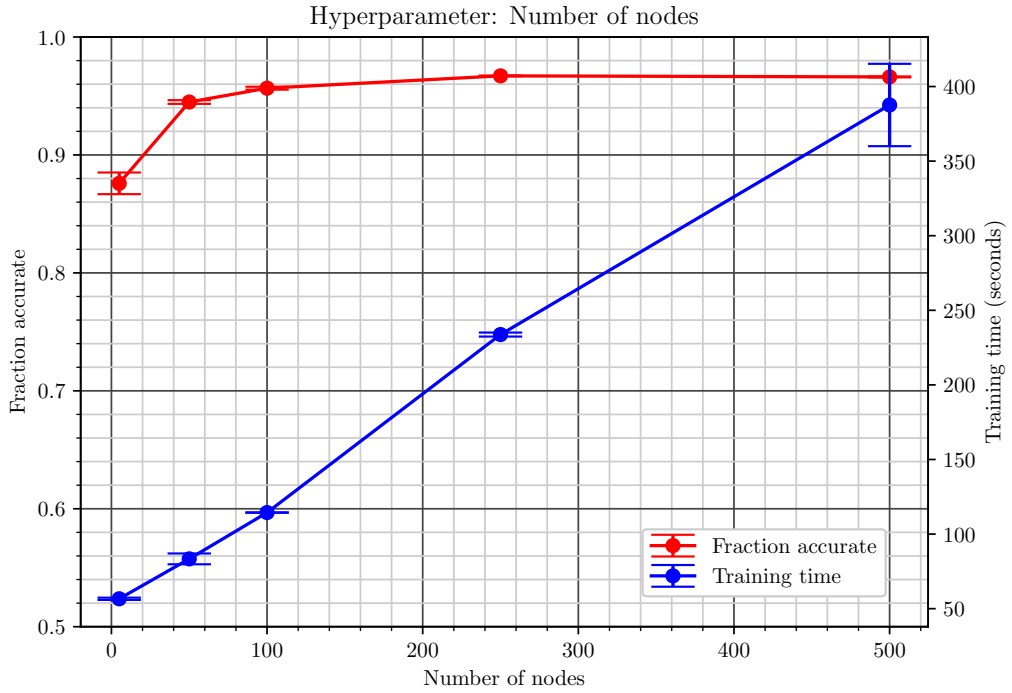

Figure 14: Accuracy and training time of pixel trim classification neural networks as a result of varying the number of (ReLU) nodes in a single hidden layer.

The above plot shows that generally, adding more nodes to a single-layer neural network improves the accuracy fraction up to a maximum of around 97%. Beyond about 250 nodes, continuing to add more no longer improves the accuracy, but still increases the training time in a linear fashion. Although a longer training time is irrelevant once training is complete (it does not affect the operation speed of the trained network), there is no benefit to increasing the number of nodes beyond this point.

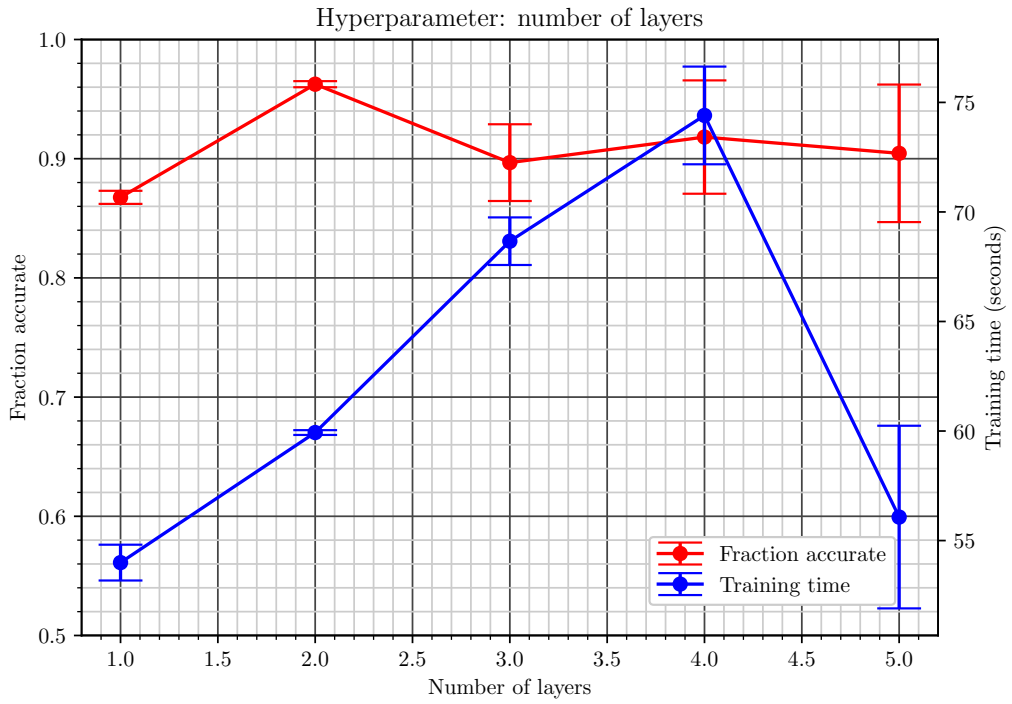

Figure 15: Accuracy and training time of pixel trim classification neural networks as a result of varying the number of 5-node hidden (ReLU) layers.

The effect of varying the number of hidden layers is less straightforward. It can be seen, for example, that two layers is better than one, but beyond this the mean accuracy appears to fall. However, the accuracy uncertainty tends to rise with the number of layers, which is likely to be due to increasing sensitivity to the random initial conditions of the neural network. At 5 layers for example, some repeats reached the maximum accuracy ($\sim 97\%$), while others got stuck at a lower fraction and were subject to the early stopping condition. This is reflected in the mean training duration, where a sudden drop can be seen when 5 layers are implemented.

