Smart mufflers: machine learning for noise quantification and removal

Phil Rinaldi, Department of Physics and Engineering Physics, Tulane University, New Orleans, LA 70118

Phillip Maffettone, National Synchrotron Light Source II, Brookhaven National Laboratory, Upton, NY 11973

Abstract

The National Synchrotron Light Source II (NSLS-II) at Brookhaven National Laboratory allows scientists from around the world to conduct extremely precise characterization experiments which drive the discovery of novel materials. A common procedure used to characterize such materials is X-ray diffraction (XRD), which illuminates atomic structure. Despite the facility's advanced instrumentation and automated procedures, diffraction patterns are inherently vulnerable to noise. In the presence of noise, smaller diffraction peaks are obfuscated, preventing researchers from ascertaining the finer details of a material's crystalline structure. If the noise in a sample can be quantified on-the-fly, it is possible to leverage adaptive measurement strategies, thus obtaining more accurate results. However, it is also necessary to reduce the impact of noise in data that has already been collected. These challenges are particularly suitable for artificial intelligence (AI). Herein, we adapt a convolutional autoencoder to the task of removing noise from accurately simulated diffraction data, resulting in a preliminary analysis tool for beamline scientists.

1. Introduction

Experimental data of all kinds is inherently vulnerable to noise. With regard to X-ray diffraction (XRD) data, aberrations and artifacts can obscure smaller diffraction peaks or even be mistaken for them. This hampers researchers' ability to determine the more subtle aspects of a material's atomic structure and renders analysis time consuming. With recent breakthroughs in autonomous experimentation at the National Synchrotron Light Source II (NSLS-II)[1] , such diffraction data is being collected at an increasingly rapid pace, while the rate data analysis lags far behind. It is thus imperative to develop tools that can help streamline the analysis process for beamline scientists.

To address this need, we trained a variational autoencoder (VAE) to remove noise from a synthetic diffraction dataset. A VAE is a self-supervised machine learning method consisting of two seperate neural networks: an encoder and a decoder. The encoder takes in a complicated input and reduces it to a low-dimensional representation in a latent space; the decoder reconstructs the original input from a sample of that space. A sampling layer serves as the glue between these two networks. It draws a representative sample from the probabilistic latent representation and passes it to the decoder. The latent space may be defined by any number of dimensions, but for visualization purposes is usually kept to two dimensions.

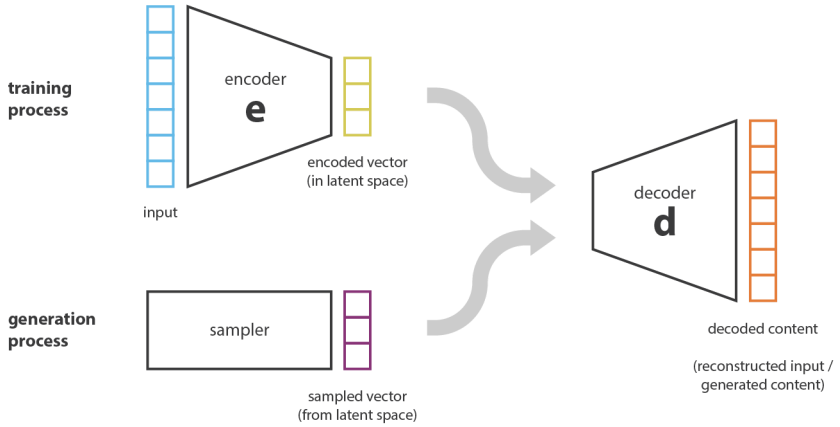

Figure 1: *Schematic of a variational autoencoder[2]. The three main components include the encoder, sampler, and decoder. The encoder reduces the input data into a low-dimensional space, the sampler draws a representative vector from this space, and the decoder takes the sample and reproduces the original input.*

2. Methods

2.1 VAE Architecture Comparison

We implemented the encoder and decoder networks of the variational autoencoder in two different ways: with dense, fully-connected layers and with convolutional layers. Dense VAEs have been employed for a wide variety of tasks, including the classification of materials by phase[3]. Convolutional (CNN) encoder and decoder networks typically find their applications in image processing. After building these two networks into the XCA[4] machine learning package using Python 3.8, Tensorflow 2.2, and the Keras module, we decided to compare the different architectures' ability to remove noise from simulated $BaTiO_3$ diffraction data. Noise was added synthetically as normally distributed fluctuations about each data point.

