[to do: Abstract]

Contents

1 Neural network triggers

1.1 Motivation

The station-level triggers in the previous chapter have been shown to perform well for the science case of the Pierre Auger Observatory. However, it has also been concluded that a lot of potential data, especially at low energies is ignored. This is by intention in order to keep DAQ readout at feasible levels.

Attempts at improving the overal efficiency of the SD triggers can be made. This is only possible to a certain level. At lowest energies the particle cascade is not big enough to warrant coincident triggers in at least 3 WCD stations. As per **??**, the lateral trigger probability a given classification algorithm can maximally achieve is given by the LPP (c.f. **??**). The T3 detection probability of such an ideal trigger, and consequently the maximal efficiency for an array with 1.5 km spacing is compared to the efficiency of classical triggers in **??**.

Of course, efficiency can be improved simply by adjusting trigger thresholds of the algorithms in **??**. However, the more lenient these thresholds are, the more background events will be detected. This quickly results in trigger rates that are unmanagable for the infrastructure at the Pierre Auger observatory. The probability with which time traces correctly raise a T2 is shown alongside the resulting random-trace trigger rate for different thresholds of classical algorithms in **??**.

Ideally, neural network architectures developed in this chapter should undercut the random-trace trigger rate of classical triggers, while retaining an overall higher accuracy. That is, they lay below and right of the operating point in **??**. For any algorithm that achieves this, the corresponding LTP will be greater than that of classical triggers, resulting in higher event detection efficiency, while not exceeding the bandwidth limitations of the underlaying hardware.

1.2 Implementation

The python library TensorFlow [1] is used as a backend to implement the individual classifiers. All discussed architectures are built and trained with the release version 2.8.4 [2]. Adjustments to the trainable parameters are calculated according to a momentum-based stochastic gradient descent (Adam [3]) on a batch level. In this context, a single batch is made up of all traces that are recorded from a single air shower event. Since batch size grows quickly with increasing energy, a generative approach, where traces are created (c.f. **??**) at runtime upon requirement, is used in building training data in

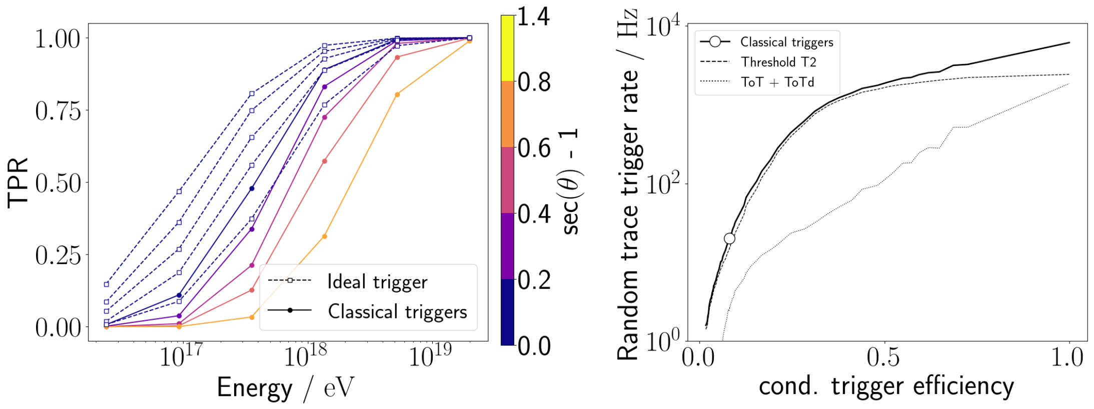

Figure 1.1: (*Left*) Comparison of an ideal trigger sensitive to any shower signal from primary energies $E \geq 10$ PeV to classical triggers. (*Right*) The noise level over calculated efficiency for classical triggers. The tail ends of the potential curve are calculated by adjusting the trigger thresholds from +250% to −95% of the nominal values. The black circle corresponds to the operational setting of a 20 Hz rate.

order to make the process as RAM-efficient as possible. This has important implications. As trace building relies heavily on randomization, the actual training data will not be the same if the random number generators are not seeded beforehand. This has been taken into account. All networks are - unless specifically stated otherwise - trained and validated using the same signal input data.

1.3 Design consideration

1.3.1 Architecture

The hardware specifications at the FPGA level, where trigger conditions are currently checked, are limited. For this reason, NN architectures should be kept as simple as possible. Most importantly, the number of weights, biases and other trainable parameters will need to be hardcoded into station software. Because of minimal available storage space, this number needs to be kept low.