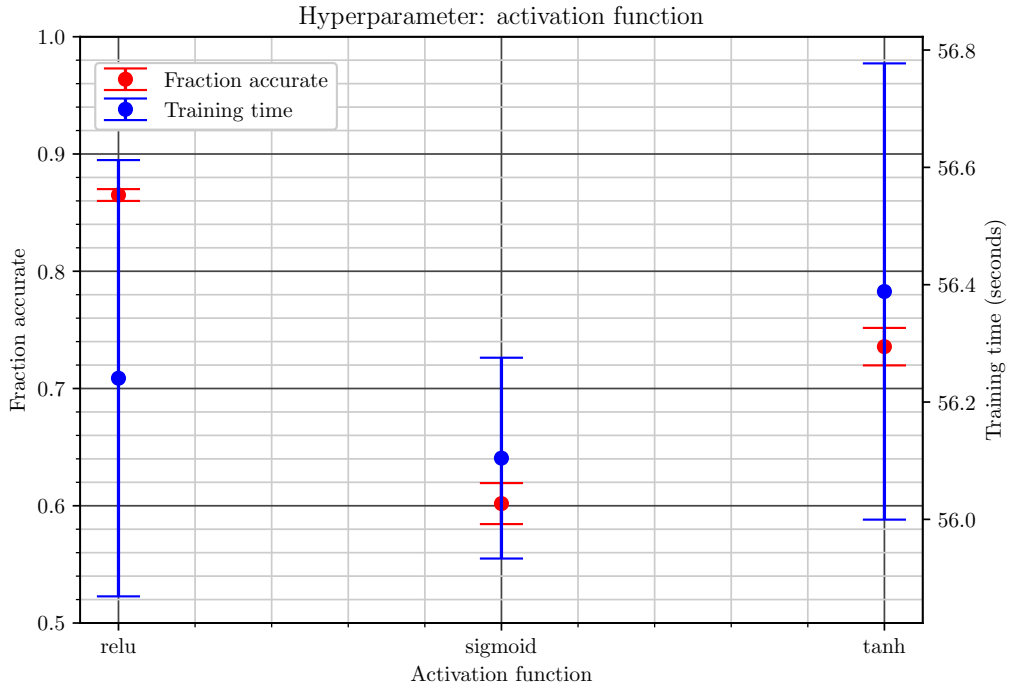

Figure 16: Accuracy and training time of pixel trim classification neural networks as a result of varying the activation function of a single 5-node hidden layer.

As previously mentioned, the rectified linear unit function (ReLU) is generally regarded as the most effective activation function for most neural network applications. This is reflected in the in above results, where the accuracy fraction achieved was significantly higher and more precise when using the ReLU function over sigmoid or tanh. The mean training time was fastest for the sigmoid function, but its low accuracy fraction immediately discredits it as a viable option.

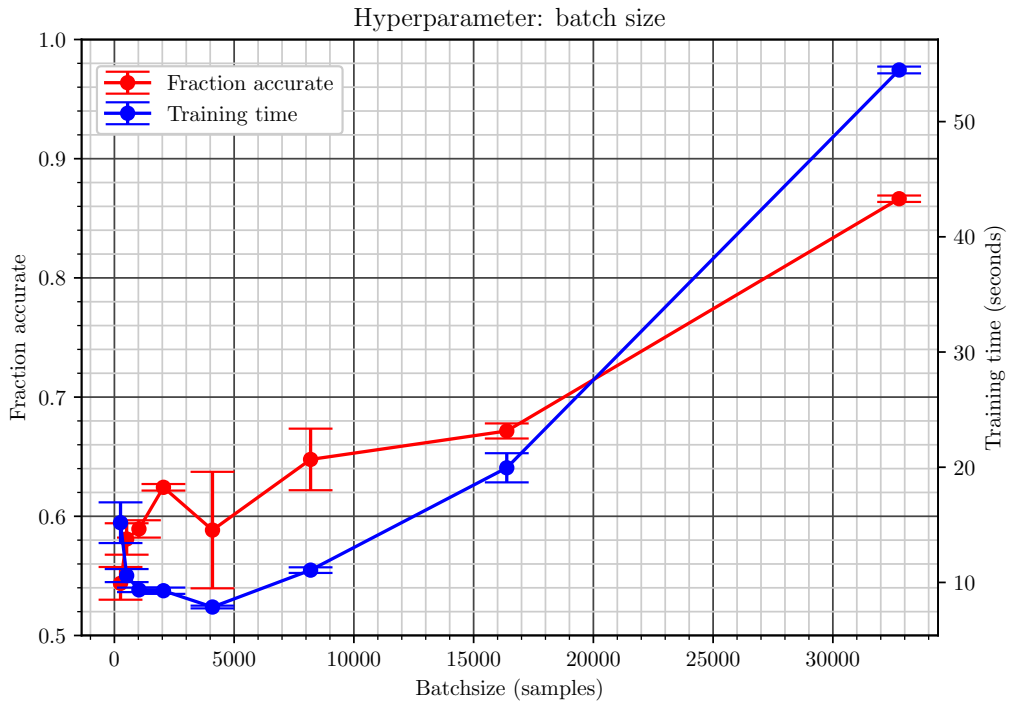

Figure 17: Accuracy and training time of pixel trim classification neural networks as a result of varying the training batchsize (using a single 5-node ReLU layer).

Figure 17 shows that in general, a larger batch size results in a greater final accuracy fraction. This is because the neural network is given more data to learn from per epoch. Interpreting the training duration of this test is more complicated the previous cases, since the early stopping condition was met for many of these networks. Smaller batchsizes resulted in much slower progress, to the point that training was terminated before a satisfactory accuracy was reached. Larger batchsizes allowed more progress to be made per epoch, which allowed training to continue and a high accuracy to be reached. In several cases, training with a large batchsize was too demanding for the computer used, causing Python to crash (on a more capable system, this issue would be less prevalent).

Considering these results, the following "best" combination of hyperparameter values was chosen to train neural networks for the 12 sensors. This structure reliably produced neural networks capable of accurately classifying $\sim 97\%$ of pixels as their "true" trim value (the maximum percentage observed throughout).

Hyperparameter	Value
Number of nodes per layer	250
Number of (hidden) layers	1
Node activation function	Rectified linear unit (ReLU)
Training batchsize	32768 pixel samples (entire training dataset)

Table 2: Optimum combination of hyperparameter values found for training pixel trim classification neural networks.

4.1.3 Widening the accuracy metric

The original accuracy metric resulted in an observed maximum accuracy of $\sim 97\%$, across all neural networks. When widened to allow trim classifications that were ± 1 from the true value, the accuracy of the same networks increased to $\geq 99.99\%$, for all 12 sensors (see Table 3 below). This shows that when a neural network prediction is incorrect, it is only incorrect by 1 trim level in virtually all cases. This result renders any further widening of the accuracy metric unnecessary, as accuracy values can never surpass 100%.

Sensor	Accuracy (original metric: true trim ± 0)	Accuracy (widened metric: true trim ± 1)
VP0-0	$96.92 \pm 0.04\%$	$100.00 \pm 0.00\%$
VP0-1	$97.77 \pm 0.01\%$	$100.00 \pm 0.00\%$
VP0-2	$97.61 \pm 0.01\%$	$100.00 \pm 0.00\%$
VP1-0	$97.83 \pm 0.01\%$	$100.00 \pm 0.00\%$
VP1-1	$97.61 \pm 0.01\%$	$100.00 \pm 0.00\%$
VP1-2	$97.95 \pm 0.02\%$	$100.00 \pm 0.00\%$
VP2-0	$97.51 \pm 0.00\%$	$100.00 \pm 0.00\%$
VP2-1	$97.69 \pm 0.01\%$	$100.00 \pm 0.00\%$
VP2-2	$97.26 \pm 0.00\%$	$100.00 \pm 0.00\%$
VP3-0	$97.01 \pm 0.01\%$	$100.00 \pm 0.00\%$
VP3-1	$97.18 \pm 0.02\%$	$100.00 \pm 0.00\%$
VP3-2	$97.27 \pm 0.01\%$	$99.99 \pm 0.00\%$

Table 3: Trained neural network accuracy values according to the original and widened accuracy metrics.

