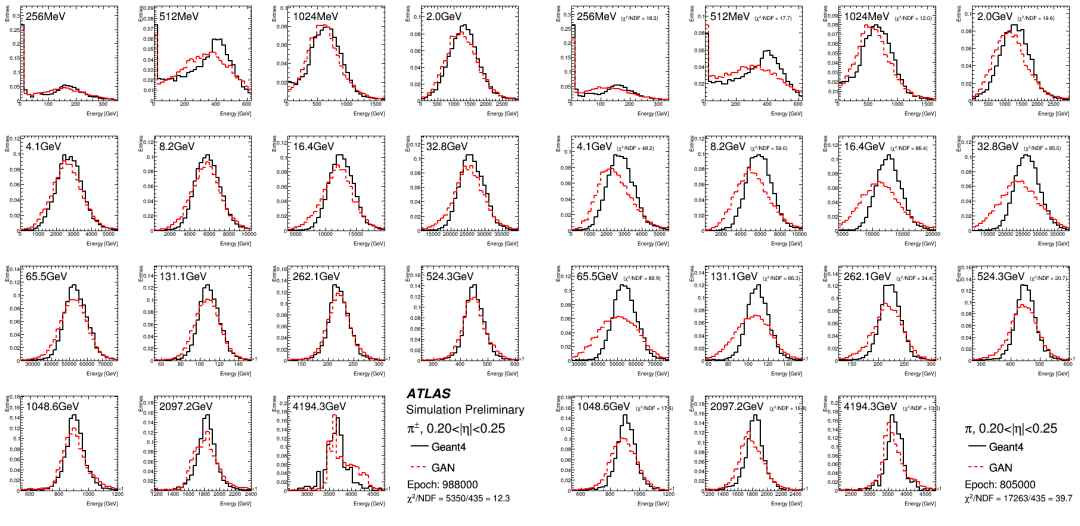
Improving FastCaloGAN Simulation of Pion Calorimeter Showers

Matthew Barber

1 Introduction

FastCaloGAN [1] is a tool that performs a fast simulation of the full ATLAS calorimeter system using unsupervised machine learning techniques, in particular generative adversarial networks (or GANs). The figure of merit used to compare different models is a $\frac{\chi^2}{\text{ndf}}$ statistic between the binned total energy distributions of the GAN and the training data. The goal of this project then was to modify the architecture of FastCaloGAN so as to achieve as low a $\frac{\chi^2}{\text{ndf}}$ as possible with minimal computational resources. My work was focused on simulating charged pions in one region of the detector ($0.2 < |\eta| < 0.25$). The number of iterations used to trained the models was kept fixed at the default value of 1000000. Evaluation of the $\frac{\chi^2}{\text{ndf}}$ was performed every 1000 iterations.

In version one of FastCaloGAN, the best $\frac{\chi^2}{\text{ndf}}$ for pions in the above-mentioned eta slice was 12.7, as shown in Figure 1a. However, the voxelisation described in [1] was later changed so that there were more voxels. This was done to increase the information available in the simulation step. Due to this change, the number of voxels rose from 536 to 1086, increasing the complexity of the problem the GAN needs to solve and thereby making training more difficult. Correspondingly, Figure 1b shows that the lowest $\frac{\chi^2}{\text{ndf}}$ found within the same number of epochs rose to 39.7. Training this model on the GPU resources available on the CERN HTCondor system takes approximately 18 hours and requires 42.6 MB of storage space. This was the state of the FastCaloGAN for pions at the beginning of this project.



(a) (b)

Figure 1: (a) shows the total energy response of the calorimeter for FastCaloGANV1. (b) shows the response once the number of voxels was increased to 1086.

