Conditional Generative Adversarial Networks for Particle Physics

Capstone 2016

Charles Guthrie (cdg356@nyu.edu)
Israel Malkin (im965@nyu.edu)
Alex Pine (akp258@nyu.edu)

Advisor: Kyle Cranmer (kyle.cranmer@nyu.edu)

***Abstract*— Existing methods for simulating particle collision experiments like those at the Large Hadron Collider (LHC) are time-consuming, slow, and expensive. In these experiments, two high-energy beams of particles are collided with one another, and the resulting "particle shower" is recorded by calorimeter sensors that surround the beams. We explore using a conditional Generative Adversarial Network (cGAN) to simulate the sensor data that is recorded from these experiments. Taking particle type (photon or pion) and momentum as inputs, it generates a two-dimensional image representing a slice from the three-dimensional array of LCD electromagnetic calorimeter (ECAL) sensors.**

Keywords—high energy physics; particle physics; conditional generative adversarial networks.

I. Background

The Large Hadron Collider (LHC) is the largest particle collider in the world, designed to shoot two high-energy beams of subatomic particles, and measure the resulting particle showers when they collide. The particle showers that are produced by these collision experiments are detected and measured with a variety of sensors. Sensors in the LHC that are designed to measure the energy levels of particles are called "calorimeters". The data provided to us by Bendavid et al. contains the calorimeter readings of nearly one million collision events.

Existing methods to simulate calorimeter sensor data rely on statistical models derived from physical theory. As a consequence, building a model for a novel experiment requires simulating underlying physics, which is often computationally expensive. Neural network models, in contrast, require no scientific expertise to construct, and are computationally inexpensive to run once they have been trained. If a neural network model could be constructed to generate realistic calorimeter data, it would significantly speed up existing computational pipelines for high-energy physics experiments.

II. Data

The LHC collision event data set was provided to us as part of the CERN Open Data Initiative (Bendavid, 2016), courtesy of Maurizio Pierini and Jean-Roch Vilmant. It contains calorimeter readings for nearly one million single-particle collision events, divided nearly evenly between two particle types: photons and neutral pions. Each collision event has its particle shower recorded by two different kinds of calorimeter sensors: an LCD electromagnetic calorimeter (ECAL) and a hadron calorimeter (HCAL). Both types of calorimeter record energy values in a three-dimensional grid. The particle beam travels through the calorimeters, along their z-axis. The ECAL consists of a 25-by-25-by-20 array of sensors, while the HCAL calorimeter is a 4-by-4-by-60 array. Each sensor records a positive number representing the energy detected at that point in space (See Fig. 1).

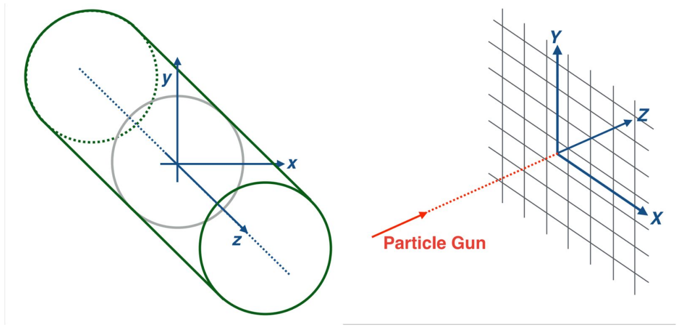

Fig. 1 (Bendavid, 2016). The particle beams travel along a cylindrical tube on the z-axis of the calorimeters, which are placed at their collision point.

The data is split across 100 different files in Hierarchical Data Format (HDF5), each containing about 10,000 different collision events, split roughly equally between each particle type. Each event has a three parts: the event's parameters, its ECAL readings, and its HCAL readings. The event parameters are given as a list of numbers containing the type of particle beam used, the initial energy of the beam, and its initial momentum vector. The ECAL and HCAL sensor readings are each given as a 25-by-25-by-20 tensor of floating point numbers. An example sensor reading is visualized in Fig. 2.