4.2 Vertex detector simulation

4.2.1 Estimating the error on efficiency

The purpose of VELO simulations performed in this investigation was to determine the track reconstruction efficiency under different conditions. In each simulation, there was a defined number of simulated tracks, n_{total}, each of which was either reconstructed or not reconstructed. Due to the binary nature of each outcome, binomial statistics can be used to estimate the error on the number of reconstructed tracks, and therefore the efficiency of the simulation. A binomial distribution has parameters n and p, which are the number of trials and probability of success, respectively. Also defined is q, the probability of failure, which is equal to $1 - p$. In this situation, n is the number of simulated tracks, p is the measured efficiency (the successfully reconstructed fraction), and q the fraction lost. Since the variance of a binomial distribution is defined as npq, the error (standard deviation) can be estimated as:

$$\Delta n_{reconstructed} = S.D. = \sqrt{npq} \tag{5}$$

Dividing this by the total number of tracks gives an estimation of the error on the efficiency:

$$\Delta(\textit{efficiency}) = \frac{\Delta n_{reconstructed}}{n_{total}} \tag{6}$$

This method was used to estimate the efficiency error for all of the following VELO simulation results.

4.2.2 Exploratory investigation

The initial "perfect" detector simulation (with no noisy or dead pixels) was able to reconstruct 825 of the 850 simulated tracks, which equates to an efficiency of $97.1 \pm 0.6\%$, hereafter considered to be the maximum possible detector efficiency. This sets the 1% acceptability threshold at 96.1%.

Three neural network repeats had been trained for each of the 12 sensors investigated, to provide an estimate of the mean and variation of the results (caused by the random nature of the initial conditions in a neural network). Once trained, the worst-case scenario noisy and dead pixel fractions of each sensor were calculated. These can be seen in the following table:

Sensor	**Noisy pixel fraction** ($\times 1000$)	**Dead pixel fraction** ($\times 1000$)
VP0-0	3.82 ± 0.02	55.58 ± 0.05
VP0-1	1.61 ± 0.03	18.83 ± 0.02
VP0-2	2.49 ± 0.04	23.73 ± 0.03
VP1-0	2.39 ± 0.10	13.70 ± 0.02
VP1-1	3.56 ± 0.02	29.08 ± 0.09
VP1-2	2.80 ± 0.02	18.27 ± 0.02
VP2-0	3.45 ± 0.05	34.31 ± 0.03
VP2-1	3.52 ± 0.02	30.71 ± 0.02
VP2-2	8.64 ± 0.01	60.51 ± 0.08
VP3-0	3.69 ± 0.01	52.82 ± 0.06
VP3-1	3.63 ± 0.02	46.79 ± 0.02
VP3-2	3.20 ± 0.01	33.37 ± 0.02

Table 4: Noisy (trim category 0, minus masked pixels) and dead (trim categories D, E, F plus masked pixels) pixel fractions of the 12 module 0 sensors, according to individually trained classification neural networks.

The entire VELO detector was simulated using each of the above fractions in turn, to observe the resulting track reconstruction efficiency values (assuming here that the noisy/dead fractions were constant across all sensors). In reality, these fractions vary between sensors, however this assumption served the purpose of providing a first insight into how the detector efficiency degrades as a result of noisy/dead pixels:

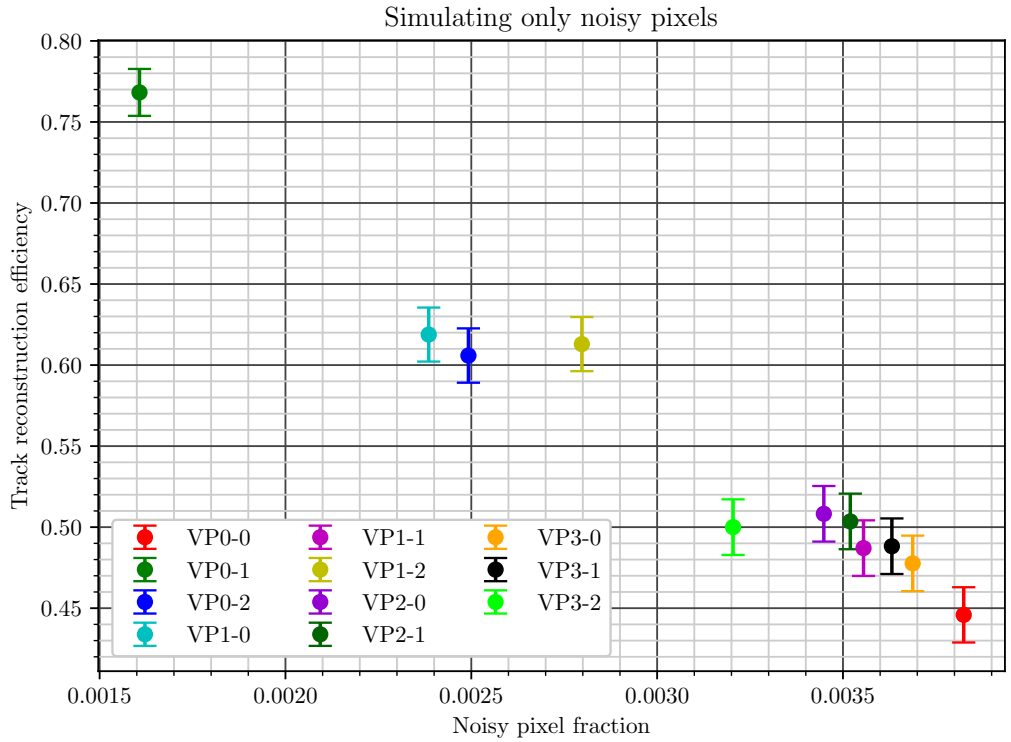

Figure 18: Track reconstruction efficiency of VELO simulations using the noisy pixel fractions of 12 sensors calculated by dedicated neural networks.

It is immediately clear that across all 12 sensors, the number of noisy pixels results in a catastrophically low fraction of reconstructed tracks: just under 76% at best and around 45% at worst. This is far below the acceptability threshold of 96.1%, so if left unmasked, these pixels would severely impact the efficiency of the detector. (Note that the VP2-2 sensor has been omitted from the above plot due its abnormally high noisy pixel fraction causing the simulation to crash, yielding no result.)