This immediately disqualifies powerful candidates like autoencoders or transformers (compare **??**) from consideration. Only relatively simple dense-, convolutional-, or recurrent neural network architectures are viable contenders and might be implementable in the SD electronics.

1.3.2 Choice of input data

As stated in the previous chapters, neural networks learn by example. It is imperative that the input data propagating through the network during training resembles later

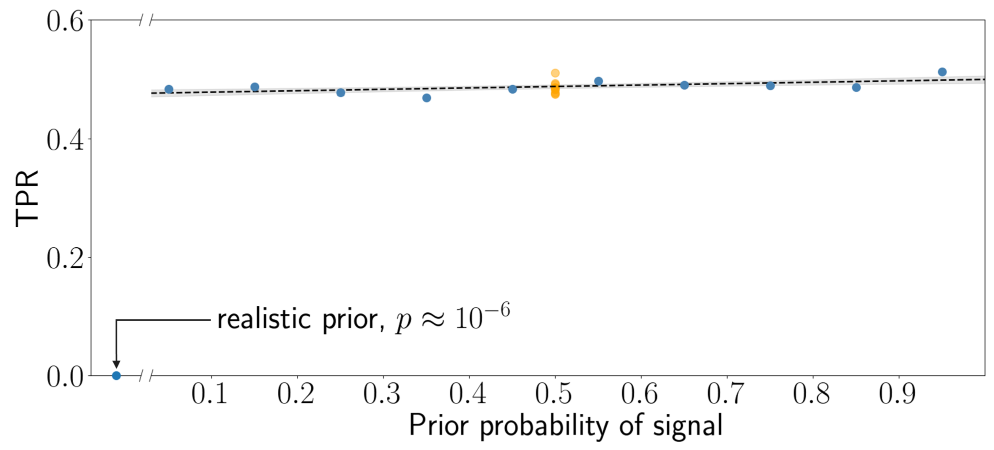

Figure 1.2: A very faint positive slope ($m = 0.02 \pm 0.01$) is observed when relating prior probability to trigger sensitivity (blue dots) of an example convolutional neural network. This could however be attributed to statistical fluctuations in the training fit. An ensemble of networks with the same architecture trained on the same data and with a single prior shows a comparable spread (orange dots).

use-case inputs as much as possible. However, some choices in how data is represented can and need to be made in order to benefit loss minimization.

Prior probability

The flux of cosmic rays with energies exceeding the proton knee is tiny ($\mathcal{O}(1\,\mathrm{m}^{-1}\,\mathrm{yr}^{-1})$ [4]). While the size of the SD guarantees decent exposure over the entire array, an individual station will mostly measure background. In fact, the prior probability p of encountering events beyond the kee in a random time trace is roughly one in one million. Of course, if this were reflected in the training dataset, neural networks would show very poor performance. On the notion of a broken clock being correct twice a day, a naive classifier would naively label every input as background and retain almost perfect accuracy. Such behaviour is not desired. The prior probability must be artificially inflated. The influence of prior probability on subsequent trigger sensitivity is shown in Figure 1.2. No strong correlation between prior and network performance is found in the range $0.05 \leq p \leq 0.95$. As long as the fraction of signal over entire training set is statistically relevant, the network learns to discern between signal and background events. For all further analysis a conservative prior of $p = 0.5$ is picked.

Filtering & Downsampling

1.3.3 Further hyperparameters

The number of ways both input data and the network architecture can be tweaked to minimize loss is extensive. Further optimization methods, like the choice of labelling, are motivated and discussed in the performance reports in section 1.4.

One promising approach of optimizing trigger performance that is not examined in this thesis is the adjustment of the neural network loss function. Several reasons exist that motivate an analysis in this direction:

- **Classwise weighting:** The classes C_1 and C_2 are seen as equally important when calculating efficiencies by default. As noted in subsection 1.3.2 an imbalance in distributions exists. Observing background is much more likely than observing an air shower. The loss function will reflect this to an extent if classification of background is weighted more heavily than classification of signal. Even a more complex loss sensitive to CR flux (compare **??**) or other parameters is possible.
- **Random-trace trigger rate pull term:** For the most part, the analysis presented here will introduce a neural network distinguished by some feature, optimize its performance, and check whether the resulting random-trace trigger rate is in an acceptable range. However, it is possible to go the opposite way and specify a target trigger rate first and then train the network. In this fashion, one could hope that the algorithm efficiency is optimized for a given bandwidth by adding a pull term to the loss function. The challenge of this approach is weighting the pull term in a way that it does not dominate the loss function in different fringe cases. An important drawback of this method is that training will be very slow.