We passed the noisy data as input to the VAEs and trained them against the corresponding smooth patterns. Training was accomplished by iteratively minimizing the mean-squared error (MSE) between the noisy inputs and (hopefully) smooth reconstructions. An additional loss, the Kullback-Leibler (KL) divergence, was optimized to enforce that the latent distributions are smooth and approximately normal. However, as we were only concerned with the quality of the reconstructions, we diminished this latter term by a factor of 1028 in the calculation of the total loss.

As illustrated on the plot in Figure 2, the dense VAE did not learn much at all -- it could not even reproduce the shape of the input pattern. The convolutional network's reconstruction

resembled the input data, but did not capture the more subtle curvature. Neither implementation was able to suitably remove noise. Clearly, the convolutional VAE was better suited for the task, indicating that it is better to treat the diffraction data as a one-dimensional image. Choosing this architecture to move forward, we focused on a simpler, analogous dataset in order to test the model's denoising capabilities.

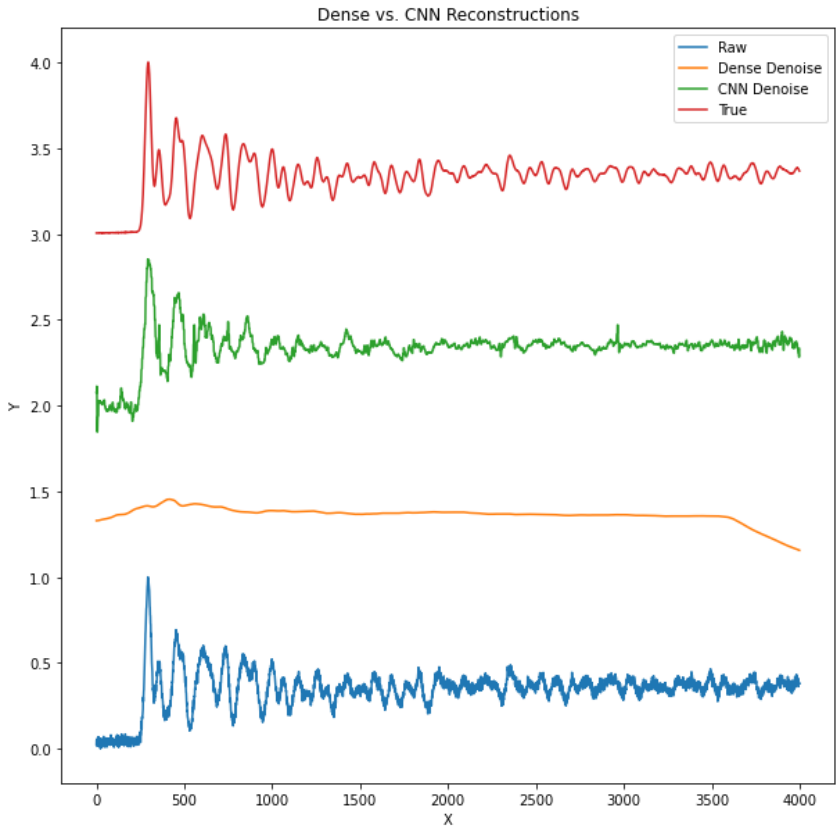

Figure 2: *Comparison of the dense and convolutional VAEs' reconstructions. The convolutional VAE reproduces the general shape of the pattern much better than the dense implementation, but neither network learns to remove noise.*

