Project Report - S8
Group 7
Material Image Generation using Deep Learning

Submit by :

MOUWAD Omar
DUPUIS Telio
NADAL Thibault
CANTY Enzo
LEGRIS Maya
JACQUES Nicolas

Supervised by :

BERTHOUMIEU Yannick
COUTINHO Pedro

15 May 2023

Contents

I. Introduction

The artificial intelligence industry has experienced explosive growth in recent years, with applications ranging from natural language processing to computer vision. One area where AI is making significant contributions is in the field of materials science. Scientists are now using AI algorithms to generate virtual materials, replacing the need for costly and time-consuming physical experimentation. The goal is to create materials with specific physical properties for use in fields such as energy production, electronics, and healthcare.

This project aims to develop a simulator for virtual materials using generative deep learning approaches. The focus will be on generating 2D and 3D images of materials with microstructures and textures that can be used to characterize their physical properties. It will explore methods such as Generative Adversarial Networks (GANs) and Variational AutoEncoders (VAEs) to generate these images.

The project will begin with an introduction to the concept of virtual materials and the importance of generating images that accurately capture their physical properties. It will then delve into the technical challenges of generating 3D images, in order to finally controlling the properties of virtual materials.

II. Generation of 2D images with a GAN

A. Presentation of the GAN

1. Architecture and principle of a GAN

Generative Adversarial Networks are machine learning algorithms that are used to generate data, in this case images, from a training database. The goal of GANs is to identify the features of the data in the training database in order to generate new images respecting these features. It works with two adversarial part : the generator and the discriminator. The objective of the generator is to generate images to mislead the discriminator while the discriminator try to predict if an image is real or generated.

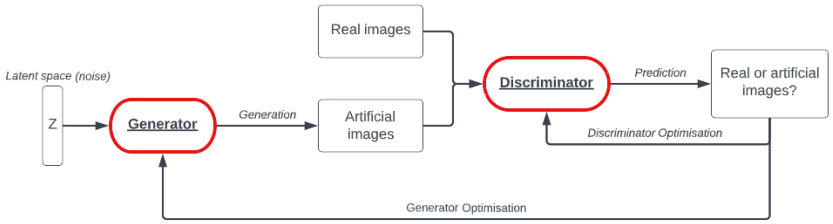

Figure 1: Architecture of the GAN

2. Cost Function

Here are the notations that one using to define the cost function of GANs.

For the Discriminator, x represent an image (generated or real) and $D(x)$ is the output of the discriminator and represent the probability that the x image is real. For the generator, z is the latent space vector and all the elements of this vector follow a normal distribution and $G(z)$ is the output of the Generator: a generated image. So, $D(G(z))$ is the probability that the output of the Generator is a real image.

Therefore, the cost function of the GAN is :

$$\min_G \max_D \mathcal{L}GAN(G, D) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p\boldsymbol{z}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))], \tag{1}$$

And it is upon this cost function that the optimization of the GAN takes place.

B. Presentation of the implementation of the GAN

Two main codes served as the foundation for implementing the GAN. The first one was derived from the project "Microstructural Materials Design via Deep Adversarial Learning Methodology" [5] for selecting appropriate parameters (number of epoch, size of latent space, convolutional layers, batch size, and dimensions) tailored to microstructures. The second code originates from Erik Lindernoren's GitHub repository [6] and was predominantly used in the implementation, as it incorporates more recent and efficient Python libraries. This decision was made due to the limitations of the original "Microstructural Materials Design via Deep Adversarial Learning Methodology" code in its current state.

The implemented GAN was applied to images from different databases (mainly NMC database), in a smaller format (64*64) for the test phase (in order to try to find the most suitable parameters and to study the influence of the choice of normalisation). Then in a larger format (longer calculation time) once the optimal parameters have been determined, to have more detailed results.