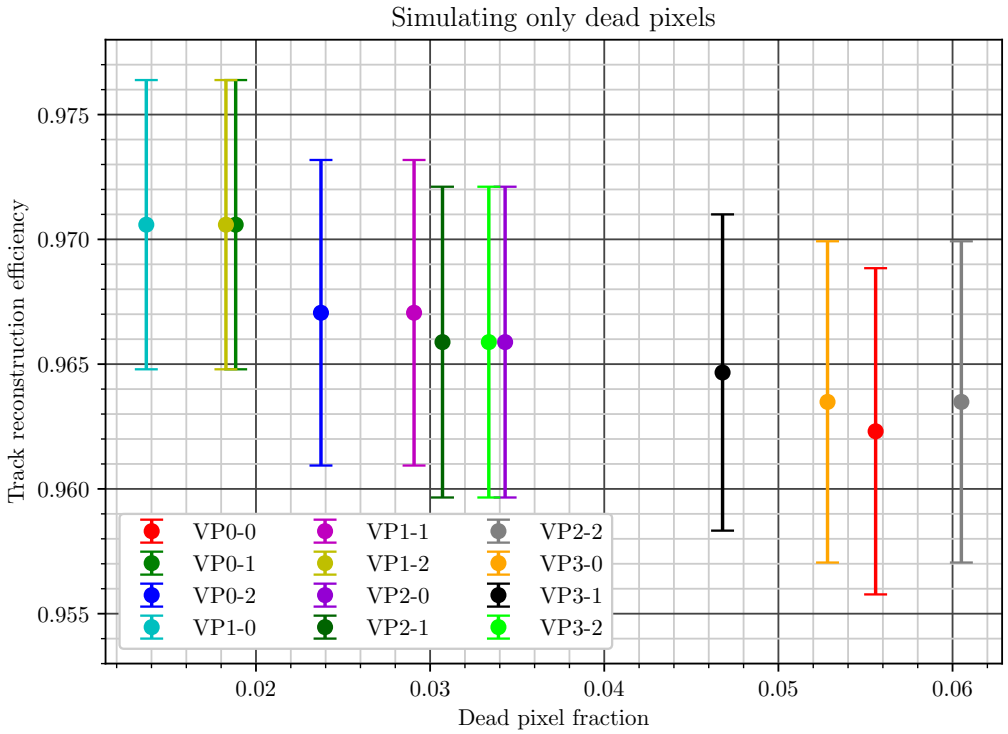

Figure 19: Track reconstruction efficiency of VELO simulations using the dead pixel fractions of 12 sensors calculated by dedicated neural networks.

It can be seen in Figure 19 that dead pixels have a much smaller impact than noisy pixels on the fraction of reconstructed tracks. (Note that the errors here are not larger than those of Figure 18. They are in fact ∼10x smaller, but the efficiency axis here covers a much smaller range.) For the sensors with the greatest number of pixels classified as dead (VP2-2, VP0-0, and VP3-0), the number of reconstructed tracks still remained above the 96.1% threshold. This result reinforces the importance of identifying and masking noisy pixels, as it can effectively transform a noisy pixel into a dead pixel.

4.2.3 Noisy pixel limit investigation

In section 4.2.1, (Figure 18) it was observed that the relationship between the number of noisy pixels and the fraction of tracks reconstructed is approximately linear, at least in the investigated range. The plot below shows that near the 96.1% acceptability level, the relationship no longer appears to be a simple proportionality, it is more curved:

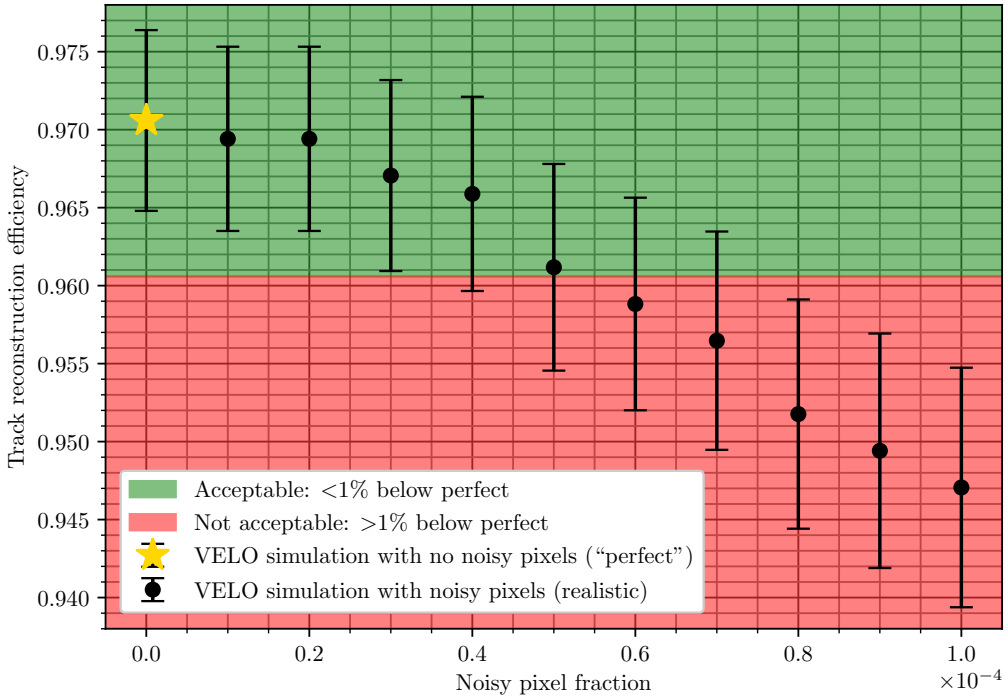

Figure 20: Close up plot of the noisy pixel fraction – track reconstruction efficiency curve as it crosses the 1% acceptability threshold.

The curve appearing to flatten as it approaches the perfect detector simulation validates the assumption that this is the greatest possible fraction of tracks able to be reconstructed.

4.2.4 Masking noisy pixels

The following plot shows that when every pixel identified as noisy is masked, the track reconstruction efficiency is pushed back up into the acceptable range. Through simulation, this was observed using the noisy and dead fractions of all 12 module 0 sensors:

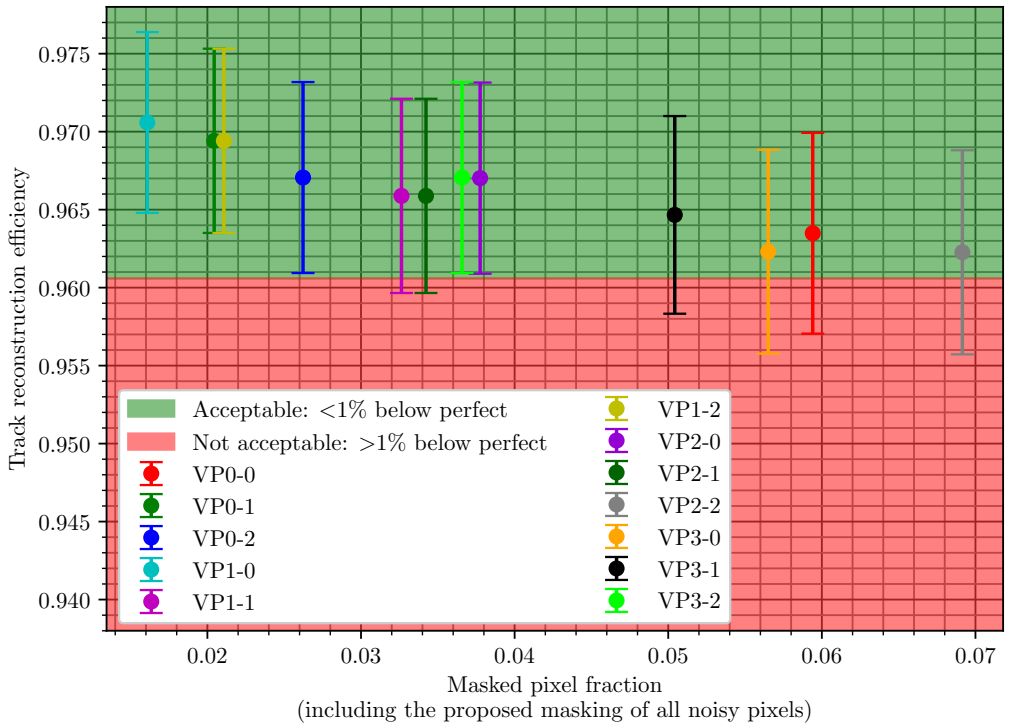

Figure 21: VELO simulation track reconstruction efficiency, in the case that all sensor pixels identified as noisy have been masked.

In reality, these 12 sensors will not represent the full range of manufactured sensor quality, and it is very unlikely that every noisy pixel could be identified and masked. This result serves as more of an indication of the benefit of masking as many noisy pixels as possible.

5 Discussion

5.1 Neural network investigation

The hyperparameter optimisation process showed that selecting an appropriate combination of hyperparameters is essential in order to train an accurate neural network. It also showed that adjusting some hyperparameters has a much greater effect on the final accuracy, while others may have other effects such as reducing the required training duration. Some hyperparameters appear to have optimal values (at least in this application), but this is not always the case. Varying the number of hidden layers (Figure 15), for example, does not appear to improve/worsen the final accuracy in any straightforward manner. The uncertainties involved would need to be reduced (possibly through the collection of more data), before any definite conclusions can be made for this hyperparameter.

The results obtained have shown that it is possible to train neural networks capable of instantly predicting an appropriate trim for a VELO sensor pixel, given some information about its noise rate and mask level. $\sim 97\%$ of the time, these neural networks predict the same trim level as determined by a more complete manual analysis. This immediately demonstrates the benefit of implementing neural networks in the trim calculation process, as reducing the time spent configuring the detector reduces the overall duration of an experiment, and therefore the cost. However, 97% is not perfect, and this may lead to the assumption that neural networks provide a faster but less reliable technique for configuring the detector. While somewhat true, the 97% accuracy value assumes that the neural networks must predict the exact trim level calculated by the manual process (which is not necessarily required). It was shown if the trim can be incorrect by ± 1 level with no consequence, the neural networks are capable of predicting an appropriate trim for all pixels.

The practicality of this of course depends on whether or not a pixels trim level being different by 1 level is problematic. Further investigation would be required to determine this, involving the analysis of the analogue pixel signal as a track passes through it. Regardless of the outcome of such an investigation, it does not apply to the $\sim 97\%$ majority of pixels, but only to the $\sim 3\%$ which were not predicted originally with perfect accuracy. With this in mind, and considering trim classification accuracy alone, it is indeed likely that neural networks could be utilised to calculate trims which are then used in the normal operation of the detector. In the case that an unconsidered factor disproves this, neural networks could instead be used to monitor the degradation of sensors, as the trim predictions will begin to change as a result of radiation damage.

5.2 Vertex detector simulation

Simulating the VELO detector has demonstrated that the fraction of noisy pixels found in a typical sensor is high enough to significantly impair the track reconstruction efficiency. It was observed, however, that dead pixels have a much smaller impact on the efficiency.

The obvious solution is to suppress noisy pixels via masking (effectively making them dead), since the significant efficiency gain from reducing the number of noisy pixels greatly outweighs the relatively small problem of creating more dead pixels. In order to mask *all* noisy pixels, they must first be identified. It has been shown that neural networks are capable of correctly categorising virtually any pixel to within the range of ± 1 level, suggesting that this is indeed possible. Due to the large number of pixels in each sensor, it is likely that at least some unproblematic pixels will be misidentified as noisy. These pixels would be masked unnecessarily, but it has also been shown that a small number of extra dead pixels would not significantly affect the detector efficiency.

The neural network predictions gave separate noisy and dead pixel fractions for each sensor. However, the VELO simulation software only allowed the noisy and dead pixel fractions of the *entire* detector to be set. For this reason, the simulations corresponding to real sensors (Figures 18, 19, 21) were performed with the assumption that these fractions were averages of the whole detector, rather than describing a specific sensor. The same analysis should be carried out using more test data (representative of the entire detector, rather than just one module), the results of which could be considered more conclusive. Nevertheless, the conclusions reached in this investigation remain valid, as the 12 module 0 sensors do in fact show significant variation in their levels of imperfection.

6 Conclusion

This project investigated the use of neural networks to predict appropriate trim levels for the sensor pixels of the newly-manufactured upgrade to the LHCb VELO detector. The reason for this investigation was that neural networks offer a potential reduction in the total time spent determining trims, as they do not require a full scan of all trim levels in order to make a prediction. The current process for determining pixel trims is slow, consuming valuable experiment time which could be better spent collecting data (or shortened to reduce total costs), so any improvement through the use of neural networks is useful. It was found that neural networks are very effective in this regard, accurately predicting trim levels using only 1/8 of the data required by the current process (as only 2 of the 16 trim levels need to be set and tested). Since testing different trim levels for every pixel is a slow process, reducing number required by a factor of 8 is a significant improvement.

In addition to the smaller drawbacks detailed in Section 5, one of the greatest assumptions made throughout the project was that *only* unmasked, trim-0 pixels were considered noisy. Naturally, widening the range of pixels considered to be noisy results in a greater noisy fraction, which beyond a certain level caused the Boole simulation to crash. For this reason, the assumption made was essentially forced by an unavoidable factor, rather than proper grounds or justification. This problem could be solved in two ways, the easier of the two involving waiting for a new version / patch of the Boole simulation, allowing more simulations to be performed at higher noisy pixel fractions and therefore a more complete investigation. A more difficult method would require a different branch of investigation to be carried out, to determine *how* noisy a pixel can be before it begins to cause problems in track reconstruction (essentially determine whether the forced assumption was valid). This is a way in which this project could be extended, but it would require additional data (different to what is currently used) and details of the track reconstruction process.