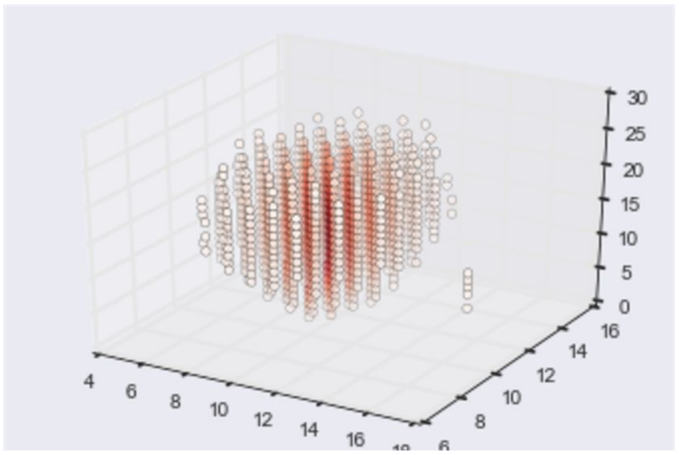

Figure 2. A 3D picture of an ECAL calorimeter reading. Darker points have higher energy readings than lighter ones.

We made a few small modifications to the data that facilitated its use with our model. First, we noticed that values 20 through 24 are always zero in the first two dimensions of the ECAL data. We do not know if this was expected or an error in the data (see Fig. 3). We discarded this portion of the data since it had no information. Additionally, since the beam enters the calorimeters exactly along the z-axis of the calorimeters, the momentum vector has zero values in its y-axis and z-axis entries, and its x-axis entry is equal to the initial total energy of the beam. The momentum vector is therefore redundant with the initial energy value. As a consequence, the particle type and the initial energy of the beam were the only two values we used to parametrize each event. Finally, we used 80 percent of the data, roughly 80,000 events, for training the model. We used the remaining 20 percent for its evaluation.

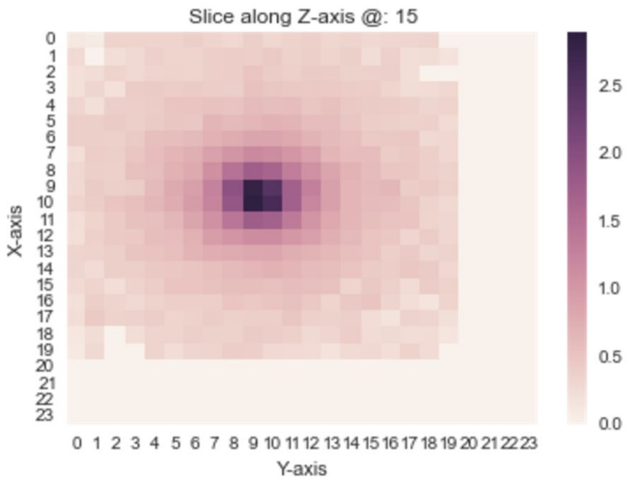

Figure 3. An example 2D slice of ECAL calorimeter. Note that the data points from entries 20 through 24 are empty.

III. Our Model

Our given task was to devise a generative model to simulate these three-dimensional calorimeter readouts. Essentially, we needed to generate three-dimensional images. We knew that recent research had seen great success using generative adversarial networks (GANs) for image generation (Goodfellow et. al, 2014). A GAN contains two neural network models within it that are trained in unison on the same data set: a “generator” network and a “discriminator” network.

The generator network takes a random vector as input and attempts to create an output image that appears as though it came from the training data set. The discriminator takes real-world- and generated images and tries to determine which ones are which. The training process alternates between training the generator and the discriminator. The generator’s loss is one minus discriminator loss. They are trained against each other - hence the “adversarial” nature of the networks. If trained properly, the result is a discriminator that is adept at distinguishing real from fake images; and more importantly, a generator that can create realistic images, or in this case, calorimeter readout simulations. Figure 4 gives the schematic of the process.