1.3.4 Training procedure

In order to make informed statements about the capability of a given network architecture an ensemble of 10 networks are trained - unless specified otherwise - for 10 epochs. Per epoch, 80% of the available showers are examined to calculate gradients, the remaining 20% are set aside for validation purposes. This is a standard procedure to prevent the network from overfitting training data [5]. Since the training dataset is extensive ($0.8 \cdot 40557 \approx 32445$ Showers), convergence to a local minimum is typically observed already within a single epoch. For this purpose training is halted early if the training accuracy is greater than $\alpha = 0.95$ and no decrease in loss has been observed for at least $75\% \hat{=} 24334$ Showers of the epoch

1.4 CNN Trigger

1.4.1 Architecture

The convolutional neural networks presented in this chapter all have a very similar architecture, which was shown to slightly outperform other candidates during prototyping tests. It is likely that this is due to design ideas that are closely related to the way information is structured in the time traces:

- A **2D convolutional input layer** (c.f. **??**) pools together the information of the three different WCD PMTs.

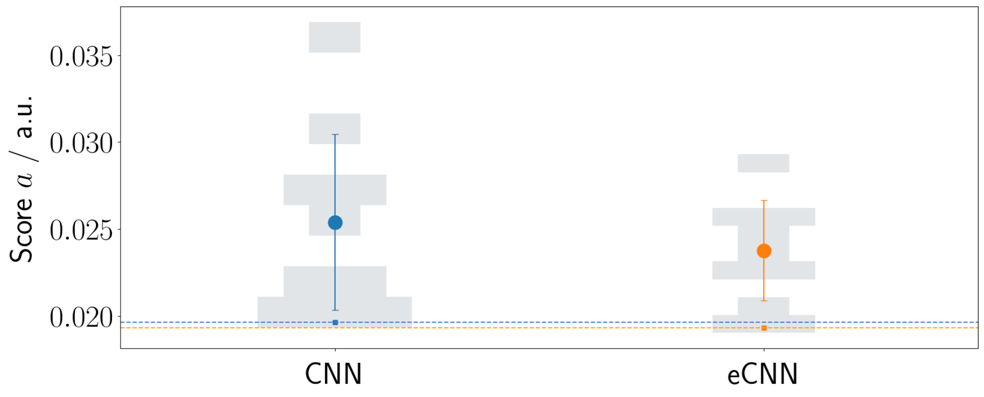

Figure 1.3: The network score as calculated in **??** is compared from original network architecture (CNN, left, in blue) to the extended architecture featuring an additional dense layer (eCNN, right, in orange). Shown are 10 networks of each architecture trained on the same dataset. Both architectures find a similar best solution (squares), while the mean and spread of the population (circles) are smaller for the extended network.

- A **1D convolutional layer** resolves temporal correlation in the output of the previous layer.
- A **Dense layer** reduces the extracted features to a binary choice of signal and background.

In subsequent tests it is furthermore concluded that extending the architecture by an intermediate dense layer between the time convolution layer and the dense output layer is on average beneficial for signal to noise ratio. However, because both approaches arrive at a similar best score, the smaller, non-extended network is seen as more viable for the use case in the Pierre Auger Observatory. An example performance of both models is given in Figure 1.3. The corresponding number of trainable parameters is given in **??**. The precise implementation designs for both cases are shown in [to do: Architecture plot]

[to do: Architecture plot] [to do: Network Parameter tweakable]

1.4.2 Performance

Training a convolutional neural network with the data from **??** as is results in near perfect accuracy. The lateral trigger probability obtained after training is equal to the LPP (see Figure 1.4 and **??**), which quantifies the probability of particle existance at station level. This implies that the networks are able to identify even single particles in the WCD with very high confidence. However, they have not learned to distinguish muonic events from actual extensive air showers. This results in a very high random-trace trigger rate of $f_{\mathrm{Random}} > 2\,\mathrm{kHz}$. In order to decrease this value, the training data must be altered.