Reflecting on what I have learned during the course of this project, the uses, theory and implementation of neural networks are the most noteworthy. Before beginning the investigation, I was confident in programming and manipulating data (which helped with the more technical aspects of the project), but had no experience whatsoever in machine learning. This project is a perfect example of its use in solving a problem / optimising an existing process, making it educational while also serving a genuine purpose. With the understanding I have gained as a result, I would feel more confident in applying machine learning techniques to other tasks / data types within or beyond the field of physics. Additionally, having the opportunity to work with real data collected from the newly-built VELO upgrade was enjoyable as it gave the work a connection to current research projects, as opposed to being purely simulated / theoretical.

Bibliography

[1] The LHCb Collaboration. LHCb VELO Upgrade Technical Design Report. Technical Report CERN/LHCC 2013-021, CERN, Geneva, Switzerland, November 2013.

[2] Jason Brownlee. How to Choose an Activation Function for Deep Learning. https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/, 2021. Accessed: 2022-02-04.

[3] Jonathan Alaria. Lecture: Photodiodes (1): Detectors. Univeristy of Liverpool: PHYS389 Semiconductor Applications, 2021. Accessed: 2022-04-20.

[4] keras.io. Simple MNIST convnet. https://keras.io/examples/vision/mnist_convnet/, 2020. Accessed: 2022-02-01.

[5] keras.io. Keras API reference. https://keras.io/api/, 2022. Accessed: 2022-02-07.

[6] numpy.org. NumPy reference. https://numpy.org/doc/stable/reference/index.html, 2022. Accessed: 2022-02-09.

[7] Peter Svihra. The LHCb VELO Upgrade I. https://cds.cern.ch/record/2725639/files/lhcb_velo_upgrade.pdf, 2020. Accessed: 2022-04-17.

[8] scipy.org. Statistical functions. https://docs.scipy.org/doc/scipy/reference/stats.html, 2022. Accessed: 2022-05-03.

[9] Stefano de Capua. The LHCb VELO Upgrade. https://cds.cern.ch/record/2630580/files/decapua_VELOupgrade%2007.07.pdf, 2018. Accessed: 2022-02-01.

Appendices

Appendix A: Additional Plots

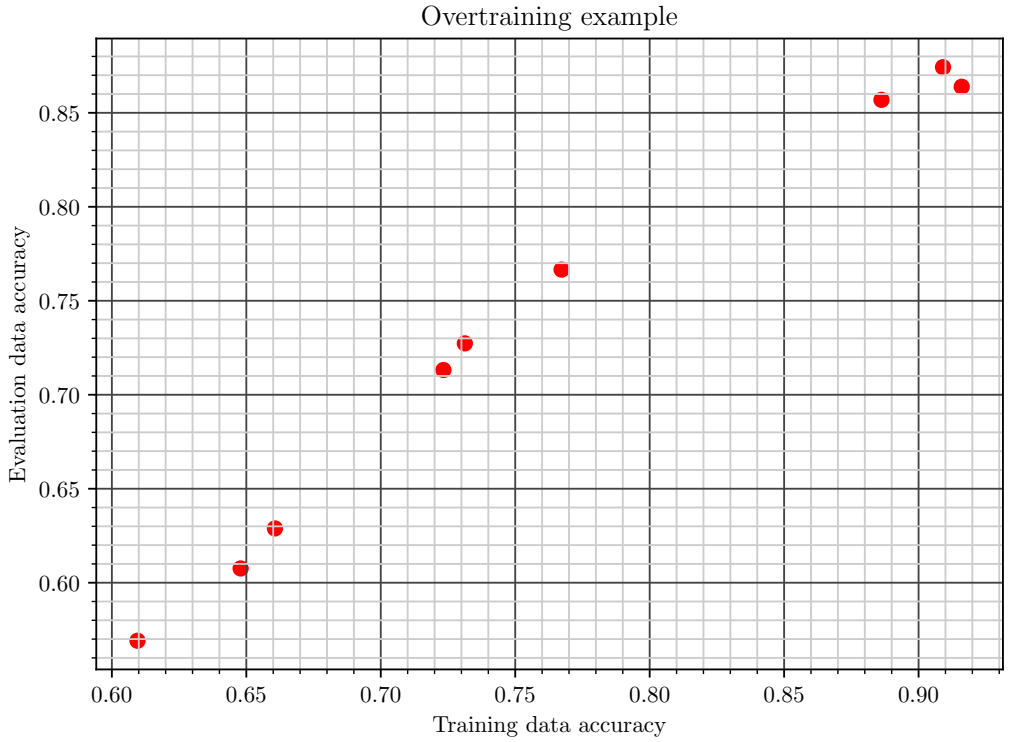

Figure A1: Example of 'overtraining' (when a neural network is trained for too long, and begins to learn specific details in the training data which are not present generally). It can be seen that beyond a certain point, the evaluated accuracy begins to fall below its previous proportionality with the training accuracy.