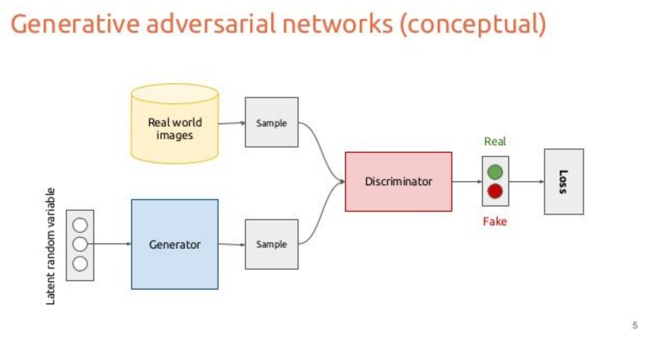

Figure 4: Schematic of a Generative Adversarial Network (McGuinness, 2016)

The conditional GAN (cGAN) (Mirza, Osindero, 2014) takes this idea a step further by allowing users to specify different types of images. For example, in the seminal paper, this meant the network could generate an MNIST digit specified by the conditioning parameter rather than simply producing one of the ten possible digits at random. The cGAN achieves this by introducing a second input vector specifying the additional parameter to both the generator and discriminator. In the MNIST case, this vector was a single scalar indicating the digit the generator should create, and was used to select an example of that digit from the training data for the discriminator to compare against.

Our approach was to use a cGAN to generate calorimeter images using momentum and particle type as the conditional inputs. Once trained, it would allow users to specify either pion or photon, and a particular momentum value for that particle, and generate a simulated calorimeter sample. To simplify the initial problem, we took a single middle slice of the calorimeter readouts to work with 2D data, rather than the entire 3D image.

IV. Evaluation Method

There is no agreed-upon general-purpose metric for evaluating the intrinsic quality of samples from a generative model. On a high-level, the generated samples should resemble examples from the real data. A common method is to measure the probability distribution of a summary statistic of the real data, and measure how well the corresponding distribution from generated samples matches it.

One simple metric to evaluate model performance is the total amount of energy present in the calorimeter data relative to the input data. The sum of the energy measured by the calorimeter should not vary significantly given a type of particle beam and the magnitude of the input energy. If the total energy in a generated sample is significantly different from that of a real calorimeter measurement with the same input parameters, then it is unlikely the generated sample came from a similar distribution to the real one.

We formalize this idea by taking all the ECAL readings in the validation data set for a single particle type, and summing up the ECAL tensor for each one. We then divide each of these total energy values by each event's corresponding input energy to find the ratio of output energy to input energy for each event. We then take all the input energy values from this dataset, and use them as input to our GAN model to generate a simulated ECAL reading for each one. As a qualitative comparison, we created a scatter plot of the two datasets to visually compare the two distributions. See figure 5, below, for an example. These gave us a simple way to determine if the model's samples were similar to those of the real dataset.

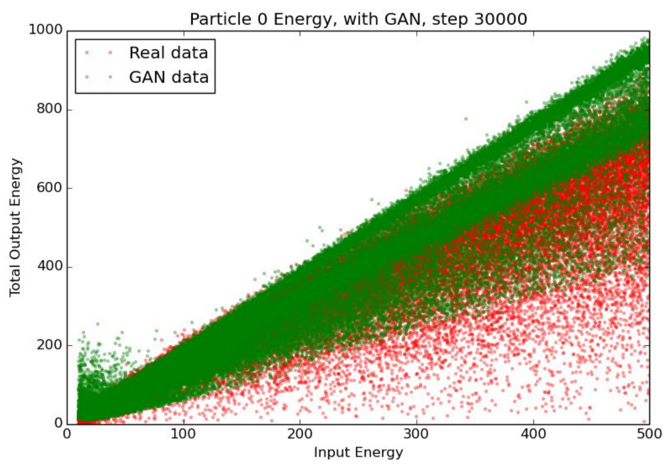

Figure 5: Scatterplot of output vs. input energy. The purpose of this plot was to check that both GAN and real data had similar output energy for a given input.

For a quantitative comparison, we create two histograms of these values, one for the total energy ratios of the real data, and another for the generated data. If the GAN model is working well, the two histograms should be very similar. We measure their similarity by finding the difference between their respective means and standard deviations. We will present a few examples of these histograms later on (Appendix, Fig. 6 and Fig. 7), and our experiments will present the difference between the mean and standard deviation of the different models we tried.