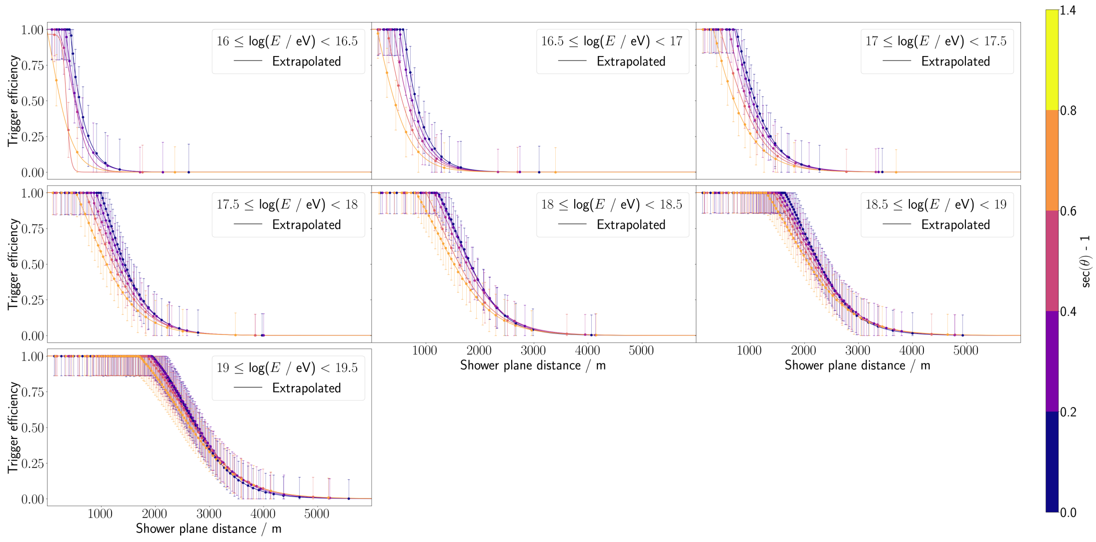

Figure 1.4: The lateral trigger probability is equal to the laterap particle probability shown in **??** for almost all E and θ. Discrepancies occur at low primary energies and high inclinations due to low statistics complicating the LTP fit. In any case, this implies that the chance of raising a T2 provided a station was hit by an air shower is 1.

Muon cut

As discussed in **??** atmospheric muons from very low-energy showers represent the dominant background (that is not just electronic noise) in measured data. The most straight forward approach of lowering the network sensitivity for such signals is requiring muonic coincidences. This means only data from tanks which were hit by at least 2, 3, ..., n_μ muons are assigned a signal label, where n_μ is a tweakable hyperparameter. Traces can be selected based on other particles as well. $\overline{\text{Off}}\underline{\text{line}}$ saves the number of muons, electrons and photons for simulated stations. All three, or even a (weighted) sum are candidate discriminators. In this work, only the muon cut is analyzed with greater detail in **??**, as it is thought to be more closely tied to the problem of distinguishing signal from background.

Several ensembles are trained with different n_μ to test the quality of this cut. Their subsequent performance is shown in Figure 1.5. The results are inconclusive. While low cut thresholds requiring a coincidence of $\gtrsim 2$ muons seem to represent a solid way of lowering f_{Random}, the trigger rates are not brought down to acceptable levels by this cut alone. Even worse, the training fits do not converge uniquely at higher cut thresholds. This indicates the number of muons is not (at least at higher muon counts) a great discriminator. This may be attributed to several reasons:

- **Lack of strict temporal coincidence**: The imposed cut only requires at least n_μ muons to hit the tank at some point during a shower event. Without a more stringent requirement it is possible that a network sees not a trace stemming from n_mu muons, but rather n_μ traces from individual particles during a sliding