C. Techniques to improve the GAN

1. Spectral normalization on Discriminator

One of the issues related to the discriminator is that it sometimes struggles to differentiate between generated images and real images, hindering the improvement of the generator. Conversely, if the generated and real images are completely different, it is possible for a discriminator to perfectly distinguish between image classes. This makes the learning of the generator challenging, as its objective is to generate realistic examples that approximate the real distribution. Spectral normalization (SN) is a method that allows to solve these problems [7]. Spectral normalization is applied to the weights of the linear layers. To spectrally normalize these weight matrices, the first step is computing the eigenvalues of the weight matrix. Then, the calculation of the spectral norm, which is simply the eigenvalue with the largest absolute value, is done. Finally, each weight in the matrix is divided by the previously calculated spectral norm.

2. Batch and Instance Normalization on Generator

2..1 Instance Normalization

In order to understand how instance normalization layer functions, consider x as a batch of N images with C channels of size H*W, taken as input to the IN layer. The channel of a training sample (an image in the case of this project) of the 4-dimensional tensor x is considered (so both N and C fixed).

Therefore, both spatial mean μ_{ni} and variance σ^2_{ni} can be estimated as follows:

$$\mu_{ni} = \frac{1}{HW} \sum_{l=1}^{W} \sum_{m=1}^{H} x_{nijk} \tag{2}$$

$$\sigma^2_{ni} = \frac{1}{HW} \sum_{l=1}^{W} \sum_{m=1}^{H} (x_{nijk} - \mu_{ni})^2 \tag{3}$$

The output of this layer is :

$$y_{nilm} = \frac{x_{nilm} - \mu_{ni}}{\sqrt{\sigma^2_{ni} + \epsilon}} * \gamma + \beta \tag{4}$$

Where ϵ corresponds to a small value added for stability purposes. Gamma and beta are values that can either be initialized or trained by the model. [9]

2..2 Batch normalization

Batch normalization (BN) operates in a similar manner to IN. However, in this case, instead of considering the training sample individually, the variance and mean are estimated over all the images for a fixed channel in

the batch x (thus considering 3 dimensions N-W-H averaging). Finally, the determination of the output follows a similar process as in the case of IN. [8]

2..3 Interest and difference between IN and BN

Firstly, both normalization methods can accelerate training and lead to faster convergence of the neural network. On one hand, the fact that BN operates on an entire mini-batch of samples makes BN dependent on the batch size in order to obtain statistically more accurate mean and variance. Therefore, the batch size needs to be sufficiently large for BN to work properly, otherwise IN would be a more appropriate solution. On the other hand, BN enables higher learning rates and improves the generalization of the neural network.

D. Comparison and choice of the parameters/ normalization

The selected parameters are:

- Latent_dim= Size_Image/2, considering the information presented in both the documentation of the article on GAN applied to microstructures [5] and the results obtained from our experiments on NMC database (tailored to NMC images)
- Batch_Size=64, for identical justifications, in particular, the observed results.
- SN, enabling enhanced convergence of the discriminator

Based on our experiments in those conditions, it has been observed that BN indeed exhibited superior performance compared to IN when batch sizes exceeded 16. Two tests were indeed conducted, utilizing either BN or IN with the parameters initialized as above over 1000 epochs using the NMC-64 database.

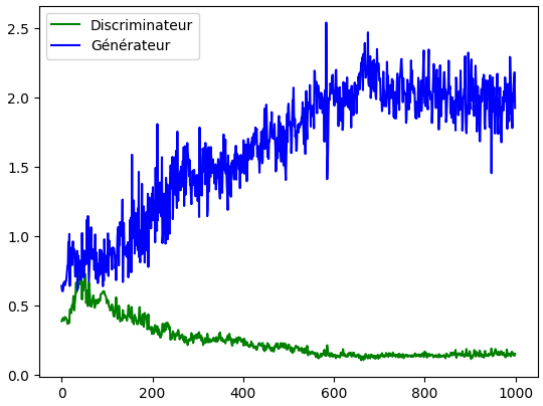

Figure 2: Cost function with SN and BN

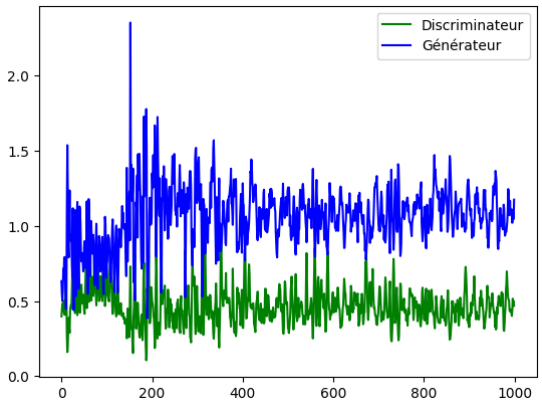

Figure 3: Cost function with SN and IN

The graph above demonstrates a faster and more significant convergence of the discriminator in the case of BN. And discriminator's convergence is a key to the generate great images.