V. Experiments

We conducted numerous experiments, focusing primarily on exploring different model architectures and training methods. We built our models using Tensorflow, and trained them on NYU's GPU-enabled High-Performance Computing clusters.

A. Model Architecture

We used an architecture borrowed from Mehdi Mirza's code (with permission) as a starting point. It was used for generating MNIST digits in the seminal paper on conditional GANs. From the initial MNIST template, we tuned parameters by performing a sort of grid search, varying some elements while keeping others constant. See appendix table 1 for list of parameters we tuned (best result for each in bold). In addition to recording loss, the models also record discriminator classification accuracy. During training, the following items get recorded periodically (typically every 10,000 training steps):

- The latest model
- Discriminator and generator loss
- Discriminator accuracy on real vs gan data
- Number of training steps completed
- Samples and plots of output for visual inspection
- Classifier results

B. Training changes

1) Asymmetric Update Steps

Initial experiments showed that the generator never had a chance to "lift-off". The discriminator error would decrease monotonically towards perfect accuracy without any oscillation due to improved samples coming from the generator. This degenerate equilibrium was avoided by updating the generator parameters k-times for every discriminator update, as suggested in Goodfellow et al. Hyperparameter search led to setting k equal to four.

2) Minibatch Discrimination

A repeating problem we encountered was that the generator would fixate on a small subset of pixels and vary those slightly, fooling the discriminator but producing very homogeneous results. Fortunately we found a blog post which said this 'collapsing' of the GAN is a common problem, and offered a solution from (Salimans et al., 2016.). The solution was to add a layer to the discriminator that tells it how much variability there is in a given minibatch. This prevents the generator from getting away with producing the same image every time, because the discriminator can see the unrealistic lack of variability across the generated data within a minibatch. This methodology was named "minibatch discrimination". To achieve better convergence, this required a smaller batch size (16 rather than 512).

Before minibatch discrimination, the distribution of generated energy ratios did not match the real data (see Appendix Fig. 6).

3) Feature Matching

Minibatch discrimination resulted in better samples, but the distribution of the energy input-output ratio produced by the generator still varied greatly from the real-data distribution. Rather than explicitly optimizing for this ratio by including it in the generator loss function, we include it in the discriminator model through a method called "feature matching" (Salimans et al., 2016). This approach suggests that you can encourage the generator to produce the feature you are interested in by explicitly giving that feature to the discriminator. In this context, we want the generator to produce samples that match the the total energy ratio statistic of a corresponding real sample. To encourage this through feature matching, we simply calculate this value and append it to the last (pre-softmax) layer of the discriminator.

4) Data Averaging

The calorimeter data is extremely sparse and heavily skewed, with the largest 15 values in the 2D slice (out of 400) accounting for 42% of the total energy in a typical experiment. This made the generators task particularly challenging and led us to average the data into evenly sized (particle-type, input momentum) bins. Each average observations was generated by collapsing 10 experiments into a single average-experiment. This dramatically improved the quality of the generated samples (visually) and significantly improved validation results

VI. Conclusions

The best model used 4 fully-connected layers of 1200 nodes each in the generator. It used 2 fully-connected layers of 1200 nodes each in the discriminator; with dropout, minibatch discriminator, and feature matching in the discriminator as well. It used a learning rate of 0.0002 with the Adam optimizer and batch size of 8.

The best model's output looked like real samples, but with realistic variation. Encouragingly, pixel intensity (measured energy) increases for particles with higher momentum. See Appendix, Fig. 7 for the energy ratio distributions generated by the best model, and Appendix, Fig. 8 for the best model's generated samples.

VII. Future Work

We learned a great deal about how what kinds of model parameters and training methods are useful for simulating ECAL readings, but there is much more work left to do until the GAN can generate useful simulations of ECAL readings.

A. Training Convergence

The biggest problem we left unsolved was that none of our models ever converged to a stable output. The training loss for both the generator and the discriminator never reliably leveled off to a single value after several hours of training. As a result, there was no clear way to determine when training was "complete".