2 Energy Labels

In the original version of FastCaloGAN, the WGAN-GP is conditioned on a parameter, a scaled energy label in the interval $(0, 1]$. In particular, if the incoming pion had kinetic energy E and the maximum kinetic energy of any incoming pion in the training data was E_{max} then the energy label would be

$$\hat{E} = \frac{E}{E_{\text{max}}}.$$

However, from physical reasoning, we expect the shower width to depend logarithmically on the kinetic energy of the incoming pion. Therefore, we expect better performance if we use the energy label

$$\hat{E} = \frac{\log\left(\frac{E}{E_{\text{min}}}\right)}{\log\left(\frac{E_{\text{max}}}{E_{\text{min}}}\right)},$$

where E_{min} is the minimum kinetic energy of the incoming pions in the training data. Indeed, Figure 2 shows that doing so reduced our primary $\frac{\chi^2}{\text{ndf}}$ statistic to 25.3 and in fact achieved a lower $\frac{\chi^2}{\text{ndf}}$ for every energy on which it was trained. This logarithmic normalisation was then adopted going forward.

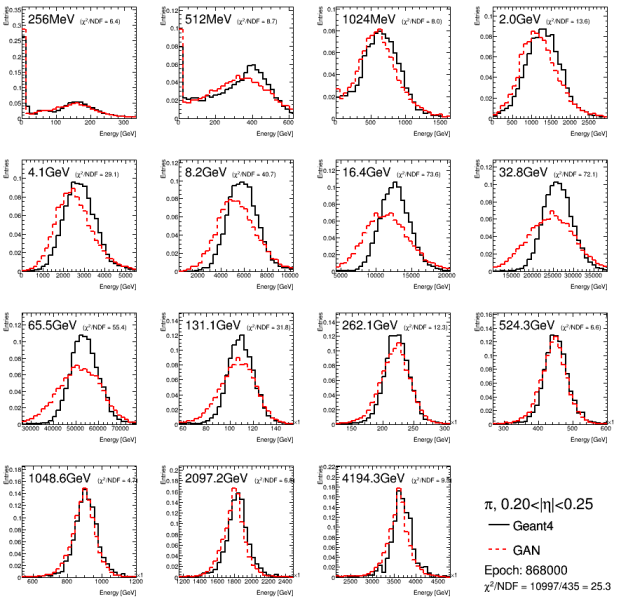


Figure 2: Total energy response of the calorimeter to pions after using logarithmically normalised energy labels.

3 Generator Size

The generator is a neural network with five layers: one input layer, three hidden layers and one output layer. In FastCaloGANV1, the input layer consisted of three nodes representing information about the incoming pion and 50 nodes, known as the latent space, whose values were

independent Uniform $([-1, 1])$ random variables. Moving forwards through the generator, the numbers of nodes in the hidden layers were 50, 100 and 200. In this section, such an architecture will be denoted by 50-50-100-200, where the first number represents the dimension of the latent space and the subsequent numbers give the numbers of nodes in the hidden layers. Analagous notation will be used for other architectures.

Table 1 compares the performance of a selection of generator architectures. We see that adopting a larger generator can give a dramatic improvement in performance. Indeed, in Figure 3, we see that the energy distribution of the GAN matches the Geant4 simulation far more closely than in Figure 2 and, correspondingly, the $\frac{\chi^2}{\text{ndf}}$ is greatly improved. This result is to be expected for a few reasons. Of course, increasing the number of nodes in the generator means that there are more weights, giving the generator more freedom to learn the desired distribution. In addition, the output layer of the generator has 1086 nodes, one for each voxel. When the final hidden layer of the generator had only 200 nodes, the increase in the number of nodes between this final hidden layer and the output layer was likely too sharp, whereas with a larger generator, the number of nodes can increase more evenly as we move forward through the network, roughly doubling each time we move to the next layer. The primary cost of this enlargement of the generator is that it increases the storage space needed to store a model from 42.6 MB to 54.2 MB. This may correspond to a significant increase in the memory requirement in the Athena service that only loads the generator part. Since memory was not a problem in version one, an increase is acceptable, especially since there are new tools (ONYX) that can reduce the memory consumption. Since the large majority of training time is spent on the discriminator, this change caused no significant increase in the training time. Overall, we believe this large improvement in the $\frac{\chi^2}{\text{ndf}}$ justifies the additional computational resources needed for training and therefore used this architecture in all subsequent sections.