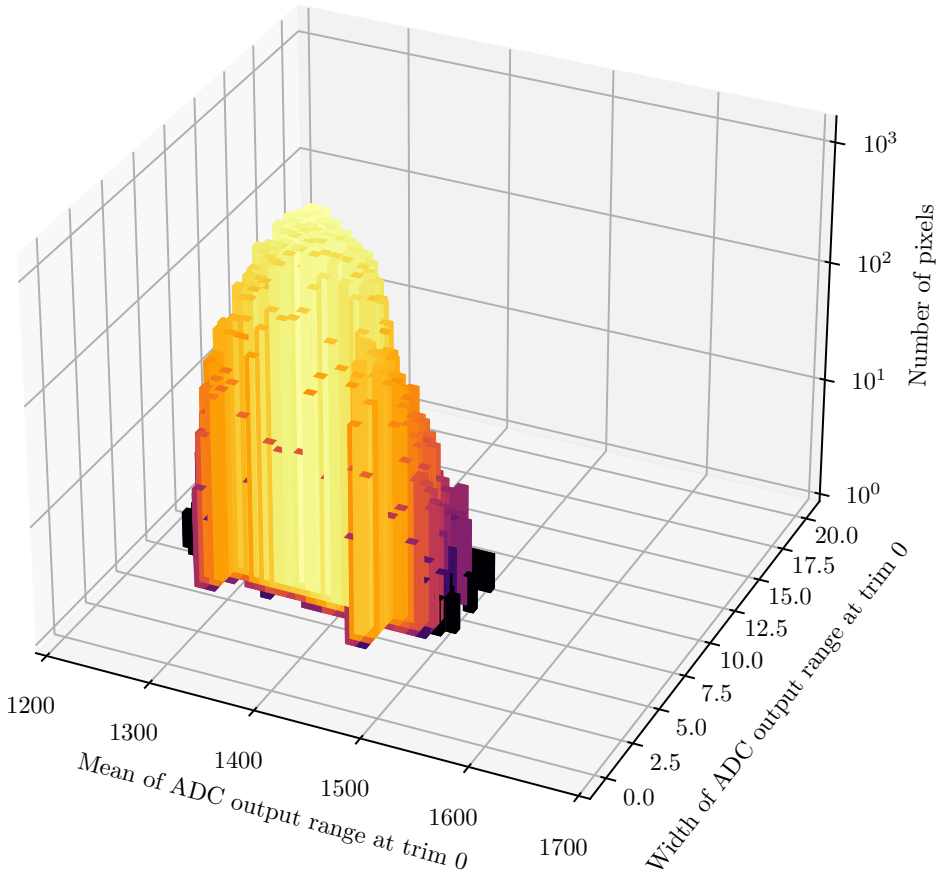

Figure A2: An additional correlation plot showing the relationship between the mean and width of the VP0-0 ADC output range at trim 0. Since this relationship is between two of the neural network input variables, it is less useful and was therefore omitted from the main report.

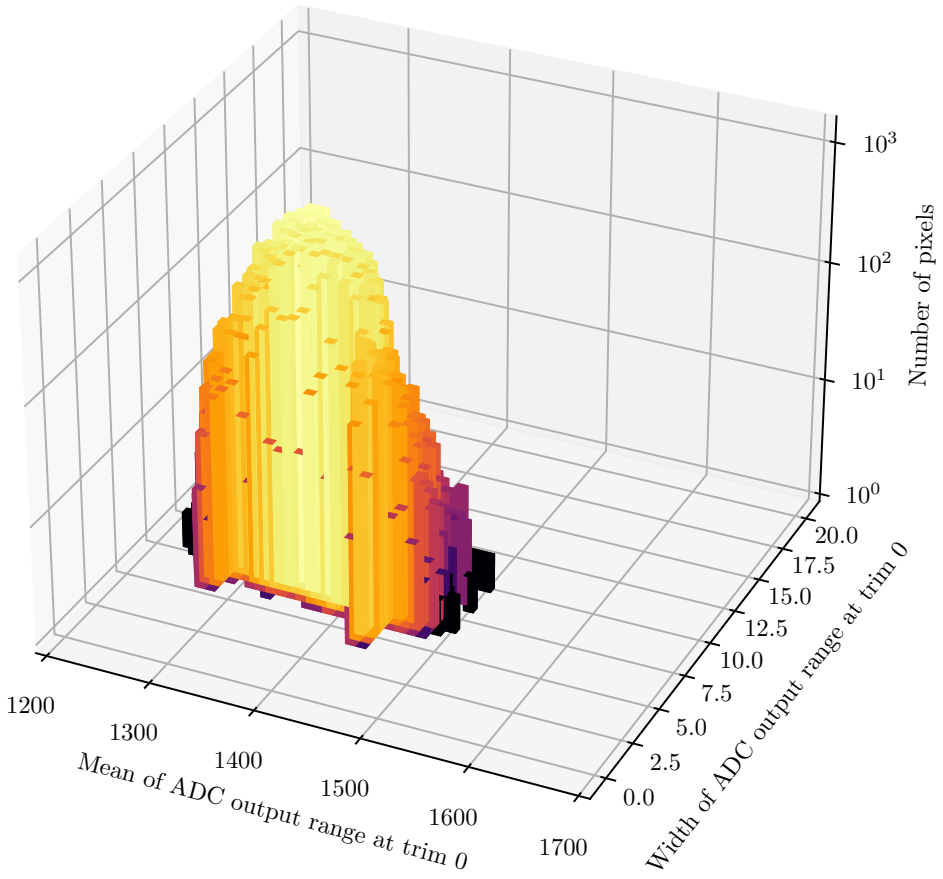

Figure A3: An additional correlation plot showing the relationship between the mean and width of the VP0-0 ADC output range at trim F. Since this relationship is between two of the neural network input variables, it is less useful and was therefore omitted from the main report.

Appendix B: Computer Programs

The following computer programs were written/used during this project (available with accompanying data at `https://github.com/dglover1/phys379-bsc-project#`):

Neural networks

- *NNData_one.ipynb* - Python notebook where neural networks were designed and trained - Daniel Glover, based on original code by David Hutchcroft (supervisor)
- *correlation.ipynb* - Python notebook where plots were created of the correlations between the neural net input/output variables - Daniel Glover
- *diagrams.ipynb* - Python notebook where plots were made to visulise neural net node activation functions - Daniel Glover
- *mnist-example.ipynb* - Keras Practice/example Python notebook which trains a neural net to categorize the standard MNIST dataset - `https://keras.io/examples/vision/mnist_convnet/`
- *plots.ipynb* - Python nodebook where plots of the neural net results are made - Daniel Glover

VELO simulation

- *Boole.py* - Python script to interface with the LHCb Boole (VELO response simulation) software - David Hutchcroft (supervisor)
- *DaVinci.py* - Python script to interface with the LHCb DaVinci (track reconstruction) software - David Hutchcroft (supervisor)
- *test.sh, initialSims.sh, curveMapping.sh, maskingNoisyPixels.sh* - Bash scripts written to allow multiple time-consuming simulations to be run in sequence - Daniel Glover
- *resultsParser.py* - Python script written to parse 'DaVinci.log' output files for rapid extraction and collection of results - Daniel Glover
- *plots.ipynb* - Python nodebook where plots of the VELO simulation results are made - Daniel Glover

Appendix C: Answers to Presentation Questions

Q1. How do you understand the differences in results with the activation functions?

Activation functions are fundamental to the training process of a neural network, as they are the mathematical operations which adjust the strengths of the connections between nodes. When the trained network is in operation, it is these connection strengths which dictate the 'route' that the input data takes through the network as it is transformed into the output data. The gradients of the activation functions play a significant role in forming these routes, as they dictate the severity of a connection strength update, and the main reason that ReLU is regarded as (and performs) better than sigmoid or tanh is that it is less susceptible to problems originating from the function gradients. Therefore, the choice of activation function has a great effect on the final accuracy achieved by the neural networks, and this is observed in Figure 16.

Q2. How did you implement the machine learning algorithm?

The machine learning algorithm was implemented using classes and routines from the Keras library, which is a Python interface to Tensorflow. The desired neural network layers were initialised from Keras classes and arranged in order, before being packaged into a Keras model object and compiled. The model.fit() routine was then called with the chosen training parameters, which begins the training cycle. After a maximum number of cycles, or when an early stopping condition is met, the training process ends and the neural network is ready for evaluation. A more detailed description of neural networks can be found in Section 2.2, and the machine learning code can be seen in the GitHub repository linked in Appendix B.

Q3. How do you verify that third-party packages are being used correctly?