2.2 Gaussian Dataset

To refine the convolutional VAE, we synthesized datasets in which each sample consisted of just two gaussian peaks. The peaks were generally well-separated, with their centers placed

randomly on either side of the origin. Additionally, to enforce similarity in peak with the $BaTiO_3$ data, we constrained the average variance of the gaussians to correspond to the same average Full Width Half Max (FWHM), in pixels. The FWHM of a peak is defined as the distance (on the x-axis) between its opposing sides at a height corresponding to half of its maximum. This ensured that the hyperparameter tuning performed on this smaller dataset would effectively map to the larger, more complicated $BaTiO_3$ dataset.

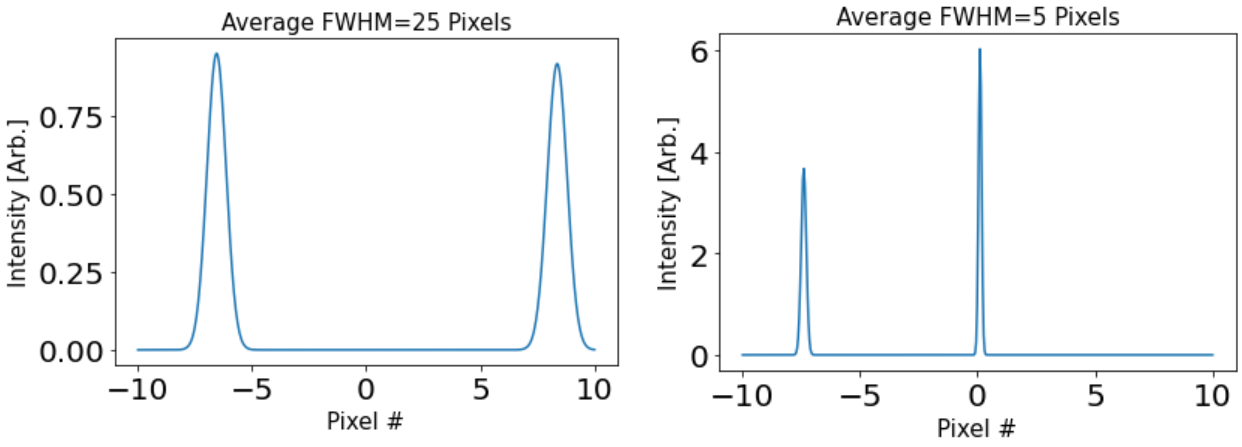

Figure 3: *Samples from datasets with different average FWHMs. The FWHM of the data is important to consider in the process of configuring hyperparameters, as the kernel/filter size is sensitive to the peaks' spread.*

2.3 Hyperparameter Tuning

We began the process of tuning the model's parameters by generating a small gaussian dataset with a FWHM similar to that of experimental $BaTiO_3$ data (which was determined by careful inspection to be $\simeq 5$). We then worked to understand how the model's performance related to its kernel size. The kernel, or filter, is what does most of the learning in a convolutional network -- it identifies important features in the data by sliding over the input pixels and adjusting its weights.

We trained the convolutional VAE on a range of kernel sizes, recorded the final value of the MSE loss, and calculated the Structural Similarity Index Measure (SSIM). The MSE and SSIM both provided measures of the model's performance. The MSE takes values in the range $[0, \infty]$, with 0 indicating that there is no difference between the input and reconstruction. The SSIM is a metric which gauges the similarity between two images, giving an output in the range [-1, 1]. Images that are extremely dissimilar have an SSIM close to -1, while those that are identical have an SSIM of 1. Both metrics were plotted against kernel size, and the results are presented in Figure 4.

Our initial hypothesis (resulting from some preliminary training) was that kernel sizes less than or equal to the FWHM of the data would produce the best results, i.e. minimize the MSE and maximize the SSIM. Thus we expected to see the MSE increase (and the SSIM decrease) for kernel sizes greater than 5. To our surprise, the maximum kernel size in the test set (16) gave the most accurate reconstructions.