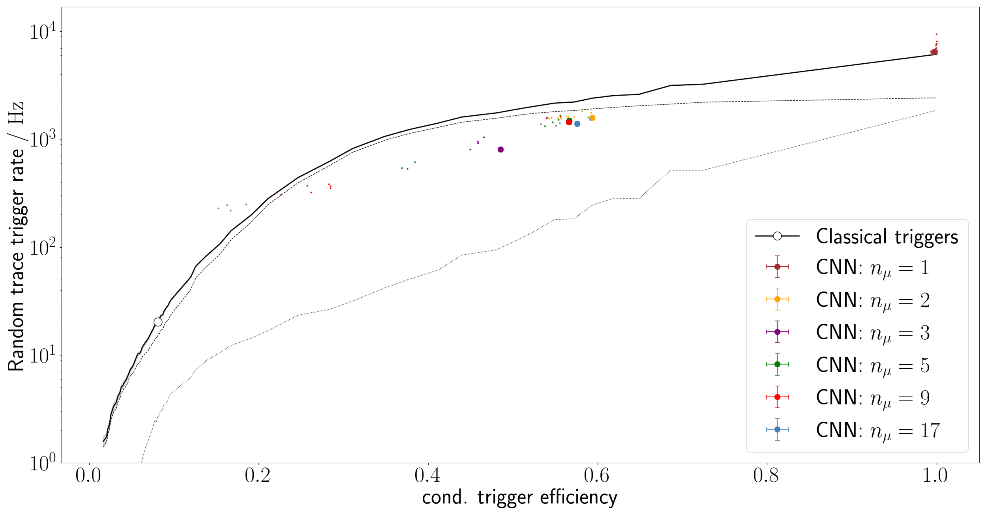

Figure 1.5: The performance of convolutional neural networks trained on data with a muon cut of different thresholds. The optimizers don't seem to identify a unique local minimum at very high cut thresholds. For all intermediate cuts $1 < n_\mu < 17$ the convolutional neural networks offer a better trigger efficiency compared to the Th-T2 trigger at the same random-trace trigger rate. The performance of the ToT trigger is not reached.

window analysis. Since all of these fit the cut criterion and are hence labelled as signal, the network does not learn to ignore such traces.

- **Variety in single muon traces**: The detector response to a single muon varies based on muon inclination, detector specifics and statistics. All of these are parameters that are not available to the CNN. The combination of all fluctuations might be too complex for a convolutional neural network with limited size to learn.

Nevertheless, a hybrid cut requiring at least 2 muons in a tank on top of some other trace characteristic might hold potential in further analysis. A muon cut by itself is shown to not work ideal for the data at hand.

Charge cut

Another method of teaching a network to ignore certain signals is given by imposing a cut t_S on the integral S of the trace. One advantage of this approach is the fact that the choice of labelling in this case is fully determined by the input data. Another edge is the granularity of the cut. Whereas only integer values of n_μ are sensible in the previous paragraph, the integral is discretized in much smaller steps of 1 $\text{VEM}_{\text{Ch.}}{}^{-1}$. This allows for finer tuning of the network performance.

By increasing t_μ, the network will learn to ignore increasingly larger signal. This is shown in Figure 1.6. The performance arising from different t_S is shown in Figure 1.7.

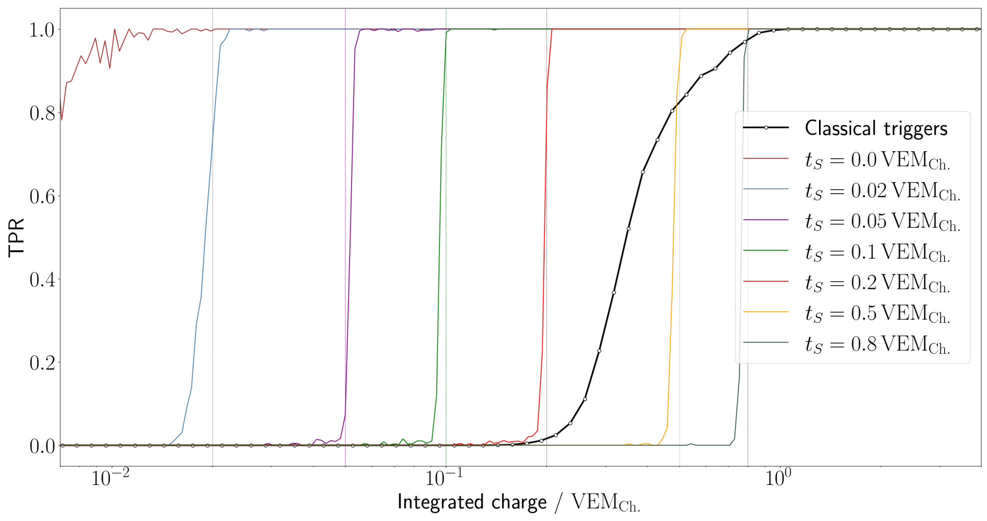

Figure 1.6: The true positive rate (TPR) in terms of deposited signal for different classifiers. Networks that were trained with a cut threshold of t_S will not trigger on traces that have a lower integral $t_S > S$, dropping the sensitivity in this region. This is precisely the desired effect. Classical triggers become fully efficient at around 1 $\mathrm{VEM_{Ch.}}$.

