Machine Learning Project Report

Jean Goudot, Madeline Casas, Andrea Combette

Contents

Chapter 1

Introduction

This project aims at discovering new structures of neural networks and their applications to cosmology. In particular, we will focus on two types of neural networks: Generative Adversarial Networks (GANs) and Diffusion Deep Probabilistic Models (DDPMs). We will study their theoretical background, their architecture and their applications to cosmology. We will also compare their performances and discuss their advantages and disadvantages.

I Context

1 Generating realistic matter field

Generating density matter fields is crucial for several reasons. First, it allows us to simulate the distribution of matter in the universe, which is fundamental to cosmology. By understanding how matter is distributed, we can gain insights into the structure and evolution of the universe. Second, these simulations can help us test theories of cosmology and astrophysics. For instance, they can be used to predict the distribution of galaxies or the cosmic microwave background radiation. Lastly, generating realistic matter fields can also be a stepping stone towards more complex simulations, such as those involving dark matter or the formation of galaxies.

Previously generating matter fields was done using N-body simulations. However, these simulations are computationally expensive and time-consuming. Therefore, it is necessary to find new methods to generate matter fields. This is where machine learning comes in, because it allows us to generate matter fields in a much faster way.

2 Machine Learning tackling the problem

To tackle this generation tasks, 3 types of neural networks could be used: Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs) and Diffusion Deep Probabilistic Models (DDPMs). These neural networks are all based on Gaussian approximations and are trained on a dataset of matter fields, and then used to generate new matter fields.

GANs

A generative adversarial network (GAN) is a class of machine learning framework and a prominent framework for approaching generative AI. The concept was initially developed by Ian Goodfellow and his colleagues in June 2014.In a GAN, two neural networks contest with each other in the form of a zero-sum game, where one agent's gain is another agent's loss.

Given a training set, this technique learns to generate new data with the same statistics as the training set. For example, a GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics. Though originally proposed as a form of generative model for unsupervised learning, GANs have also proved useful for semi-supervised learning,fully supervised learning, and reinforcement learning.

The core idea of a GAN is based on the "indirect" training through the discriminator, another neural network that can tell how "realistic" the input seems, which itself is also being updated dynamically.This means that the generator is not trained to minimize the distance to a specific image, but rather to fool the discriminator. This enables the model to learn in an unsupervised manner.

GANs are similar to mimicry in evolutionary biology, with an evolutionary arms race between both networks.

DDPM

In machine learning, diffusion models, also known as diffusion probabilistic models or score-based generative models, are a class of generative models. The goal of diffusion models is to learn a diffusion process that generates the probability distribution of a given dataset. It mainly consists of three major components: the forward process, the reverse process, and the sampling procedure. Three examples of generic diffusion modeling frameworks used in computer vision are denoising diffusion probabilistic models, noise conditioned score networks, and stochastic differential equations.

Diffusion models can be applied to a variety of tasks, including image denoising, inpainting, super-resolution, and image generation. For example, in image generation, a neural network is trained to denoise images with added gaussian noise by learning to remove the noise.After the training is complete, it can then be used for image generation by supplying an image composed of random noise for the network to denoise.

Diffusion models have been applied to generate many kinds of real-world data, the most famous of which are text-conditional image generators like DALL-E and Stable Diffusion. More examples are in a later section in the article.

VAE

In machine learning, a variational autoencoder (VAE) is an artificial neural network architecture introduced by Diederik P. Kingma and Max Welling. It is part of the families of probabilistic graphical models and variational Bayesian methods.

Variational autoencoders are often associated with the autoencoder model because of its architectural affinity, but with significant differences in the goal and mathematical formulation. Variational autoencoders are probabilistic generative models that require neural networks as only a part of their overall structure. The neural network components are typically referred to as the encoder and decoder for the first and second component respectively. The first neural network maps the input variable to a latent space that corresponds to the parameters of a variational distribution. In this way, the encoder can produce multiple different samples that all come from the same distribution. The decoder has the opposite function, which is to map from the latent space to the input space, in order to produce or generate data points. Both networks are typically trained together with the usage of the reparameterization trick, although the variance of the noise model can be learned separately.