	50-50-100-200	100-100-150-200	100-128-256-512	100-200-400-800	100-256-512-1024
$\frac{\chi^2}{\text{ndf}}$	25.3	24.2	8.2	5.5	5.9

Table 1: The lowest $\frac{\chi^2}{\text{ndf}}$ found for various generator architectures.

We also note briefly that the 100-256-512-1024 architecture seemed to perform worse than the 100-200-400-800 architecture, despite having a larger generator. We have several candidate explanations for this. One possibility is that it takes more iterations to optimise the GAN's weights when there are more to optimise and that the smaller architecture therefore performs better when training to only 1000000 epochs. Another is that having too large a generator makes it too powerful relative to the discriminator, therby reducing the effectiveness of training.

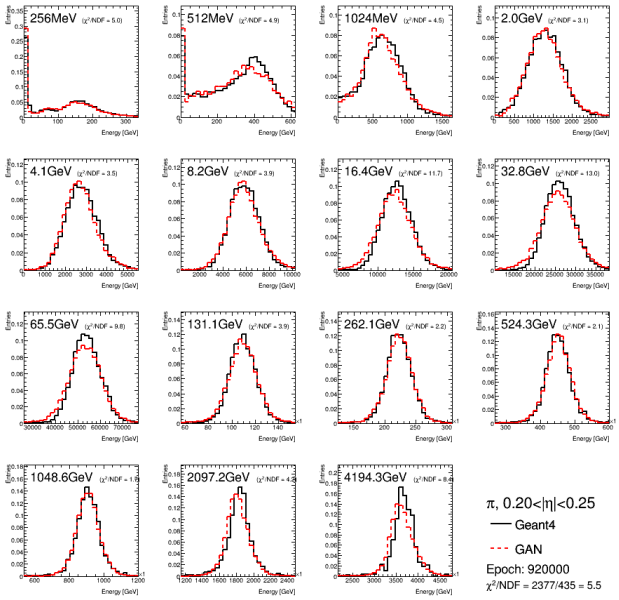


Figure 3: Total energy response of the calorimeter to pions with a 100-200-400-800 generator architecture.

4 Discriminator Size

Like the generator, the discriminator is a neural network with three hidden layers. In version one of FastCaloGAN, the number of nodes in each hidden layer was equal to the number of voxels. However, it was discovered that by reducing the number of nodes so that there were 800, 400 and 200 nodes in the three hidden layers respectively as you move forward through the discriminator, a $\frac{\chi^2}{\text{ndf}}$ of 4.2 could be achieved. Moreover, this reduced the storage space required to save a model from 54.2 MB to 28.9 MB. The resulting energy response is shown in Figure 4.

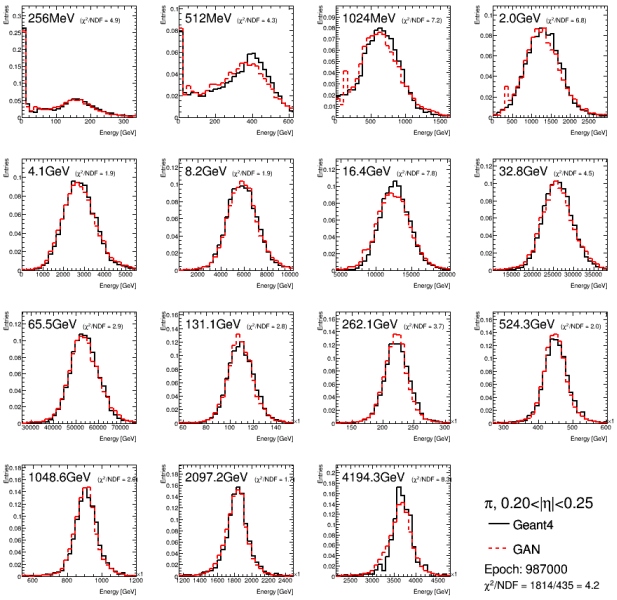


Figure 4: Total energy response of the calorimeter to pions with a discriminator whose hidden layers have 800, 400 and 200 nodes respectively, moving forward through the network.