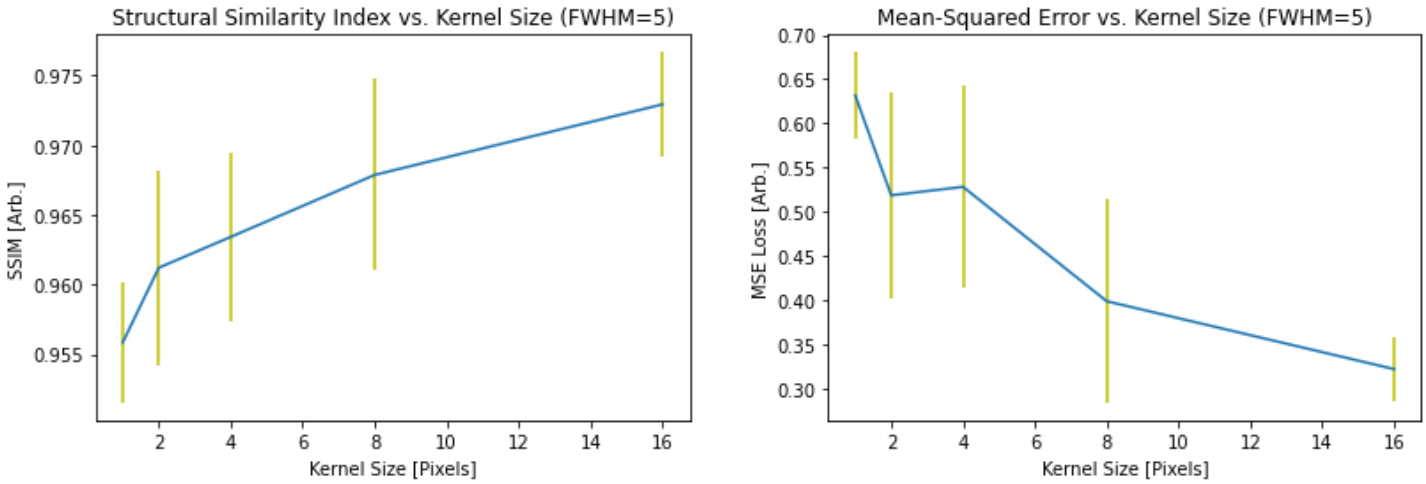

Figure 4: *Plots of the SSIM (left) and MSE (right) versus kernel size. As expected, an increase in the SSIM (an accuracy metric) corresponds to a decrease in the MSE (a loss metric). Each data point represents the average over five trials, with standard deviations of the samples visualized*

by error bars. From these plots it is clear that a kernel size of 16 is best for the model, as it minimizes the MSE and maximizes the SSIM.

With the kernel size configured, we had obtained the final model architecture. The encoder network consists of three 1D convolutional layers with channel dimensions 32, 32 and 16. A kernel size of 16 is utilized for all convolutional layers. The final, dense layer comprises 100 neurons, corresponding to the dimensionality of the latent space. The decoder network is essentially the inverse of the encoder, with three 1D convolutional transpose layers (channel dimensions 16, 32 and 32) followed by a dense layer. The dense layer's dimensionality matches that of the input. Like the encoder, all kernel sizes in the decoder's convolutional layers are of size 16. An outline of the full model architecture (prepared for training on the simple dataset) as returned by Keras.summary() is given in Figure 5.