Therefore, the BN was selected over IN for the rest of the project.

E. Final results

The GAN that generated the images in Figure 5 was trained with the parameters given above on NMC-128 database over 10 000 epochs. The generated images closely resemble real images; however, they exhibit a high degree of similarity among themselves, and some generated images appear slightly blurry.

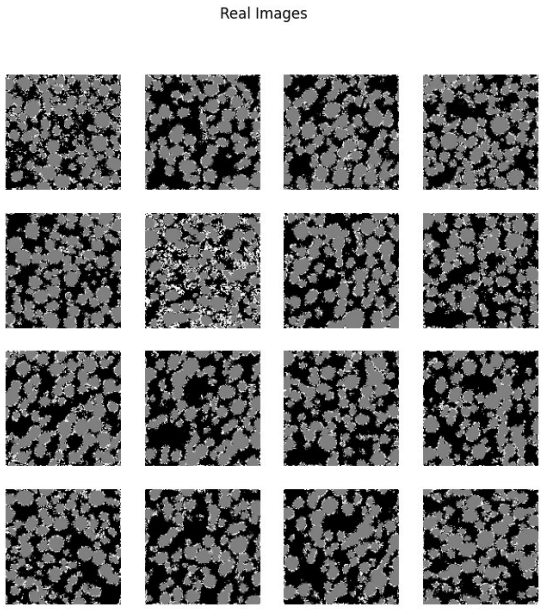

Figure 4: Real data

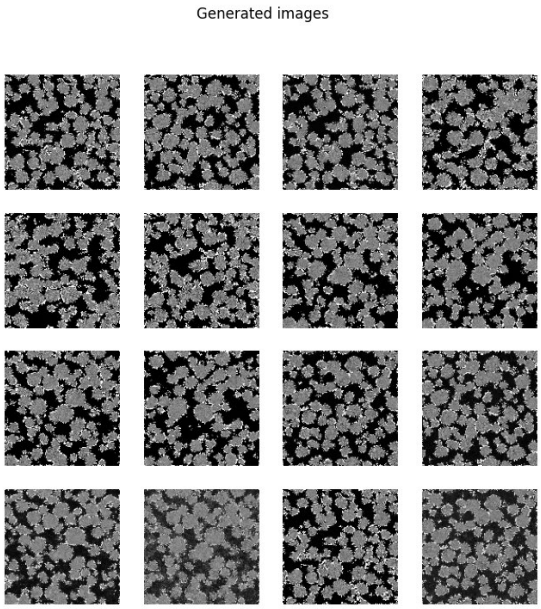

Figure 5: Generated images

III. Generation of 2D images with a VAE

A. Principle of VAE

The principle of a Variational AutoEncoder (VAE) is to learn a compressed representation of input data, also known as latent space, while simultaneously generating new data samples from this learned representation. This feature is critical because it enables more controlled manipulation while generating new samples compared to GANs.

A VAE consists of two main components: an encoder and a decoder neural network. The encoder network maps the input data to a lower-dimensional latent space, while the decoder network maps the latent space back to the input space. During training, the VAE optimizes a loss function that balances the reconstruction error of the decoder and the divergence between the learned latent space distribution and a chosen prior distribution (usually a standard normal distribution).

Let x, an input data (a generated material image) and z, a latent representation whose dimension is less than x. The objective of the encoder is to learn the parameters θ of a probability distribution $q_\theta(z|x)$ to approximate the true distribution q(z|x). The objective of the decoder is the inverse one, finding the parameters ϕ of the probability distribution $p_\phi(x|z)$ which best fit to get the input image x back from the latent space z.