5 Further Improvements

Normal Latent Space Variables and Batch Size As mentioned previously, in FastCaloGANV1, the latent space inputs were independent uniform random variables. Instead, it was found to be more effective to replace them with independent normal random variables with mean $\frac{1}{2}$ and variance $\frac{1}{4}$. This lead to a $\frac{\chi^2}{\text{ndf}}$ of 3.8 being achieved. Another change resulting in a similar improvement was to increase the batch size from 128 to 512. When this was implemented using uniform random variables in the latent space, the resultant best $\frac{\chi^2}{\text{ndf}}$ was also 3.8. However, when these two changes were combined, a model with a $\frac{\chi^2}{\text{ndf}}$ of 2.7 was found. However, increasing the batch size to 512 meant that the model now took roughtly 56 hours to train, compared to 18 hours before. Nevertheless, it was decided that we would adopt these changes due to the improved energy distribution.

Discriminator/Generator Training Ratio In FastCaloGAN the discriminator is trained multiple times each time the generator is trained. In particular, in version 1, this training ratio was set to be five. However, we found that we could reduce this ratio to three without any noticeable reduction in performance. Doing so reduced the training time for the GAN from about 56 hours to around 34. The resulting energy response is shown in Figure 5. However, when this ratio was reduced even further to one, the performance was dramatically worse, with the lowest $\frac{\chi^2}{\text{ndf}}$ being 139.6.

TensorFlow 2 Rui Zhang refactored much of the implementation of FastCaloGAN to upgrade the code to use TensorFlow 2, rather than TensorFlow 1. When the previously described pion architecture was implemented, the training time was significantly improved from around 34 hours to approximately 7 hours.

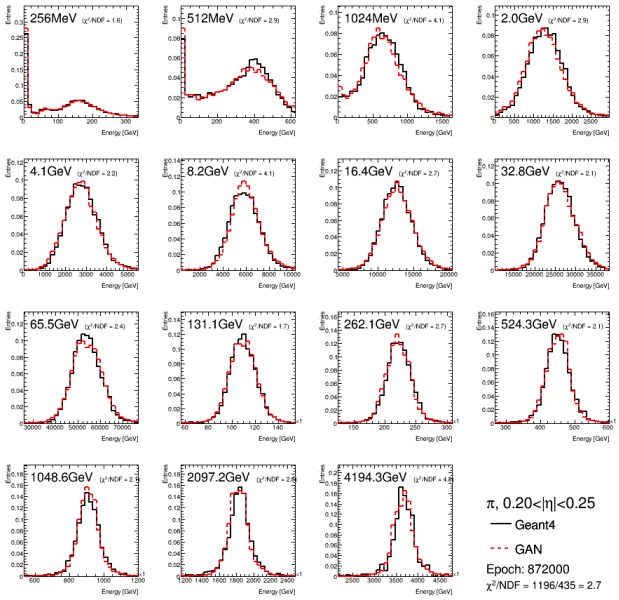


Figure 5: Total energy response of the calorimeter to pions with normal random variables in its latent space, a batch size of 512 and a training ratio of three.

6 Conclusion

When we combine all of the changes described above, the net result it to take us from a $\frac{\chi^2}{\text{ndf}}$ of 39.7, a training time of 18 hours and a required storage space of 42.6 MB to a $\frac{\chi^2}{\text{ndf}}$ of 2.7, a training time of 7 hours and a required storage space of 28.9 MB. We have therefore achieved significant improvement both in terms of the performance of the model and in the computational resources required to train it, just as we set out to at the beginning of this project.

7 Acknowledgements

Several people played an extremely important role in helping me throughout this project and I would like to thank them all. Firstly, thank you in particular to Michele Faucci Giannelli, who supervised and guided me throughout my investigations. Rui Zhang did much of the work merging our codes together and helping me understand his TensorFlow 2 implementation. Finally, Mikel Zemborain wrote the project README file, which was particularly helpful at the beginning of the project.

References

[1] The ATLAS Collaboration, Fast simulation of the ATLAS calorimeter system with Generative Adversarial Networks, ATL-SOFT-PUB-2020-006, 2020, URL: `https://cds.cern.ch/record/2746032/files/ATL-SOFT-PUB-2020-006.pdf`