Although this type of model was initially designed for unsupervised learning, its effectiveness has been proven for semi-supervised learning and supervised learning.

3 Datasets specifications

Quijote Simulations

Quijote provides not only thousands of simulations on different latin-hypercubes, but the a total number of 44,100 N-body simulations, with billion of halos, galaxies, voids and millions of summary statistics such as power spectra, bispectra. . . et, to train machine learning algorithms. In our case we will use the fiducial simulations.Those are simulations with a fiducial cosmology consistent with Planck. They only vary the initial random seed. We will only use $N^3 = 256^3$ cubes with a redshift parameter $z = 0$, to get the filamentary structures of density fields (Gaussian -> filamentary structures). These cubes are avarage over 2 voxels in z-direction, to get a more isotropic dataset. The size of the cubes being $L = 1Gpc/h$. Then each 256 voxels width squares are sliced in 16 voxels width squares. This finally leads to the following nyquist wave vector

$$k_{\text{nyquist}} = \pi \frac{N}{L} \approx 0.8 Mpc/h^{-1}$$

.

Datasets dimensions and transformations

The raw datasets is composed of 50000 density fields of size 64^3 voxels. And each voxel as a value between $[-1, 235.2]$. In order to make the DDPM work we must normalize it between $[-1, 1]$[cite] to be in the range of a Gaussian distribution.

Chapter 2

DDPM neural Network

I Theoretical background

A diffusion model has two main process a forward process and a reverse process. The forward process consists of a series of steps, where each datasets image is transformed into a more noisy image. Given these noisy images, it's not possible directly to learn how to clean them, the backward process consists in learning this cleanning step for each image. This give the following mathematical scheme :

Given a general distribution of the input image : P we want to reach a $\mathcal{N}(0, I)$ distribution given n steps of diffusion. This is done by applying the following transformation to the input image x_0 :

$$
\begin{aligned}
x_1 &= x_0\sqrt{1-\beta_0} + \beta_0\epsilon \\
&\vdots \\
x_n &= x_{n-1}\sqrt{1-\beta_{n-1}} + \beta_{n-1}\epsilon
\end{aligned}
$$

Where $\epsilon \sim \mathcal{N}(0, I)$ and β_i is in the range [0.0001, 0.02]. Given this scheme it's possible to knoew exactly x_t with x_0. Defining : $\alpha_i = 1 - \beta_i$ and $\bar{\alpha}_i = \prod_{j=0}^{n} \alpha_j$ we have :

$$x_i = \sqrt{\bar{\alpha}_i}x_o + \sqrt{1-\bar{\alpha}_i}\epsilon$$

Then the learning process consits of finding the parameters of the backward process $p(x_i ß)$, which has the same functionnal form that the forward process transformation (William Feller, *Diffusion process in one dimension*). The transformation is the following :

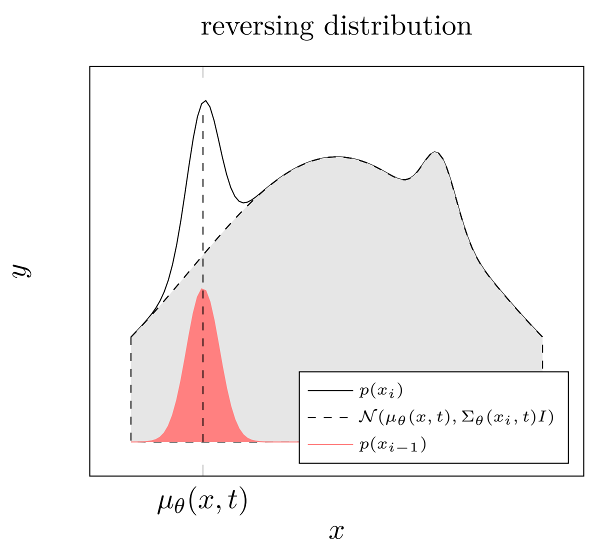
$$\bar{x}_{i-1} = \mu_\theta(x,t) + \sqrt{\Sigma_\theta(x_i,t)}\epsilon$$