Theoretically, $q_\theta(z|x)$ and $p_\phi(x|z)$ could be found using the Bayes Theorem and probability chain rule, $p(z|x) = \frac{p(x|z)p(z)\cdot p(z)}{\int p(x|z)\cdot p(z)\, dz}$. Without any simplification, the integral is intractable due to its exponential complexity (all the z combinations for an unique input x must be computed). To avoid this, the statistical inference problem is turned into an optimization problem by minimizing the KL divergence:

$$\phi, \theta = argmin_{\phi,\theta} KL(q_\phi(z|x)||p_\theta(z|x)) \tag{5}$$

with $D_{KL}(p(z|x)||q(x|z)) = \int p(z|x) \log\left(\frac{p(z|x)}{q(x|z)}\right)\, dz$

With after some simplifications, the loss function is obtained:

$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_{q_\phi(z|x)}\left[-\log p_\theta(x|z)\right] + KL\left(q(z|x)||p(z)\right) \tag{6}$$

The first term is called the image loss and the second one is the latent loss.

To train the encoder and the decoder, an optimization algorithm like the SGD (Stochastic Gradient Descent) or a variant is used to minimize $\mathcal{L}$ and find the more accurate parameters. The use of a probabilistic model and the incorporation of a regularizing term in the loss function makes the VAE more robust to overfitting and helps to generate new data samples from the learned latent space distribution.

B. VAE implementation

The same was done with GANs, the dataset consists of 384 images of a porous material structure. Regarding the VAE structural choices, the latent chosen space dimension is 64, making a 256 ratio between x and z sizes. In order to obtain this reduction, the neural networks will have 5 layers. After 1000 epochs, here is the evolution of the loss function:

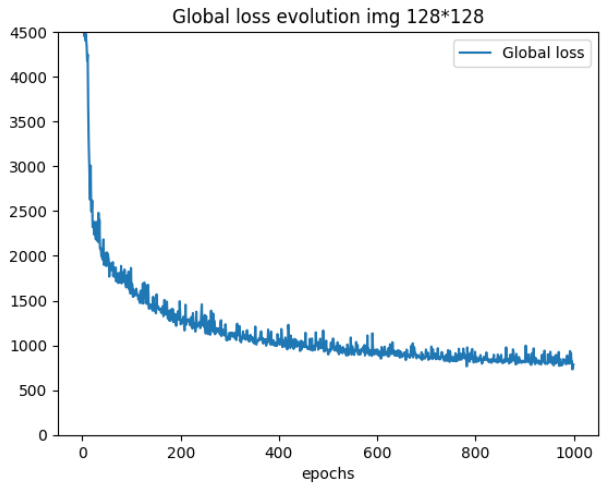

Figure 6: VAE loss evolution along the epochs

The loss function has converged enough to its limits, showing that the training has been successful after 125 minutes of training on an 11Th generation i5 Intel processor.

Figure 7 below is a comparison between the images provided by the dataset and the new ones generated by the trained decoder from Gaussian latent vector.

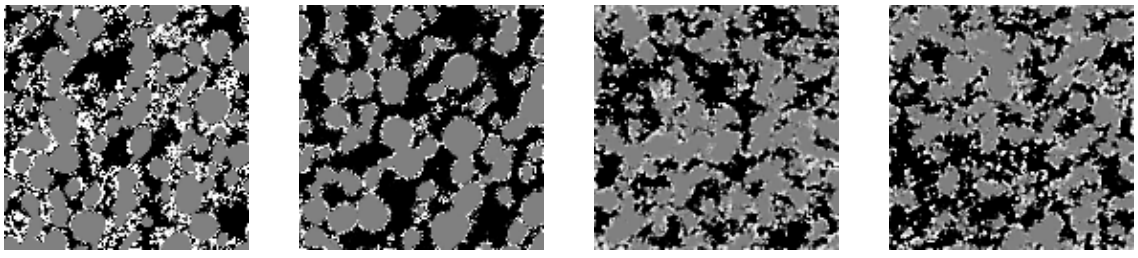

Figure 7: VAE outputs comparison with the database (right: database image / left : output images)

C. VAE performance and limitations

Based on the results of the VAE model, it can be seen that the global structure of the image is preserved, but the images generated are blurry compared to those generated by GANs. This drawback has led to the development

of variations of VAE, such as the VAE/GAN, which combines the benefits of both models. The addition of the $\mathcal{L}_{GAN}$ (such as $\mathcal{L}_{dec} = \mathcal{L}_{image} - \mathcal{L}_{GAN}$) loss function can modify the decoder training and potentially reduce the blurriness and multiple artifacts in the generated images.

