Understanding Machine Learning Generative Models - GANs and VAEs

Srishti Goel

Contents

Chapter 1

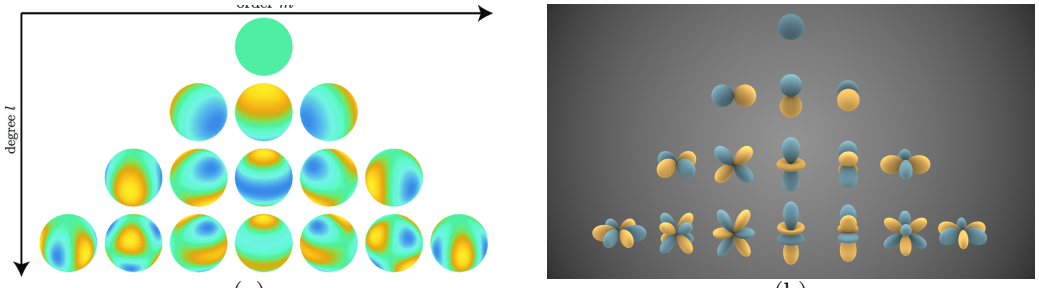
Introduction and Context

1.1 Cosmic Microwave Background(CMB)

The Cosmic Microwave Background(CMB) is the remnant light from the early universe, at the point in time when protons and electrons had cooled down enough to form neutral hydrogen making the universe transparent to radiation. This is was the process of *recombination*.

The CMB is the most dominant source of electromagnetic radiation beyond the Milky Way. It covers the full sky with the light spectrum of a nearly perfect black-body at a temperature of around 2.7K. Due to the expansion of the universe, light from a universe at around 4000K has been red-shifted to peak in the microwave wavelength. In other words, this black-body appears cooler than the universe they represent. The point of time when these photons last scattered in the universe before hitting the Earth is called the *surface of least scattering*.

The CMB photons follow distributions and patterns dictated by the physics of the universe when they last scattered. Thus, they contain important information to help us understand the early universe. For example, the temperature of the CMB black-body can tell us about the energy scales at which recombination occurred. Anisotropies in the temperature and polarization of these photons can tell us about the characteristic length-scale of the early universe's Baryonic Oscillations and other phenomena, the origin of the large-scale structure of the universe, etc.



(a) (b)

Figure 1.1: Spherical harmonics with varying order and degree. (a) uses color as intensity, (b) uses radius as intensity

Since wish to study the length-scales of the universe more than their exact coordinates in the sky, analysis of the same is usually done in the *spherical harmonics*. Similar to the Fourier transforms, this means describing the signal in terms of some basis functions of varying parameters. These functions In 2D maps like the surface of the sphere, there are 2 such parameters denoted as the order 'l' and the degree 'm'. The order specifies the angular resolution of the harmonic (a.k.a, how many times the wave divides the sky) and the degree specifies its direction. These harmonics are depicted in Fig. 1.1. The degree at a certain order 'l' may take integer value in the set: $m \in [-l, l]$.

It may become apparent that when depicted with radial intensities, these seem similar to the atomic orbitals in chemistry. This is no surprise.

We usually express the temperature of the CMB as:

$$T(\theta, \phi) = \sum_{l} \sum_{m} a_{lm} Y_{lm}(\theta, \phi) \tag{1.1}$$

where Y_{lm} denotes the spherical harmonic of order l, degree m; and a_{lm} is a spherical harmonic coefficient for the same.

The power at the order l is calculated as:

$$C_l = \frac{l}{4\pi} \sum_{m} |a_{lm}|^2 \tag{1.2}$$

By observing the power spectrum of the CMB at all the orders of spherical harmonics, we get a graph such as Fig. 1.2.

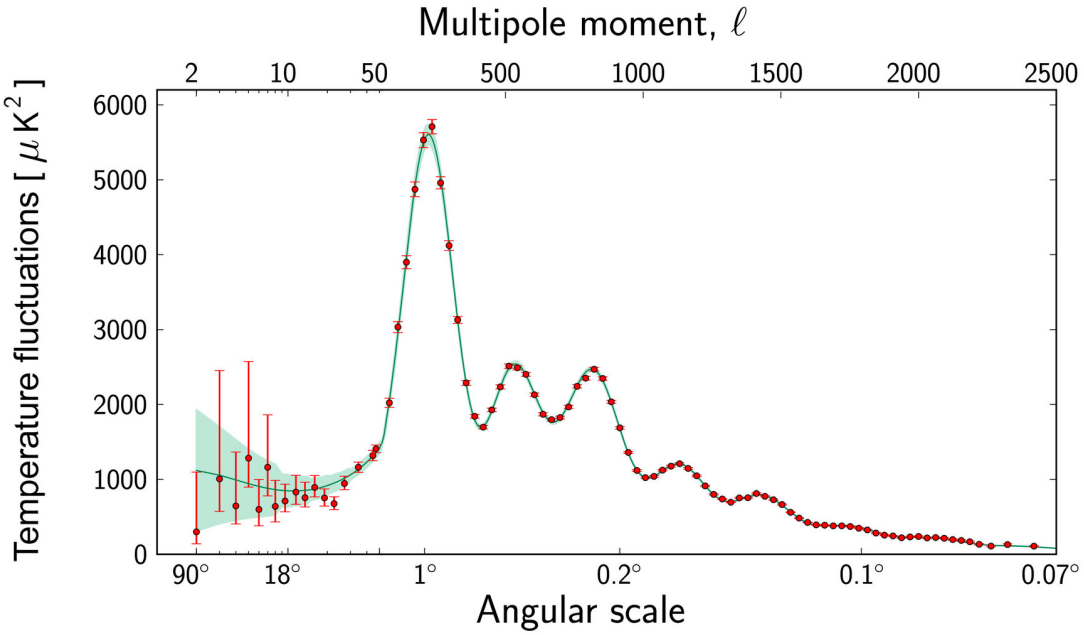


Figure 1.2: Power Spectrum of the CMB

Order l = 0 corresponds to the monopole, and represents the average temperature. It is equivalent to the DC component in spherical coordinates. This is the value used as the temperature of the CMB = $2.7255 \pm 0.0006K$

Order l = 1 corresponds to the dipole, and is interpretted as the Doppler boosting of the monopole due to movement of the Solar-System and the galaxy[1] relative to the rest-frame of the surface of last scattering.

The higher order multipoles are mostly due to the perturbations in the early universe. For example, the length scale of the first and biggest peak in Fig. 1.2 corresponds to the characteristic length scale of the sound/pressure waves due to the gravitational potential. Baryonic rarefications suppress the second harmonic, and as the baryonic number increases, the curvature decreases.

The green region in Fig. 1.2 represents the error bar. In addition to instrument-noise, as we reach the lower multipoles, there are fewer degrees

[1]Which surprisingly are in almost the same direction with respect to the CMB rest-frame.

to average over, which increases the error bar. This is the *cosmic variance*.

Density variations of the early universe can be one of 2 categories - Adiabetic density perturbations and Isocurvature perturbations.

So how does this help constrain the physics of the early universe?

Different models of the early universe predict different correlations between the orders of the spherical harmonics, and the power-spectrum. For example, the slow-roll theory of inflation predicts that the CMB fluctuations are purely Gaussian, and are completely defined by the power-spectrum. Thus, we notice PRIMODIAL non-Gaussianities in the CMB, it would start making us second-guess the slow-roll inflation theory.

1.2 Foregrounds Considered

However, as the CMB photons travel from the surface of last-scattering, they get perturbed by certain foregrounds which introduce effects like non-Gaussianities. In order to detect primodial non-Gaussianity, we need to accurately undo the effects of these foregrounds.

1.2.1 Gravitational Lensing

Gravitational Lensing is the phenomenon of bending of light around a massive body. When the CMB photons undergo this bending, it affects the distribution of the radiation field. Gravitationally overdense regions(have more matter-density than the average) cause photons to lose energy on the way out, causing them to redshift. On the other hand, Under-dense regions (lesser matter-density than the average) cause photons to gain energy on the way in, causing them to blueshift. In short, overdensities show up as cold spots, and underdensities show up as hot spots.

This effect has been detected in the CMB with an over 1000σ detection. It is mapped as a lensing convergence map, which is expected to be nearly Gaussian with linear isotropic perturbations. The fact that we know this

distribution means that we do not need to use Machine Learning to create it (and perhaps, as the results on the blobs dataset described in this report show, it's possibly worse to learn a Machine Learning to generate a dataset with a known distribution). This will become important in our methodology later.

1.2.2 Thermal Sunyaev-Zeldovich Effect (tSZ)

The Sunyaev-Zeldovich (SZ) effects are perturbations that occur when CMB photons hit galaxies with clouds of hot electrons as found in star-forming clouds.

The Thermal SZ effect(tSZ effect) is a kind of inverse-Compton effect(see Fig. 1.4). In the Compton effect, a highly energetic photon hits a slow-moving electron, and the electron gets energized, and the photon gets red-shifted. However, in the tSZ effect, a low energy photon hits a relativistically moving electron. In this process, the photon gets blue-shifted, and the electron slows down. In terms of the spectrum, this appears as in Fig. 1.3. It shows how the photon count of the lower-energy photons has been decreased, and a subsequent increase is seen in the high-energy photon-count. Note that the spectrum of a Black-Blody at the CMB temperature has been scaled to fit in this scale. The SZ effects of the CMB are smaller than the Gravitational Lensing effects, and have only been detected to a 30σ detection.

1.2.3 Kinetic Sunyaev-Zeldovich Effect (kSZ)

The Kinetic SZ effect is a milder still effect. It is caused when a galaxy cluster is moving relative to the Hubble flow, in an effect similar to the Doppler effect of the scattered photons.

$$\Delta T_{kSZ} = -T_{CMB}\frac{v_p}{c}\tau \tag{1.3}$$

where T_{CMB} is the temperature of the CMB black-body, v_p is the peculiar velocity of the galaxy cluster (velocity more than the Hubble flow veloc-

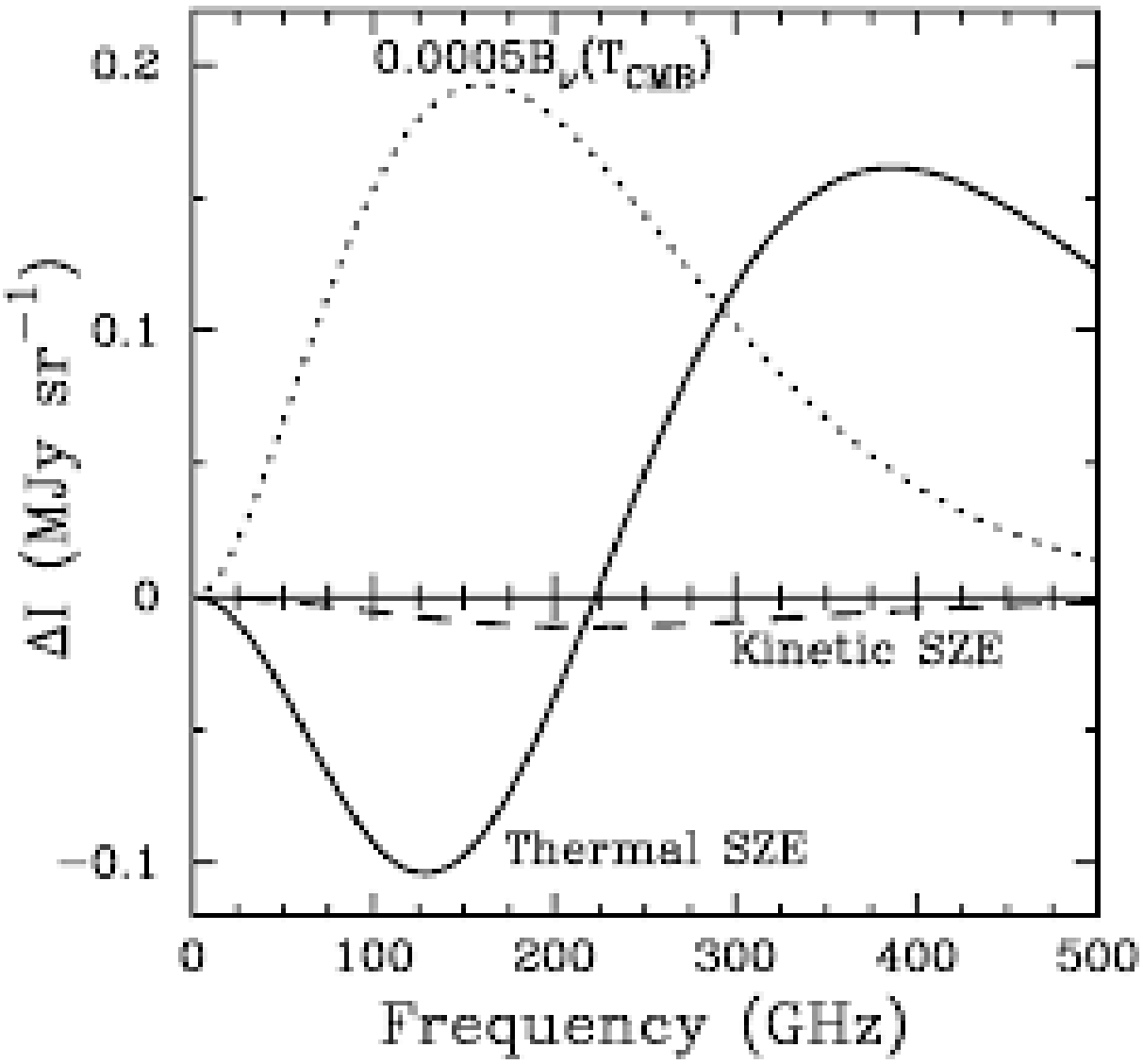


Figure 1.3: The Spectrum of tSZ and kSZ effects

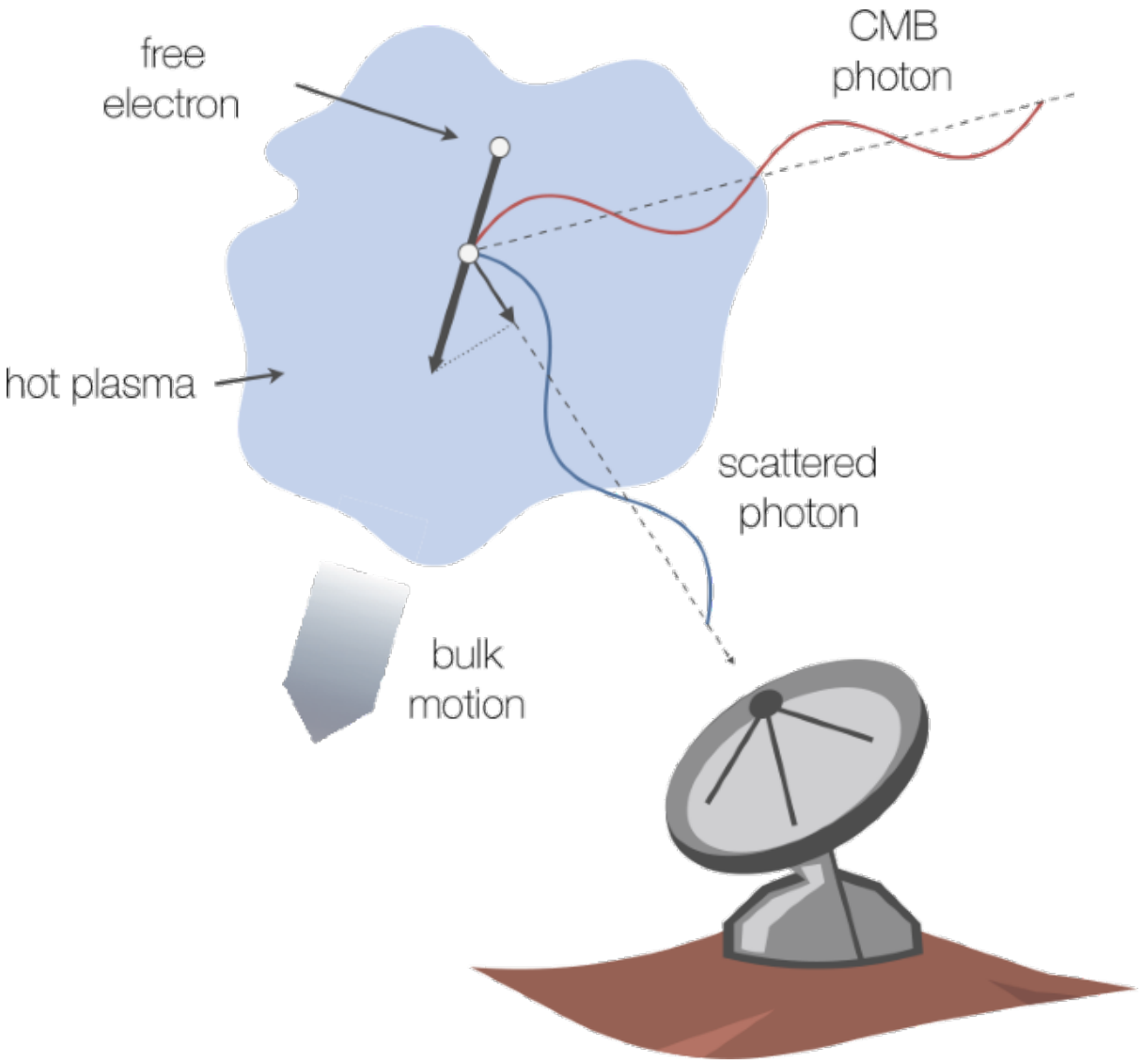


Figure 1.4: Thermal SZ effect

ity), and τ is the optical depth, which takes into account of the Dispersion Measure. Using gravitational lensing, the peculiar velocity can be used to determine other velocity components for a specific galaxy cluster.

1.2.4 Radio Galaxies

Radio Galaxies are galaxies with large structures of radio emission extending beyond its visible structure. These energetic radio lobes are powered by jets from its active galactic nucleus. Radio-loud active galaxies can be detected at large distances, making them valuable tools for observational cosmology.

Since these are strong radio sources, they add to the sensed radio emissions, and affect the black-body temperature of the CMB.

1.2.5 Cosmic Infre-Red Background Radiation (CIB)

The Cosmic Infra-Red Background(CIB) is the background of infra-red radiation that is caused by diffused starlight scattered by stellar dust. It peaks around $50\mu m$ and is important in studying distant quasars, etc. Since Star-Formation Rate(SFR) of galaxies peaked around redshift z = 1-3, this is also the strongest source of the diffused starlight that forms the CIB. The redshift of this light has moved the peak close to the CMB peak wavelength, and thus, this forms an important foreground source to understand.

Another important component of the CIB is the infrared emission by quasars.

There are correlations between all the foregrounds, such as the correlation between the CIB and the tSZ as explored in [1]

1.3 The Seghal 2010 (S10) simulations

Current state-of-the-art removal of the foregrounds involves making 3D simulations of the universe with N bodies, and projecting the foregrounds to

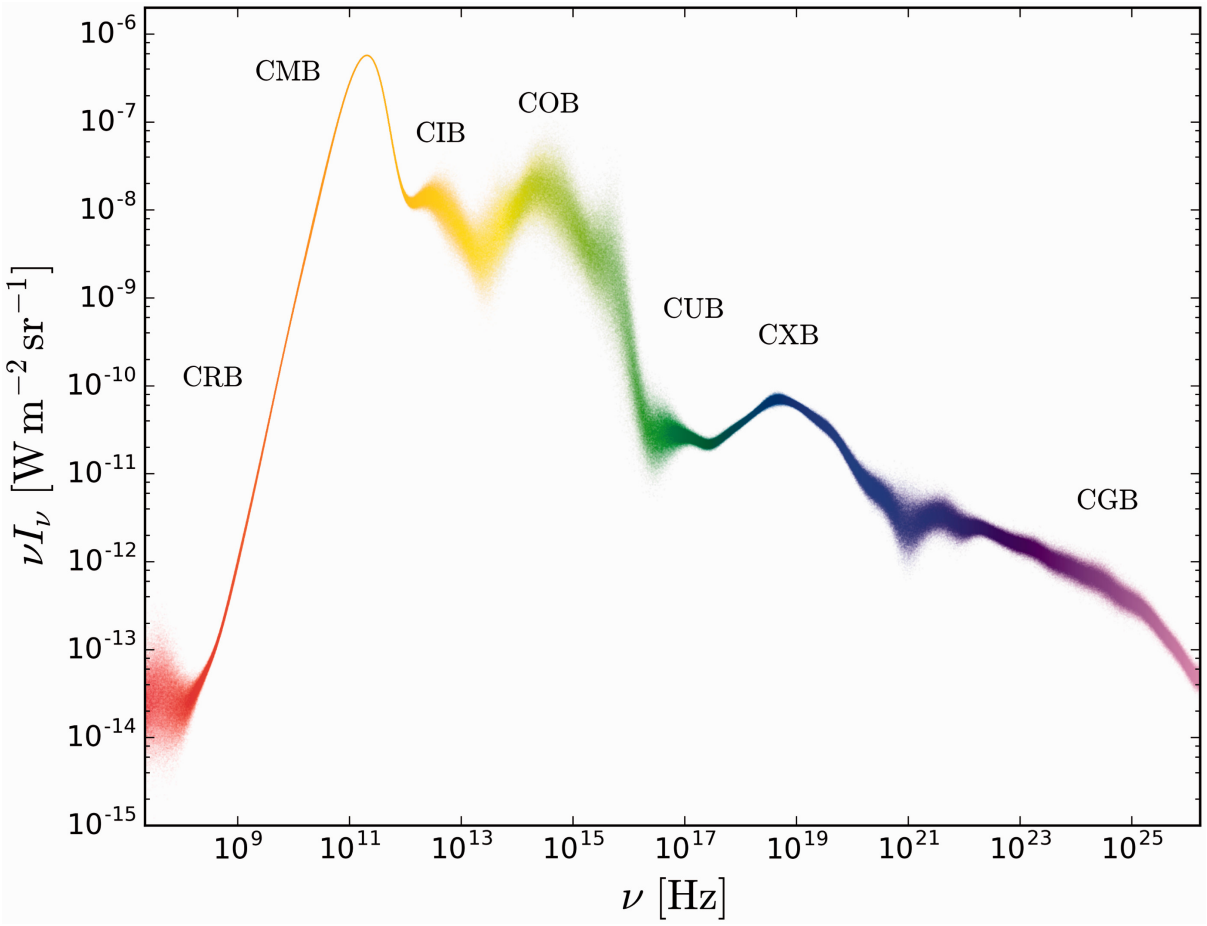


Figure 1.5: A depiction of the Cosmic Background Spectrum

the 2D full-sky map. These full-sky maps are then used to eliminate these foregrounds.

However, making 3D simulations of the universe is a computationally very expensive method. Instead, the authors of [2] (which is the paper we try to reproduce the results of in this report) use one full-sky map simulation to study the statistics of the foregrounds using Machine Learning (ML) and produce 50 full-sky maps which replicate the statistics of the full-sky map it was trained upon.

The full-sky map the authors of [2] use to train the model are the S10 simulations, which adapt the WMAP-5 cosmological parameters and was developed for the Atacama Cosmology Telescope(ACT) team to test their data-analysis pipeline. The angular resolution of these maps are .5-arcminute, and are saved in the form of NSIDE = 4096 HEALPix maps. These simulations are discussed in [3] and have been used to develop full-sky foreground maps for frequencies beyond the ACT frequency channels too.

However, the S10 simulations are only unique in 1 octant of the full-sky.

Chapter 2

Generative Adversarial Networks (GANs)

Generative adversarial networks are a class of generative models that use unsupervised learning to generate data (in this case, 2D images) similar to the input data.

The question then arises, what does "similar" even mean in this context? The problems that GANs are applied on do not have a fixed framework of similarity, and thus, allow the model to learn its own meaning of similarity. The problem statement does not give it any signals of whether it is doing better or worse (this is what it means to do "unsupervised learning").

GANs acheive this goal by setting up an adversarial game between 2 neural networks - the Generator and the Discriminator. The core ideology behind this? *Competition drives progress.*

The two neural networks are discussed below:

- **Generator** : The generator gets an input random vector of a fixed size (understood as a set of a fixed number of random variables).[1] It then uses these random variables to generate an image. In doing so, its objective is to make images that the discriminator cannot distinguish from the actual input images.

- **Discriminator** : The discriminator gets an input image (from the gen-

[1] This is the latent vector.

erator or the input images), and assigns it a particular real valued score. In doing so, it tries to distinguish input images from input images by assigning a high score to the input images, and a low score to the generated images. This score can thus be seen as a measure of "realness" as understood by the discriminator.

During training, the generator and discriminator engage in a competitive process. The generator aims to produce realistic data, and the discriminator tries to understand key features in the input data to distinguish input data from generated data.

GANs are considered a type of implicit density models. They do not assume any likelihood on either the input random vector, or the data. Thus, it is more difficult to use a likelihood estimation, and use a deterministic mapping from noise to samples (the mapping is the discriminator).

GANs have demonstrated impressive capabilities in various domains. They have been successfully applied to tasks such as image synthesis, style transfer, image-to-image translation, text generation, and even video generation. GANs have also been instrumental in advancing the field of computer vision, as they can learn to generate highly realistic images that resemble the training dataset.

Upon applying a simplistic GAN model on the MNIST dataset, which has handwritten digits in every image, and are classified by digits ('0', '1', etc.), the evolution of generated images over the course of training was observed. (Fig. 2.1, Chapter 2)[2]

However, training GANs can be challenging. Finding the right balance between the generator and discriminator networks is a delicate process, and GANs are known for being sensitive to hyperparameter settings. Training instability manifest as mode collapse[3], and convergence issues[4], gradient van-

[2]While the training was not perfect at the end of epoch 60, the losses seemed to have converged, perhaps due to the simplicity of the model, or other factors as discussed.

[3]The generator appears to get fixated on the same or a small subset of images like the input images.

[4]The apparent improvement of one of the models leads to an apparent worsening of the other and this causes a sort of oscillation as seen in Fig. 2.4. Combined losses implies the sum of the losses of the generator and discriminator and is a function of just the discriminator acting on a morphed version of input images.

Figure 2.1: Original images from the MNIST dataset

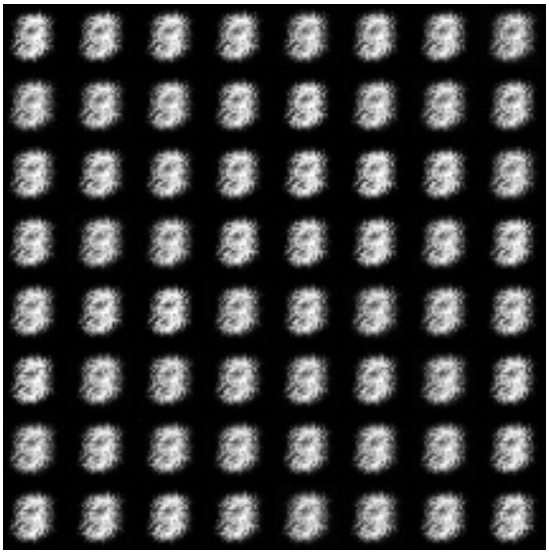


Figure 2.2: Output after 1 epoch

Figure 2.3: Output after 60 epochs

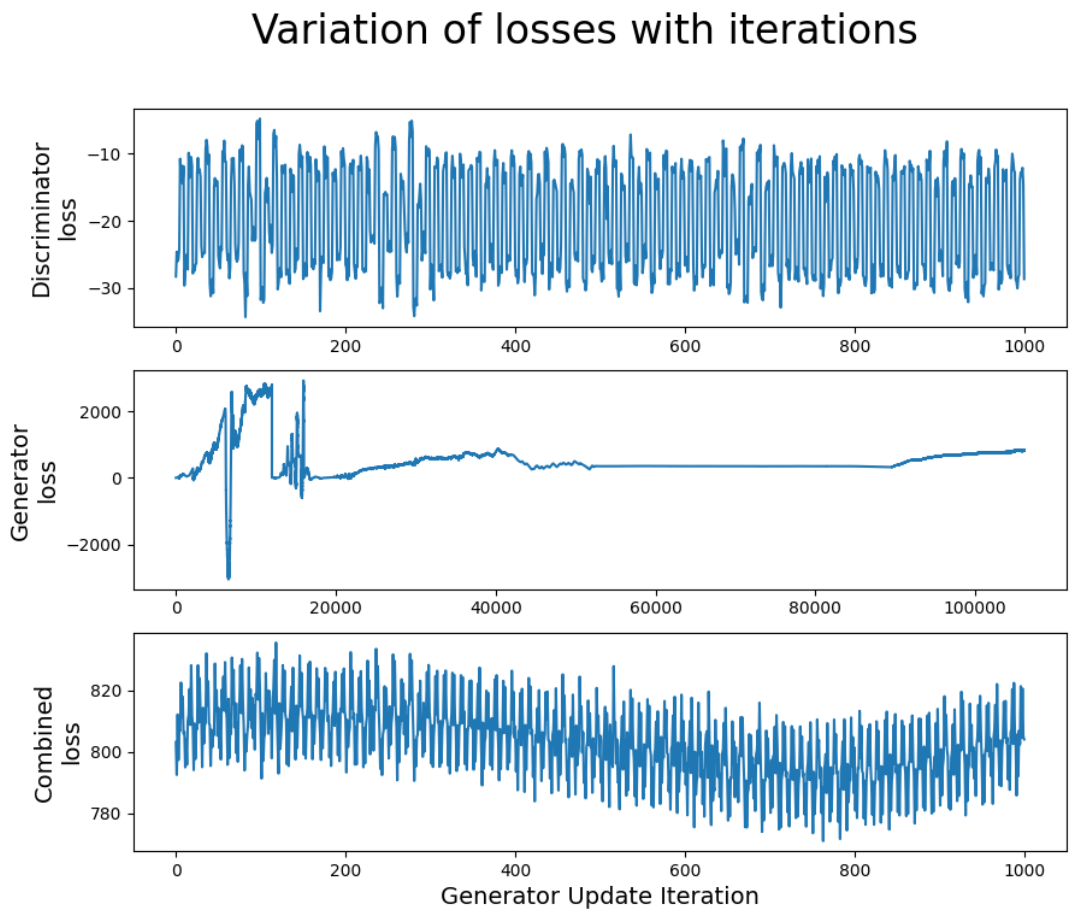


Figure 2.4: Example of the situation where the losses begin to oscillate

ishing[5], mode dropping[6] and hyperparameter sensitivity[7]are some common challenges faced when working with GANs.

Addressing training instability in GANs is an active area of research, and several techniques have been proposed to mitigate these issues. Some approaches include architectural modifications, improved loss functions, regularization techniques, and more stable training algorithms such as Wasserstein GANs or spectral normalization. Researchers are continually exploring new methods to enhance the stability and convergence of GAN training.

[5]If the gradients become too small to make meaningful upates, or too large, leading to unstable training updates. This was removed from our models by adding a factor of the gradient to the loss itself.

[6]The discriminator becomes fixated on particular features that are not actually relevant, but the generator accordingly fixates on them too, missing some key modes/patterns in the input data.

[7]GANs are sensitive to hyperparameter settings such as learning rates, network architectures, regularization techniques, and more. Suboptimal choices of these hyperparameters can cause training instability, making it challenging to achieve good results.

Despite these limitations, GANs provide stunning images , given all the parameters tuned right. This is why they still remain an area of active research to this day.

Chapter 3

Wasserstein Loss in GANs

The loss function used in the Han et al. paper was the Wasserstein loss function (for all the 3 models).

Using the notation:

- $D(i)$ is the discriminator score for an image i (which may be an input image, or a generated image)
- I is an input image
- $\tilde{I}$ is a generated image
- λ_{GP} is a programmer-defined (hyperparameter) weight to the regularization term
- I_{interp} is the interpolated image described below
- $\nabla_{\mathbf{y}} f(\mathbf{y})$ represents the vector gradient of the function $f(\mathbf{y})$ acting on a multi-dimensional input $\mathbf{y}$. In case this is an image, we mean the gradient of $f(\mathbf{y})$ with respect to each pixel of $\mathbf{y}$. This is calculated by using the library function in *Pytorch*:

```
torch.autograd.grad(input_value, output_value)
```

- $\|\mathbf{y}\|$ represents the vector norm of $\mathbf{y}$

The vanilla Wasserstein loss defines the loss of the Generator and Discriminator as:

$$\text{Loss of the Generator} = 0 - D(\tilde{I}) \tag{3.1}$$

$$\text{Loss of the Discriminator} = D(\tilde{I}) - D(I) \tag{3.2}$$

Thus, the Generator aims to increase the Discriminator's score on generated images. The Discriminator, on the other hand, aims to increase the divide between the score of generated images and score of input images. It does this by trying to maximize the score of input images, while decreasing the score of generated images. This is how the score is a measure of "realness" of an image.

However, to counter the mode-collapse faced by GANs, some works introduce an extra regularization factor to the loss of the Discriminator. This extra regularization is related to the gradient of the discriminator score on an interpolated image between input and generated image.

First, to generate this interpolated image, for each generated image, a random weight ϵ is selected uniformly between 0 and 1.

$$I_{\text{interp}} = \epsilon I + (1 - \epsilon)\tilde{I} \tag{3.3}$$

Thus, the interpolated image is a weighted average of the input and generated images in the image space.

Thus, this variation of the Wasserstein loss function (called the Wasserstein Loss function with Gradient Policy in the future) is calculated as:

$$\text{Loss of the Generator} = 0 - D(\tilde{I}) \tag{3.4}$$

$$\text{Loss of the Discriminator} = D(\tilde{I}) - D(I) + \lambda_{GP}\|\nabla_{I_{\text{interp}}} D(I_{\text{interp}})\| \tag{3.5}$$

The regularization term, thus penalizes the Discriminator for large gradients between generated and input images. If there are large gradients, the

Discriminator may be over-fitting to differentiate input and generated images. It would appear to introduce more twists and turns to differentiate all the generated images.

During training, MINI-BATCHES are batches of input images over which the loss is calculated, after which an update step is taken). While using these mini-batches, the regularization term uses its mean value over a mini-batch-long number of generated images matched to a corresponding input image each. The discriminator score is also similarly averaged over the mini-batch.

The question of how to choose the hyperparameter λ_{GP} remains an unanswered question right now.

Chapter 4

Overview of Methodology

The authors of [2] use an elaborate combination of 3 Machine Learning (ML) models, in particular, Generative Adversarial Networks (GANs) to replicate the statistics of the full-sky maps it learns on (the S10 simulations).

All three GAN models use the same architecture of the Discriminator. This architecture is given in Fig. 4.2. They all use the Wasserstein loss function as discussed in the previous chapter.

Since the universe is isotropic, we can assume that any patch of the sky is statistically similar to another patch of the same size. They will share the same moments up to the angular size of the patch. For example, a 1° x 1° patch corresponds to spherical harmonics of order $l \geq 200$, and thus, every patch of this size will have the same power-spectrum and bispectrum at these l's. Thus, the entire full-sky has been divided into small patches of angular size 1° x 1° (by the method discussed in a future chapter), which are projected from the HEALPix map to a Cartesian Coordinates. The ML models learn to replicate these patches for all the foregrounds, which are components of this patches.

The first model (in Training Step 1) is used to increase the size of the dataset maintaining the correlations between the components. This model is a Deep Convolutional GAN (DCGAN). The architecture of this model's generator is shown in Fig. 4.3, and has the least number of parameters to train. Thus, on learning a simple GAN model, we increase the size of the

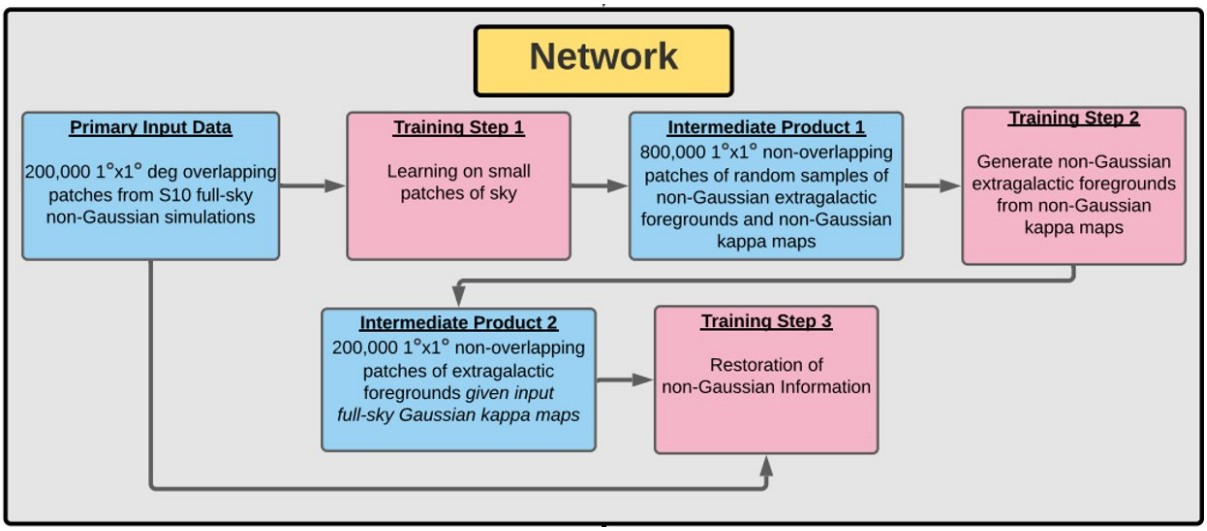


Figure 4.1: A schematic of the overall procedure to train the network and generate the output maps from the input data

dataset from 200,000 patches in the Primary Input Dataset to 800,000 in the Intermediate Product 1. However, since we have learnt the statistics and made an ML model replicate them, there is bound to be some form of loss of information in this step.

The second model then learns to input a patch with only the lensing convergence component, and generate the corresponding patches of the rest of the components. By doing so, the authors were able to generate a full-sky lensing convergence map and make patches of this full-sky maps. Then, the corresponding foregrounds for that patch of the sky could be generated, and placed back at the same position as the initial patch along with some interpolations (as discussed later). Thus, full-sky correlated maps of the different components can be generated. The reason the lensing convergence component was chosen as the base component was that these maps have been observed to be almost perfectly Gaussian, and thus, are assumed to be perfectly Gaussian in this work. By assuming Gaussianity, generating a full-sky map is simple, and does not require Machine Learning. This learns on the Intermediate Product 1 in Training Step 2, and has the architecture of the generator similar to a U-NET, and is called a PIXGAN (pixelwise GAN)

Layer	Activation Function	Output Shape of Tensor	# of Trainable Parameters
Input Map	N/A	*N*x128x128	N/A
Conv 4x4	LReLU	64x64x64	5,184
Conv 4x4	LReLU	128x32x32	131,200
Conv 4x4	LReLU	256x16x16	524,544
Conv 4x4	LReLU	512x8x8	2,097,664
Conv 4x4	LReLU	1024x4x4	8,389,632
Linear	N/A	1	16,385
Total Trainable Parameters			11,164,609

Table I: Summary of the discriminator model described in Section III A. We use the same discriminator model throughout all three training steps described in Section III. The first and second columns list the layers and the corresponding activation functions. The third and the fourth columns give the shapes of the output arrays and the number of trainable parameters in each layer. Here, N is the number of input components, i.e. $N = 5$ for DCWGAN-GP and PIXGAN and $N = 10$ for VAEGAN. Conv 4×4 indicates a two-dimensional convolution layer with a 4×4 kernel, and LReLU stands for a leaky rectified linear unit activation with $\alpha = 0.2$ (see Section III A for details). Note that Conv 4×4 downsamples an image by a factor of two, and increases the number of features to consider.

Figure 4.2: Discriminator used for all 3 GAN models

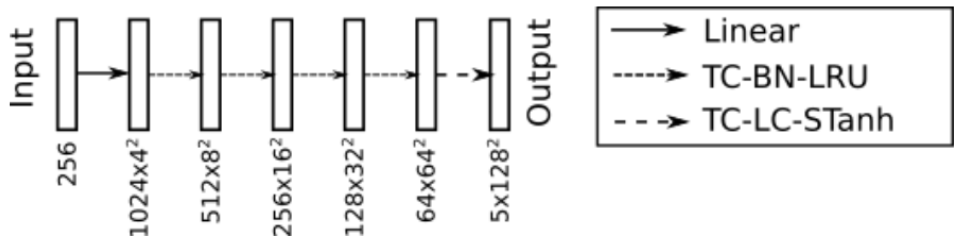


Figure 2: Shown is a diagram of the DCWGAN-GP generator model described in Section III A and indicated in the schematic of Figure 1 as Training Step 1. The input to this generator is a latent-vector consisting of 256 random numbers drawn from a Gaussian distribution with mean equal to zero and variance equal to one ($\mu = 0, \sigma = 1$). The output is one set of 1° x 1° maps consisting of the five foreground components. The direction of data flow is indicated by arrows. A solid arrow is a fully connected Linear layer. A short-dashed arrow, labeled TC-BN-LRU, represents a sequence of 1) a 4×4 two-dimensional transposed convolution, 2) a batch normalization, and 3) a leaky rectified linear unit activation function with $\alpha = 0.2$. Lastly, a long-dashed arrow, labeled TC-LC-STanh, is a sequence of 1) a 4×4 two-dimensional transposed convolution, 2) a Linear Channel, and 3) a Scaled Tanh activation function. The Linear Channel and Scaled Tanh are defined in Section III A. Note that a transposed convolution upsamples an image by a factor of two.

Figure 4.3: The architecture of the DCGAN-WP generator

as in Fig 4.4. The output of this step is 200,000 patches of the foregrounds given one realization of the full-sky lensing convergence map (κ map) in Intermediate Product 2..

However, the authors of [2] admit that in the process of generating these images, some information of the initial full-sky maps, and in particular, the non-Gaussian information, was lost. In order to restore this information, they include a last GAN model at the end. This takes in all 5 components as input and returns 5 components as output. This is trained on a combination of both Primary Input Data and Intermediate Product 2. The architecture of the generator is similar to a combination of the Variational Autoencoders and GANs called VAEGAN, as given in Fig. 4.5. The output of this model is the final output map's patches.

In generating the final simulations, initially, a full-sky κ map is generated with the known Gaussian statistics, and using the PIXGAN, corresponding maps of the other components are generated. These are then passed through the VAEGAN generator to refine and add missed non-Gaussian statistics. The patches are then put together and interpolated to form a full-sky map.

To shift to the other frequencies, we can adopt the following method. Since the κ map is frequency independent, it can be generated the same

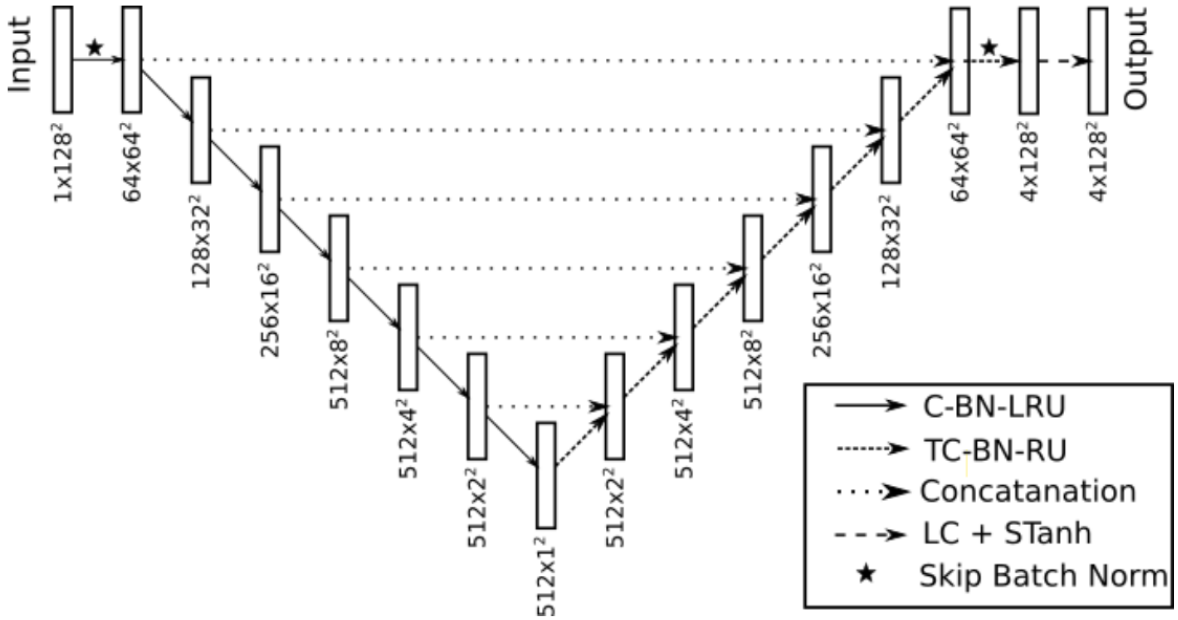


Figure 3: Shown is a diagram of the UNET generator model described in Section III B and indicated in the schematic of Figure 1 as Training Step 2. The generator takes as input a 1° x 1° κ map, and it outputs a set of 1° x 1° extragalactic foreground maps consisting of tSZ, kSZ, CIB, and Radio components. The direction of the data flow is indicated by the arrows. A solid arrow, labeled C-BN-LRU, represents a sequence of 1) a 4×4 two-dimensional convolution, 2) a batch normalization, and 3) a leaky rectified linear unit activation function. A short-dashed arrow, labeled TC-BN-RU, represents a sequence of 1) a 4×4 two-dimensional transposed convolution, 2) a batch normalization, and 3) a rectified linear unit activation function. A two-dimensional convolution downgrades an image by a factor of two, while a transposed convolution upsamples an image by the same factor. A dotted arrow is a concatenation operation, which concatenates the output of an originating box to the output of a destination box. A long-dashed arrow, labeled LC + STanh, on the top right represents a sequence of a Linear Channel and a Scaled Tanh activation function. Linear Channel and Scaled Tanh are defined in Section III A. Deviating from our notation, we do not apply a batch normalization to the first and the last convolution layers, indicated by a star over a line.

Figure 4.4: The architecture of the PIXGAN generator

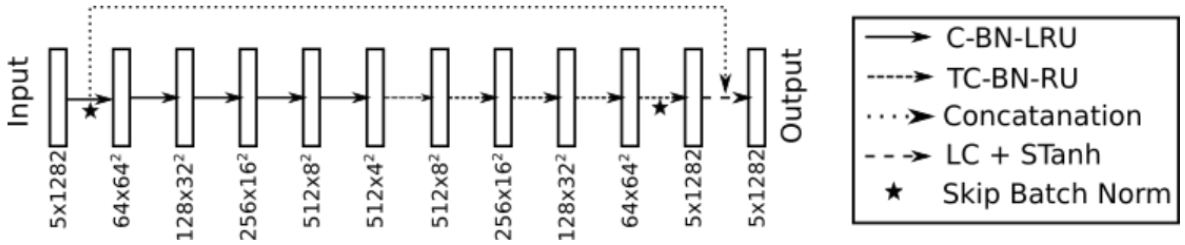


Figure 4: Shown is a diagram of the VAEGAN generator model described in Section III C and indicated in the schematic of Figure 1 as Training Step 3. The input to this generator is a set of 1° x 1° *Gaussian* κ maps and their corresponding extragalactic foreground maps from the PIXGAN network in Figure 3 (i.e. Intermediate Product 2). The output are a set of non-Gaussian κ maps and their corresponding non-Gaussian extragalactic foreground maps. The direction of the data flow is indicated by arrows. A solid arrow, labeled C-BN-LRU, represents a sequence of 1) a 4×4 two-dimensional convolution, 2) a batch normalization, and 3) a leaky rectified linear unit activation function. A short-dashed arrow, labeled TC-BN-RU, represents a sequence of 1) a 4×4 two-dimensional transposed convolution, 2) a batch normalization, and 3) a rectified linear unit activation function. The two-dimensional convolution downgrades the image by a factor of two, while the transposed convolution upsamples the image by the same factor. Shown as a dotted arrow is a concatenation operation between the input Intermediate Product 2 and the VAEGAN network output, which focuses the network on the residual difference between the two; note that we apply the inverse of Scaled Tanh to this input prior to the concatenation. A long-dashed arrow, labeled LC + STanh, represents a sequence of a Linear Channel and a Scaled Tanh activation function; they are described in Section III A. Deviating from our notation, we do not apply a batch normalization to the first and the last convolutions layers, indicated by a star below the arrow.

Figure 4.5: The architecture of the VAEGAN generator

way. The other components can be found for the ACT frequency (148 GHz chosen by [2]) using the PIXGAN and VAEGAN. Next, the tSZ map can be shifted to the desired frequency ν by the known analytic formula:

$$M^{tSZ}(\nu) = \frac{a(\nu)}{a(148GHz)} M^{tSZ}(148GHz) \tag{4.1}$$

where the map is given at a frequency f as $M|tSZ(f)$, and $a(f)$ is the frequency-dependent scaling factor. For the CIB and Radio maps, we find the mean spectral indices between each frequency and the ACT frequency of 148 GHz, directly from the S10 catalogue. For each source in the 148 GHz map, a sample from a Gaussian distribution with the same standard deviation and mean is selected and the flux is scaled accordingly.

Chapter 5

Projection of Full-sky Maps for Patch Dataset

Full-sky images are essentially projections of the sky onto the 2D celestial sphere. There have been many ideas about how to best save these images without using the entire 3D space. One such method is the HEALPix map. However, since these are on a spherical surface, they must be projected to a flat 2D space to be visualized as an image. Some views of the full-sky map include the Mollewide view or the Gnomic view. However, grid lines on these views are curved. To convert to a Cartesian coordinate system, these maps must be projected in a more complicated fashion.

Each full-sky map is divided into 200,000 overlapping patches of angular size 1° x 1° by taking the middle latitudes ($-10°$ to $10°$). This map is then rotated to shift a different part of the sky to the middle, and taking the same cutouts. This process is repeated for 20 total positions.

5.1 HEALPix Projections

HEALPix (Hierarchical Equal Area isoLatitude Pixelisation) is a kind of projection used for CMB maps which involves dividing the full-sky into 12 diamond-structures with 4 on the equator and 4 on each side of this. For further resolution, each of these are binary-divided into N-side x N-side "pix-

els" as shown in Fig. 5.1[1] where the lines define the pixel boundaries and the dots represent the center of the pixel. The number of pixels in an N-side map is given by:

$$\text{No. of pixels in the map } = 12 * (\text{N-side})^2 \tag{5.1}$$

Healpy saves maps as 1D arrays of these pixels, and has 2 formats of reading this array - nested or ring. In the default ring setting, pixels are ordered (using their pixel-centers) in rings of equal latitude and read from north to south, west to east. In the nest configuration, the pixels of each of the 12 initial diamond structures that make N-side 1 are saved one-after-the-other.

5.2 Cartesian (CAR) Projection

As seen in Fig. 5.2, the flattened version of the HEALPix map has curved pixel shapes especially at the poles, and are not of equal sizes. This is not conducive for a regular CNN. Thus, in this project, the maps are projected to a Cartesian Projection with Equirectangular pixels. This allows each pixel to be of a regular shape and size.

However, this conversion from the HEALPix projection to the CAR projection introduces a certain measure of stretching, as the same HEALPix pixel is counted for multiple CAR pixels. In particular, the pixels at the poles are highly distorted as a single point (the pole) gets distorted to an entire line.

Thus, to avoid the stretching as best as possible, we first divide the components into 2 sets -

- **Ones with point-like sources**: These are the CIB and Radio Galaxy foregrounds
- **Ones with extended sources**: These are the lensing convergence

[1]Since the pixels are divided using a binary-division, N-side must be an integral power of 2.

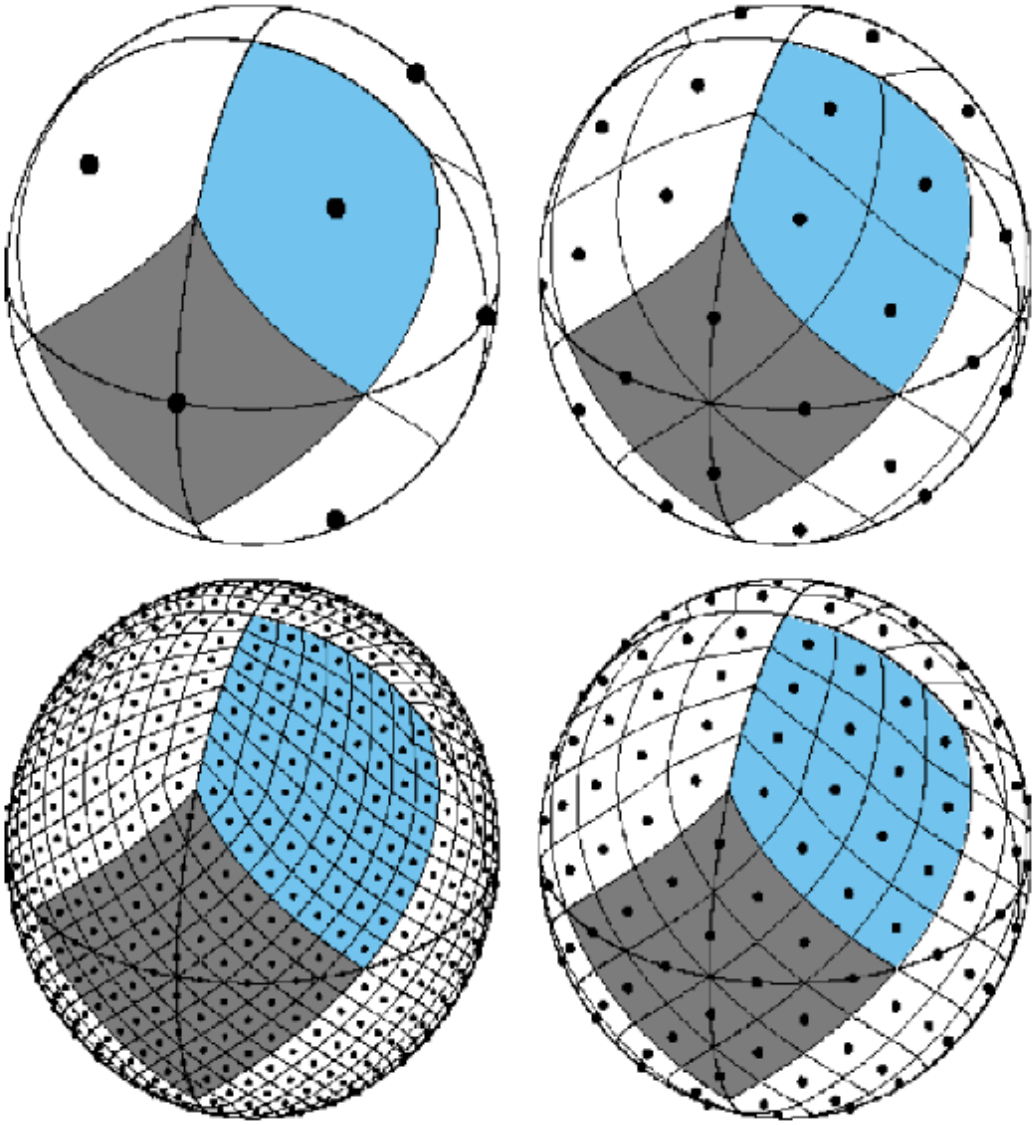


Figure 5.1: HEALPix maps "pixels" of N-side 1, 2, 4, 8

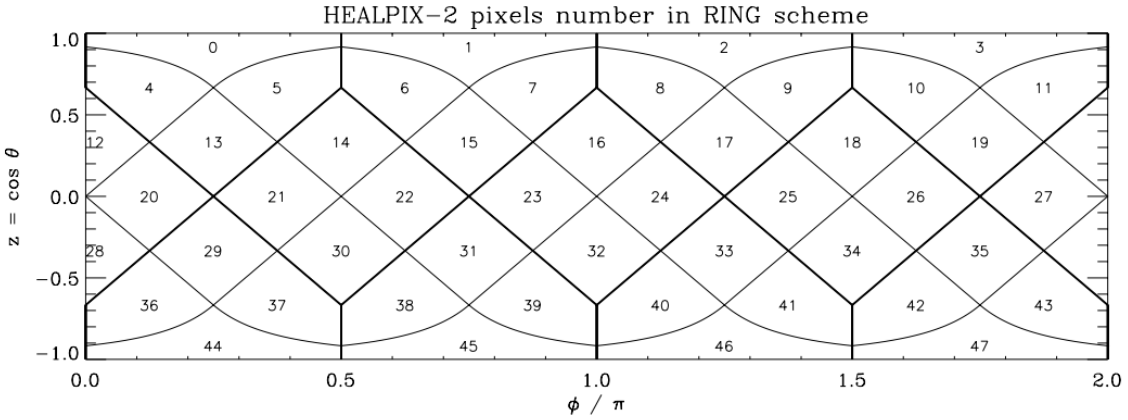


Figure 5.2: A flattened HEALPix full-sky map with the pixels counted in nest format and labeled (N-side = 2)

maps, the tSZ maps and the kSZ maps

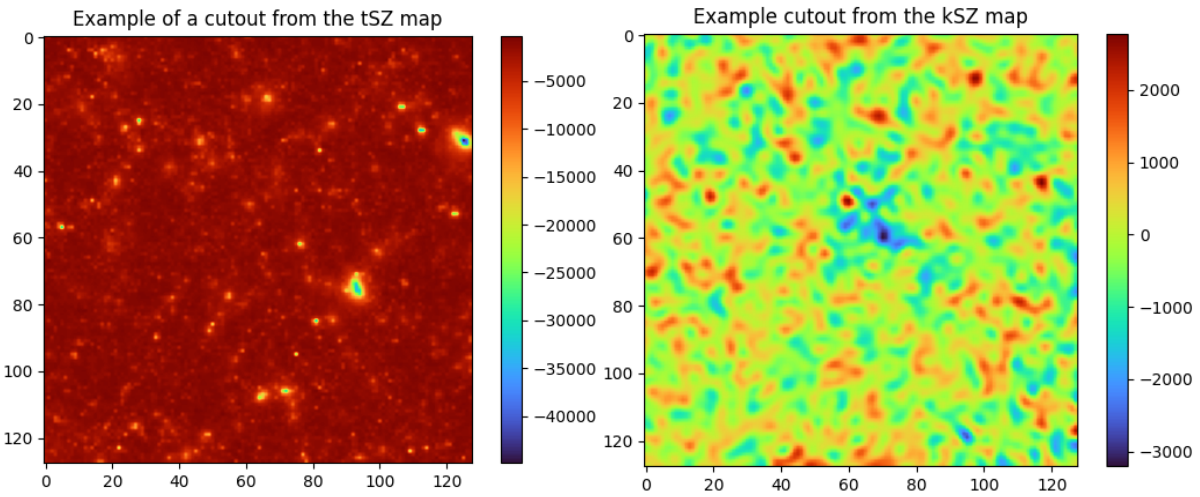
5.3 Reprojection of Components with Point-Like Sources

For the CIB and Radio Galaxy components, sources are identified on the full-sky map using star-apperture photometry. These coordinates are then found in the cutouts, and a source of the known flux is placed there. This procedure is followed to preventing ringing of the point-like sources upon reprojection, rotating or cutouts.

5.4 Reprojection of Components with Extended Sources

The tSZ, kSZ and κ map patches are formed by projecting the middle latitudes onto a Cartesian system, and then making cutouts from this long patch.

Each patch, before being fed as a training example to the GAN, may be randomly augmented to increase the size of the dataset. This is done by random vertical/horizontal flips, or rotations by 90°. While this does not give the model the same amount of information as a brand new image, it may introduce the information that the parity of the image, or the orientation do



(a) (b)

Figure 5.3: Example cutout of the components with extended sources (a) tSZ effect (b) kSZ effect. NOTE: These are unscaled images from the HEALPix maps. They must be scaled to be in units of μK

not matter, only the sizes of the structure in it.

5.5 Normalization

The samples are then normalized to speed-up and stabilize training. The normalization accounts for the fact that the probability distribution function(pdf) of the foreground components come from different distributions (i.e., the tSZ, CIV and Radio Galaxies are roughly Poisson distributed, but κ and kSZ are closer to Gaussian distributions), and that the dynamic ranges are drastically different between components. The DCGAN-WP has been shown in this project to learn smaller dynamic ranges faster and more easily than larger ones[2]. The κ maps range between (-1.5, 1.5) but the Radio Galaxies map range between (0, 852)μK.

To convert the pdfs to the Standard Gaussians, the Poisson distributed components (tSZ, CIB, Radio Galaxies) are altered pixelwise such that the

[2]For the Gaussian Blob datasets, we saw that if the images were scaled by an order of magnitude, the GAN prefers converging to a 0 solution than to learn large weights and biases. Further, the GAN learns Gaussian pdfs more easily than the other distributions.

value of each pixel with initial value x is changed to:

$$x' = f(x) = sign(x) \ln [\frac{|x|}{\sigma_x} + 1] \tag{5.2}$$

Here, the $sign(x)$ gives the sign of x, σ_x gives the standard deviation of the pixel pdf of the map.

To tackle the dynamic range, there is an additional scaling added to every pixel with value x':

$$x" = g(x') = \frac{(x' - \mu_{x'})}{\sigma_{x'}} \tag{5.3}$$

Here, $\mu_{x'}$ is the mean of the pixel value x' and $\sigma_{x'}$ is the standard deviation of the same.

These normalization steps help convert the pdf of the components closer to the standard gaussians (we still want to learn the non-Gaussianities, but they are more controlled now).

Chapter 6

Results on the Blob-Datasets

The first ML model (the DCGAN-WP with generator architecture in Fig. 4.3) was implemented. To understand how it works, we try to replicate images from simpler datasets that we think we understand better. These are:

- **Single Gaussian Blob**: There is a single Gaussian blob placed randomly on the image with its center lying anywhere on the image with a uniform probability. This can be thought of as a single point source convolved with a Gaussian point-spread function/Gaussian-beam. Ideally, there would be 2 terms to encode the entirety of the image - the X and Y coordinates of the center of the blob.

- **10 Gaussian Blobs**: There are exactly 10 Gaussian blobs placed throughout the image (centers are again uniformly chosen), with some overlapping blobs causing distortions that may be interpreted as multiple separate circular blobs, or a single abnormally-shaped blob. All blobs have the same intensity though. In addition, each image was multiplied by a scaling factor that made the maximum value of any pixel in the image = 1. Ideally, there would be 20 terms to encode the entirety of this image - the X and Y coordinates of the centers of all the blobs.

- **50 Gaussian Blobs**: There are exactly 50 Gaussian blobs placed throughout the image (centers are again uniformly chosen), with some

overlapping blobs causing distortions that may be interpreted as multiple separate circular blobs, or a single abnormally-shaped blob. However, each image is not scaled to have a maximum of 1, but is allowed to take higher values as dictated by the overlapping of blobs.

- **1000 Gaussian Blobs**: There are exactly 1000 Gaussian blobs placed throughout the image (centers are again uniformly chosen), with some overlapping blobs causing distortions that may be interpreted as multiple separate circular blobs, or a single abnormally-shaped blob. All blobs do not have the same intensity, but rather, their intensity distribution is given by a folded-Gaussian and a threshold was applied to the intensity of each blob. Further, each image is also not scaled to have a maximum of 1.

The training was chosen to be stopped when the losses of the discriminator and generator individually appeared to stop changing. This was defined as the mean of the past 50 losses being within 10% of the mean of the 50 losses before it. Defining convergence to a stronger threshold (e.x. 1% or 0.1% convergences) will no-doubt make these models better, but this was chosen as a proof of the concept.

6.1 Outputs on Single Gaussian Blob Dataset

To analyse the generated images, we check a number of statistics on them. These include visually checking the images, comparing the histogram generated by comparing the pixels in 128 images (which gives a total of $128*128^2 = 2,097,152$ pixels to find the histogram. This large number is taken to discount the effects like large amplitude noises in the generated images, etc.

Next, we add the image pixel-wise in the X and Y direction. A truly Gaussian peak would appear Gaussian in both these directions. Since we are considering circularly symmetric Gaussian beams, these peaks should also share the same widths. Further, multiple beams at the same/closeby lo-

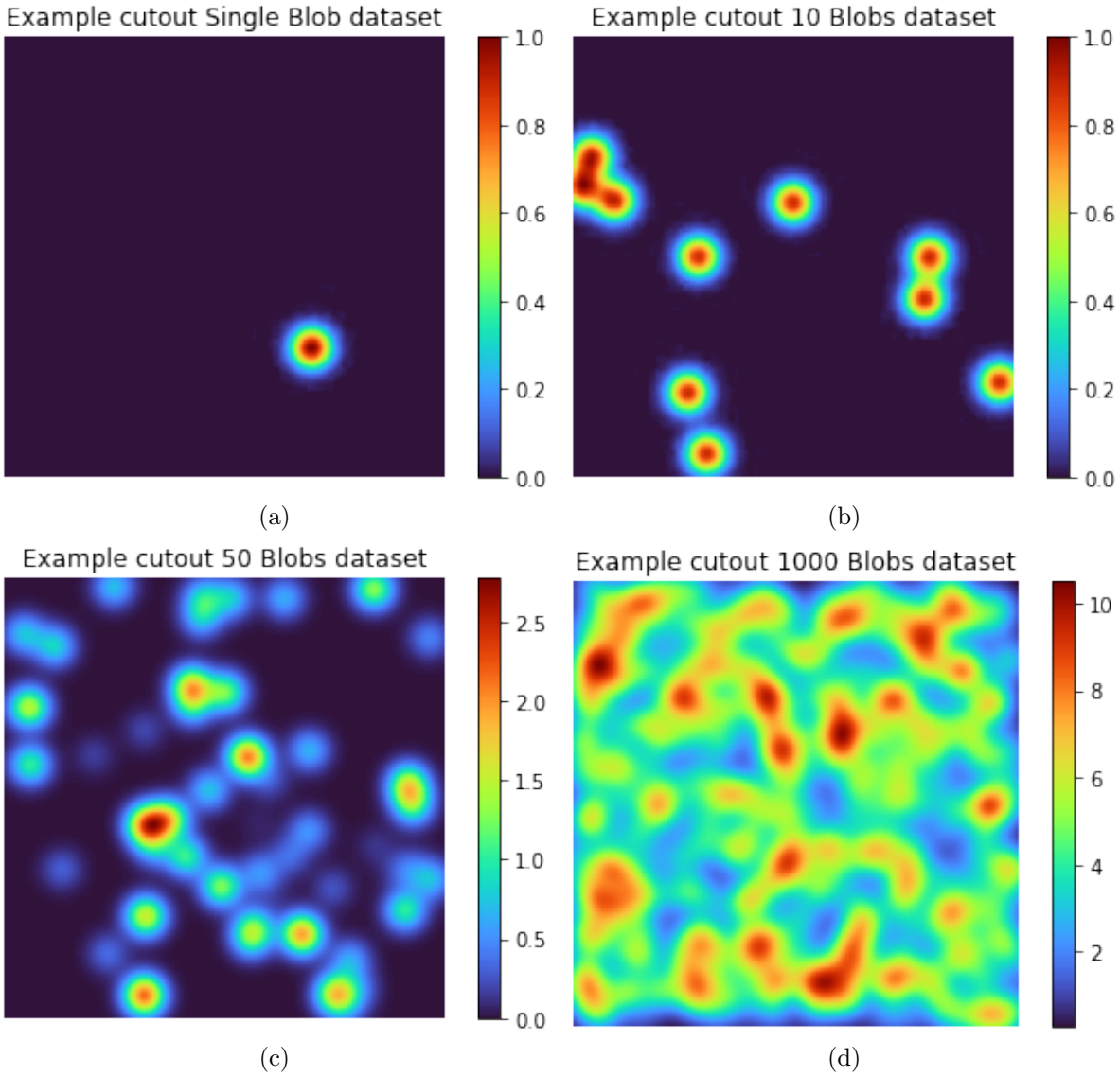


Figure 6.1: Examples from each of the datasets: (a)Single Gaussian Blob, (b)10 Gaussian Blobs, (c)50 Gaussian Blobs, (d)1000 Gaussian Blobs

cations can be distinguished by this method as they would appear as distinct local maximas.

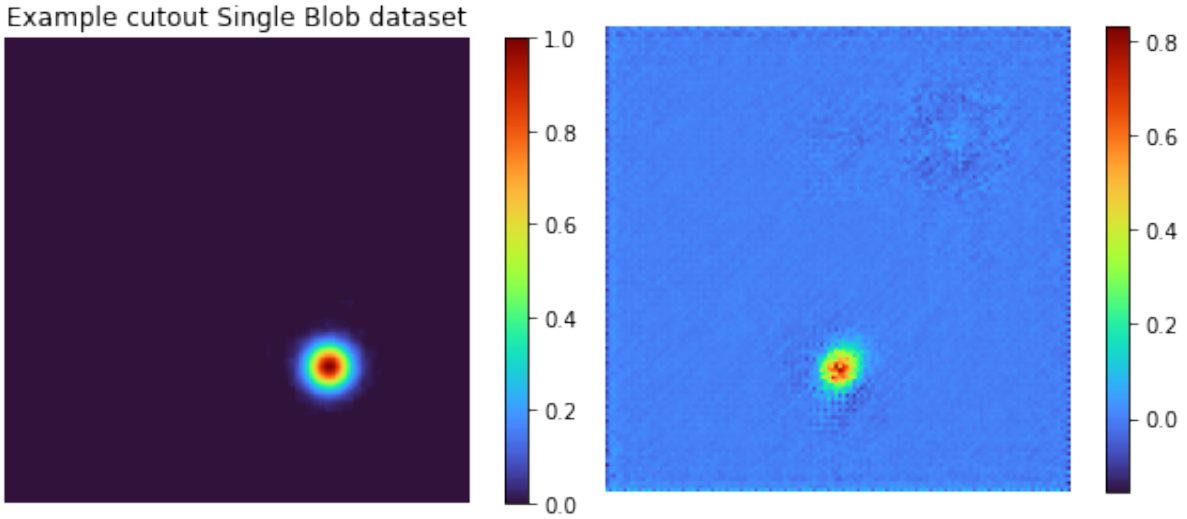
The next statistic of the images that we plot is a stacked version of the peaks. We find peaks in the image by comparing each pixel to its adjacent pixels to find the local maxima. Then, we take a 21pixel x 21pixel cutout around this peak. These cutouts are then stacked by taking an average value of each pixel (summing up the cutouts and then dividing by the number of cutouts). These must appear as radially-Gaussian blobs. Hence, we also plot the radial intensity variation of the peak average thus found. Any tendency of the model to make distorted peaks will be clearly visible in this statistic.

The last statistic we compare is the spatial Fast-Fourier transform (2D FFT), or alternatively, the spatial power-spectrum of the image (which is simply the square of its 2D FFT). While the FFT of a single Gaussian blob is still a Gaussian, multiple blobs add some artifacts of their own.

For the model trained on the single Gaussian Blob dataset, the visual comparison of the images is shown in Fig. 6.2. While it has learnt to make a Gaussian blob, it occasionally ends up making 2 blobs with different intensities. It also has a wider dynamic range than the input image which lies strictly between 0 and 1. The histogram is shown in Fig. 6.3. It reflects this difference in dynamic range. The marginal sum in the X and Y directions is shown in Fig. 6.4. This image is also an example of the kind of image that has 2 blobs in it. The stacked peaks are shown in Fig. 6.5. The 2D FFT is shown in Fig. 6.6. Note that the colorbar in the 2D FFT has different color-schemes for the input and generated images.

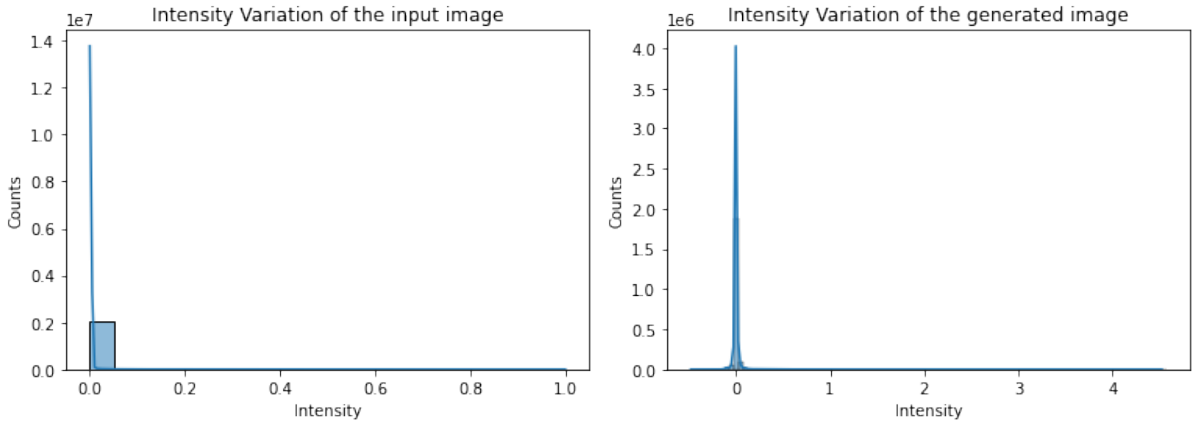
6.2 Outputs on 10-Gaussian Blob Dataset

A generated sample from the model trained on the 10-blob dataset is shown in Fig.6.7. Clearly, it has not learnt to normalize the images to make a maximum of 1. We hypothesize that this is because normalizing the image



(a) (b)

Figure 6.2: Examples from the (a)input image and (b) generated image



(a) (b)

Figure 6.3: Intensity histogram of the (a) input image, (b) generated image

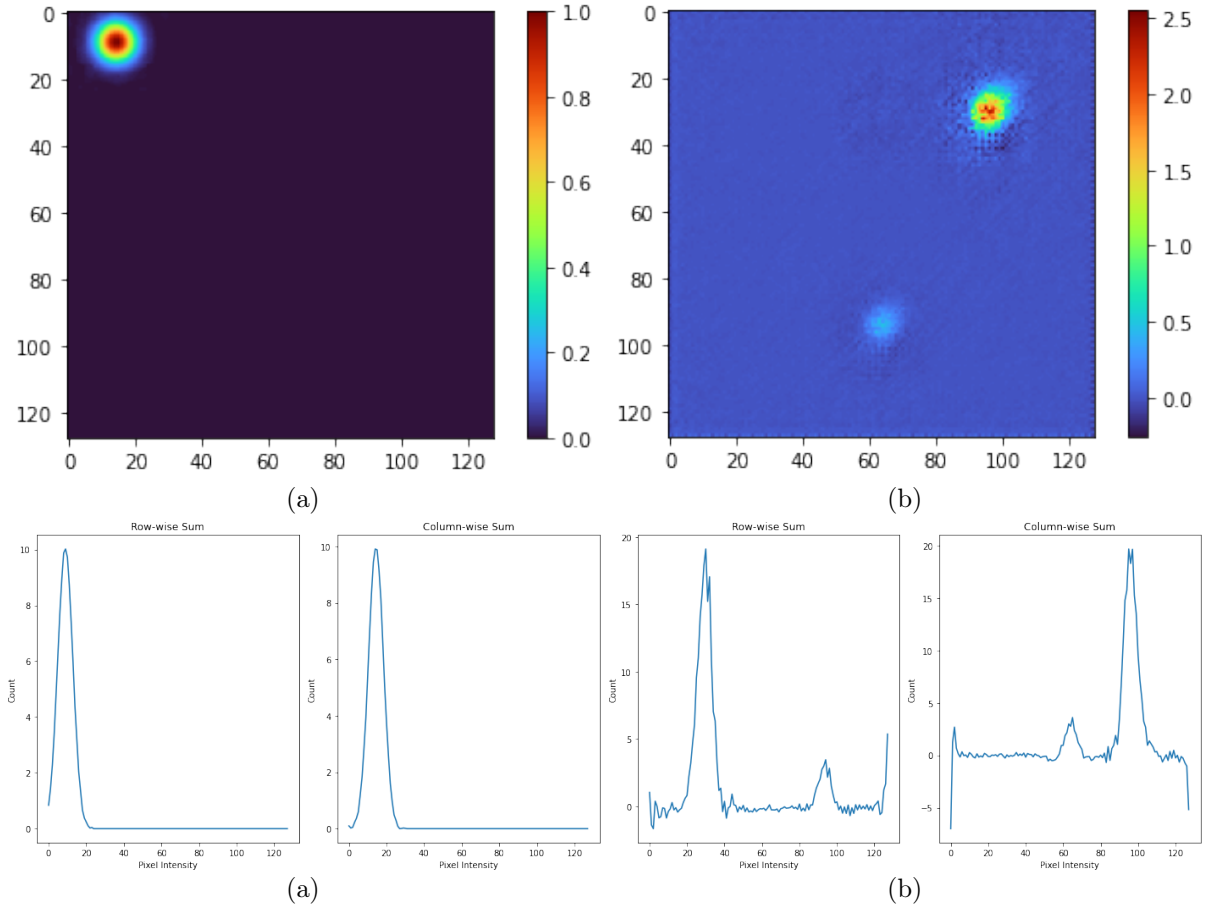


Figure 6.4: Example marginal sum (row and column sum) of (a),(c) input image, (b),(d) generated image

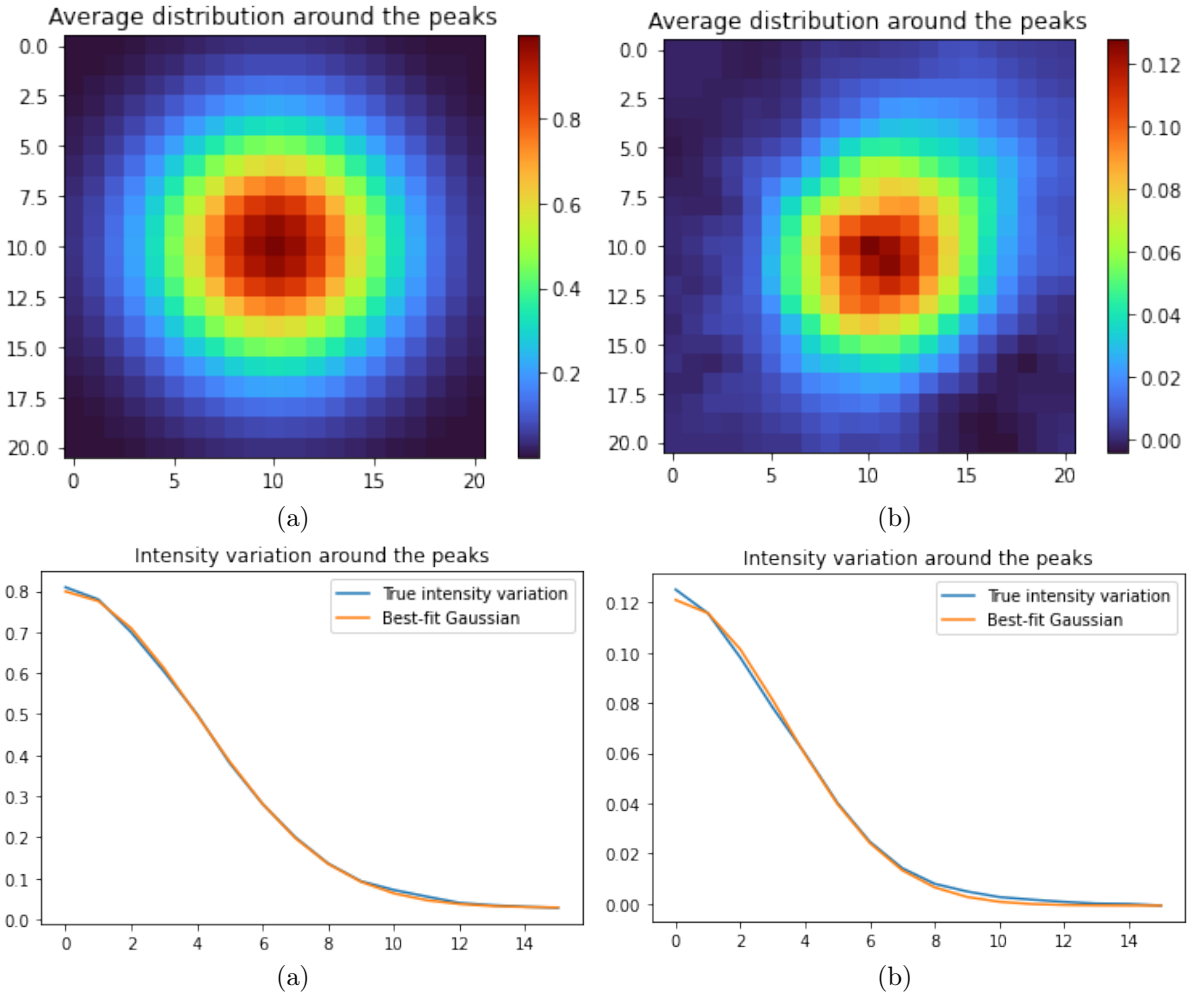


Figure 6.5: Stacked (sum) peaks by centering the local maxima and the radial distribution for (a),(c) input images and (b),(d) generated images

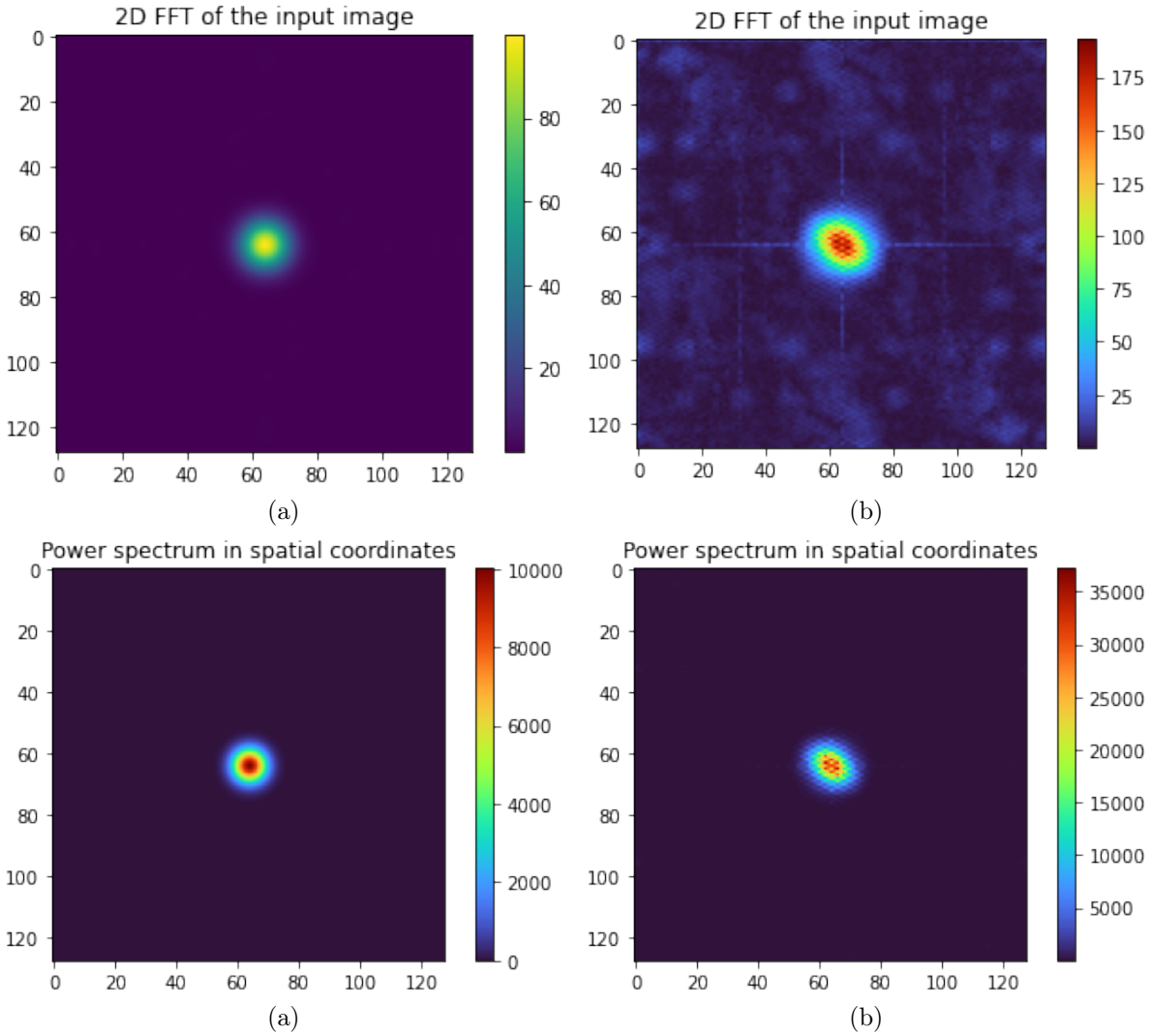
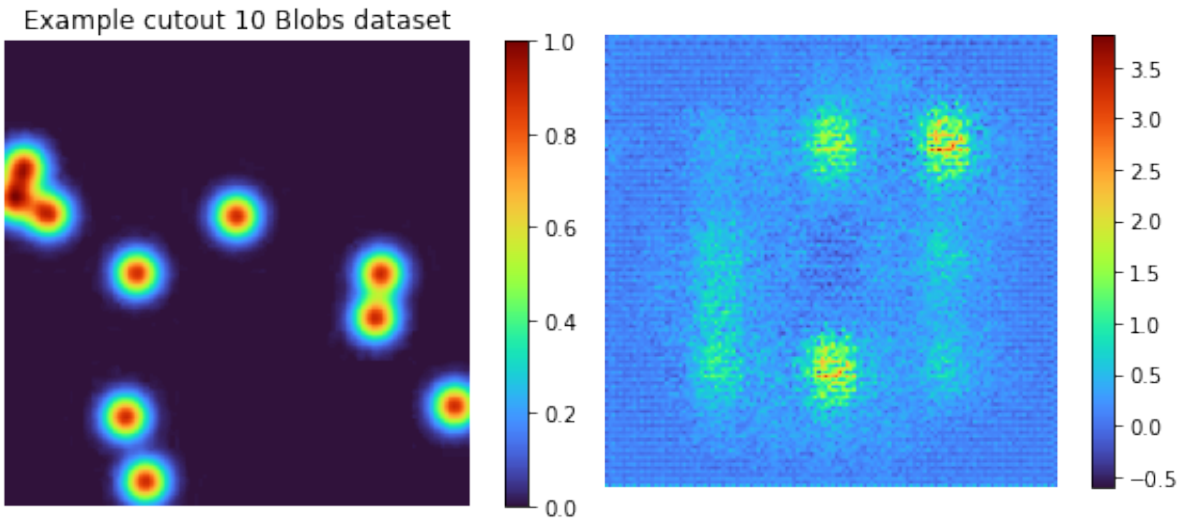


Figure 6.6: 2D FFT and spatial power-spectrum of (a),(c) input image and (b),(d) generated image



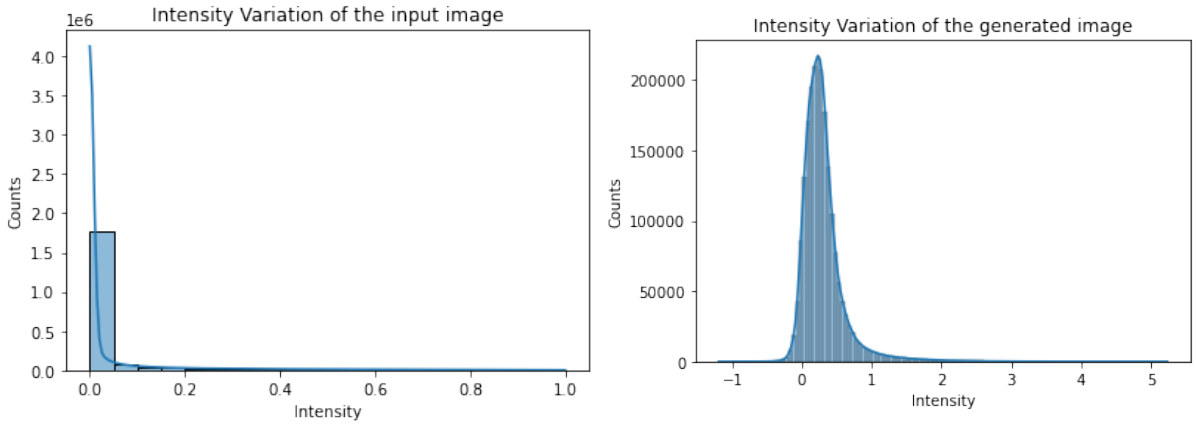
(a) (b)

Figure 6.7: Examples from the (a)input image and (b) generated image

is a statistic that is not localized to one part of the image unlike the blobs. This may make it a harder aspect of the image to learn than we thought. The histogram is shown in Fig. 6.8. While there is an asymmetry with the sharp edge on the left, it still has a different dynamic range. This could be due to the fact that this model has not learnt to normalize the image. Its marginal sum in the X and Y directions is given in Fig. 6.9. The stacked sum of the peaks is shown in Fig. 6.10. Its power-spectrum and 2D FFT is shown in Fig. 6.11.

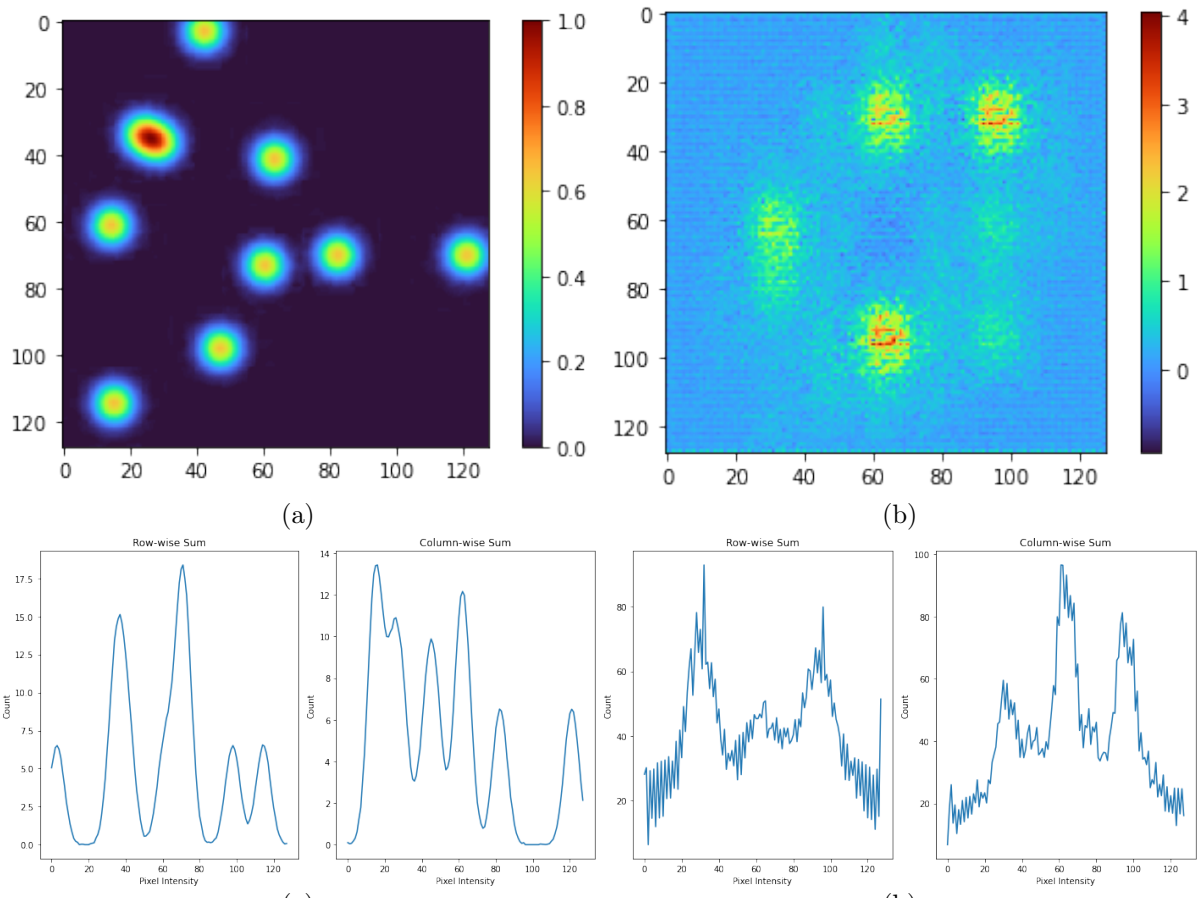
6.3 Outputs on 50-Gaussian Blob Dataset

An example image from the generated 50-Blob dataset is shown in Fig. 6.12. The model learns the non-Gaussianity in the probability-distribution function as seen in Fig. 6.13. The marginal sums of the image in the X and Y direction are shown in Fig. 6.14. The stacked images of the peaks as seen in Fig. 6.15 have a heavy tail, which is due to the fact that they tend to be distorted, and not perfectly Gaussian. The 2D FFT and power spectrum are shown in Fig. 6.16.



(a) (b)

Figure 6.8: Intensity histogram of the (a) input image, (b) generated image



(a) (b)

Figure 6.9: Example marginal sum (row and column sum) of (a),(c) input image, (b),(d) generated image

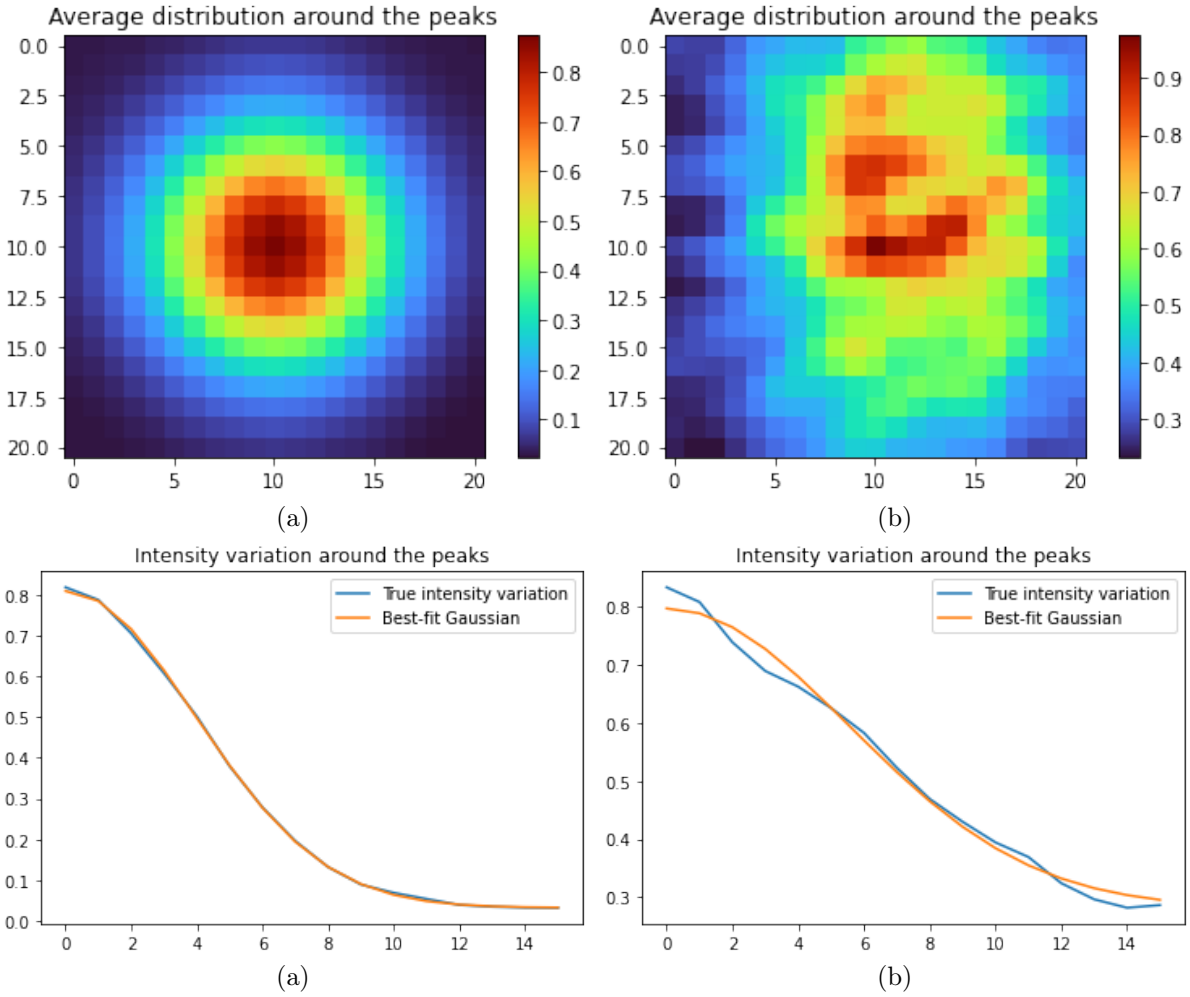


Figure 6.10: Stacked (sum) peaks by centering the local maxima and the radial distribution for (a),(c) input images and (b),(d) generated images

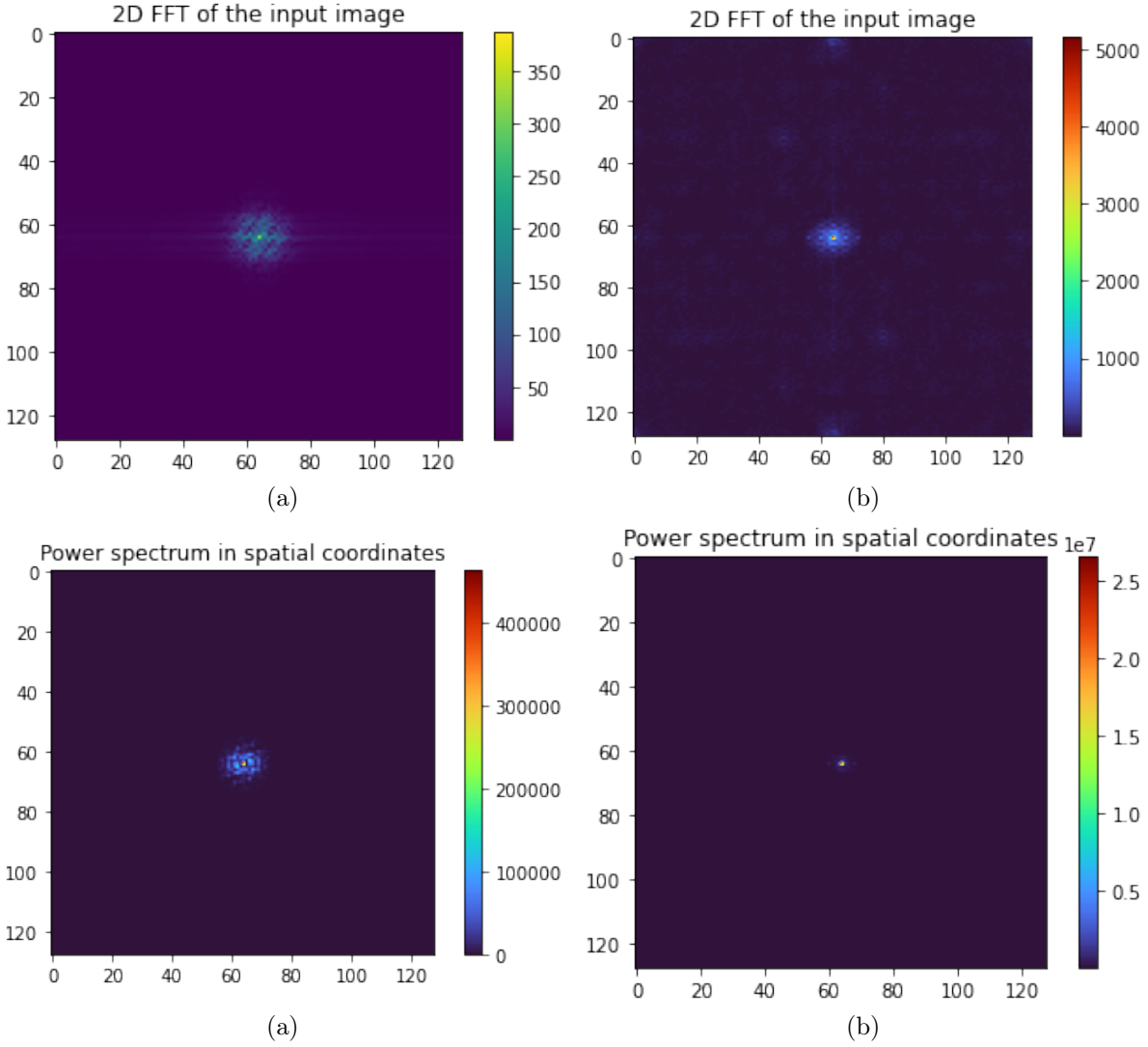
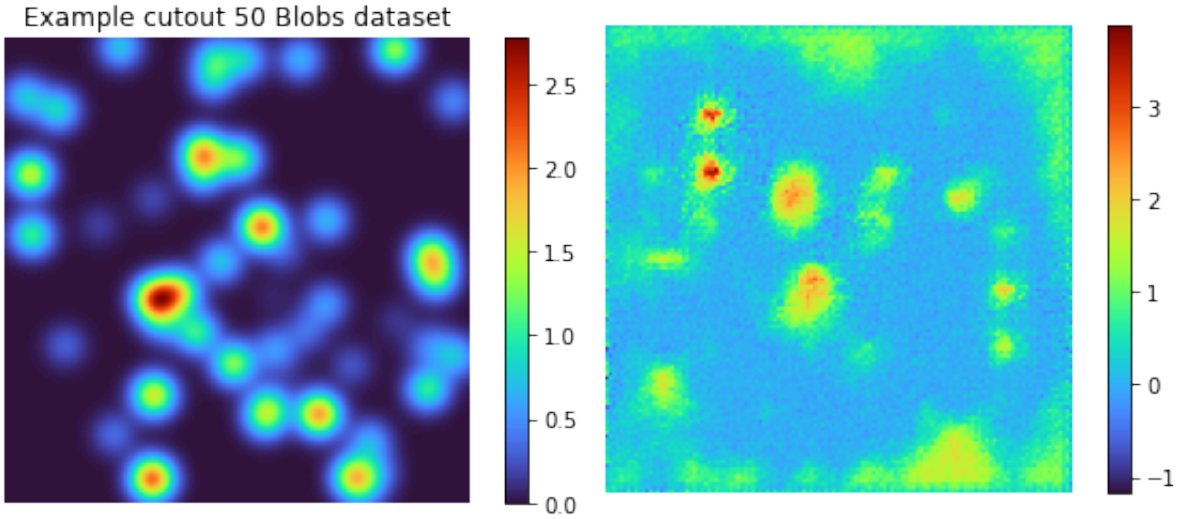
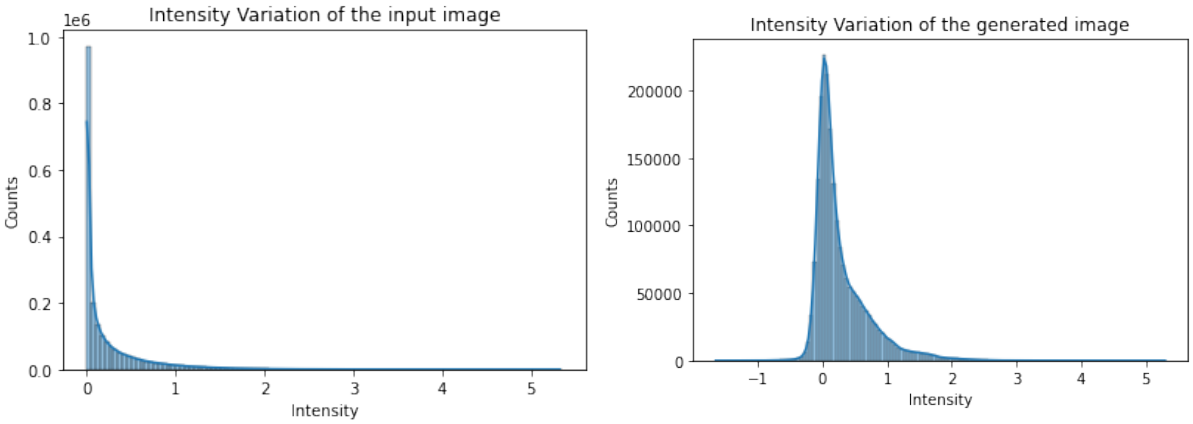


Figure 6.11: 2D FFT and spatial power-spectrum of (a),(c) input image and (b),(d) generated image



(a) (b)

Figure 6.12: Examples from the (a)input image and (b) generated image



(a) (b)

Figure 6.13: Intensity histogram of the (a) input image, (b) generated image

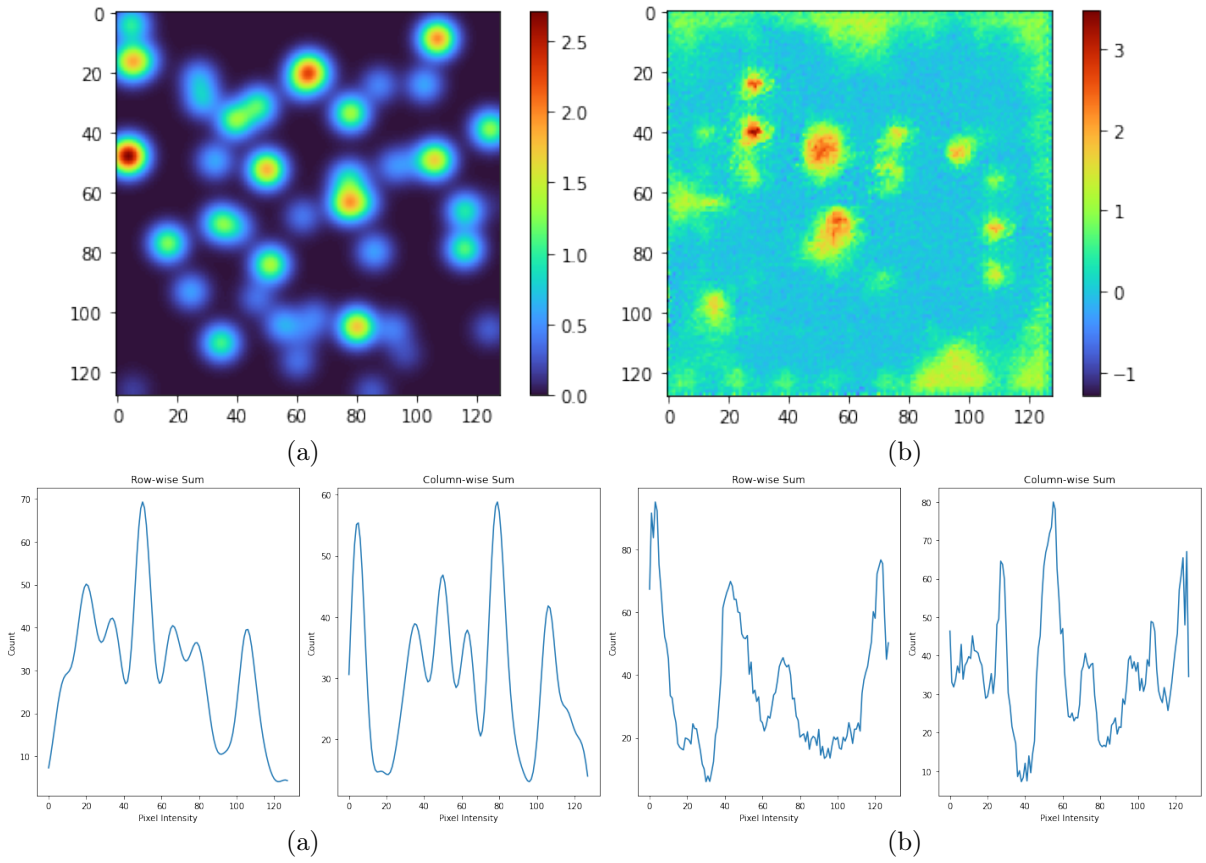


Figure 6.14: Example marginal sum (row and column sum) of (a),(c) input image, (b),(d) generated image

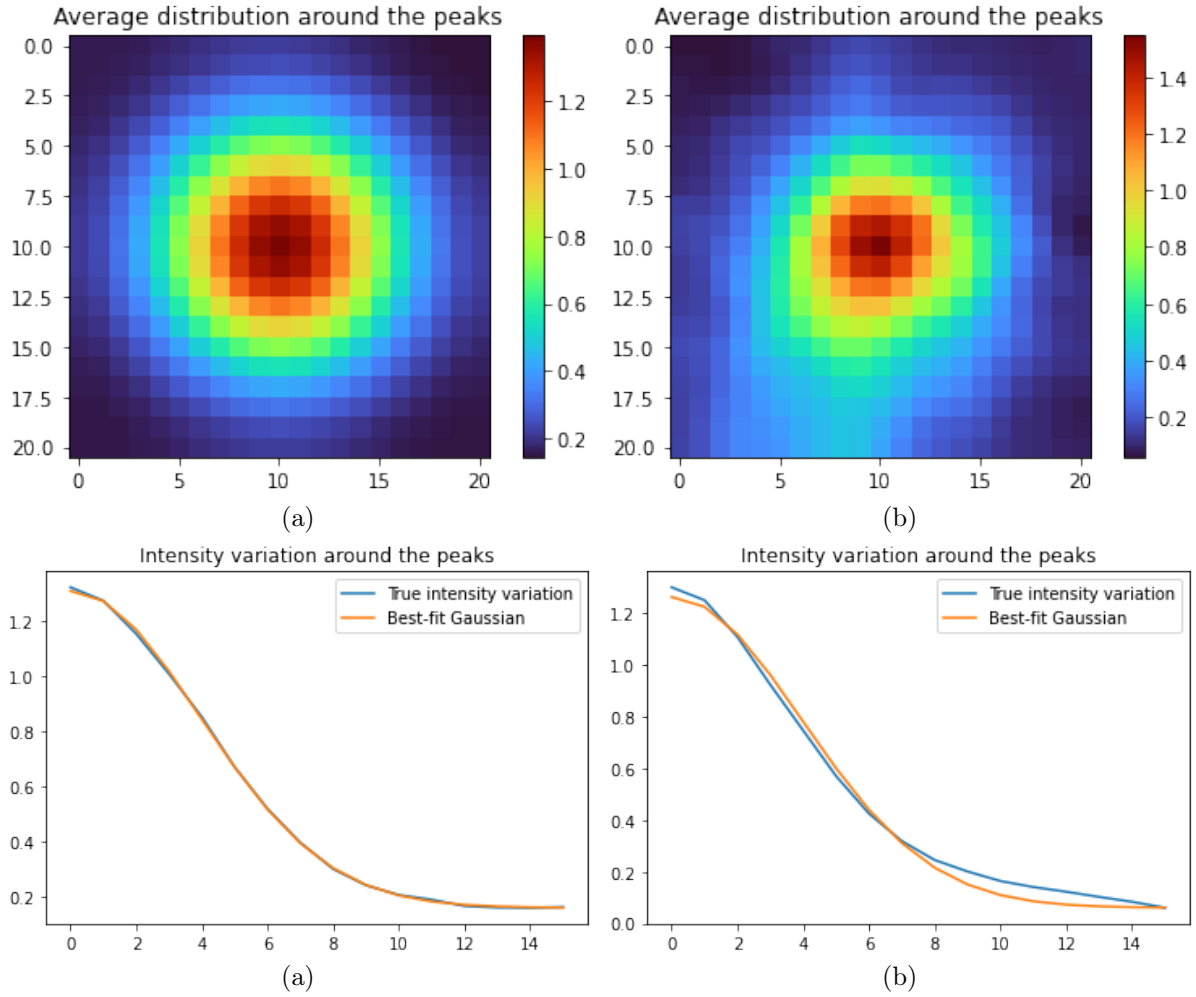


Figure 6.15: Stacked (sum) peaks by centering the local maxima and the radial distribution for (a),(c) input images and (b),(d) generated images

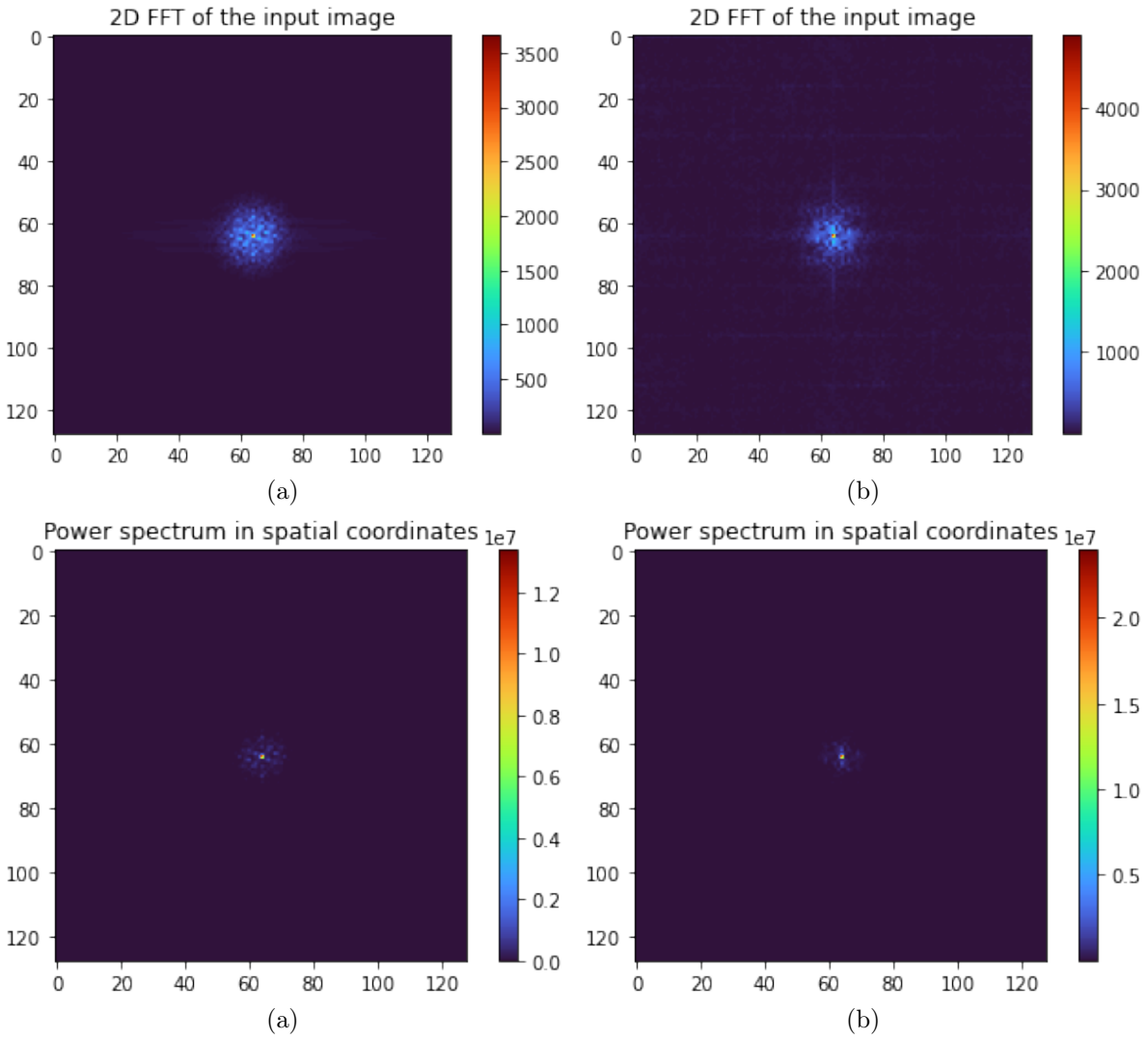
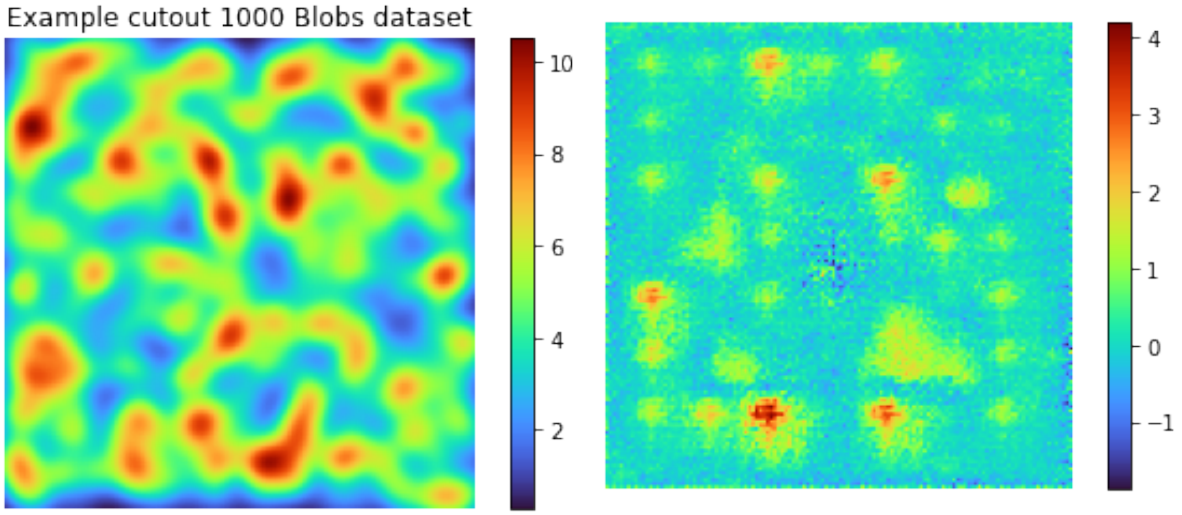


Figure 6.16: 2D FFT and spatial power-spectrum of (a),(c) input image and (b),(d) generated image



(a) (b)

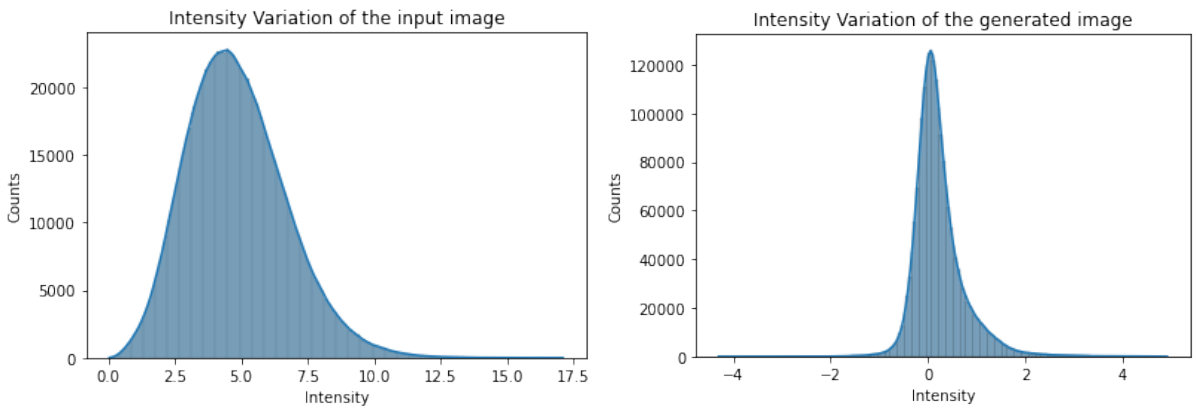
Figure 6.17: Examples from the (a)input image and (b) generated image

6.4 Outputs on 1000-Gaussian Blob Dataset

As seen in the visual example images in Fig. 6.17, the generated images of the model trained on the 1000-Gaussian Blobs dataset have some level of salt-and-pepper-noise, and have a lower dynamic range than the input images. However, the machine learns the non-Gaussianity in the probability-distribution function, as seen in Fig. 6.18. The marginal sums are shown in Fig. 6.19, and the stacked peaks in Fig. 6.20. The 2D FFT and the power-spectrum are more spatially spread out as compared to the input images as seen in Fig. 6.21

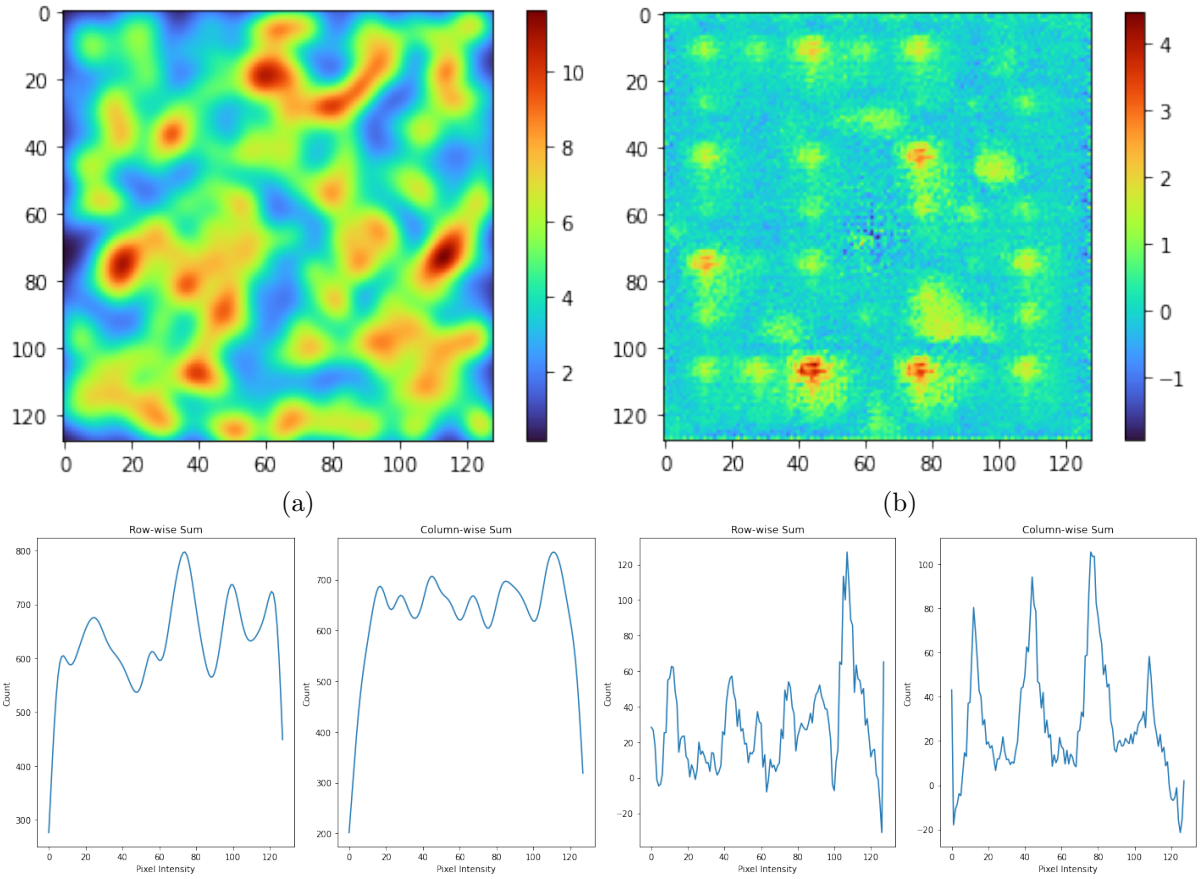
6.5 Alternative to Finding Peaks

It was noted that due to the fact that peaks were found by a method of finding local minima that only takes into consideration the immediate neighbours of every pixel, these statistics may introduce a certain bias on stacking, especially on images with high levels of white noise. This would result in a higher peak in the middle (distance 0 in radial variation), and may even lead to distortions while stacking the non-perfect Gaussian peaks. To counter



(a) (b)

Figure 6.18: Intensity histogram of the (a) input image, (b) generated image



(a) (b)

(a) (b)

Figure 6.19: Example marginal sum (row and column sum) of (a),(c) input image, (b),(d) generated image

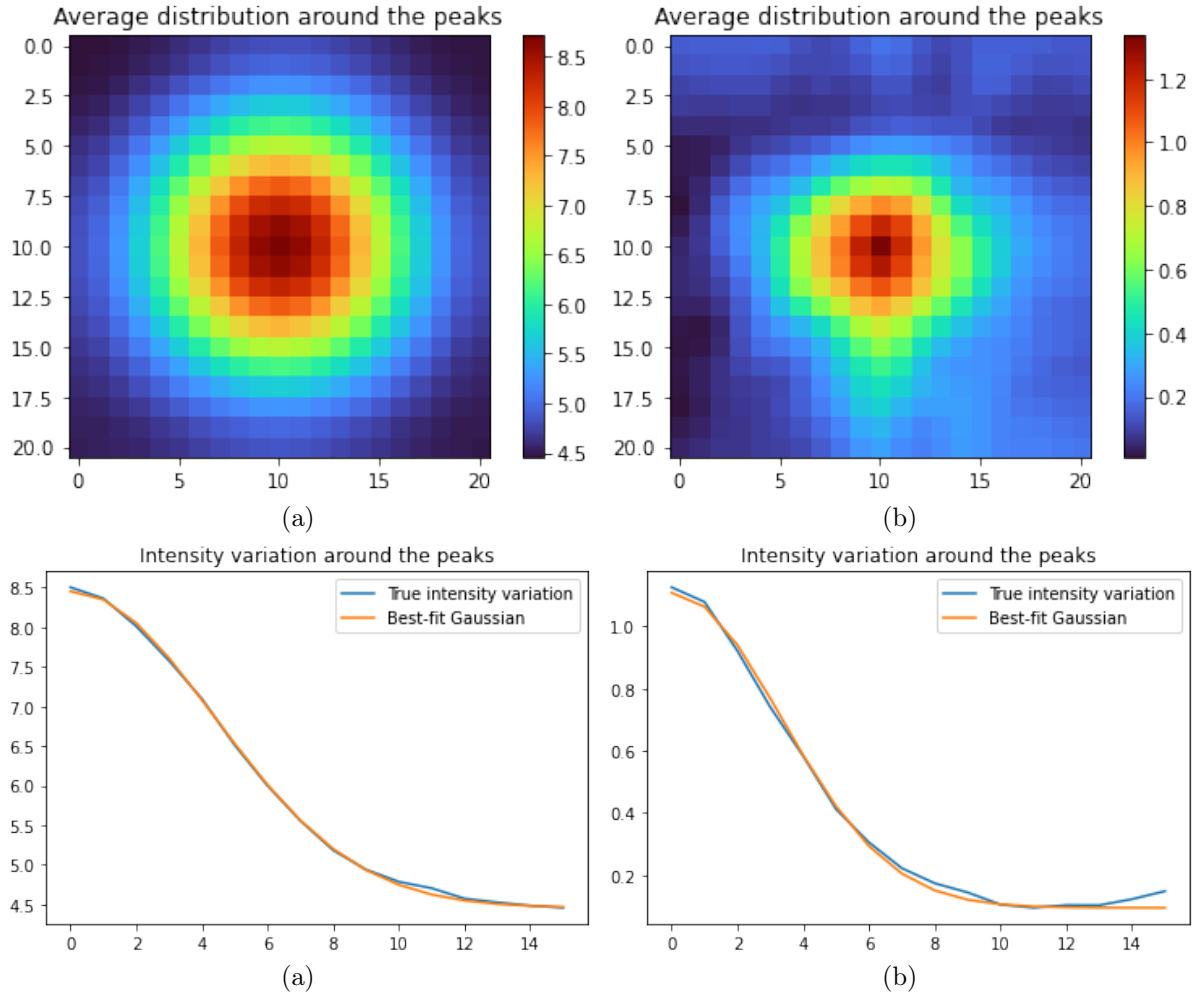


Figure 6.20: Stacked (sum) peaks by centering the local maxima and the radial distribution for (a),(c) input images and (b),(d) generated images

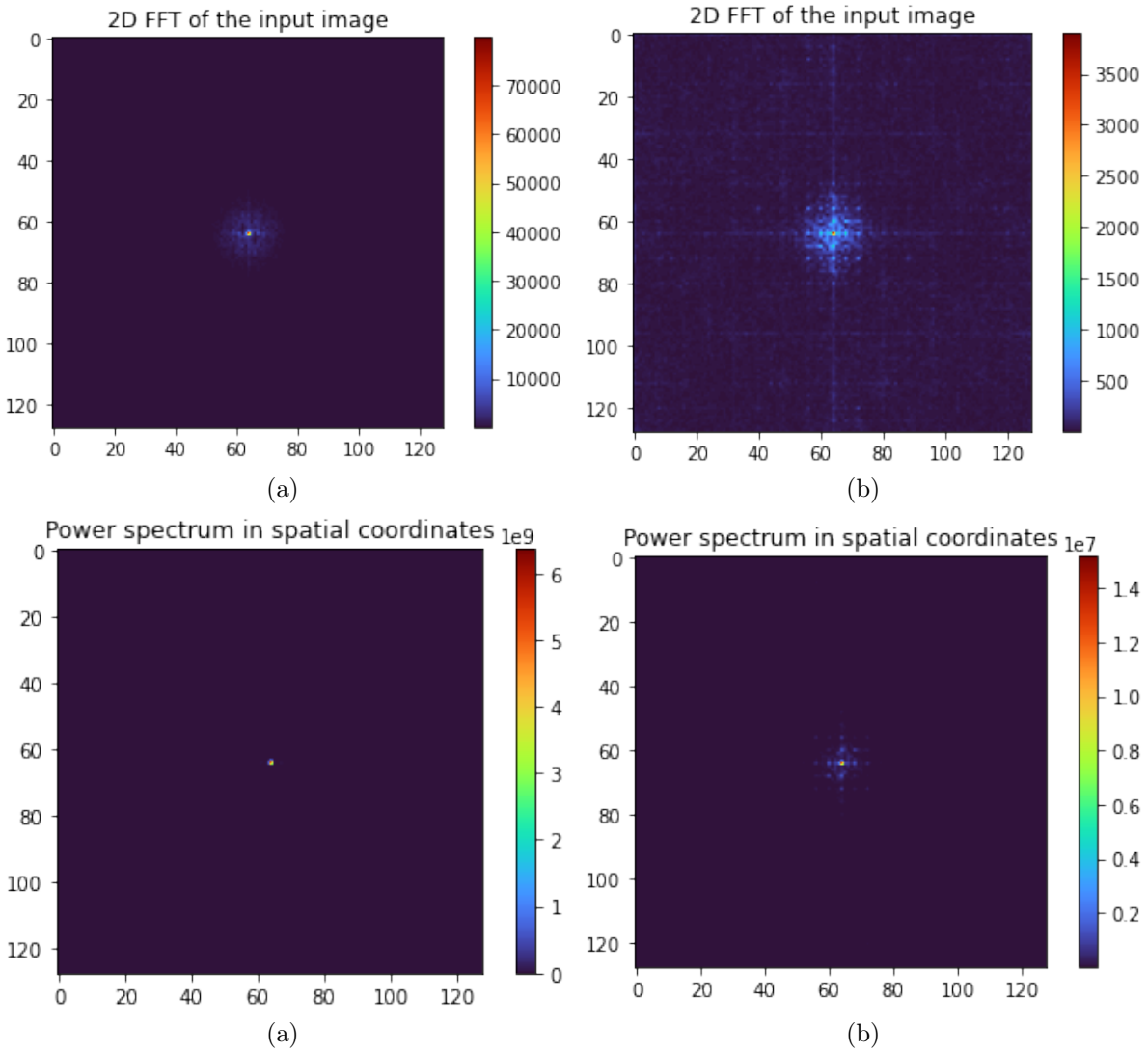


Figure 6.21: 2D FFT and spatial power-spectrum of (a),(c) input image and (b),(d) generated image

this, it was suggested to find peaks by convolving with a Gaussian beam of the same size as the telescope beam (in this case, the size of the input Gaussian blobs), and finding maximas to subsequently remove from the image. This is similar to the method used to find stars/sources in the night-sky images. However, this method was not implemented in this project.

6.6 Transfer Learning

Since all the datasets have Gaussian blobs of the same size, we used Transfer Learning from one dataset to the next to speed up training.

Transfer Learning is the idea in Machine Learning to reuse the weights and biases of a pre-trained problem for another similar problem, and updating them for the current problem. It exploits the knowledge already learnt (in this case, we suspect it is the ability to make Gaussian Blobs), and then refines it to learn the details of the current problem. It not only reduces the time required for training, but also the need for more data-points (images to train on).

After achieving a model that can make a single blob, we used this model, and trained it to make multiple blobs.

Chapter 7

Future Ideas

In the future, after the implementing all 3 models in the methodology discussed, they ought to be trained on simulation based on more recent updates of the cosmological constants. The S10 simulations are based on the WMAP-5 constants, but WMAP-9 and Planck results lead to different values of the relevant parameters such as the Hubble constant.

After this, the simulation outputs can be used to probe the relation between these foregrounds such as the distinct correlation between the CIB and the tSZ maps further.

Yet another interesting thing to probe in the future is to test other kinds of bispectra on the simulated maps such as the scattering coefficient, to test how well the machine has learnt to replicate the maps. This would help us understand if these maps are really reliable sources to study the cosmology of the maps they have been trained upon. If they do, then perhaps we can explore the weights and biases themselves to understand more physics.

Another avenue to explore is the error bars associated with results of these machine learning simulations. Analytic procedures have well-studied error-bar calculations, but can the same be said about the ML-generated results? Can other kinds of scores on GAN outputs such as the Coverage Metric, average log likelihood, KID score, etc. be used to calculate the same?

Chapter 8

Variational Auto-Encoders (VAEs)

Over the course of this project, we learnt that the GANs are very sensitive to the hyperparameters such as training learning rate, the training optimizer, or the size of the kernels ued in the model architecture itself. This led us to explore other generative models such as Variational Auto-Encoders (VAEs). Ultimately, however, we realized that GANs do provide us with better-looking images than the VAEs if all the hyperparameters are tuned well[1]

Variational Auto-Encoders are a class of probabilistic generative models. They try to infer the distributions of the latent variables representing the observed data. The encoder, instead of mapping input images to the latent space (which is chosen to be 'n' dimensional, as a hyper-parameter of the model), maps input images to distributions of latent-space variables. The red box returns a single sample point from the (normal) distribution. This point (called the latent vector) is passed to the decoder, which then generates an image similar to the input image. However, in describing the distributions output from the encoder, matters can quickly become intractable. The key is to assume a multivariate Gaussian distributions (which simplifies the math, while easily mapping a large subset of distributions). The encoder must then simply "encode" each input image into a multi-dimensional mean and

[1]For context, the Han et.al paper uses 148 hyperparameters in their entire process, each of which may have been manually tuned to achieve their results. There is also the possibility to use some computationally optimizing technique over these hyperparameters, or even applied another ML model for it. The latter would be the domain of Auto-ML.

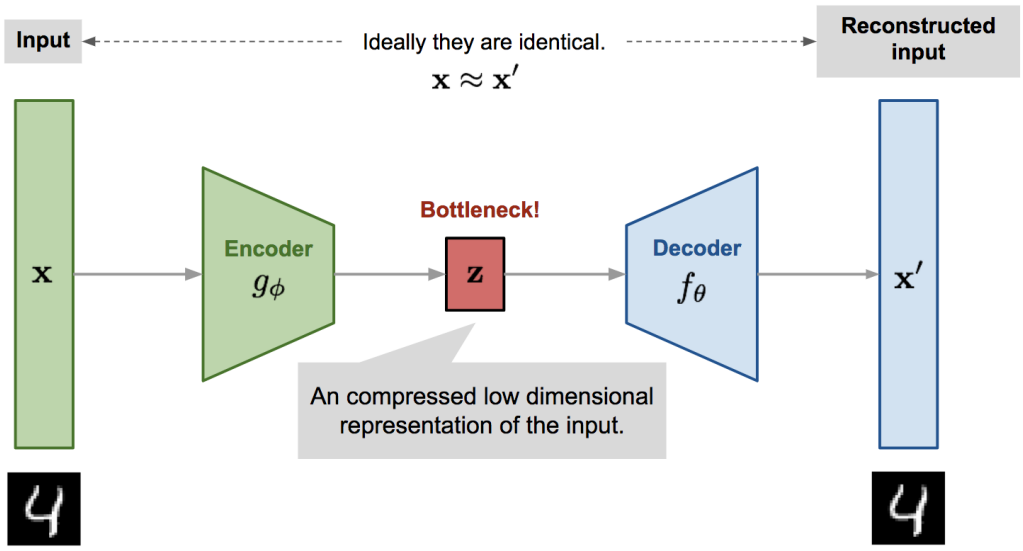


Figure 8.1: The general pipeline of a VAE

variance vector (the dimensions of which are at the choice of the engineer designing this machine).

While sampling an element of the distribution encoded by the encoder, we use a normal random vector with mean 0, variance 1. We then convert this into a vector from the encoded distribution as:

$$\mathbf{X} \sim \mathcal{N}(0, I)$$

$$\rightarrow \mathbf{Z} = \sigma\mathbf{X} + \mu \sim \mathcal{N}(\mu, \sigma^2 I)$$

(In our implementation, we ensure our encoder gives us an output of $\log \sigma^2$ instead of σ to simplify calculations and avoid non-linearities.)

Using the sample of $\mathbf{Z}$ (using the above trick, which is called the "reparameterization trick"), the decoder uses its CNN model to reproduce the desired image.

In the generation of images with many classes, we want to separate out different classes in the distribution space. To understand this, we plot the variation of each latent variable, as encoded by the encoder. A trained model, should start differentiating the classes as in Fig. 8.2. This model assumes a 2 dimensional latent vector for visualization clarity, but it need not be the

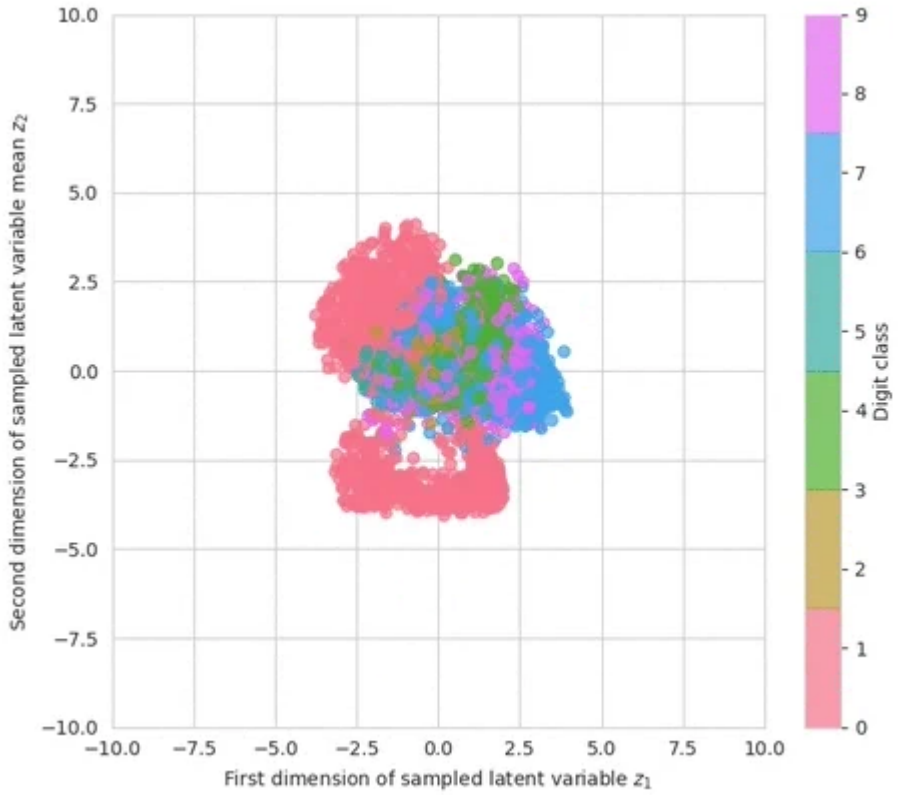


Figure 8.2: Class-wise separation of encoder output variation
Disclaimer, this image was taken from this link

case.

Further, I trained a model on the Fashion MNIST dataset which contains images of various clothing items like shoes, shirts, pants. I then created a map, where each grid-box contains an image generated by a sample of the distribution by varying the sample variance in 2 dimensions of the 10 latent variables. In a trained model (Fig. 8.6, the adjacent boxes of this map appear similar, but the gradual change across the map is apparent. On the top right corner of the variance map shows images of shoes, and the bottom left shows images of shirts. Clearly, this difference has been consistently developing with training. Initially, no matter the distribution, all the images looked alike, but over time, these differences emerge. Similar variations occur at other pairs of dimensions as well.

Moving on to the loss function used.

At any given snapshot / instance of training, there is a fixed relationship

between the latent vector $\mathbf{z}$ and the generated image $\hat{x}$. This is the decoder, and there are no stochastic layers in the decoder.

The source of the stochasticity is the red box in Fig. 8.1. This box takes in a distribution (further represented by a parameter λ), and generates a sample latent vector $\mathbf{z}$ from this multivariate (here Gaussian) distribution.

We want to ensure that there is least loss of information across this stochastic process. Since the parameters of the encoder and decoder are separately updated (in the sense that the update of a parameter 'a' assumes that another parameter 'b' ($b \neq a$) is fixed), they collectively minimize the loss function. This method is in contrast to the GANs, which have the generator and discriminator work against each other. This also ensures that we will find an optimum mappings of images to distributions, and from samples in this distribution to generated images.

Let us denote the distribution "encoded" by the encoder for an input image x by a placeholderλ. For the Gaussian distributions, $\lambda = (\mu, \Sigma)$[2]. The probability of a latent vector output from the red box given the distribution λ is denoted as:

$$q_\lambda(\mathbf{z}|x) = \frac{1}{\sqrt{2\pi \prod_i \sigma_i^2}} \exp\{\sum_{i=1}^{n} -\frac{(z_i - \mu_i)^2}{2\sigma_i^2}\} \tag{8.1}$$

Since the latent vector is a random vector following a certain distribution, the output of the decoder acting on these latent vectors ($f_\theta(\mathbf{z})$ is also a random image following another distribution). Assume there is a true probability distribution $p(\mathbf{z}|x)$ of latent-vectors (given input images) that generates images with a distribution similar to the input images set.[3].

Thus, the notation is summarized as:

- ϕ parameterizes the encoder mapping (as this mapping evolves with

[2]The mean and Variance describe everything about the Gaussian distribution. Further, we assume that the components of the multivariate distribution are independent (covariance = 0), and replace the matrix Σ by a vector σ

[3]Ideally, these distributions would be the same for every input image of a certain class (the CMB dataset, or the Gaussian Blob dataset has only 1 class).

training)

- θ parameterizes the decode mapping (as this mapping evolves with training)
- x is an input image
- The encoder g_ϕ takes an input image x and computes a mean μ and (co-)variance Σ.
- λ parameterizes the Gaussian distribution. Thus, $\lambda = (\mu, \sigma)$
- $q_\lambda(y|x) = \mathcal{N}(y|g_\phi(x) = \lambda = (\mu, \sigma))$
- $\mathbf{z}$ is the latent variable, a single sample point drawn from the multivariate normal distribution q.
- The decoder f_θ takes the latent variable z and generates an image x'.
- $p(y|x)$ is the true desired distribution of the latent variable, at a fixed θ, or equivalently f_θ.

We want to find a distribution among our class of distributions (Gaussians with varying mean and variance) which most closely resembles this true distribution. Thus, we want to minimize the Kullback–Leibler (KL) divergence between $q_\lambda(\mathbf{z}|x)$ and $p(\mathbf{z}|x)$

$$D_{KL}(q_\lambda(\mathbf{z}|x) \,||\, p(\mathbf{z}|x)) \equiv \int_{\mathbf{z}} q_\lambda(\mathbf{z}|x) \log \frac{q_\lambda(\mathbf{z}|x)}{p(\mathbf{z}|x)} \, d\mathbf{z} \tag{8.2}$$

$$= E_q\{\log \frac{q_\lambda(\mathbf{z}|x)}{p(\mathbf{z}|x)}\} \tag{8.3}$$

$$= E_q\{\log q_\lambda(\mathbf{z}|x) - \log p(\mathbf{z}|x)\} \tag{8.4}$$

$$= E_q\{\log q_\lambda(\mathbf{z}|x)\} - E\{\log \frac{p(\mathbf{z}, x)}{p(x)}\} \tag{8.5}$$

$$= E_q\{\log q_\lambda(\mathbf{z}|x)\} - E\{\log p(\mathbf{z}, x)\} + \log p(x) \tag{8.6}$$

Thus, minimizing $D_{KL}(q_\lambda(\mathbf{z}|x) \mid\mid p(\mathbf{z}|x))$ is equal to minimizing $E_q\{\log q_\lambda(\mathbf{z}|x)\} - E\{\log p(\mathbf{z}, x)\}$.

We define the negative of this term as the Evidence Lower Bound:

$$\text{ELBO} = -(E_q\{\log q_\lambda(\mathbf{z}|x)\} - E_q\{\log p(\mathbf{z}, x)\}) \tag{8.7}$$

$$\boxed{\text{ELBO} = E_q\{\log p(\mathbf{z}, x)\} - E_q\{\log q_\lambda(\mathbf{z}|x)\}} \tag{8.8}$$

Thus, the loss used is ELBO loss.[4]

To calculate the loss more efficiently, we use a single sample Monte-Carlo estimate of the expectation over the decoder. We do this believing that the huge number of iterations allows us to freely use the laws of large numbers. Thus, even if the model takes a few bad steps because of a bad estimate, we will ultimately converge to the optimal value. However, since we are no longer using the KL divergence, this loss need not be convex. There may exist non-global minima.

We now try to see the benefit of using a Gaussian distribution:
Representing $\mathbf{z}$ as the encoder output distribution of input image x.

$$\log p(x|x') = \log p(x' \sim z) \tag{8.13}$$

$$= \log \left\{ \prod_i \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp \frac{-(x_i - \mu_i)^2}{2\sigma_i^2} \right\} \tag{8.14}$$

$$= \sum_i \left[\frac{(x_i - \mu_i)^2}{2\sigma_i^2} - \frac{1}{2}\log(2\pi) - \log \sigma_i \right] \tag{8.15}$$

This term is clearly easier to compute and learn across, since it no longer contains log terms over the mean, or the decoder.

.

[4] Calculated equivalently as:

$$\text{ELBO} = E_q\{\log p(x|\mathbf{z})\} + E_q\{\log p(\mathbf{z})\} - E_q\{\log q_\lambda(\mathbf{z}|x)\} \tag{8.9}$$

$$= E_q\{\log p(x|\mathbf{z})\} + E_q\{\log p(\mathbf{z})\} - E_q\{\log \frac{q_\lambda(\mathbf{z}|x)}{p(x|x)}\} \tag{8.10}$$

$$= E_q\{\log p(x|\mathbf{z})\} + E_q\{\log p(\mathbf{z})\} - \text{Cross-entropy between x and x'} \tag{8.11}$$

$$\tag{8.12}$$

These are terms we can now easily compute.

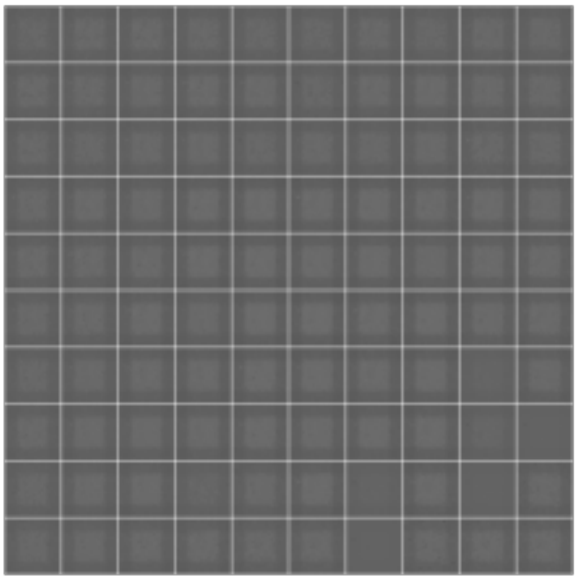


Figure 8.3: Untrained model map

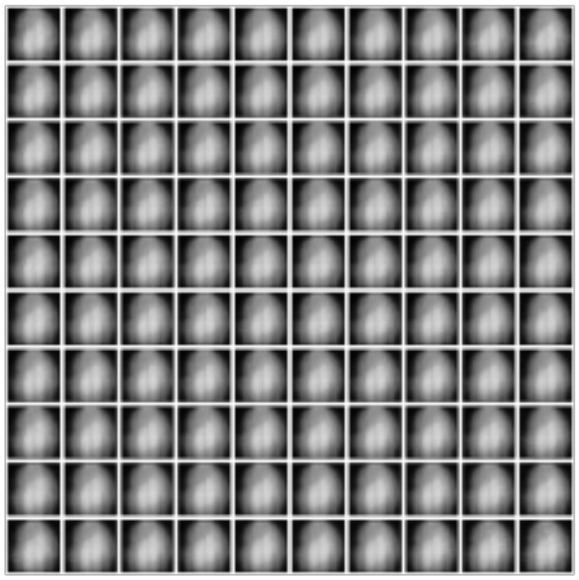


Figure 8.4: Training Epoch 10

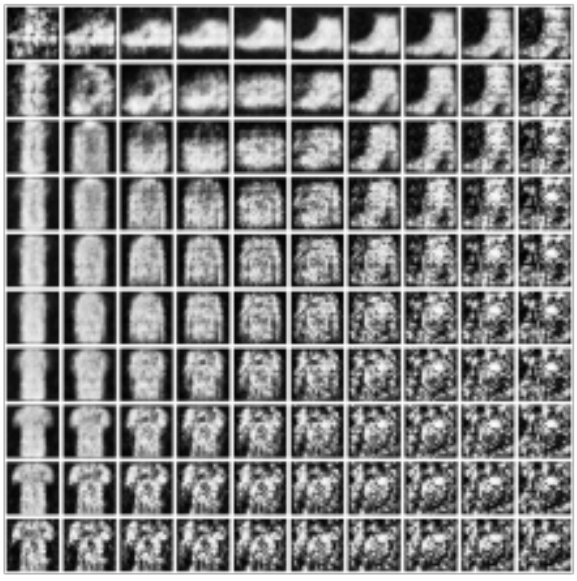


Figure 8.5: Training Epoch 20

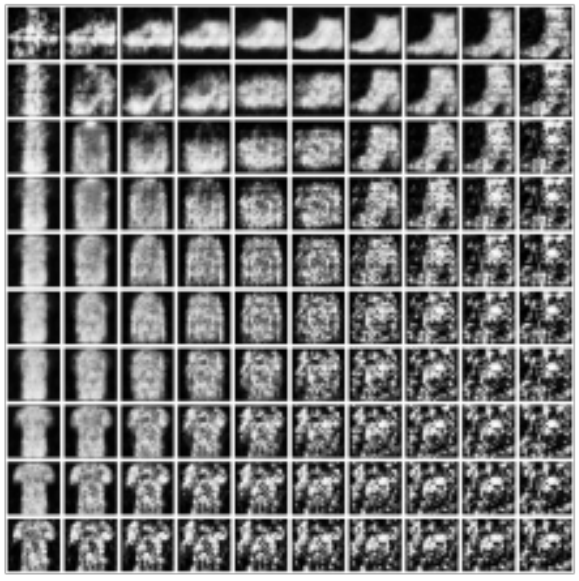


Figure 8.6: Training Epoch 30 (i.e., maximum training)

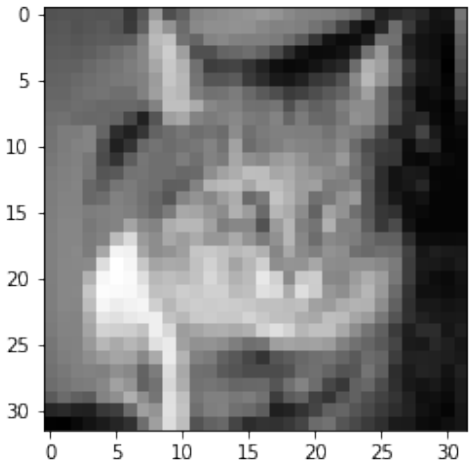


Figure 8.7: Original CIFAR10 image, which is not very clear.

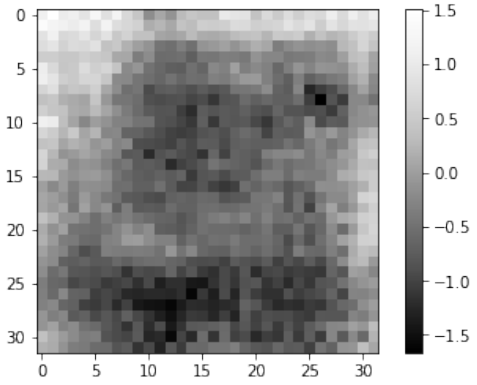


Figure 8.8: Generated image on the CIFAR10 dataset that may look like a small teddy-bear

Despite this, VAEs do face some challenges - most notably *interference suboptimality* (i.e. faults caused due to premature stopping of the encoder). This manifests itself in the form of:

- Reduced expressiveness of the decoder due to limited variance of the latent variables (This is called **variational approximation**.)
- Missed discrepancy between the input and the generated images (This is called **KL Divergence approximation**.)
- Decoder becoming fixated on a small subset of the input data (This is called **model collapse** and may be caused due to small probability variations in the input data being blown up by the model, or other factors.)
- Latent space distortions/blurry images as seen in the training of a simpler model on the CIFAR10 dataset of objects from everyday life in Fig. 8.7, Fig. 8.8.

Another dataset this model was tried on was on the MNIST dataset (of handwritten digits). The generated and real images are as shown in Fig. 8.9 Fig. 8.10. Note that while neither of the Fashion-MNIST or MNIST images were taken after the model had completed training, these were taken after

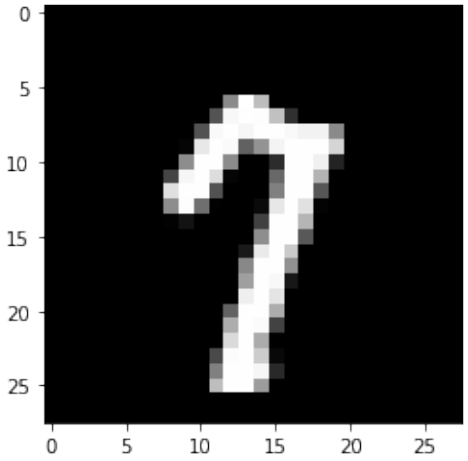


Figure 8.9: Original MNIST image

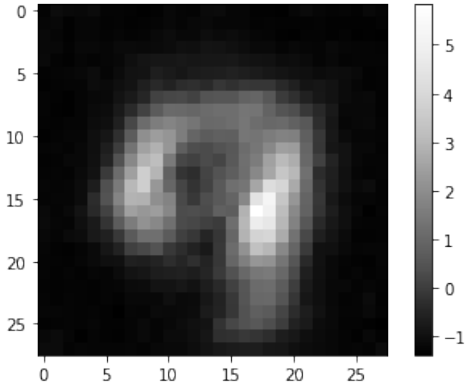


Figure 8.10: Generated image on the MNIST dataset that may look like a 9

the images appeared to take the form they started resembling the original dataset. The CIFAR10 images were taken at convergence (i.e., end of model training).

Bibliography

[1] Hurier, G., "Predicting the cib-tamination in the cross-correlation of the tsz effect and," *A&A*, vol. 575, p. L11, 2015.

[2] D. Han, N. Sehgal, and F. Villaescusa-Navarro, "Deep learning simulations of the microwave sky," *Physical Review D*, vol. 104, dec 2021.

[3] N. Sehgal, P. Bode, S. Das, C. Hernandez-Monteagudo, K. Huffenberger, Y.-T. Lin, J. P. Ostriker, and H. Trac, "SIMULATIONS OF THE MICROWAVE SKY," *The Astrophysical Journal*, vol. 709, pp. 920–936, jan 2010.