Model: "CNN_encoder"

Layer (type)	Output Shape	Param #
X (InputLayer)	[(None, 500, 1)]	0
conv_0 (Conv1D)	(None, 500, 32)	544
average_pooling1d (AveragePooli	(None, 500, 32)	0
conv_1 (Conv1D)	(None, 500, 32)	16416
average_pooling1d_1 (AveragePoo	(None, 500, 32)	0
conv_2 (Conv1D)	(None, 500, 16)	8208
average_pooling1d_2 (AveragePoo	(None, 500, 16)	0
flatten (Flatten)	(None, 8000)	0
dropout (Dropout)	(None, 8000)	0
z_mean (Dense)	(None, 100)	800100
z_log_sig (Dense)	(None, 100)	800100

Model: "CNN_decoder"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 100)]	0
dense (Dense)	(None, 8000)	808000
reshape (Reshape)	(None, 500, 16)	0
conv_transpose0 (Conv1DTrans	(None, 500, 16)	4112
conv_transpose1 (Conv1DTrans	(None, 500, 32)	8224
conv_transpose2 (Conv1DTrans	(None, 500, 32)	16416
flatten_1 (Flatten)	(None, 16000)	0
dense_out (Dense)	(None, 500)	8000500
reshape_1 (Reshape)	(None, 500, 1)	0

Figure 5: *Final model architecture (encoder and decoder) as returned by Keras.summary(). For each layer the type, output shape, and parameter count are given.*

3. Results and Discussion

We trained and tested the model on both the gaussian dataset and the simulated $BaTiO_3$ diffraction patterns (testing here refers to the process of passing noisy data through the trained denoiser); the results are depicted in Figure 6. In both plots, the reconstructions match the true (smooth) pattern quite well. Inspecting the residuals, it is clear that most of the error lies in the reproduction of the peaks, rather than in the removal of the noise between them. This is not cause for concern, as it's clear that the important features of the peaks (i.e. their height and width) are retained. All of the static between the peaks is shaved off nicely.

Now, at first glance, the reconstruction for $BaTiO_3$ is impressively accurate. Among such intense (frankly, obnoxious) noise, the model was able to resolve even the smallest peaks. But since the model is only passed the noisy data as input, the fidelity of the reconstruction raises a bit of suspicion -- how could the autoencoder possibly recognize such tiny features of the true pattern given an input whose noise far overshadows them? We are concerned that the model might be learning the actual diffraction pattern of Barium Titanate, rather than how to remove noise from it in a general sense. Thus, we deem the more substantial outcome here to be the success with which the model cleaned up the gaussian data.