Before implementing the machine learning algorithm with the Keras library, I experimented with the Keras MNIST example[4] (see Appendix B), which trains a neural network to perform handwritten character recognition using the popular MNIST machine-learning example dataset. This demonstrated how a Keras model is set up and used, as well as the program outputs to expect during throughout the various stages.

In addition to practicing with examples, referring to the official documentation of third party packages is generally the best way to ensure that they are being used correctly. An example of this occurred during the implementation of the RMS error calculation code on the mean of the neural network final accuracy values. After some research, it became apparent that there were two popular functions for performing such a calculation. The numpy.std() function calculates the standard deviation of a set of measurements, while the scipy.stats.sem() function calculates the standard error on the mean. After testing both methods, it was observed that they gave similar but slightly different results. It was unclear which function was returning the correct values until the documentation of each function was read, which explained that the scipy.stats.sem() function implements Bessel's correction by default, and numpy.std() does not. Both allow the default option

to be changed, so either function was appropriate providing Bessel's correction was being used. This demonstrates the benefit of referring to the official documentation of third party packages to ensure their correct use.

Appendix D: Initial Project Plan (February 2022)

Aim / Objectives

The focus of this project is the use (and development) of machine learning models to evaluate the noise and dead pixels in test data from the newly manufactured LHCb vertex detector modules. The results of these models will be used to simulate the tracking of charged particles by the detector, and investigate the effect of realistic conditions (due to the various imperfections in realistic devices). These simulations will then be compared to the current 'ideal' simulations and their relative differences evaluated, to gain insight into the manufactured quality and expected performance of these modules in the detector.

Methods / Techniques

This is a computational project which will be carried out using Python (via Jupyter notebooks) with scientific (NumPy) and machine learning (Keras/TensorFlow) packages.

Timeline

Week 1	• Become familiar with project details, aims etc. • Setup programming environment (to point where neural net code runs) • Attempt to optimise first neural net model (maximise accuracy)
Week 2	• Start investigating the effectiveness of different neural net models (altering model/hyperparameters) • Complete risk assessment, project proposal (short overview & timeline) • Submit risk assessment/proposal (Fri 11th Feb 5PM)
Week 3	• Continue trying different neural net models, recording details/results of each (structure of model, accuracy, epoch number etc.)
Week 4	• Extract data from the neural net models (investigate how to convert the model outputs to inputs for the simulations below)
Week 5	• Begin working with simulations: tracking charged particles in the detector (based on input position).
Week 6	• Continue work on simulations – investigate performance under realistic conditions (the effect of various 'failures' in a realistic detector: varying noise, efficiency, ghost tracks, clones, etc.).
Week 7	• Continue above work on simulations • Start planning presentation
Week 8	• Practice, deliver presentation
Week 9	• Continue above work on simulations
Week 10	• Finish simulation work – compare the 'realistic' simulations to the current 'ideal' simulation • Work on writing report
Week 11	• Complete report
Week 12	• Final checks, tidying up • Submit report and portfolio of activity (Fri 13th May 5PM)

Appendix E: Risk Assessment

UNIVERSITY OF LIVERPOOL

RISK ASSESSMENT FORM

School/Department: Physical Sciences	**Building: Home**
Task: Computer-based BSc Project ("Event Pileup and the LHCb Vertex Detector")	
Persons who can be adversely affected by the activity: **Daniel Glover (Student)**	

Section 1: Is there potential for one or more of the issues below to lead to injury/ill health (tick relevant boxes)

People and animals/Behaviour hazards

Allergies		Too few people		Horseplay		Repetitive action		Farm animals	
Disabilities		Too many people		Violence/aggression		Standing for long periods		Small animals	
Poor training		Non-employees		Stress		Fatigue		Physical size, strength, shape	
Poor supervision		Illness/disease		Pregnancy/expectant mothers		Awkward body postures		Potential for human error	
Lack of experience		Lack of insurance		Static body postures	X	Lack of or poor communication		Taking short cuts	
Children		Rushing		Lack of mental ability		Language difficulties		Vulnerable adult group	

What controls measures are in place or need to be introduced to address the issues identified?

Identified hazards	**What controls are currently planned or in place to ensure that the hazard identified does not lead to injury or ill-health?**	**RISK SCORE**			**Is there anything more that you can do to reduce the risk score in addition to what is already planned or in place?**	**RESIDUAL RISK SCORE**		
		L	**C**	**R**		**L**	**C**	**R**

Static body postures	**Attempt to maintain good posture while sat at a computer for long periods to avoid back/neck strain.**	**2**	**2**	**4**	**Take regular breaks during long work sessions**	**1**	**2**	**2**

L = likelihood; C = consequence; R = overall risk rating

Section 2: Common Workplace hazards. Is there potential for one or more of the issues below to lead to injury/ill health (tick relevant boxes)

Fall from height		Poor lighting		Portable tools		Fire hazards		Chemicals		Asbestos	
Falling objects		Poor heating or ventilation		Powered/moving machinery		Vehicles		Biological agents		Explosives	
Slips, trips, falls		Poor space design		Lifting equipment		Radiation sources		Waste materials		Genetic modification work	
Manual handling		Poor welfare facilities		Pressure vessels		Lasers		Nanotechnology		Magnetic devices	
Display screen equipment		Electrical equipment		Noise or vibration		Confined spaces		Gases		Extraction systems	
Temperature extremes		Sharps		Drones		Cryogenics		Legionella		Robotics	
Home working		Poor signage		Overseas work		Overnight experiments		Unusual events		Community visits	
Late/lone working		Lack of/poor selection of PPE		Night work		Long hours		Weather extremes		Diving	

What controls measures are in place or need to be introduced to address the issues identified?

Identified hazards	**What controls are currently planned or in place to ensure that the hazard identified does not lead to injury or ill-health?**	**RISK SCORE**			**Is there anything more that you can do to reduce the risk score in addition to what is already planned or in place?**	**RESIDUAL RISK SCORE**		
		L	**C**	**R**		**L**	**C**	**R**
n/a					**n/a**			

L = likelihood; C = consequence; R = overall risk rating

Section 3: Additional hazards: are there further hazards NOT IDENTIFIED ABOVE that need to be considered and what controls are in place or needed? (list below)

Identified hazards	What controls are currently planned or in place to ensure that the hazard identified does not lead to injury or ill-health?	RISK SCORE			Is there anything more that you can do to reduce the risk score in addition to what is already planned or in place?	RESIDUAL RISK SCORE		
		L	C	R		L	C	R
Eye strain caused by long hours at computer screen	Take regular breaks from looking at screen (recommended every 20 minutes)	3	1	3	n/a			

Section 4: Emergency arrangements (List any additional controls that are required to deal with the potential emergency situation)

Emergency situation	Additional control required
n/a	

Risk assessor (signature).....Daniel Glover.....Date 7/2/2022

Authorised by (signature) .....David Hutchcroft..... Date 7/2/2022