. Then we can show that the parameter

$$\mu_\theta(x,t) = \frac{1}{\sqrt{\bar{x}_t}}[x_t - \frac{\beta_t}{\sqrt{(1-\bar{\alpha}_t)}}\epsilon_\theta(x.t)]$$

and expressing $\Sigma_\theta(x_i, t)$ as a combination of α_i and β_i :

$$\Sigma_\theta(x_i, i) = I\frac{1-\bar{\alpha}_{i-1}}{1-\bar{\alpha}_i}\beta_i$$



we can find the parameters of the backward process just by knowing the noise level of the image, this is learned with a U-net neural network composed of a encoder and a decoder. By minimization of the given loss :

$$\mathcal{L} = \mathbb{E}\|\epsilon - \epsilon_\theta(x_i, i)\|^2 \tag{2.1}$$

Then we can generate a new image by applying the backward process to a noise image $\epsilon \sim \mathcal{N}(0, I)$. The network will then estimate the noise level and apply the desired transformation to the image. which results normally in a new image with a realistic distribution of density.

II Architecture

The architecture used in this diffusion model is a simple encoder-decoder,

Table 2.1: layer dimensions

Layer	shape
Decoder Input	(1, 64, 64), 2 x (10, 64, 64)
Bottlneck	(160, 1, 1), 2 x (160, 1, 1)
Decoder Output	(20, 64,64), 2 x (10, 64,64)

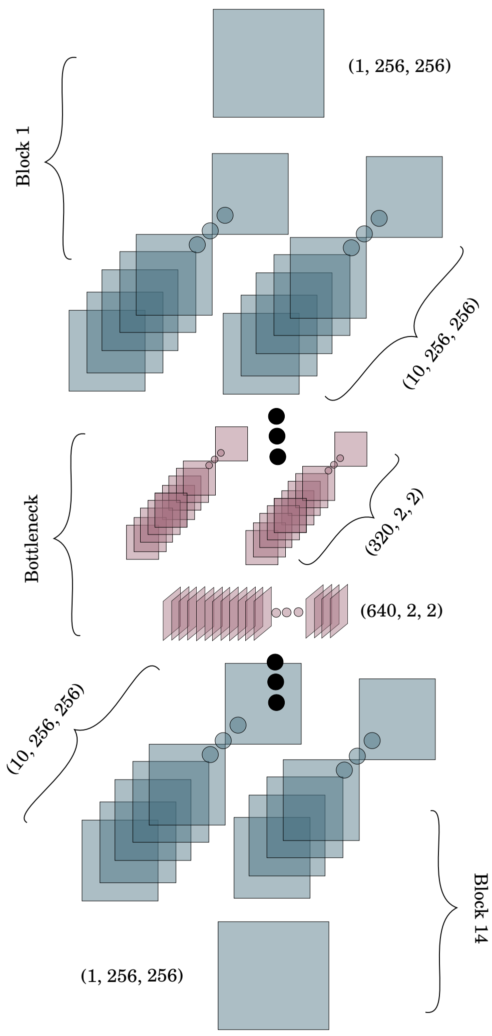
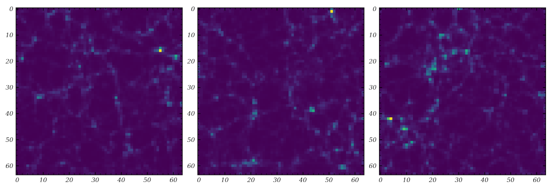
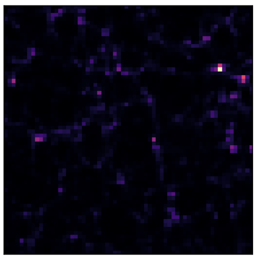


Figure 2.1: Unet convolutional structure

III Testing



(a) Training image



(b) Generated image

Figure 2.2: Overview of the NN data

The training images presents some specificities, we can clearly notice the presence of node like structures, which results from a large aggragation of galaxies. And as the priiomordial universe converged to a filamentary-like structure, the training image presents clear and noticeable filaments, so called "cosmic-web". In the datasets these type of struucture are also recognizable, however one clear observation is the mean value of the field which is not correct at all, 2.2b, therefore we can just conclude as first sight that the distrubution is range relevant.

1 Loss function

In section 2.I we have defined the loss function as a simple difference between real noise value, and predicted one (2.1), We can then study the convergence of this loss function during the training process. for each epoch and for each batch, this gives the following graph :

(Must retrained the NN to save this data)

2 Physics likelihood

Histograms

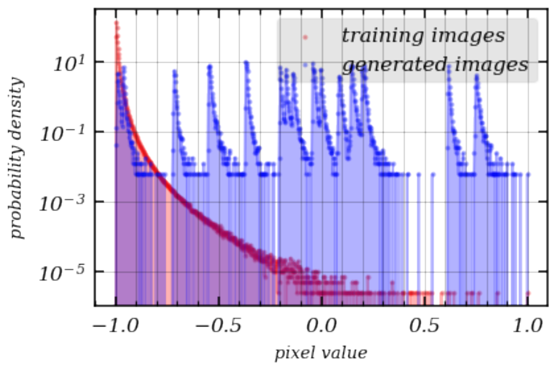


Figure 2.3: Histogran of intenties for both training and generated images, in red line the training images and in blue the generated images

Wuth 2.3, ze can lay the emphasis on the distribution similarity between generated and training images, however we can notice that the generated images mean values are not the same as opposed to the training images as we discussed in **2.III**. I've not find the cause of this issue . . . .

Power spectrum

As we discussed before, the generated images are not perfect, however besided the mean value issue, the range of the distribution seems correct. To study more precisely the characteristics of the images, one common way in astronomy is to study the power spectrum of the images. We can suppose the power spectrum isotropic in space, meaned over each y-voxel. The power spectrum is defined as the fourier transform of I^2 wuth I the pixel intensity. It describes the repartition in wave vector of the power, which is not related to with the mean value of the given image, which is a good point to study the generated images. Here we will study the normalized power spectrum, which is defined as : [cite]

$$P(k) = k^3 p(k)$$

with $p(k)$ the power spectrum and k the wave vector.

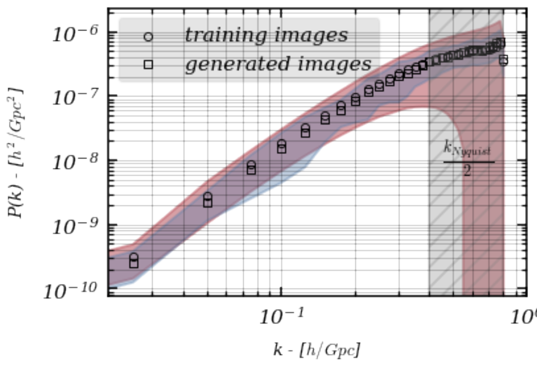


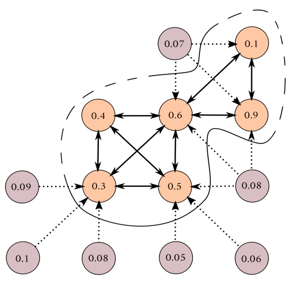
Figure 2.4: The blue fill is the standard deviation of the generated datasets, and the red one corresponds to the training image deviation. The grey shaded domain delimits the k_{nyquist} validity domain, due to the descretization of space. The psd leads to the same conclusion, the generated images and the training image have the same power spectrum, which strengthened the previous conclusion

```python
def compute_isotropic_psd(img : Iterable) -> tuple:
    """
    compute_isotropic_psd compute_isotropic_psd compute isotropic power distribution of a given image

    Parameters
    ----------
    img : Iterable
        _description_

    Returns
    -------
    tuple
        tuple of Iterable (wave vector k, power vector)
    """

    delta_x = 250/64
    k, p = welch(img, fs = 2 * np.pi / delta_x, axis = 0, scaling= 'spectrum', nperseg = 64)

    return k, np.mean(p, axis = 1)
```

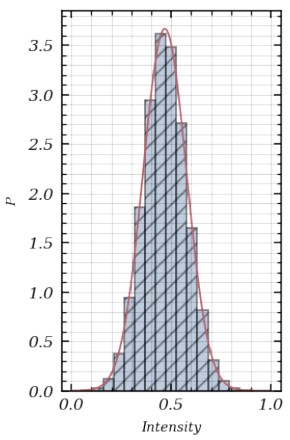
3 Third order image analysis

The goal of this analysis is to isolate complex structures such as filaments or nodes and to study their statistical characteristics. For isolating such structures, we used a graph approach, considering every pixel as a graph node connected to its nearest neighbors, this connection is removed if the pixel values is smaller than a typical threshold value fixed by the typical deviation of the gaussian noise distribution of the primary universe. To find this typical length we use *Fiducial* simulations at $z = 127$, which gives after renormalization process :

$$\sigma_0 = 0.107$$

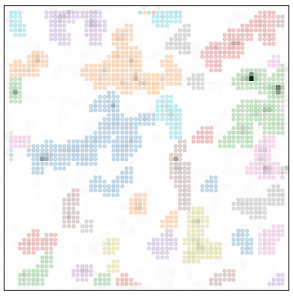


(a) Graph construction of the density field

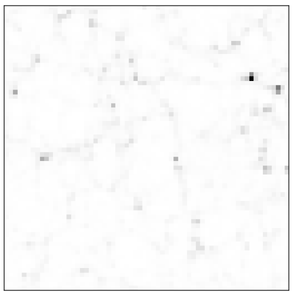


(b) Density distribution for $z = 127$

Figure 2.5



(a) Analyzed image with many filament cluster



(b) Generated image

Figure 2.6

4 Non linear contrasting

To improve the contrast of previous images, a solution has been proposed by [cite], it consists of the following transformation :

$$\sigma : x \mapsto \frac{x}{x+a}$$

Let's consider the number $\mathcal{C}_{nl} = \frac{\sigma_0}{(\sqrt{a}-a)}$. The intersting point about this typical number is the following. If $\mathcal{C}_{nl} \ll 1$, the contrasting functions is too weak for the filament structure, whearas for $\mathcal{C}_{nl} \gg 1$ the contrasting function is too strong for the noise. Therefore we must choose a, such that $\mathcal{C}_{nl} \sim 1$. The idea behind this is the following if $\mathcal{C}_{nl} \ll 1$ the derivative of the contrasting function at the typical length σ_0 is too big, it means that we do not isolate the filamentary structure. On the other hand if $\mathcal{C}_{nl} \gg 1$ the derivative of the contrasting function at the typical length σ_0 is too small, it means that we isolate the noise..

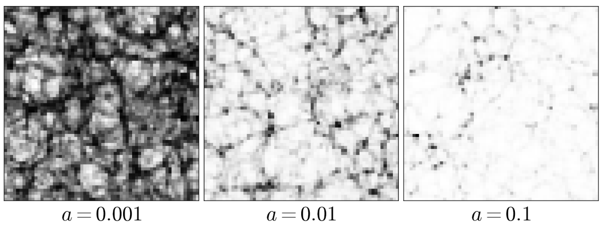
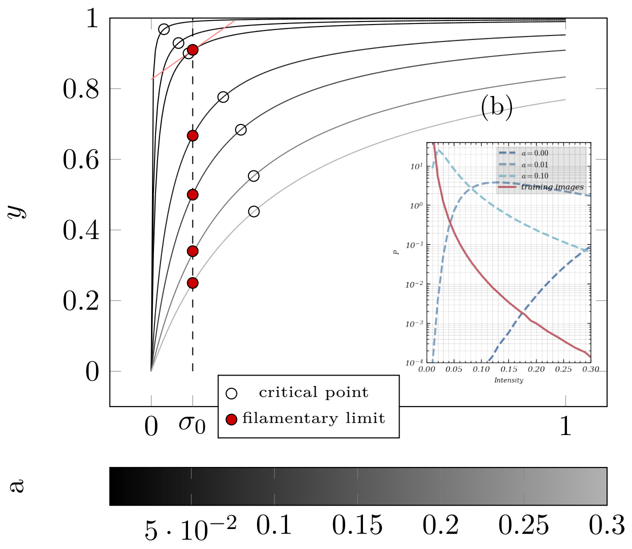


Figure 2.7: transformations for multiple values of a, the extreme loglike behaviour at low a is totally deforming the physics matter field. which should drastically lower the learning efficiency of the NN. For relatively high $a = 0.1$ we do nor increase a lot the filaments contrast as opposed to medium a values, where the filamentary structure is clearly distinguishable, without deforming the physics field 2.8.



(a) Analyzed image with many filament cluster

Figure 2.8: In the Contrast transformations where the parameter a is chosen to optimize the contrats given the previous normalization process. These functions allow us to shrink the inage domain after a critical value (2.8a), which should be choosen accordingly to threshold used in the clustering process. Indeed we wanna isolate the filamentary structures from the noisy background, which is not too small, otherwise we will isolate the noise, and not too big, otherwise we will not isolate the filaments. Given 2.8b we can observe the migration of intensity distribution for low a, it means that a lot of noisy pixel have been strenghtened, which is not what we want. For relatively high $a \sim 0.3$ we do ont isoliate the filaments anymore, because the crticial intensity is much higher

Chapter 3

GAN neural Network

I Theoretical background

1. G generates 32 images from 32 sets of 100 random numbers
2. D discriminates the images from the 32 real images (labelled 1) and the 32 images generated by G (labelled 0)
3. D learns from its mistakes
4. G generates 32 images from 32 sets of 100 new random numbers
5. D gives probabilities for the 32 generated images
6. G learns from the mistakes of D
7. we plot 16 generated images from 16 * 100 new random numbers to see if G always generates the same images
8. we plot a generated image and a real image to compare them
9. we print the outputs of the loss functions for this epoch

II Architecture

```python
class Discriminator(nn.Module):
    def __init__(self):
        self.model = nn.Sequential(
            nn.Linear(4096, 2048),  # 64x64 input image
            # = 4096 pixels
            nn.ReLU(),
            nn.Dropout(.3),
            nn.Linear(2048, 1024),
            nn.ReLU(),
            nn.Dropout(.3),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Dropout(.3),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(.3),
            nn.Linear(256, 1),
            nn.Sigmoid(),  # Discriminator gives a
            # probability as output
            )

    def forward(self, x):
        x = x.view(x.size(0), 4096)  # Vectorization
        output = self.model(x)
        return output

```

Listing 3.1: Python example

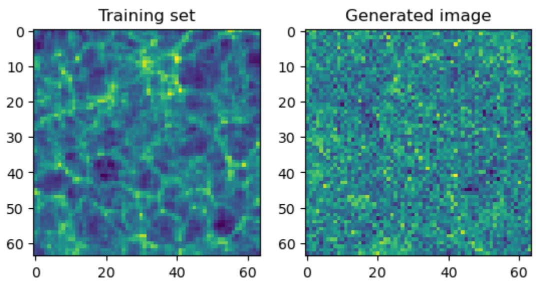
Training the model

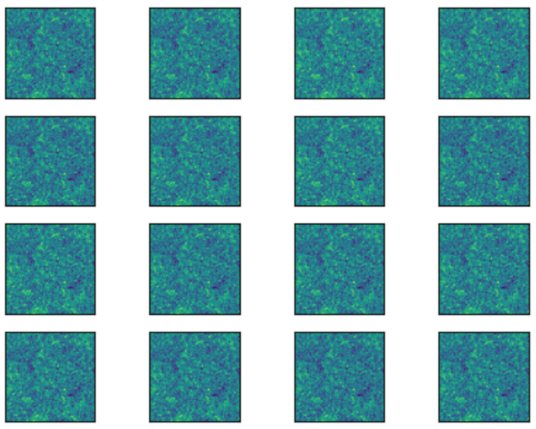
```python
plt.ion() # to be able to plot at every epoch

batch_size = 32
train_loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size, shuffle=True)

for epoch in range(num_epochs):
    for n, (real_samples, useless_labels) in enumerate(train_loader):
        # Data for training the discriminator
        real_samples = real_samples
        real_samples_labels = torch.ones((batch_size, 1))
        latent_space_samples = torch.randn((batch_size, 100))
        generated_samples = generator(latent_space_samples)
        generated_samples_labels = torch.zeros((batch_size, 1))
        all_samples = torch.cat((real_samples, generated_samples))
        all_samples_labels = torch.cat((real_samples_labels, generated_samples_labels))

        # Training the discriminator
        discriminator.zero_grad()
        output_discriminator = discriminator(all_samples)
        loss_discriminator = loss_function(output_discriminator, all_samples_labels)
        loss_discriminator.backward()
        optimizer_discriminator.step()

        # Data for training the generator
        latent_space_samples = torch.
    randn((batch_size, 100))

        # Training the generator
        generator.zero_grad()
        generated_samples = generator(
    latent_space_samples)
        output_discriminator_generated =
     discriminator(generated_samples)
        loss_generator = loss_function(
    output_discriminator_generated,
    real_samples_labels)
        loss_generator.backward()
        optimizer_generator.step()

    # Plotting generated data
    latent_space_samples = torch.randn
    ((16, 100))
    generated_samples = generator(
    latent_space_samples).cpu().detach()

        # to see if it is stuck
    for i in range(16):
        ax = plt.subplot(4, 4, i+1)
        plt.imshow(generated_samples[i].
    reshape(64, 64))
        plt.xticks([])
        plt.yticks([])

        # to compare with dataset
    fig, (ax1, ax2) = plt.subplots(1, 2)
    ax1.imshow(real_samples[2].reshape
    (64,64))
    ax1.set_title('Training set')
    ax2.imshow(generated_samples[2].
    reshape(64,64))
    ax2.set_title('Generated image')
    plt.show()

    print(f"Epoch: {epoch} Loss D.: {
    loss_discriminator}")
    print(f"Epoch: {epoch} Loss G.: {
    loss_generator}")

plt.ioff()
```

III Testing

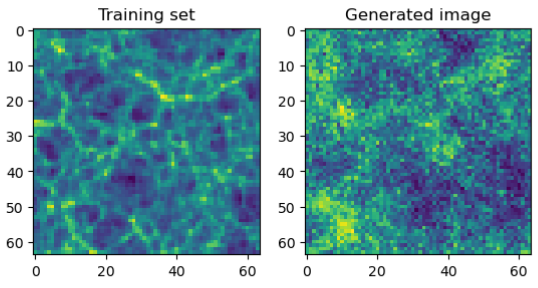
1 Results

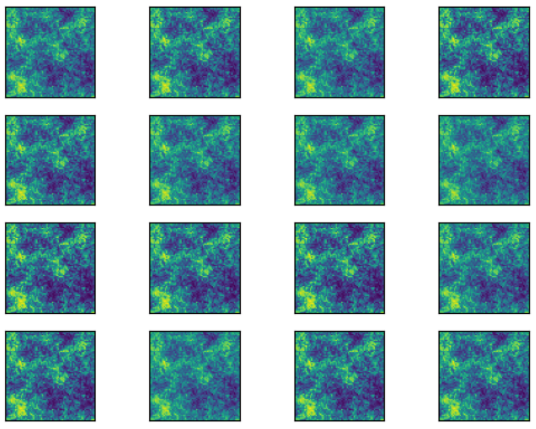
After 10 epochs (15 min):





After 100 epochs (one night):





- Visually, the ouput of the generator does not seem to depend on its input as show the 16 very similar images produced by different inputs for epochs 10 and 100.
- D and G are trained on a 32 size dataset. If we want to use the whole dataset, then I

have to choose between either loading the whole dataset in the train loader but then I improve D and G every 3 hours, or picking 32 new images at random for every epoch but then it's not an epoch anymore, and additionally my computer has a tendency to crash.

- Need for a statistical tool to analyse the images.

2 Loss function

3 Physics likelihood

Histograms

Power spectrum

Chapter 4

Discussions