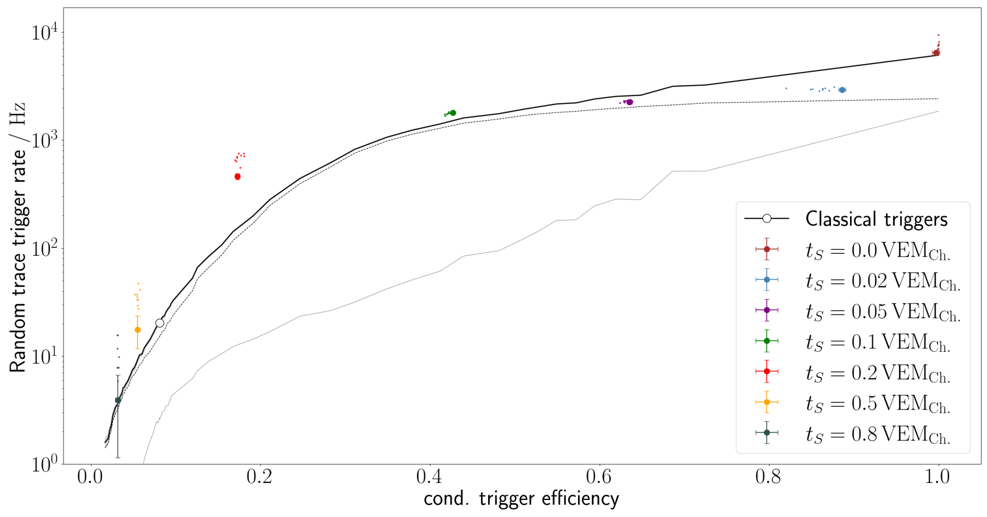

Figure 1.7: Network performance in relation to different integral thresholds. With an increasingly stringent cut, the networks learn to ignore more and more low-energy events, which quickly drops the trigger efficiency. On the other hand, the classifiers random-trace trigger rate drops to more acceptable levels.

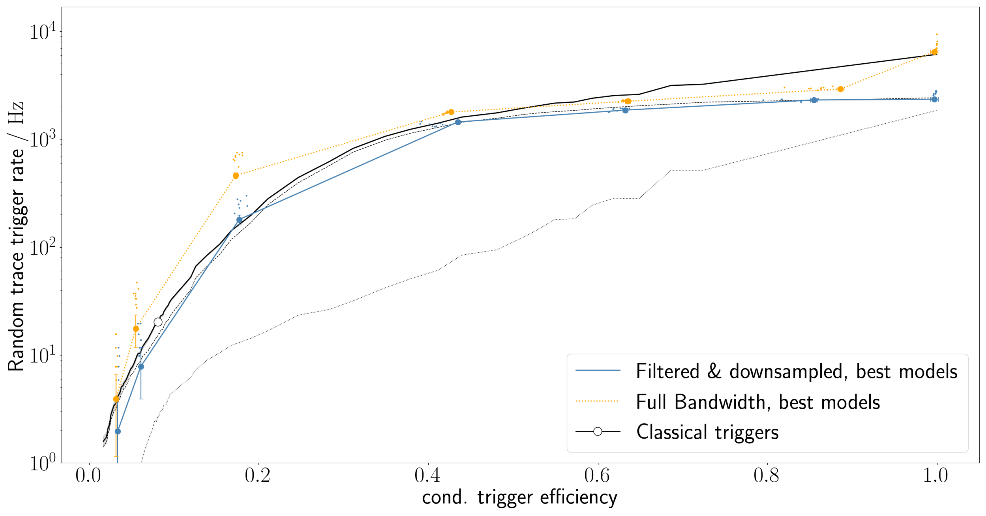

Figure 1.8: Neural networks trained on compatibility mode traces compared to networks trained on full bandwidth traces. Compatibility-trained networks consistently have a lower random-trace trigger rate

Full Bandwidth vs. Filtering & Downsampling

Charge cutting provides a powerful method to control the conservativity of the classifier and force discrimination between muonic background and extensive air showers. Equipped with this tool, other questions can be answered in an attempt to optimize network performance.