Lack of convergence is a common problem training GANs, and has no theoretical solution (Salimans et al., 2016). When attempting to generate realistic images, convergence is not strictly needed, since the output can be evaluated qualitatively. Doing qualitative analysis of generated calorimeter readings is not as clear cut, so a convergence is required to be sure that a given model has reached its potential. There are a number of different techniques that have been proposed to deal with this that we have not yet tried, such as “historical averaging”, “one-sided label smoothing”, and “virtual batch normalization” (Salimans et al., 2016).

B. Three-Dimensional ECAL Simulation

Our model only attempted to generate simulations of a two dimensional slice of ECAL readings. Although this was a good starting point, our model will not be useful to other researchers until it can generate the full three-dimensional simulation of an ECAL reading.

We believe that it is likely that a three-dimensional model can be created through a few small architectural modifications to our two-dimensional model. We believe this because most of the improvements (and pitfalls) we discovered in constructing the two-dimensional model came from modifying how the model was trained, and how the data was represented. The details of the model architecture had relatively little effect on the quality of its output, leading us to think that may be true of GANs generally.

C. Additional Evaluation Methods

There are several ways one could evaluate the quality of the generated calorimeter readings, in addition to our energy ratio histograms. We attempted another technique as well: building a classifier that could identify the type of particle (photon or neutral pion) associated with a given example of real calorimeter data. If the classifier is able to reliably identify the particle type associated with real data, and if our GAN is working well, then the classifier should be able to reliably identify the particle types used as input to the GAN’s simulations.

Towards this end, we trained several different types of models: logistic regression, decision tree, support vector machine, random forest, and multilayer perceptron, but none of them were able to reliably classifier the calorimeter readings by their particle type. It is possible that this may not have worked because we limited the classifier to a two-dimensional slice of the calorimeter data, because that is what our GAN was designed to generate. There are reasons from physics to believe the full three-dimensional tensor of calorimeter data could have been more easily classified. If, in the future, we are able to create a generative model that can simulate three-dimensional calorimeter readings, we will retry this approach to evaluate its creations.

References

[1] Bendavid, Josh, et al. “Imaging Calorimeter Data for Machine Learning Applications in HEP.” 2016.

[2] Goodfellow, Ian, et al. "Generative adversarial nets." Advances in Neural Information Processing Systems. 2014.

[3] Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." arXiv preprint arXiv:1411.1784 (2014).

[4] Denton, Emily L., Soumith Chintala, and Rob Fergus. "Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks." Advances in neural information processing systems. 2015.

[5] Salimans, Tim, et al. "Improved techniques for training gans." Advances in Neural Information Processing Systems. 2016.

[6] Chen, Xi, et al. "Infogan: Interpretable representation learning by information maximizing generative adversarial nets." Advances in Neural Information Processing Systems. 2016.

[7] Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv preprint arXiv:1511.06434 (2015).

[8] Glover, John. "An Introduction to Generative Adversarial Networks (with Code in TensorFlow) - AYLIEN." AYLIEN. N.p., 26 Aug. 2016. Web. 12 Dec. 2016.

[9] McGuinness, Kevin “Generative Models and Adversarial Training” http://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative -models-and-adversarial-training -upc-2016

APPENDIX

Parameter	**Values**
Number of layers in networks	2, **4**
Nodes per layer in generator network	600, **1200**
Number of generator training steps per discriminator training step	1,3,**4**
Activation function	Sigmoid, **ReLU**
Learning Rate	0.1, 0.01, 0.001, **0.0002**
Batch Size	8, **16**, 64, 256, 512

Table 1. Tuned Parameters (Best Value in Bold)