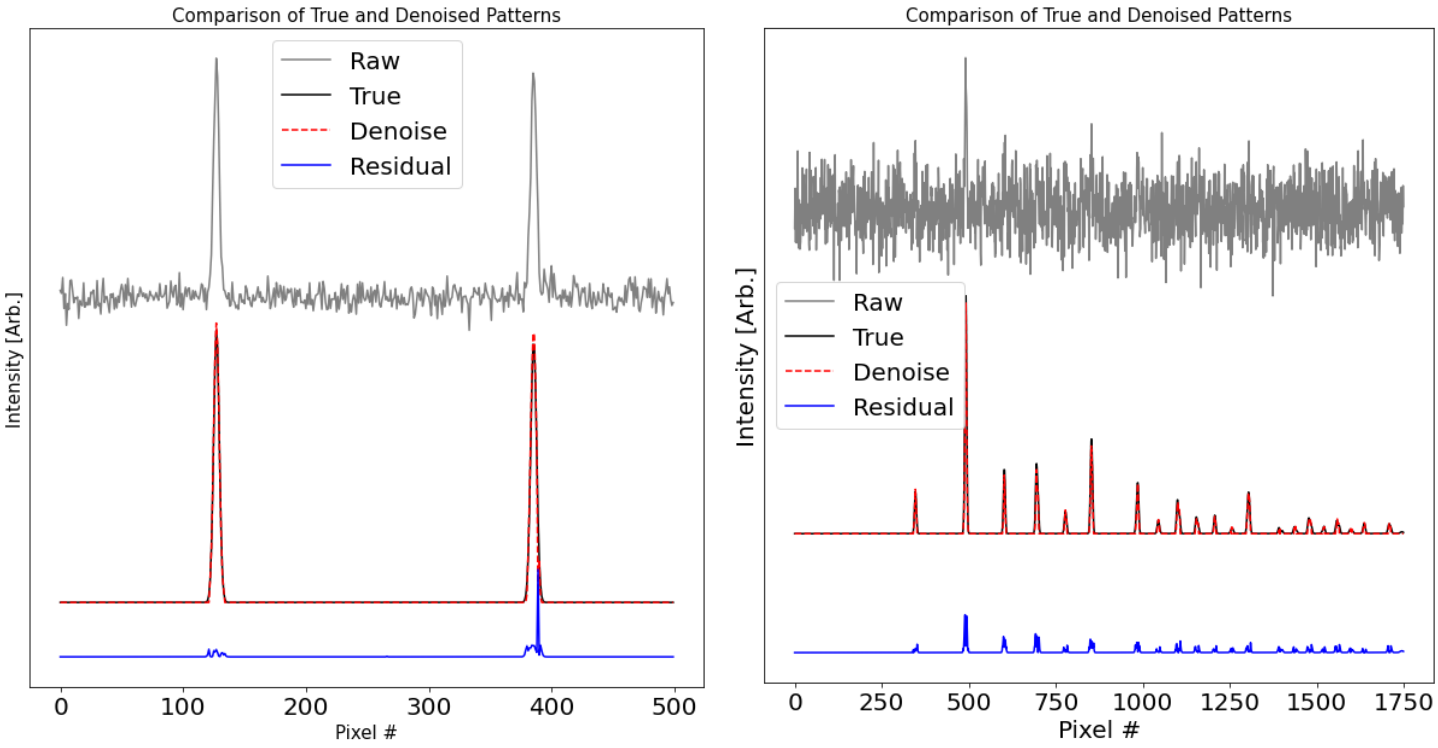

Figure 6: *Plots comparing the convolutional VAE's reconstruction (red dashed line) to the true patterns (black line) given the noisy patterns (grey line) as input. Residuals are plotted for easier visualization of discrepancies. On the left is the comparison for the gaussian dataset; on the right, the comparison for simulated* $BaTiO_3$.

4. Conclusion

We have successfully adapted the architecture of a convolutional VAE to the problem of removing noise from synthetic diffraction patterns. Though our results for $BaTiO_3$ look promising, we hesitate to declare true success in removing noise from the data as the model may simply be learning the material's crystal structure. Moving forward, we hope to train the autoencoder on a more diverse dataset that includes a variety of materials so as to ensure its robustness. Additionally, the model will need to be tested on experimental data, as it may eventually be useful for on-the-fly quantification of noise at the beamline.

5. Code Availability

The code used to generate the figures, synthesize the gaussian dataset, and build/train/test the convolutional autoencoder can be found in a suite of notebooks located at https://github.com/prinaldi3/Denoising.

6. References

[1]Maffettone, Phillip & Lynch, Joshua & Caswell, Thomas & Cook, Clara & Campbell, Stuart & Olds, Daniel. (2021). Gaming the beamlines—employing reinforcement learning to maximize scientific outcomes at large-scale user facilities. Machine Learning: Science and Technology. 2. 025025. 10.1088/2632-2153/abc9fc.

[2]Rocca, Joseph. "Understanding Variational Autoencoders (Vaes)." *Medium*, Towards Data Science, 21 Mar. 2021, <towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73.>

[3]Banko, L., Maffettone, P.M., Naujoks, D. *et al.* Deep learning for visualization and novelty detection in large X-ray diffraction datasets. *npj Comput Mater* 7, 104 (2021). https://doi.org/10.1038/s41524-021-00575-9

[4]https://github.com/maffettone/xca

7. Acknowledgements

This project was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internships Program (SULI). I would like to specifically thank Phillip

Maffettone for his generous support and guidance throughout this project, as well as Daniel Olds, Andi Barbour, and Thomas Caswell for providing me with this internship opportunity.