For example, classical triggers are run in compatibility mode, where 3 UUB bins (8.3 ns) are filtered and downsampled to resemble a single UB bin (25 ns). Some information about the trace shape is lost in the process. Logically, a neural network should be able to identify original UUB signal traces with more ease than their filtered and downsampled counterparts. Ideal triggers should have a higher efficiency when tested on full bandwidth traces.

This does not render compatibility mode uninteresting for analysis. An effective way of minimizing the trainable parameters in a neural network is reducing its' number of input parameters. Filtering and downsampling encodes - albeit imperfectly - the input data and reduces the dimensionality by a factor of three. This is done in such a way that the UB-like trace contains trace information from a time window that is three times as long for a given number of bins.

Consequently, by filtering and downsampling input data it becomes possible to hand the classifiers additional information while keeping the network size at a fixed level. The effects this has on predictive power of neural networks is shown in Figure 1.8.

Classifiers trained on compatibility mode traces consistently have a lower random-trace trigger rate while operating at the same efficiency. Whether this is caused by the above discussed difference in (temporal) input size or other trace characteristics is determined in the next paragraph.

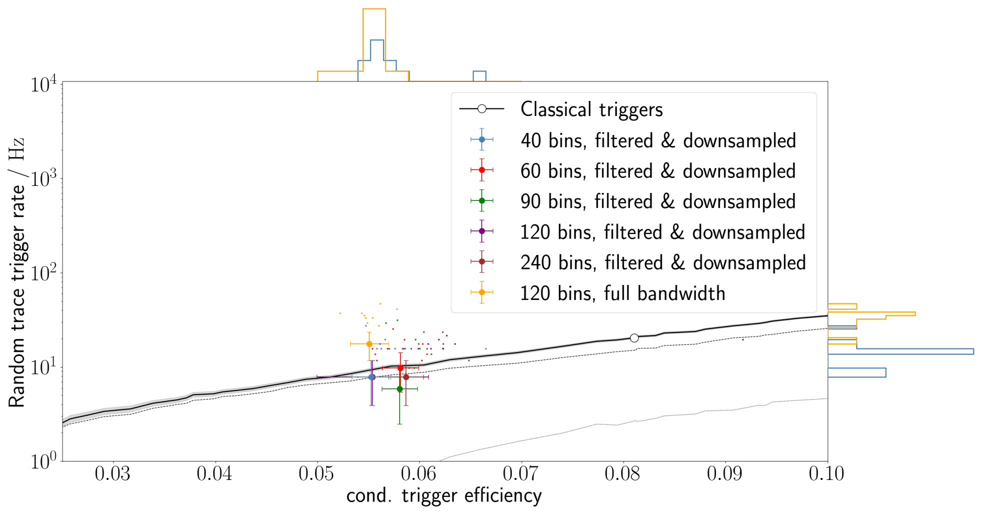

Figure 1.9: The comparison between a compatibility mode network (blue) that has the same input size in time to a full bandwidth network (orange) reveals the advantage of filtering & downsampling. Both networks have a comparable efficiency, but the compatibility mode network is less sensitive to background triggers from random-traces.

Input size

To test whether filtering and downsampling input data gives the classifiers an edge when predicting labels, multiple ensembles are trained on the same dataset with the same charge cut $t_S = 0.5\,\text{VEM}_{\text{Ch.}}$. Each ensemble differs in the size of the input trace. If compatibility-trained networks are still outperforming full bandwith ones that have an input size three times the size, filtering & downsampling is the preferred method for classifying air showers with convolutional neural networks.

Indeed, this is found. While the compared networks converge to a similar trigger efficiency, the networks that receive filtered and downsampled traces show a lower random-trace trigger rate. Furthermore, a larger input size slightly positively influences trigger efficiency at the cost of increased network size. This is expected.

1.4.3 Kernel size

The presented models so far show a performance that is similar to the Th-T2 threshold trigger. In an attempt to boost performance, the network architecture is altered. Namely, the kernel size of the 2D convolutional input layer is increased. While this extends the network size, it could be expected that the consideration of a larger window during convolution helps the network extract more meaningful features, and increases predictive power in the process.