Figure 8 shows an implementation of a VAE/GAN model for a previously used dataset. The aim of this study was to explore the capabilities of VAE/GAN in generating realistic and diverse images. The model is trained over 25 epochs. The following presents the progression of the reconstruction loss throughout the training process:

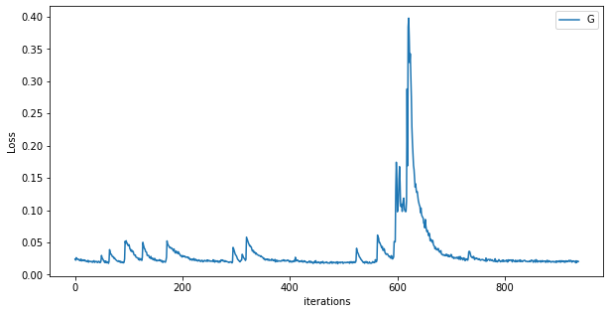

Figure 8: Plot of Reconstruction Loss vs Iterations

Here is a comparison between the images provided by the dataset and the new ones generated:

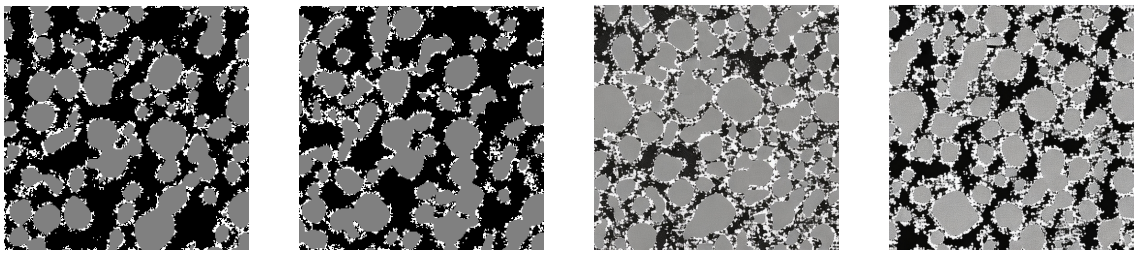

Figure 9: VAE/GAN outputs comparison with the database (right: database image / left : output images)

IV. Generation of 3D volumes with SliceGAN

A. Presentation of SliceGAN

In this part, a GAN architecture, SliceGAN, which is able to synthesize high fidelity 3D datasets using a single representative 2D image is introduced [3]. Indeed, 3D training data is very challenging to obtain. This architecture implements the concept of uniform information density, which ensures that generated volumes are equally high quality at all points in space, and that arbitrarily large volumes can be generated. Like a GAN, the quality of generated micrographs is shown through a statistical comparison of synthetic and real datasets of a battery electrode in terms of key microstructural metrics. The concept of this specific GAN can be summarised in the following figure:

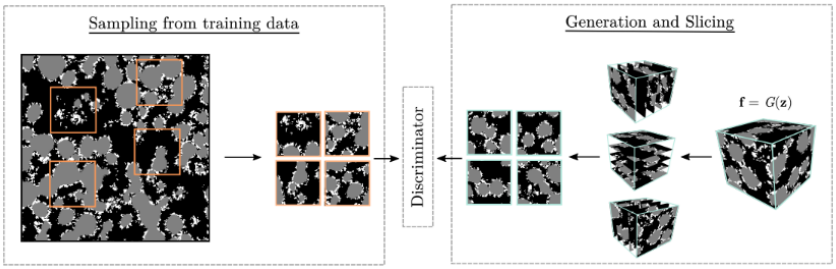

Figure 10: SliceGAN training procedure

The fundamental role of SliceGAN is to resolve the dimensionality incompatibility between a 2D training image and 3D generated volumes. This is achieved by incorporating a slicing step before fake instances from the 3D generator are sent to the 2D discriminator. For a generated cubic volume of edge length l voxels, 3*l 2D images are obtained by taking slices along the x, y and z directions at 1 voxel increments.