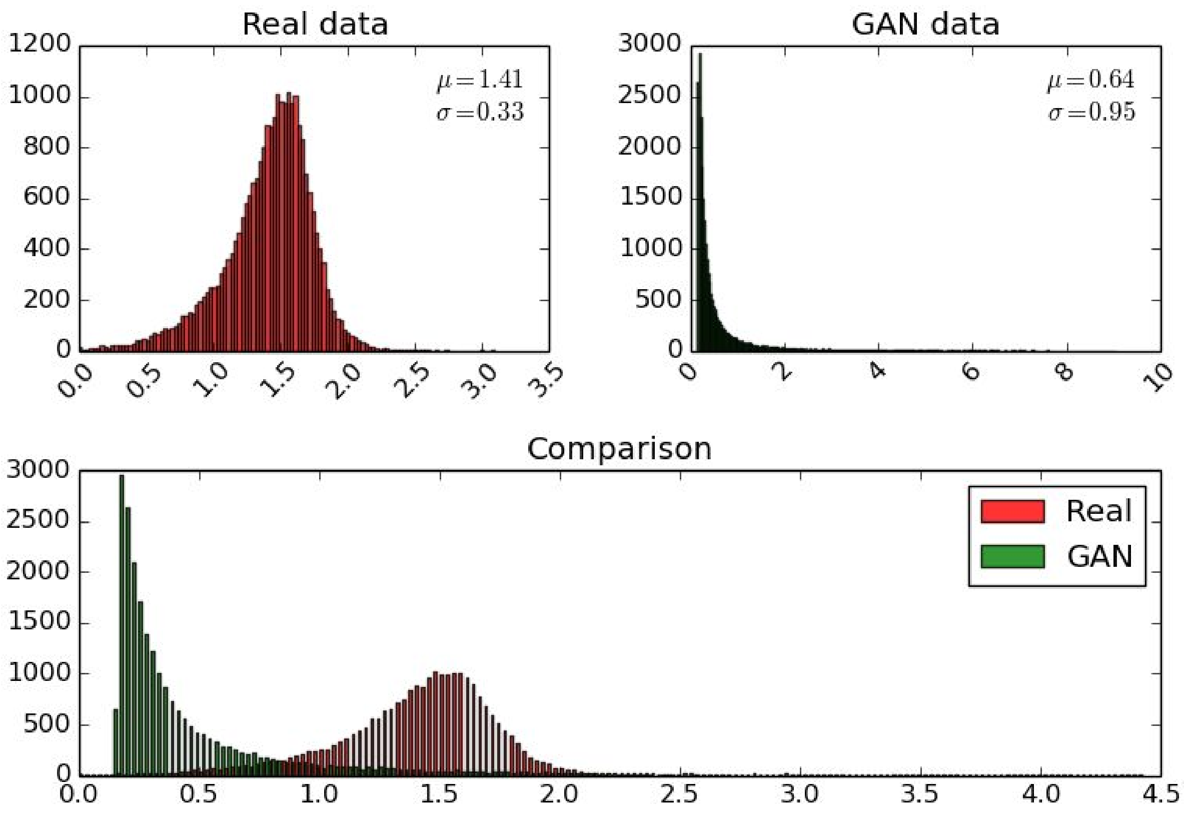

Figure 6: Ratio of measured energy to input energy of real and generated events. In this early experiment, the distribution of generated ratios did not match the real data's distribution.

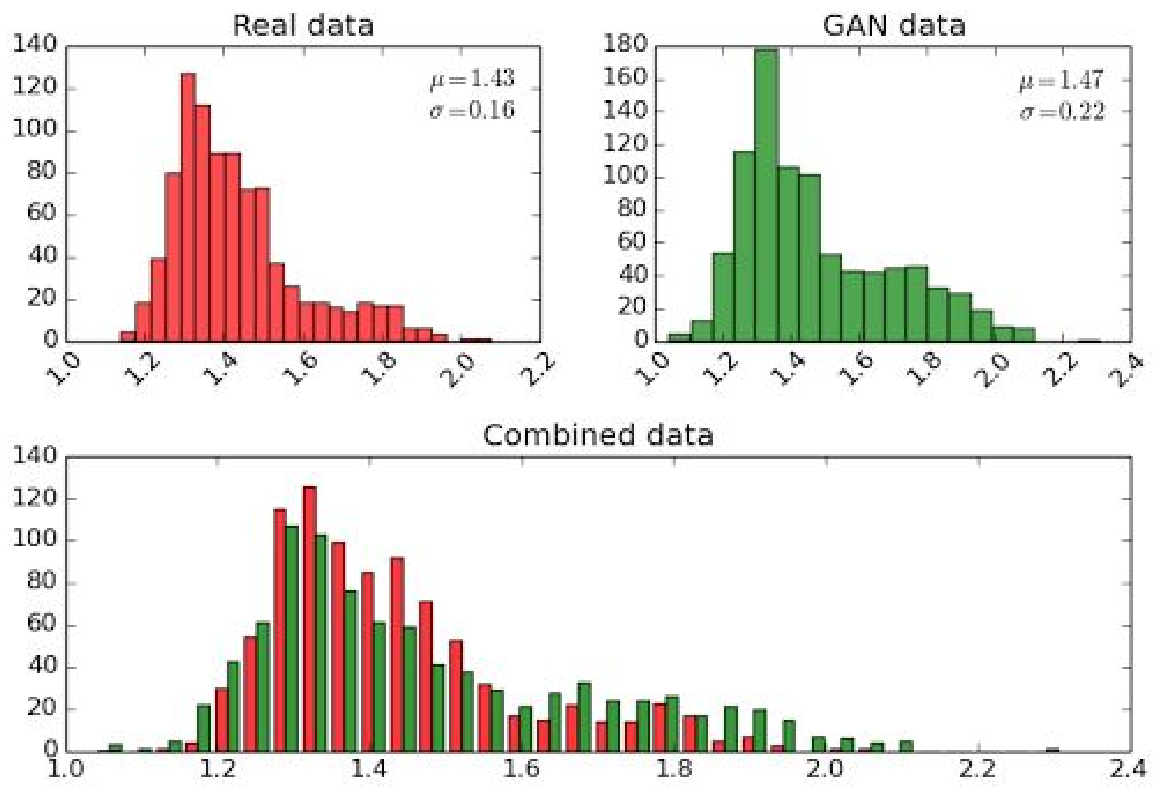

Fig. 7: After minibatch discrimination, feature matching and data averaging, distributions matched much more closely

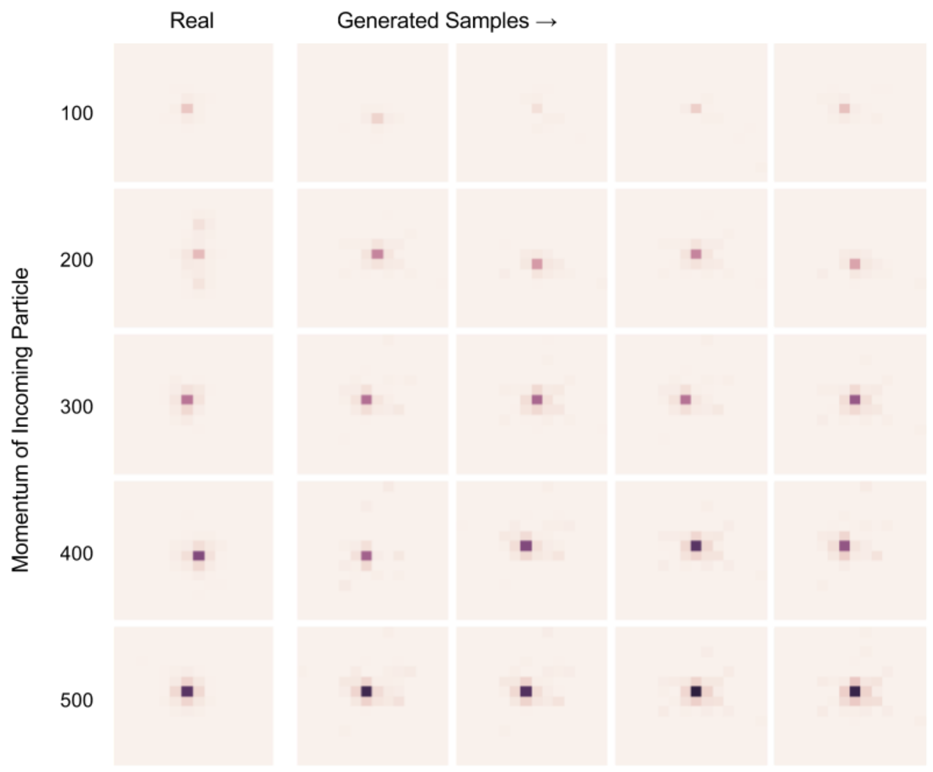

Fig. 8: Real and generated samples from our best model. Appropriately, pixel intensity increased with increasing momentum, but there was still variability in generated images.