Three architectures are examined to check for a trend in prediction efficiency. The original $(3, 3)$-kernel only takes a minimum window into consideration. It has the lowest accuracy out of all models. The two other candidates have a kernel size of

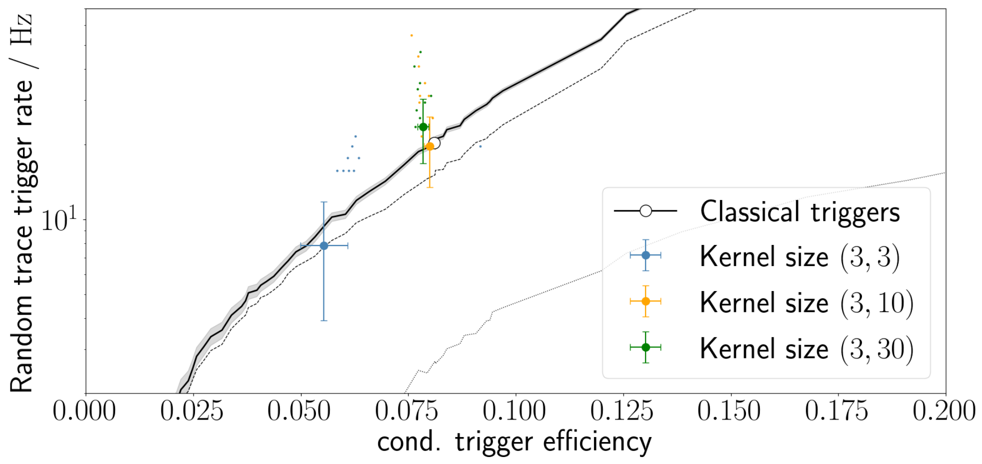

Figure 1.10: All listed networks are trained with $t_S = 0.5\,\mathrm{VEM}_{\mathrm{Ch.}}$. An increase in kernel size benefits trigger efficiency to some extent. On the other hand, random-trace trigger rate increases. No clear trend in network score can be seen.

$(3, 10)$ and $(3, 30)$ respectively. Both have a very similar performance as can be seen in Figure 1.10.

It seems trigger efficiency is not strongly correlated to kernel size beyond a certain level. An increase in network score is not achieved by increasing it. All architectures have a performance very similar to the threshold trigger.

1.5 LSTM Trigger

1.5.1 Architecture

1.5.2 Charge cut

[to do: signal-to-noise-ratio plot]

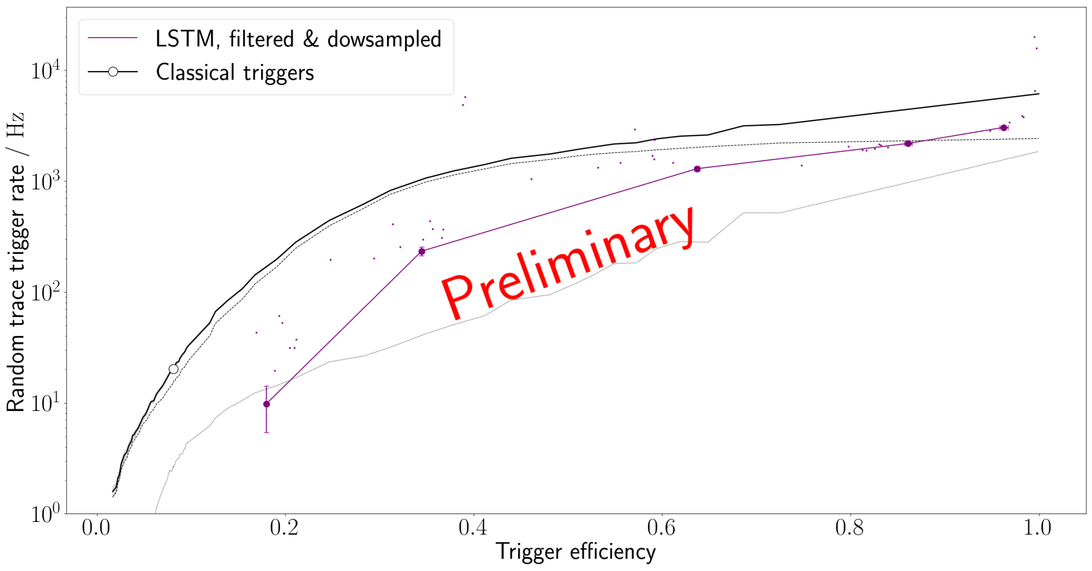

Figure 1.11: performance of LSTM