During the training of D, for each fake 2D slice, a real 2D image is also sampled and fed to the discriminator, such that it learns equally from real and fake instances, like shown in figure 10 above. To train G, the same slicing procedure is followed; however, a larger batch size (mG) compared to the discriminator (mD) is used. This rebalances the effect of training D on numerous slices per generated sample. Experimentally, mG = 2 mD typically results in the best efficiency. Importantly, the particular architecture of G used in this study means that each slice in any given set of 32 adjacent planes is synthesised through a unique combination of kernel elements. As such, a minimum of 32 slices in each direction must be shown to D in order to ensure each path through the generator is trained. After slicing, a standard Wasserstein loss function was used to encourage stable training.

B. Properties of the generated microstructure

1. Isotropy

Isotropy refers to the uniformity of properties in all orientations. Conversely, anisotropy is used to describe situations where properties systematically vary depending on the direction. It is crucial to note that SliceGAN is specifically designed for isotropic materials, where a single 2D image can be utilized to train slices from the x, y, and z orientations of the material. However, when dealing with anisotropic materials, an extension of the architecture is necessary to accommodate the directional variations in properties. Next figure is the representation of 6 successive layers of the generated isotropic microstructure:

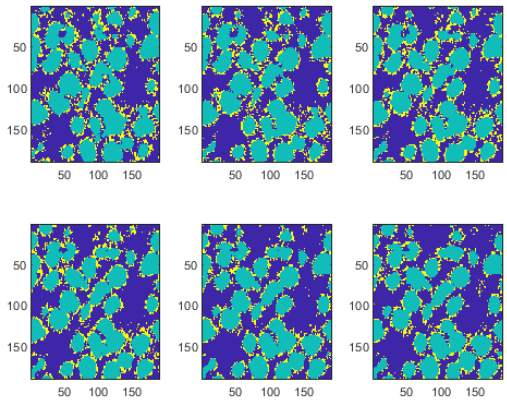

Figure 11: SliceGAN slices

In this particular study, the material under consideration for implementing the method is an NMC (Nickel Manganese Cobalt) battery. This type of battery consists of three distinct phases and exhibits isotropy, meaning it possesses the same properties regardless of the direction. In the provided figure, the consecutive layers appear identical, indicating that SliceGAN simultaneously alters the data in all three dimensions (x, y, and z) during the training process.

2. Composition and density

A key strength of deep convolutional neural nets, like SliceGan, is their ability to efficiently capture the diverse and complex features of an image without any user defined statistical metrics as inputs. Indeed, no parameters about the complex composition of the material is necessary to launch this program. The density of the material is the count of the number of pixels in a 2D image that are on the same phase divided by the total number of pixels, as shown in the first graph of figure 12 below. Meanwhile, the second one shows the densities of each phase.

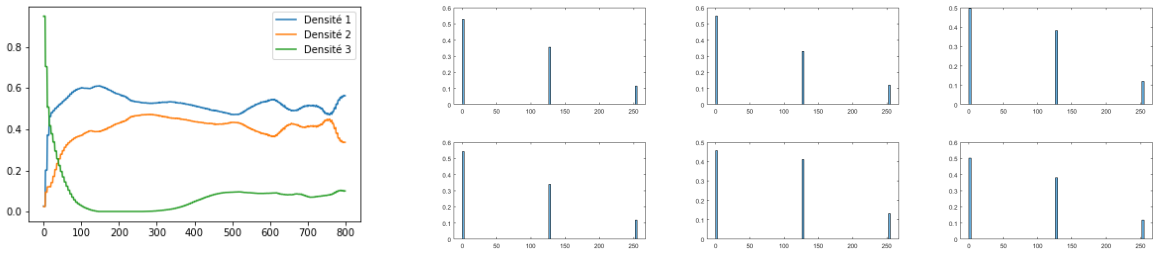

Figure 12: General density and Phase densities in different portions of the volume generated

The sum of each density for one iteration equals one, as expected. Furthermore, in the second graph, the densities of each phase exhibit notable distinctions, indicating that each block of the generated material possesses its unique density. As a result, they bear a resemblance to some extent: the first phase being predominant, followed by the second and then the third.

C. Control over the microstructure's properties

In order to exert control over the properties of the generated material, it becomes imperative to make modifications to the loss function of SliceGAN, as the training process hinges on this function. Consequently, it is necessary to devise a strategy for altering the loss function of both the Generator and Discriminator. This can be accomplished by introducing a density control mechanism, which will serve as input parameters for achieving the desired density. The revised loss function will thus encompass the previous formulation, augmented by the disparity between the density control value and the computed density. Figure 13 shows the different densities of the microstructure along with the number of iteration.

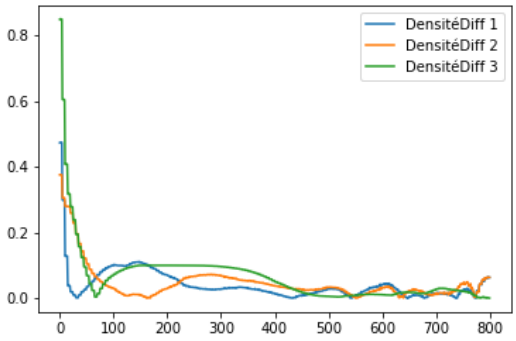

Figure 13: Density differences

In this context, it is worth noting that the disparities in density between each phase exhibit a declining trend throughout the iterations. Consequently, the generated densities are gradually converging towards the targeted values. This indicates an increasing proficiency of the generator as the training progresses.

The outcomes presented in this study showcase the proficiency of SliceGAN in accurately synthesizing 3D n-phase media from 2D micrographs, while also affording control over the desired properties.

V. Conclusion

In conclusion, the goal of this project was to develop a simulator for virtual materials using generative deep learning approaches such as GANs and VAEs. The focus is on generating 2D and 3D images of materials with microstructures and textures that can be used to characterize their physical properties. This report started with an introduction to the concept of virtual materials and the importance of generating images that accurately capture their physical properties. Then, it delved into the technical challenges of generating 3D images and controlling the properties of virtual materials.

The project implemented a GAN to generate 2D images of microstructures. The results obtained are convincing, with some of the generated microstructures closely resembling the real images. This report also discussed techniques to improve GANs, such as spectral normalization and instance normalization. The project then introduced the principle of VAEs, which can learn a compressed representation of input data, also known as a latent space, while simultaneously generating new data samples from this learned representation. Moreover, SliceGAN has proven to be an exceptional tool for synthesizing high-fidelity 3D datasets and generating customizable microstructures with modifiable properties, such as phase densities.

List of Figures

References

[1] A.D.L. Larsen, S.K. Sønderby, H. Larochelle, and O. Winther (2016) : Autoencoding beyond pixels using a learned similarity metric, in arXiv.

[2] A. Sardeshmukh, S. Reddy, P. Gautham, and P. Bhattacharyya (2021) : TextureVAE: Learning Interpretable Representations of Material Microstructures Using Variational Autoencoders, in AAAI Spring Symposium.

[3] S. Kench and S.J. Cooper (2021) : GENERATING 3D STRUCTURES FROM A 2D SLICE WITH GAN-BASED DIMENSIONALITY EXPANSION, in SliceGan

[4] S. Kench and S.J. Cooper (2021) : Codes used for SliceGan, in SliceGan github

[5] Zijiang Yanga, Xiaolin Lib, L. Catherine Brinsond, Alok N. Choudharya, Wei Chenc, Ankit Agrawala Microstructural Materials Design via Deep Adversarial Learning Methodology GAN tailoredfor microstructure

[6] Erik Lindernoren GAN, MNIST, batch normalization. Black refactoring. GAN github

[7] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, Yuichi Yoshida SPECTRAL NORMALIZATION FOR GENERATIVE ADVERSARIAL NETWORKS Spectral normalization

[8] Akbar Karimi Instance vs Batch Normalization Batch and Instance nomalization

[9] Instance nomalization Dmitry Ulyanov, Andrea Vedaldi Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis