Generation of Texture: A Case Study with Steel Microstructure Images

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology

in

Computer Science

by

Soumee Guha

[Roll No: CS-1806]

under the guidance of

Prof. Dipti Prasad Mukherjee

Electronics and Communication Sciences Unit

Indian Statistical Institute

Kolkata-700108, India

July, 2020

DECLARATION

This is to certify that the dissertation entitled **Generation of Texture: A Case Study with Steel Microstructure Images** submitted by **Soumee Guha** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of Master of Technology in Computer Science is a bonafide record of work carried out by her under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Prof. Dipti Prasad Mukherjee
Electronics and Communication Sciences Unit,
Indian Statistical Institute, Kolkata

Acknowledgements

First of all, I would like to express my deepest gratitude to my supervisor, Prof. Dipti Prasad Mukherjee, Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata for his patience, motivation and immense knowledge. He has guided me throughout this work.

I would like to sincerely thank Prof. Nilanjan Ray, Department of Computing Science, University of Alberta for his valuable insights and guidance.

I would like to thank my lab-mates for their continuous support.

I am deeply indebted to the professors of Indian Statistical Institute, Kolkata for their valuable guidance and encouragement.

I would like to thank all my friends for their continuous support.

Last but not the least, I would like to acknowledge my deepest gratitude to my family for supporting me morally and loving me unconditionally.

Soumee Guha
Indian Statistical Institute, Kolkata
July, 2020

Abstract

A lot of work has been done on texture generation techniques. Deep learning based image generation techniques have been extremely successful in generating realistic images. Moreover, reaction-diffusion systems have also been successful in generating a wide variety of textures. However, the reaction-diffusion systems have never been incorporated in modern deep learning architectures. On the other hand, although a wide variety of images have been generated using traditional computer vision algorithms and deep learning models, very little work has been done on generating the microstructures that are found in abundance in nature. We have explored two established texture generation algorithms for generating steel microstructure images: *PatchMatch* and *DCGAN*. We have also tried to combine the reaction-diffusion systems with deep learning architectures and have explored the possibility of its success in generating the steel microstructure images.

Keywords: *reaction-diffusion, PatchMatch, DCGAN, steel microstructure images*

Table of Contents

Chapter 1

Introduction

1.1 Objective

A repetitive, periodic or quasi-periodic pattern displayed in an image is texture. Such repetitive patterns can be seen on any textured surface as well. Texture can also be defined as a collection of texture elements (texels) that occur in some regular pattern. Naturally observed textures are extremely difficult to analyse and model due to their diversity and complexity. Natural textures often have very irregular structural patterns which cannot be modelled by the traditional modelling tools. Some natural textures are shown in Figure 1.1. Texture patterns, specifically microstructure patterns are found in abundance all around us. The objective of this work is two-fold:

- Generation of textures
- Generation of steel microstructure

Figure 1.1: Some natural textures [1]

The quality of steel can be determined by analysing the microstructure pattern (Figure 1.2) of steel [2]. Observing a polished block of steel with a light microscope can reveal such patterns. If these microstructures were based on a certain mathematical model, then it would have been easier to obtain different varieties of steel by

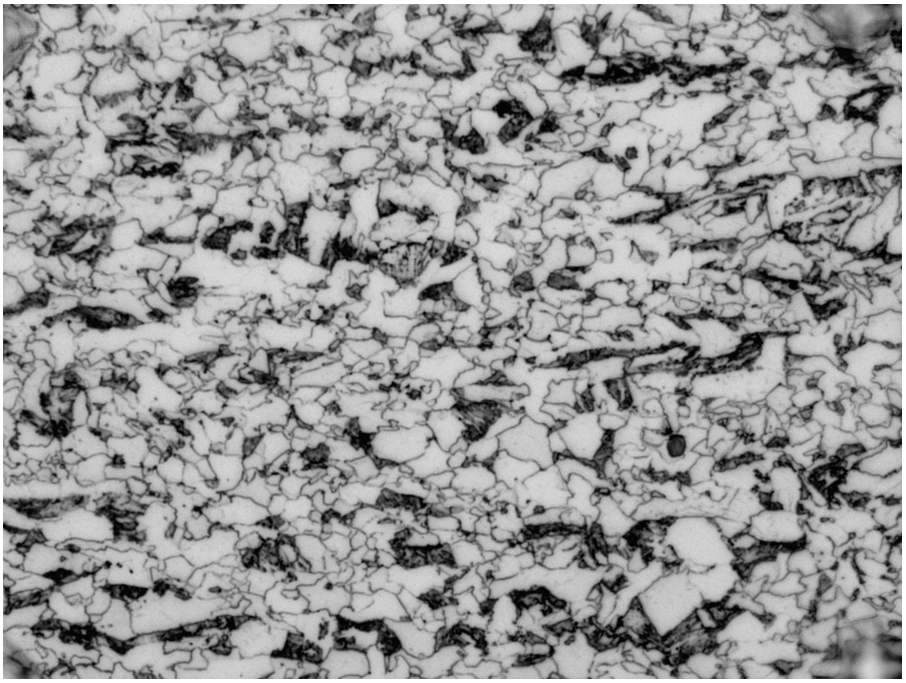


Figure 1.2: Steel microstructure

changing the model parameters slightly. However, there does not exist any mathematical or physical (metallurgical) model for generating the steel microstructure textures. Since texture modelling and texture generation are closely related, the generation of these microstructures can provide a means of modelling them. This project aims at developing a computational model to create such microstructures.

1.2 Literature Survey

Texture modelling has been an active research topic in the domain of computer vision since a very long time. The algorithm proposed in [1] generates realistic textures and takes a sample texture pattern as input. [3] improved the algorithm proposed in [1] and made it suitable for generating textures having irregular shapes. For textures having complex structures, these greedy algorithms sometimes generate results which are inconsistent. The algorithm proposed in [4] generates textures that are globally consistent by modelling the texture completion problem as a global optimization problem. Approaches involving patch optimization have been used in [5–7]. Computational complexity increases with the improvement in the quality

of the generated textures using patch optimization techniques. This problem further escalates when the dimensions of the sample texture increase. In order to solve this problem, tree-based structures have been proposed in [1, 4, 5] and approximate search methods have been proposed in [8, 9]. The algorithm proposed in [3] employs a local propagation method that exploits the interdependence between pixels and hence limits the total number of searches required for texture generation. [10] introduces a randomized algorithm that generates textures by approximating the nearest neighbour matches between two given images.

Reaction-diffusion method based texture generation techniques were first proposed in [11]. It was shown that stable patterns could be generated by the interaction of two or more chemicals. [12–14] have shown that reaction-diffusion systems can generate patterns like stripes and spots by varying the parameters of the reaction-diffusion systems. Generation of patterns in accordance with the geometry of a surface has been proposed in [15]. The major drawback of these algorithms is the estimation of the reaction and diffusion constants required for generating a stable pattern.

With the invention of Generative Adversarial Networks (GANs) in 2014 [16,17], the image synthesis algorithms have progressed remarkably. GANs do not require Markov chains or inference approximations during either the model's training phase or while generating new samples with the trained model [16]. These models are trained with back propagation techniques and do not require explicit definition of a probability distribution. In contrast, the models are trained for generating samples from a desired distribution. Unlike the generative stochastic networks (GSNs) [18] and generalized denoising auto-encoders [19], adversarial networks do not involve Markov chains. However, GANs are difficult to train and often generate meaningless outputs. Deep Convolutional GAN proposed in [20] addresses this problem with the use of deep convolutional architectures for both the generator and the discriminator. Even after training these models sufficiently, they might still be unstable and can collapse. A few hacks have been found useful for stabilizing these models to some extent. However, GANs are still notoriously difficult to train and several other attempts have been made to solve this problem by modifying the loss function [21], introducing regularization techniques [22] and redefining the model

architectures [23, 24]. [25] proposes a collaborative sampling technique between the generator and the discriminator for refining the samples generated at a particular layer of the generator by reducing the distance between the generator distribution and the distribution of the training images.

1.3 Dataset

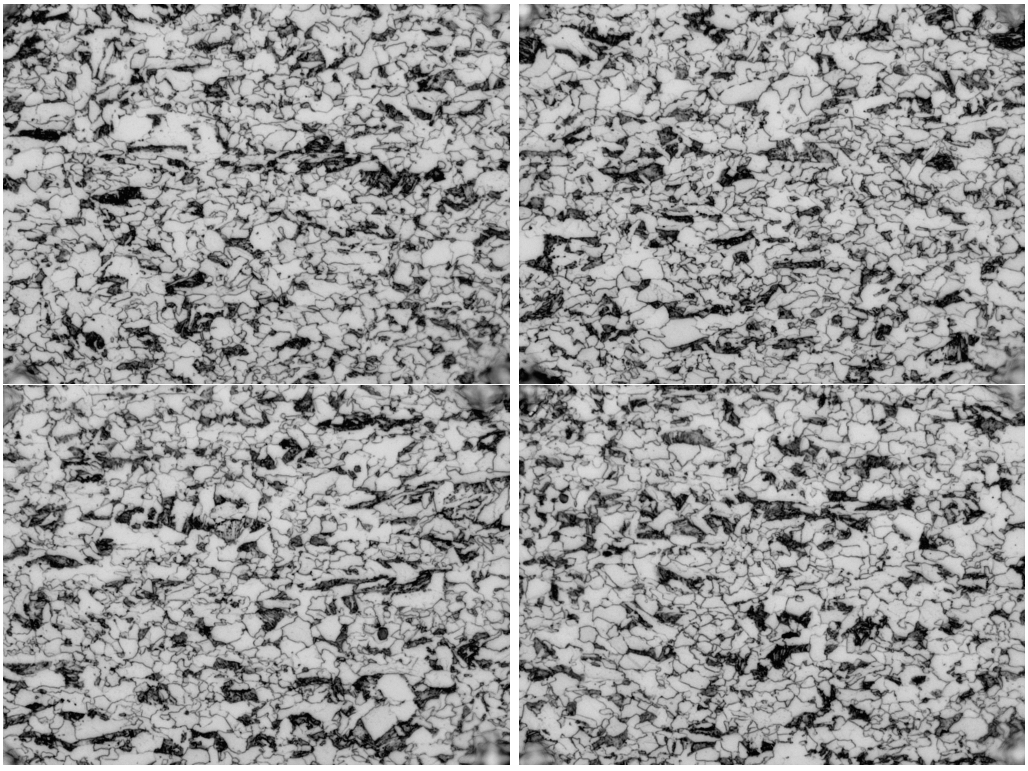


Figure 1.3: Sample images of steel microstructure

Deep learning architectures require lots of training data. At first, a sample of steel is grounded and a mirror-finish surface is obtained by polishing it. After this step, the polished surface is etched with *Le Pera* etchant, which is a mixture of Picric acid in ethyl alcohol and sodium metabisulphite in distilled water. Indentations are made on the steel sample with a microhardness testing machine. The indentation array are at a distance of 178 μm in the horizontal direction and 133 μm in the vertical direction. The steel sample enclosed by four indentations is imaged using a light microscope at 500x magnification [2].

There were initially 20 images of steel microstructure textures. The dimension

of each image was 1920 x 2560 x 3. These images were RGB images, hence each image had three channels. Figure 1.3 shows some of the microstructure images. The actual dataset that has been used for this work has been generated by randomly cropping the microstructure images. Each cropped image is of size 512 x 512 x 3.

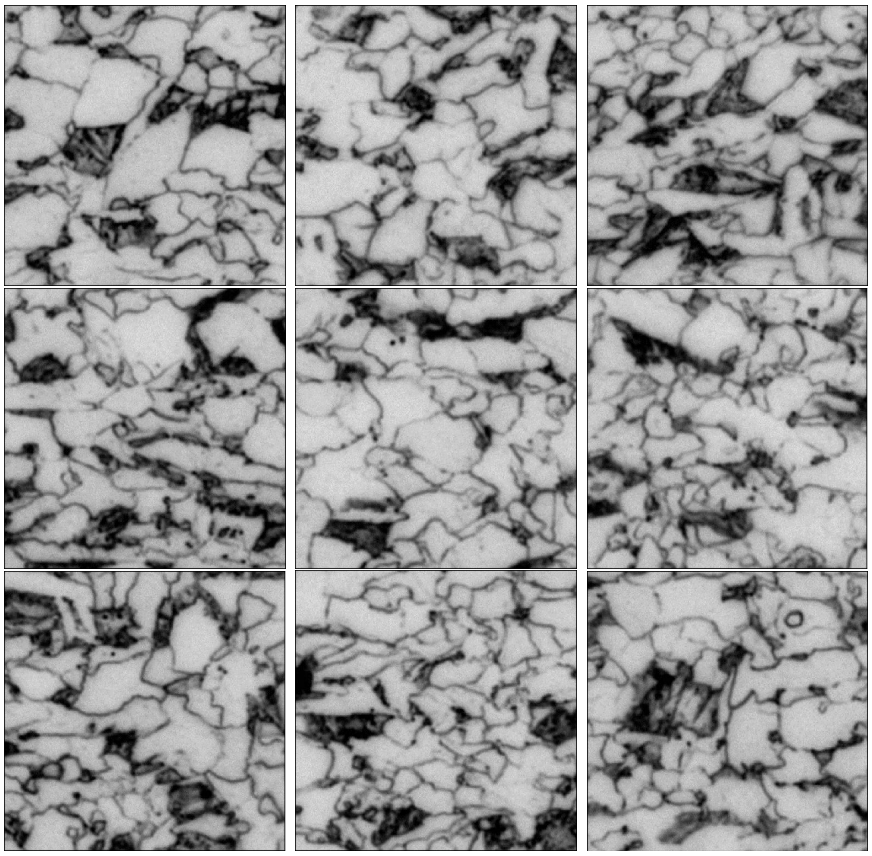


Figure 1.4: Sample images from the steel microstructure image dataset

Figure 1.4 shows some of the cropped images obtained from the steel microstructure images. We have obtained 300 cropped images with which we have performed the experiments.

1.4 Contributions

In this work, we have explored two established texture generation methods. The first one is a traditional method, known as the PatchMatch algorithm [10]. The second one is Deep Convolutional Generative Adversarial Network (DCGAN) [20]. We have generated the steel microstructure images with these two methods. The

original DCGAN paper generates images having dimensions 64 x 64 x 3. However, the modified network can generate images of much larger dimensions (512 x 512 x 3) very efficiently. Thus, the original DCGAN architecture has been modified to suit our requirements. We have also discussed the reaction-diffusion systems and have identified the implementation issues that are often faced while working with the traditional simulations. We have proposed a deep neural architecture which can generate the reaction-diffusion patterns without any knowledge of the parameters of the actual reaction-diffusion system.

1.5 Organization of thesis

Chapter 2 discusses the PatchMatch algorithm and DCGAN, their training procedures and the corresponding results. In Chapters 3 and 4 we have discussed the proposed method and the experimental results respectively. Chapter 6 concludes the thesis.

Chapter 2

Related Works

2.1 PatchMatch

PatchMatch [10] is a randomized algorithm that can efficiently generate different types of textures. Given a reference texture (I_{IN}), it can generate another texture (I_{GEN}) which is similar to the given reference texture. The textures I_{IN} and I_{GEN} can be RGB images or grayscale images. Considering two images X and Y, for every texture patch in X, the objective is to find the closest patch in Y under a chosen distance metric. This mapping is known as the Nearest-Neighbour Field (NNF). Assuming a small patch of texture with n pixels, the brute force method of searching for its closest patch in an image of size N pixels is extremely expensive, O(n*N*N).

The following observations make PatchMatch extremely efficient:

- The patch space is sparsely populated (O(N) patches) and operating in the two-dimensional patch space reduces the total time complexity and increases memory efficiency.
- In every image, adjacent pixels are not independent and almost all images have a natural structure. Searching for a better candidate in the adjacent pixels interdependently improves the efficiency of the algorithm.
- A fairly large number of random patch assignments can lead to a good patch selection even though any one random choice of patch might not be a good candidate.

I_{GEN} is initially empty and PatchMatch starts by randomly selecting pixels from I_{IN} and assigning them to I_{GEN}. After the initialization step, the algorithm iteratively updates the NNF by two separate processes, **propagation** and **random**

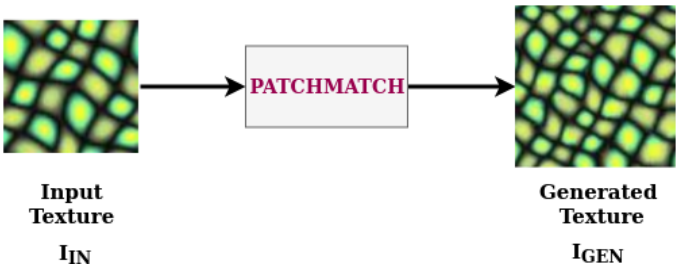


Figure 2.1: Block Diagram of PatchMatch [10]

search. The pixels and the corresponding neighbourhoods are scanned from the top left corner and every pixel undergoes propagation and random search, one following the other. Figure 2.1 shows the PatchMatch algorithm.

- **Propagation** uses coherence for dissemination of good texture patch choices to the adjacent pixels. Let us consider a pixel p_G in I_{GEN}. Let us assume that the candidate pixel for p_G is at index (i,j) of I_{IN}. Here $0 \leq i < H$ and $0 \leq j < W$, where the H and W are the height and width of I_{IN} respectively. The pixels at index positions (i-1, j) and (i, j-1) of I_{IN} are also considered and their neighbourhoods are examined for a closer match. This is because the neighbouring pixels in an image are interdependent and hence their neighbourhoods are also similar.

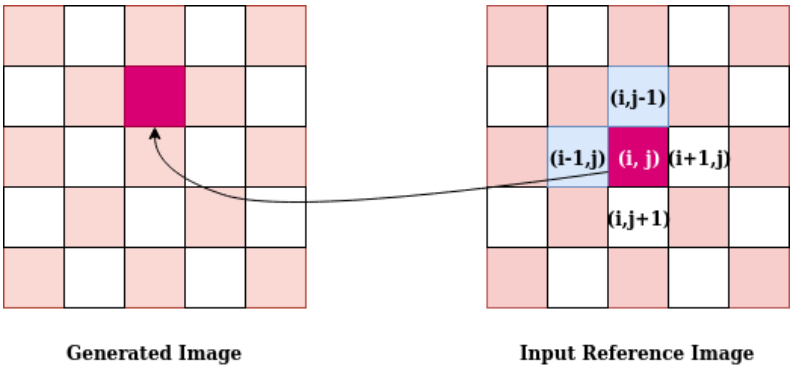


Figure 2.2: Propagation

Figure 2.2 shows that the pixel at index (i,j) of the input reference image is initially the chosen candidate pixel for the pixel shown in the left image. Let us assume that the PatchMatch method is executed for ITR number of iterations. $\forall$ $iter \in ITR$, if $iter$ is odd, then it is an odd iteration, else it is

even. On odd iterations, the pixels at indices (i, j-1) and (i-1, j) are examined to see whether any of these are a better match for the selected pixel in the generated image whereas on even iterations, the pixels at indices (i, j+1) and (i+1, j) are examined for the same.

- **Random search** tries to improve the candidate pixel p selected for a particular index position (i, j) by examining the neighbourhoods of a list of candidates selected randomly within a prespecified exponentially decreasing radius around p.

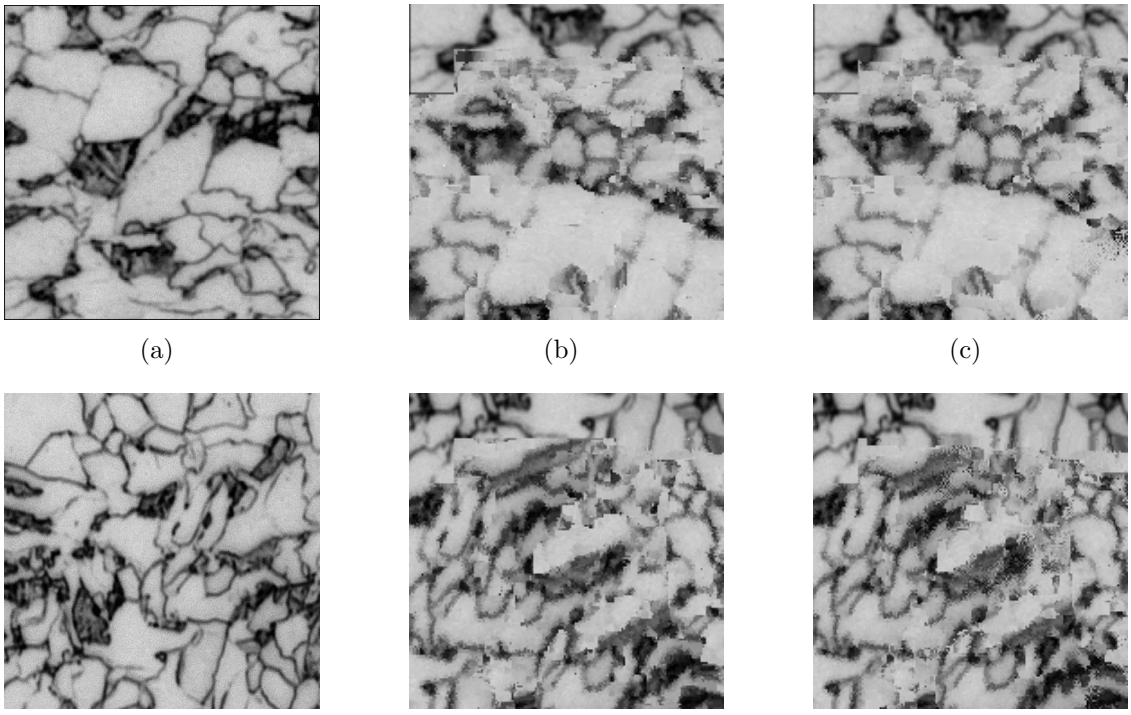
$$p_{iter} = p + R_{max}\beta^{iter}U_{iter}. \tag{2.1}$$

In equation 2.1, U_{iter} is a random number in [-1, 1] x [-1, 1], R_{max} is the maximum radius available for searching and β is a constant ratio. The search starts with iter = 0 and is continued for higher values of iter until the radius of search $R_{max}\beta^{iter}$ becomes less than unity.

2.1.1 Experiments and Results

Images of size 200 x 200 x 3 were used as reference textures for obtaining similar textures having the same dimensions. The single resolution method was used with a neighbourhood of size 51 x 51. For better results, the overall procedure was repeated twice.

- For generating texture images of size 200 x 200 x 3, in the initializing step, a matrix (I_{GEN}) of size 200 x 200 is created. Each index of I_{GEN} is assigned a randomly selected pixel (all the three, i.e, R, G and B channels) from the reference image.
- for $iter$ in $TotalNumberofIterations$:
  - for all pixels in I_{GEN}:
    - Propagation
    - Random Search
- A new empty matrix having size 200 x 200 x 3 is created. In the new matrix, the pixel values of the candidate pixels are put and the final texture is obtained.



(a) (b) (c)

(d) (e) (f)

Figure 2.3: PatchMatch results. Details are given in Section 2.1.1

The image obtained at the end of each iteration has been shown. Since we have repeated the entire PatchMatch method twice, for a particular input reference image, we have got two generated images. The results are shown in Figure 2.3. The results corresponding to two different steel microstructure images have been shown. Figures 2.3(a) and 2.3(d) are the two reference images that have been used for this experiment. Figures 2.3(b) and 2.3(c) are the images generated at the end of the first and second iterations with Figure 2.3(a) as the input image. Similarly, with Figure 2.3(d), the images generated at the end of the first and second iterations are shown in Figures 2.3(e) and 2.3(f) respectively.

2.2 Deep Convolutional GAN (DCGAN)

Generative Adversarial Networks (GANs) were introduced in [16]. The term **generative** denotes that this model aims at generating new data points similar to the ones in the training set. The term **adversarial** indicates that it is a two player game and the loss function is a minimax function. The term **networks** is used as

this is a deep neural network architecture. A discriminative model aims at learning the inherent features of a training set and thereafter classify them into various categories. In contrast, a generative model aims at learning the underlying distribution of the training set for sampling new data points. Mathematically, for a set of data points A and a set of labels B, the generative models try to learn the joint probability $P(A, B)$ whereas the discriminative models aim at learning the conditional probability $P(B|$ A).

GANs have two independent networks, the **generator** and the **discriminator**. The generator aims at generating data points similar to the ones in the training set whereas the discriminator tries to distinguish between real and fake data points. During the training process, for a batch of input images, the discriminator estimates the probability that the batch belongs to the actual training dataset. For real images, this probability should be very close to 1 and for fake images, it should be close to 0. In adversarial networks, the generator and the discriminator work as adversaries and this competition forces both of these networks to improve their performance.

Figure 2.4 shows the DCGAN architecture. A batch of 1-dimensional Gaussian noise vectors (Z) is the input to the generator G. The output of G is a batch of fake images $G(Z)$. A batch of real images and $G(Z)$ are alternatively fed to the discriminator D. The training procedures of G and D are interleaved, yet independent. In DCGAN [20], both G and D are two separate and independent deep convolutional neural networks and binary cross entropy loss (BCE loss) is used for both the networks.

$$V(D) = E_{X \sim Prob_{real}(X)}[log(D(X))] + E_{Z \sim Prob_{fake}(Z)}[log(1 - D(G(Z))]. \tag{2.2}$$

$$V(G) = E_{Z \sim Prob_{fake}(Z)}[log(1 - D(G(Z))]. \tag{2.3}$$

Equations 2.2 and 2.3 are the objective functions of the discriminator network and the generator network respectively. The discriminator network aims at maximizing its objective function. Hence, the output of the discriminator network ($D(X)$) for a batch of real images (X) should be as close to 1 as possible and the output ($D(G(Z))$) corresponding to a batch of fake images ($G(Z)$) should be as close to 0 as possible. On the contrary, the generator network tries to minimize

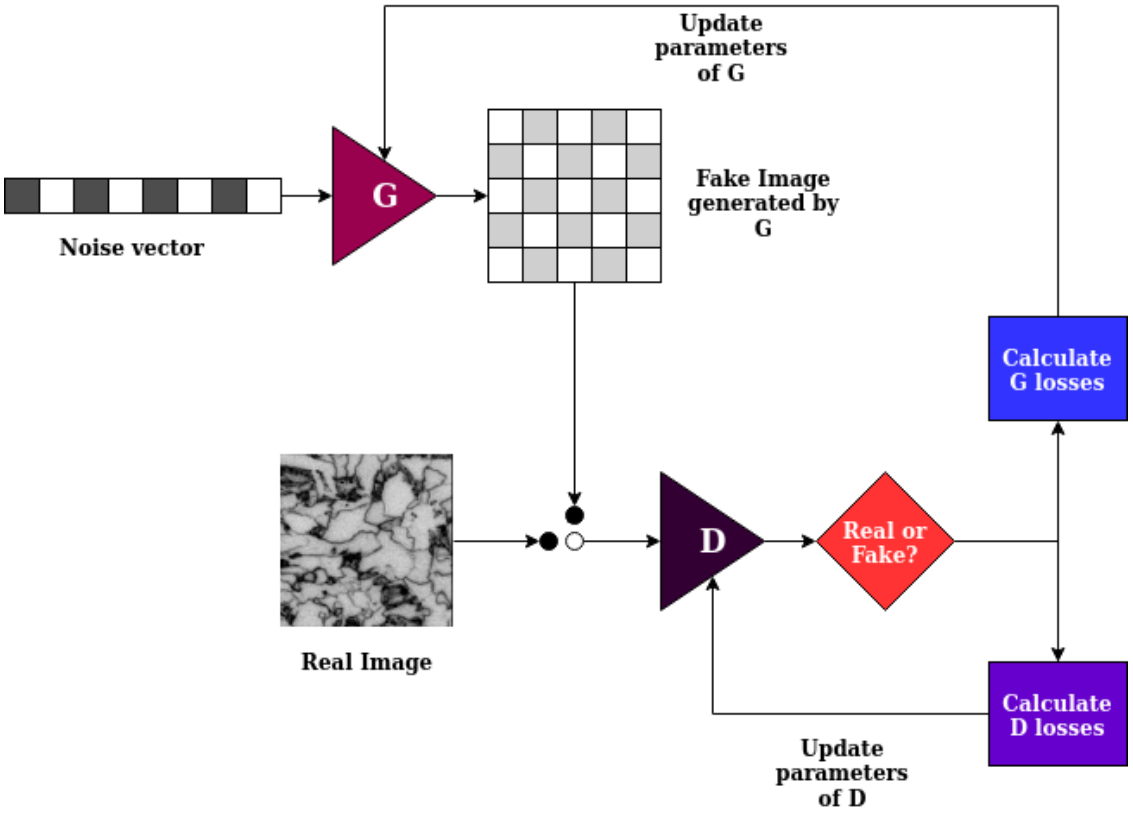


Figure 2.4: DCGAN architecture

its objective function. Therefore, the generator tries to generate images which are very similar to the images belonging to the actual training dataset so that the value of $D(G(Z))$ is close to 1. Equation 2.4 is the overall objective function [16, 20].

$$\min_{G}\max_{D} V(D,G) = E_{X\sim Prob_{real}(X)}[log(D(X))] + E_{Z\sim Prob_{fake}(Z)}[log(1-D(G(Z))]. \tag{2.4}$$

The discriminator and generator networks designed for the steel microstructure dataset are shown in Figures 2.5 and 2.6 respectively. The pooling layers are replaced with fractionally strided convolutions and strided convolutions in the generator and discriminator so that the network can learn the spatial upsampling and downsampling respectively. The input to the generator is a 1D uniform noise vector Z and it is reshaped as a 4-dimensional tensor. Batch Normalization layers are used in all the layers of the discriminator network except the input layer. In case of the generator network, Batch Normalization is applied to all layers except the

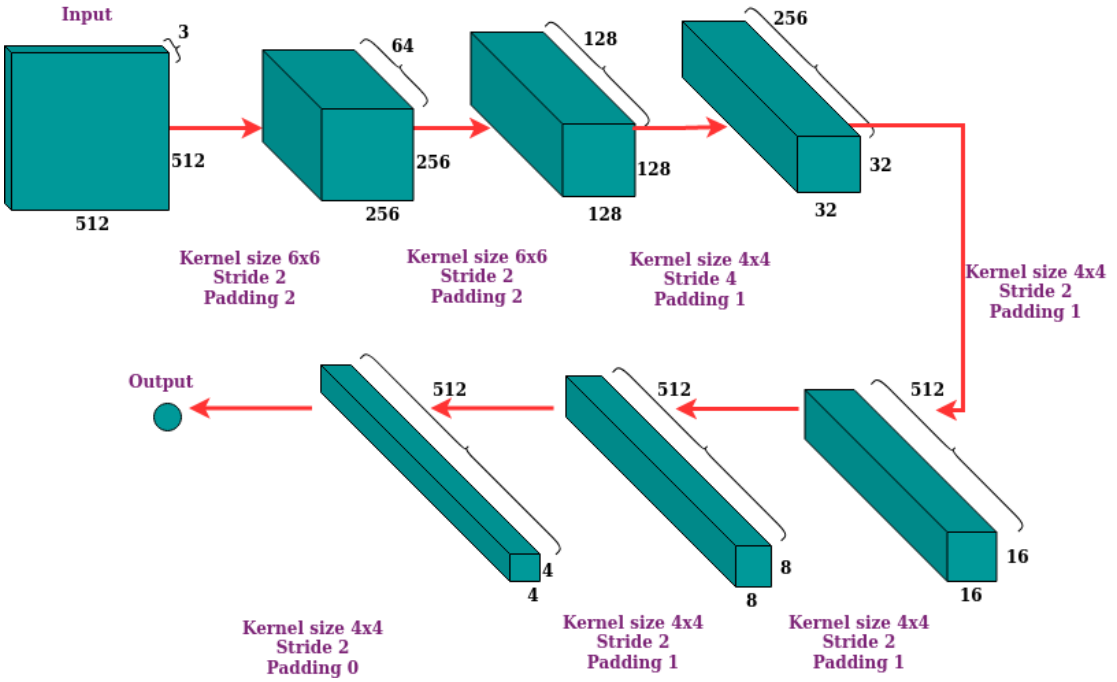


Figure 2.5: Discriminator

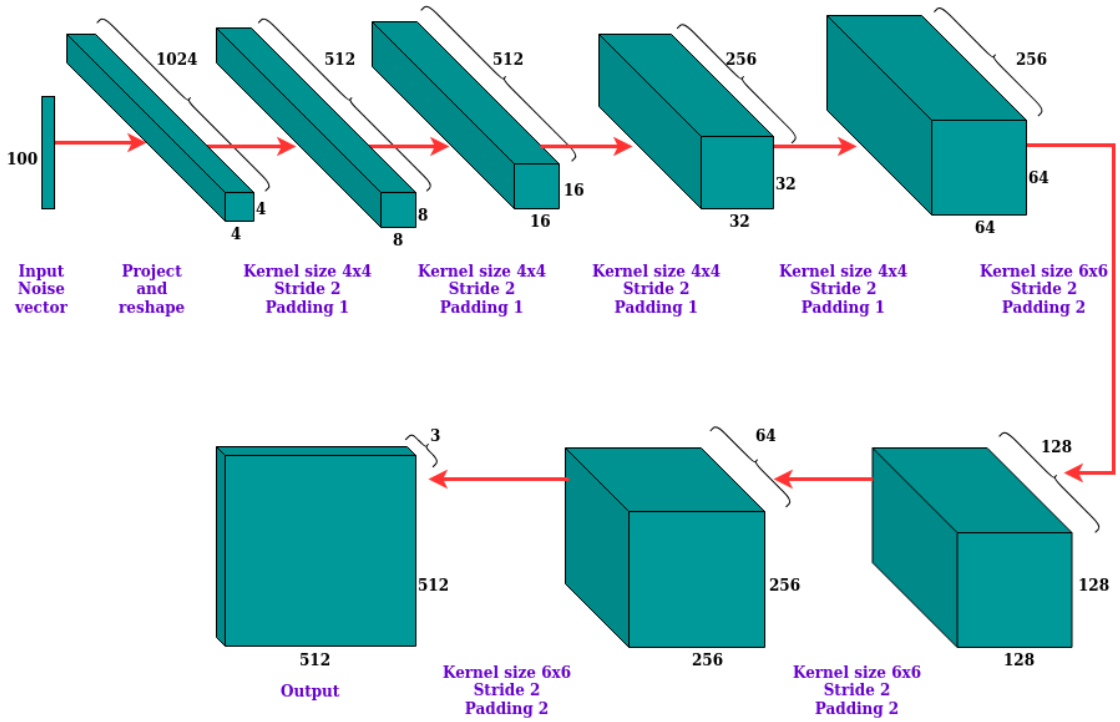


Figure 2.6: Generator

output layer. Both the generator and the discriminator networks use Leaky ReLU activation function at all layers except the output layer. The output layer of the

generator network uses the Tanh activation function whereas the output layer of the discriminator network uses the Sigmoid activation function. Equations 2.5 and 2.6 represent the ReLU and Leaky ReLU activation functions respectively where a is the input value and β is a small, non-zero constant.

$$f(a) = \begin{cases} 0 & a \leq 0 \\ a & a > 0 \end{cases} \tag{2.5}$$

$$f(\beta, a) = \begin{cases} \beta a & a \leq 0 \\ a & a > 0 \end{cases} \tag{2.6}$$

Each epoch of the training process is as follows:

- A batch of real images is sent to the discriminator network and the output is obtained. The loss corresponding to the real batch is calculated using the first term of equation 2.2 and it is back propagated and the parameters of D are updated.
- The generator network generates a batch of fake images $G(z)$ from a batch of 1D noise vectors. The discriminator's output corresponding to $G(z)$ is obtained.
- Depending on the discriminator's output, the loss corresponding to the fake batch is calculated using the second term in equation 2.2 and the network parameters are updated again.
- The generator losses are also calculated according to equation 2.3 are the parameters of the generator network are updated.

2.2.1 Experiments and Results

The discriminator and generator networks shown in Figures 2.5 and 2.6 were trained with the steel microstructure dataset. Images of size 512 x 512 x 3 were used without any pre-processing. The model was trained for 2000 epochs with Adam optimizer with a learning rate of 0.0002. The size of each mini-batch was 4. The weights of the networks were initialized with a normal distribution having mean 0 and standard

deviation 0.02. Leaky ReLU having slopes 0.0002 and 0.00002 were used in the discriminator and generator respectively. For improving the training procedure, noisy labels were used for the discriminator network. In each epoch, the real and fake labels were randomly selected between [0.8, 1] and [0, 0.2] respectively. A batch of training images is shown in Figure 2.7. The modified DCGAN architecture is trained twice and at the end of each training session, a batch of fake images is obtained. Figure 2.8 shows the two batches of fake images generated by DCGAN.

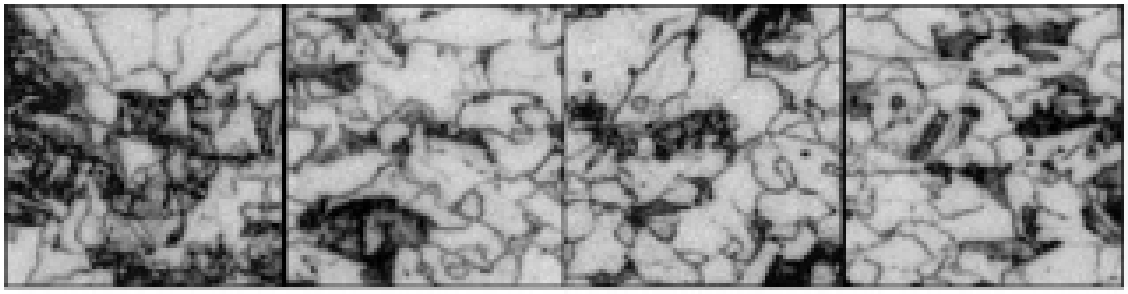


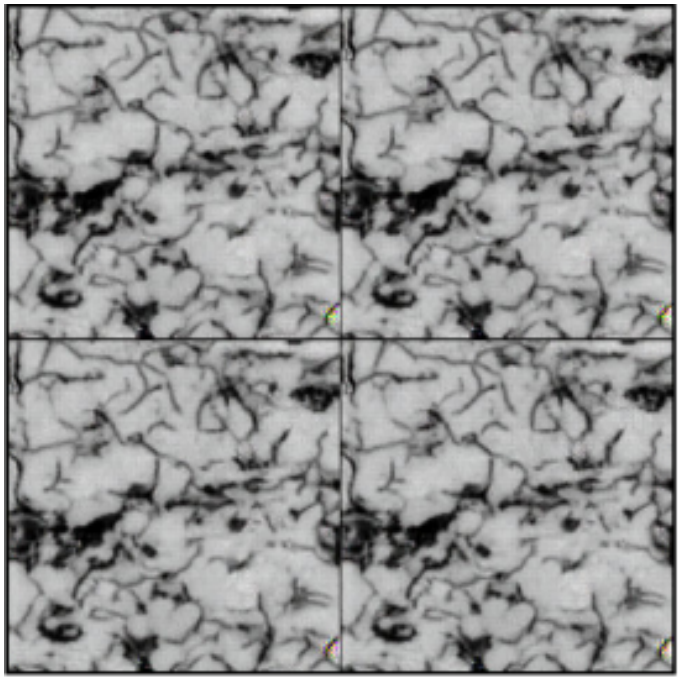
Figure 2.7: A batch of real images. Details about the training process are given in Section 2.2.1

2.2.2 Discussion about the modified architecture

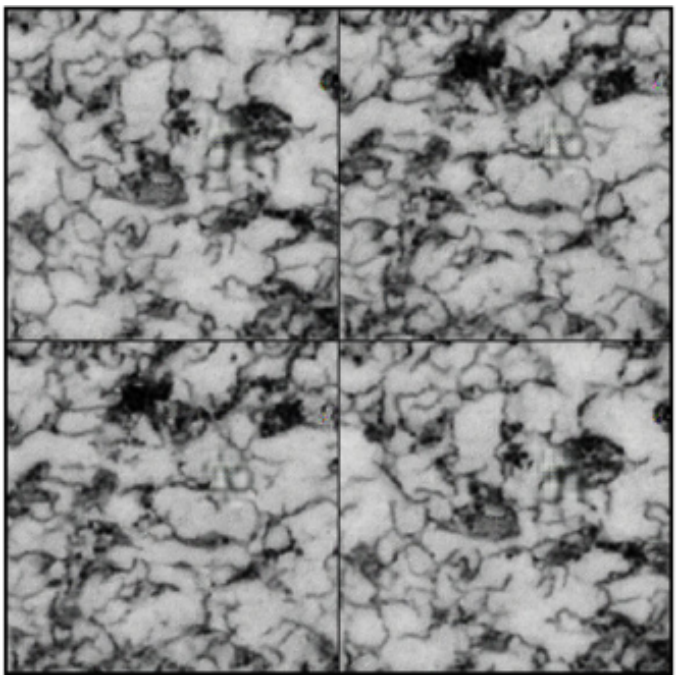
In this approach, we have expanded the discriminator and the generator architectures for generating large sized images efficiently. In the proposed DCGAN architecture [20], the dimensions of both the training images and the generated images are 64 x 64 x 3. Hence kernels of size 4 x 4 are used in [20]. However, since we are using the modified architecture for generating images of size 512 x 512 x 3, the kernel sizes have been increased to 6 x 6 in the first two layers of the discriminator network and the last two layers of the generator network as shown in Figures 2.5 and 2.6 respectively. This is because larger sized kernels can capture the required information needed for extracting suitable features.

2.3 Summary

In this chapter, we have explored two different approaches for generating the steel microstructure images. The first method is a traditional one, whereas the second method is a more recent architecture. The results generated by the PatchMatch



(a)



(b)

Figure 2.8: DCGAN results. Details about the generation process are given in Section 2.2.1

algorithm are not very satisfactory. Moreover, the system doesn't **learn** anything. DCGAN, on the other hand, has been quite successful in generating a wide variety of images. In the original DCGAN paper [20], the size of each image that was used for training was 64 x 64 x 3. The generated images were also of the same dimension. The images in our steel dataset have intricate structures. When they were resized to 64 x 64 x 3, the details were lost. Hence, we have modified the DCGAN architecture and made it suitable for generating images of size 512 x 512 x 3. The results shown in Figure 2.8 are generated by two separate training methods. From Figure 2.8, it is evident that all the images that are generated after a single training session are almost identical, i.e, the architecture suffers from mode collapse. Aiming to solve the shortcomings of the methods discussed in this chapter, we explore another method of texture generation: the **reaction-diffusion** method.

Chapter 3

Proposed Method

3.1 Reaction-Diffusion

The reaction-diffusion system of pattern synthesis was first proposed in [11] where the author has described how two or more chemicals can form stable patterns after diffusing across a surface. The system is defined by a set of partial differential equations, where each equation comprises a diffusion term and one or more reaction terms. The diffusion method simulates the movement of a chemical substance from an area of higher concentration to the neighbouring areas having lower concentration. The diffusion process has the effect of adaptively smoothing the image, whereas, the reaction method generates distinct textures in it. The reaction term in the equation accounts for the non-linearity that is usually observed in nature.

The most elementary form of the reaction-diffusion system can be described with the help of two different chemicals having different rates of diffusion. The initial variation in the concentration levels of these chemicals tend to make the system unstable initially and ultimately help in the formation of patterns by varying their concentration across the surface. A reaction-diffusion system comprising two chemicals can be described by the following partial differential equations:

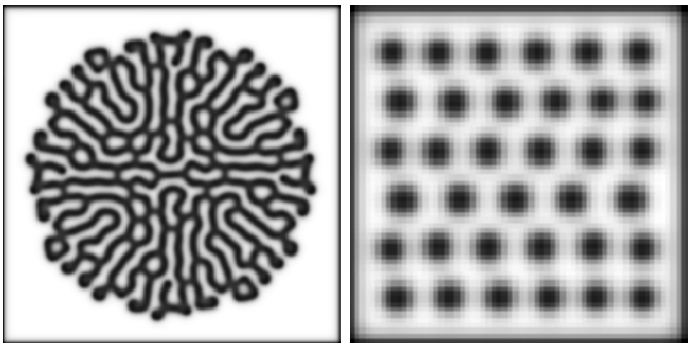
$$\frac{\partial p}{\partial t} = D_p \nabla^2 p + R_p(p, q). \tag{3.1}$$

$$\frac{\partial q}{\partial t} = D_q \nabla^2 q + R_q(p, q). \tag{3.2}$$

These equations indicate that the change in the concentration levels of any one of the chemicals at any given instant depends on the sum of the reaction and diffusion components for that chemical. The terms D_p and D_q are constants, representing the diffusion rates for the chemicals p and q respectively. The chemicals p and q are defined in a two-dimensional space (i,j) and $\nabla^2 = \frac{\partial^2}{\partial^2 p} + \frac{\partial^2}{\partial^2 q}$. The terms $\nabla^2 p$ and $\nabla^2 q$ indicate the concentration level of a particular chemical with

respect to the neighbouring areas. If the neighbouring areas have higher concentration of a particular chemical, the Laplacian will be positive and the chemical will diffuse towards that region and vice-versa. A reaction-diffusion system iteratively computes the local concentrations $R_p(p, q)$ and $R_q(p, q)$ until a stable pattern is generated [15, 26].

In this method, at any given time instant, only one of the chemicals can be present at a particular point on the surface across which the reaction and diffusion occur, i.e, the chemicals are locally exclusive. Various patterns have been generated by different models in the past. The generated patterns can be broadly classified into two categories, **spots** and **stripes**. These patterns depend largely on the non-linear terms present in the reaction function [27]. Figure 3.1 shows two images generated by the traditional reaction-diffusion systems.



(a) (b)

Figure 3.1: Images generated by traditional reaction-diffusion systems

The simulation process of any reaction-diffusion system is iterative. A fixed number of substrates react with each other to produce a specific pattern. For a reaction-diffusion system involving two substrates A and B, let us consider the simulation process shown in Figure 3.2. A and B are actually two matrices representing the two substrates which will react with each other to form a stable pattern. During diffusion, A and B are convolved with a diffusion filter. For generating any specific pattern, this filter should be known. In the reaction process, A and B react with each other according to a set of equations. In Figure 3.2, C is a matrix representing scalar values. The dimensions of matrices A, B and C are same. I_{R_A} and I_{R_B} are the reaction images obtained from A and B while I_{D_A} and I_{D_B} are

the diffusion images obtained from them. The terms rA and rB are the reaction coefficients for the substrates A and B while dA and dB represent their diffusion coefficients. Apart from the diffusion filter, the exact values of rA, rB, dA and dB also need to be known for generating any desired pattern. Finally the update equations show how A and B are updated.

for iter in range(Total Iterations):

Diffusion Process	$I_{D_A} = conv(A, \text{Diffusionfilter})$ $I_{D_B} = conv(B, \text{Diffusionfilter})$
Reaction Process	$I_{R_A} = 16 - A \times B$ $I_{R_B} = A \times B - B - C$
Incremental Changes	$deltaA = rA \cdot I_{R_A} + dA \cdot I_{D_A}$ $deltaB = rB \cdot I_{R_B} + dB \cdot I_{D_B}$
Update Equations	$A = A + deltaA$ $B = B + deltaB$

Figure 3.2: Reaction-Diffusion Simulation

The four steps mentioned in Figure 3.2 are repeated for a number of iterations. In each iteration, the reaction process develops intricate structures while the diffusion method blurs them. The fine balance between these two methods can ultimately generate a wide variety of textures.

3.2 Simulation with Neural Networks

For a pattern for which the diffusion filter, reaction terms and the coefficients of reaction and diffusion are unknown, the simulation of a reaction-diffusion system can be extremely difficult. This is because these systems are very unstable and the slightest of deviation of these parameters from the actual values can make the system completely collapse. Generation of the steel microstructure images with reaction-diffusion systems would require the knowledge of these parameters. For obtaining these parameters, we have explored three different ways in this work:

1. Fully Connected Neural Network (FCNN)

2. Convolutional Neural Network (CNN)

3. Generative Adversarial Network (GAN)

Using each of the above methods, the objective is to mimic the reaction and diffusion process shown in Figure 3.2 without any knowledge about the reaction terms, the diffusion filter and the coefficient terms. Starting with a noise matrix, the objective is to obtain a desired pattern with the reaction-diffusion process.

for iter in range(Total Iterations):

Reaction Diffusion

$$I_D = conv(I_{iter-1}, \text{Diffusionfilter})$$
$$I_R = Reaction(I_{iter-1})$$

Update Equations

$$I_{iter} = I_{iter-1} + I_D + I_R$$

Figure 3.3: RDNet

As shown in Figure 3.3, like the original reaction diffusion system, our method is also iterative. The only difference is that we are trying to estimate the reaction terms, the diffusion filter and the coefficients for a desired pattern with a chosen neural network architecture. In Figure 3.3, the process **Reaction** represents a model that would generate reaction terms. The input to RDNet, as shown in Figure 3.4, is a noise matrix and the output is the desired pattern.

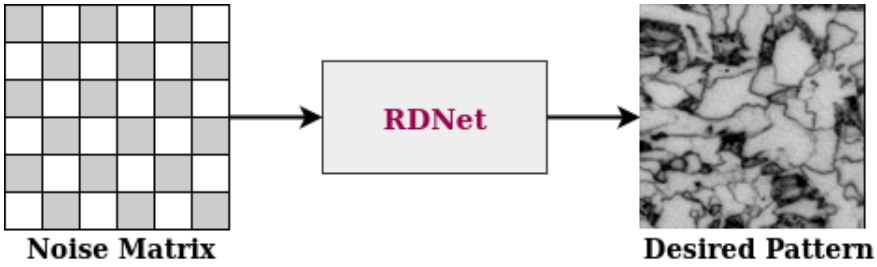


Figure 3.4: Block Diagram representing RDNet

RDNet starts by initializing the diffusion mask as the Laplacian filter. The diffusion mask has different components, one for each substrate. All the components of the diffusion mask are stacked together and hence the dimension of the mask is 3

x 3 x n_groups, where n_groups is the number of substrates. During the i^{th} iteration of the reaction-diffusion mechanism, the image (I_{i-1}) obtained from the $(i-1)^{th}$ iteration is fed to RDNet. I_{i-1} passes through the different layers along with the intermediate activation functions and the image that is obtained at the output of this entire network, I_{R_i} is the image obtained from the i^{th} reaction process. For diffusion, I_{i-1} is convolved with the diffusion mask to generate the image I_{D_i} that is obtained due to the diffusion process at the i^{th} iteration. The resultant image obtained by adding I_{R_i} and I_{D_i} is I_i, i.e, the image that is finally obtained at the end of the i^{th} iteration.

The overall simulation process for the FCNN and CNN RDNet architectures is shown in Figure 3.5. Since neural networks can generalize a wide class of functions, we have designed these architectures. For these architectures, the networks are tuned for generating any desired pattern from a random noise matrix. The process is repeated for a fixed number of epochs (total_epoch). In each epoch, the MSE loss between the desired pattern and the generated pattern is computed and back propagated to update the network parameters. This is shown in Figure 3.5. Equation 3.3 shows the MSE loss between two images of size $N \times N$. The terms $I(i)$ and $I'(i)$ denote the i^{th} pixel values in the images I and I' respectively and N^2 denotes the total number of pixels in each of the two images.

$$MSELoss = \frac{1}{N^2} \sum_{i=0}^{N^2-1} |I(i) - I'(i)|^2. \tag{3.3}$$

In the remaining part of this chapter, we describe the architecture and the methodology for each of the three methods.

3.2.1 Fully Connected Neural Networks

In this method, a four-layer fully connected neural network is the RDNet. The block diagram of this method is shown in Figure 3.6. The ReLU activation function is used between any two consecutive layers. Inside the RDNet, the image obtained at the end of the $(i-1)^{th}$ iteration is the input to the i^{th} iteration. Since the output obtained from layer 4 at $(i-1)^{th}$ iteration is the input to layer 1 for the i^{th} iteration, the dimensions of both of these images must be identical. Hence, the number of input neurons in layer 1 and the number output neurons in layer 4 must be equal.

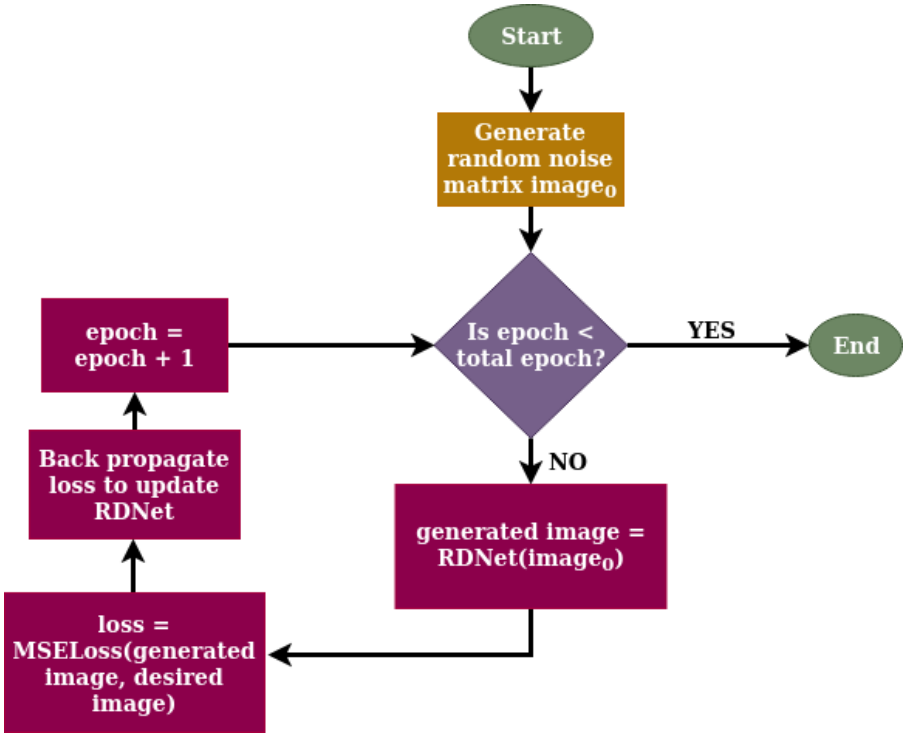


Figure 3.5: Simulation process for FCNN RDNet and CNN RDNet

The number of input neurons in layer 1 and the number output neurons in layer 4 are equal to the number of substrates required for generating any selected texture.

At the beginning of every iteration, each channel of the image is flattened. For an image of size $N \times N \times n\_groups$, the size of the flattened image is $N^2 \times n\_groups$. The flattened image passes through layer 1 and its shape becomes $N^2 \times n\_neurons$. This is again passed through the layers 2 and 3. In layers 2 and 3, the number of input and output neurons have been taken to be the same ($n\_neurons$). Hence the shape of the output from layer 3 remains $N^2 \times n\_neurons$. This passes through layer 4 and the shape of the output obtained is $N^2 \times n\_groups$. It is finally reshaped back to the shape of the input image, i.e, $N \times N \times n\_groups$. The term $n\_neurons$ is a parameter of this network and can be changed. We have taken $n\_neurons$ to be 100. Considering RDNet as a black box, the input and output method is shown in Figure 3.4.

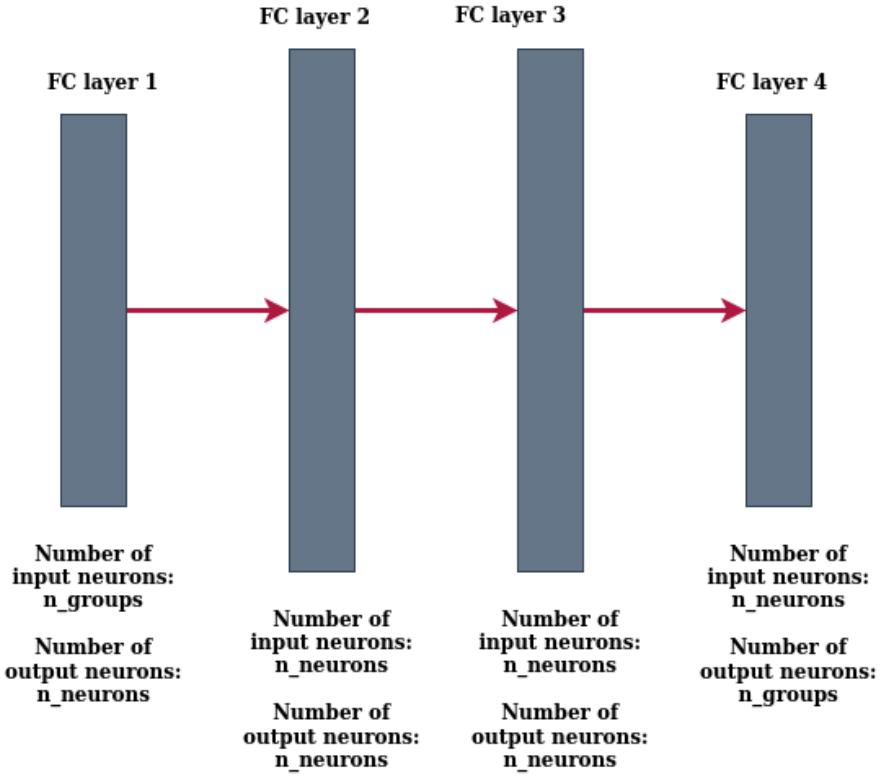


Figure 3.6: Block Diagram representing the FCNN RDNet

3.2.2 Convolutional Neural Network

Now, the fully connected layers have been replaced by convolutional layers inside the RDNet. Convolutional neural networks consist of several layers. Higher level features are extracted by the top layers of the network whereas the lower layers contain information about the lower level features. The entire network is trained simultaneously in order to optimize a given cost function. Figure 3.7 shows how the input image progresses through the network. The training procedure of a convolutional neural network comprises two major steps, the forward propagation and back propagation. During forward propagation, the input image passes through the successive layers and an output is obtained. Back propagation tries to minimise the deviation of the obtained output and the desired output. The forward propagation process at any intermediate l^{th} layer convolves the feature maps obtained from the

$(l-1)^{th}$ layer with a convolutional kernel.

$$M^l_{C_{out_j}} = f(bias^l_{C_{out_j}} + \sum_{k=0}^{C_{in}-1} (kernel^l_{C_{out_j}k} * M^{l-1}_k)). \tag{3.4}$$

Equations 3.4 depicts the operations involved in the forward propagation process. C_{in} and C_{out_j} denote the total number of input channels and the j^{th} output channel respectively. M^l_j denotes the j^{th} channel of the feature map corresponding to the l^{th} layer of the network and f is the activation function. The term $kernel^l_{ab}$ refers to the convolutional kernel of the l^{th} layer corresponding to the a^{th} input channel and b^{th} output channel and $bias^l_a$ of the l^{th} layer is the bias term corresponding to channel a.

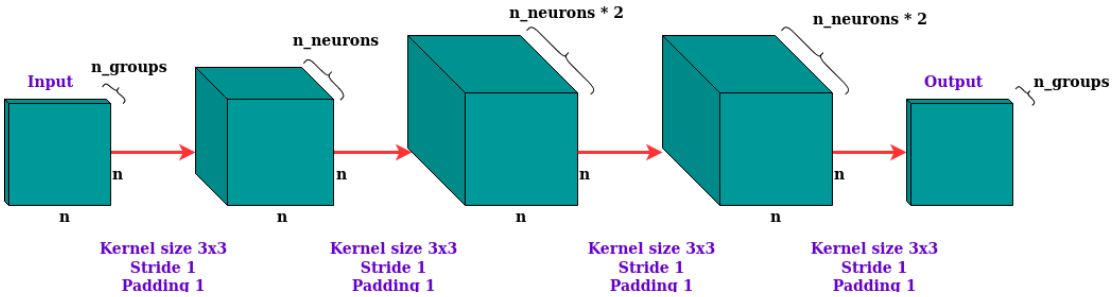


Figure 3.7: CNN RDNet

Figure 3.7 shows the CNN RDNet. The dimensions of the feature maps are shown at every step along with the convolutional details. Every feature map undergoes goes convolution with a kernel of a given size. Depending on the stride and the padding, the shape of the map changes. The architecture has 4 convolutional layers as depicted in Figure 3.7 and ReLU activation function is applied at the end of each layer except the last layer which uses the Tanh activation function (equation 3.5 where a is the input value). Batch Normalization is applied to the feature maps obtained from the second and third convolutional layers. The use of the Tanh activation function at the end of the network makes the optimization during the training robust [28].

$$f(a) = \frac{1-\exp(-2a)}{1+\exp(-2a)}. \tag{3.5}$$

3.3 Reaction-Diffusion GAN

The generator of this architecture is the CNN RDNet. The discriminator is very similar to the discriminator used in DCGAN. The discriminator designed for the reaction-diffusion GAN (RDGAN) has six layers including the input and the output layers and Leaky ReLU activation function has been used. Batch Normalisation has been used in all layers except the first and the last layers and the end of the discriminator network has the Sigmoid activation function. The discriminator of this GAN model is shown in Figure 3.8.

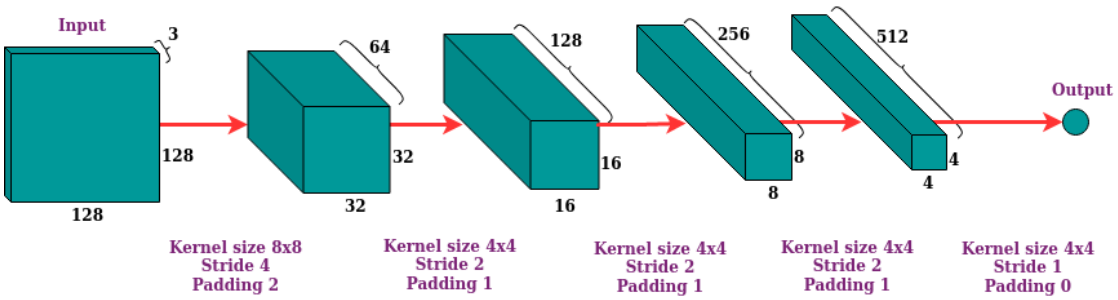


Figure 3.8: Discriminator of Reaction-Diffusion GAN

Three different losses are used for the RDGAN architecture:

1. **Adversarial Loss:** The discriminator network shown in Figure 3.8 is trained simultaneously with the generator network. We have used binary cross entropy (BCE) loss as our adversarial loss. The minimization of the adversarial loss indirectly trains the generator to synthesize realistic images.

2. **Map Loss:** Inner layers of a deep convolutional network contain semantic information about the training images. In [29], inner layers of a pre-trained VGG-19 network have been used for tuning the architecture. In a similar way, we have fine tuned a pre-trained VGG-19 with our dataset. With the tuned network, we have obtained the maps corresponding to the real and fake batches from an inner layer (ReLU 4_2). The mean squared difference between these maps is termed as the map loss.

3. **Quality Loss:** In [30], a novel loss is proposed that can consistently determine the perceptual quality of an image. This loss has been motivated

by the gradient magnitude similarity deviation ($GMSD$) which is obtained in three steps. The gradient magnitudes of the the real ($grad_R$) and fake images ($grad_F$) are obtained. The gradient magnitude similarity (GMS) is obtained with equation 3.6, where the term c is a positive constant that is introduced for maintaining the numerical stability. The terms $GMS(i)$, $grad_R(i)$ and $grad_F(i)$ are the corresponding values at the i^{th} index of the GMS, the gradient of the real image batch and the gradient of the fake image batch respectively.

$$GMS(i) = \frac{2 \cdot grad_R(i) \cdot grad_F(i) + c}{grad_R^2(i) + grad_F^2(i) + c}. \tag{3.6}$$

The gradient magnitude similarity mean ($GMSM$) is obtained according to equation 3.7 [30], where N is the total number of pixels in the real and fake images.

$$GMSM = \frac{1}{N} \sum_{i=1}^{N} GMS(i). \tag{3.7}$$

Finally, the gradient magnitude similarity deviation [30] is calculated as:

$$GMSD = \sqrt{(\frac{1}{N} \sum_{i=1}^{N} (GMS(i) - GMSM)^2)}. \tag{3.8}$$

The quality loss between the real and fake images is defined as the $GMSD$ between real and fake image batches.

Let L_{ADV}, L_{MAP} and L_Q represent the adversarial loss, the map loss and the quality loss respectively. During the training process, the objective is to minimize these losses. If w_{ADV}, w_{MAP} and w_Q represent the coefficients of L_{ADV}, L_{MAP} and L_Q respectively, the objective function becomes:

$$L = w_{ADV} L_{ADV} + w_{MAP} L_{MAP} + w_Q L_Q. \tag{3.9}$$

3.4 Summary

In this chapter, we have described three different architectures. The objective is to generate images by reaction-diffusion method without actually knowing the parameters of the system. The models have been described in such a way that each model is more complex than the preceding one. With the proposed architectures, we have conducted the experiments which are described in the next chapter.

Chapter 4

Experiments

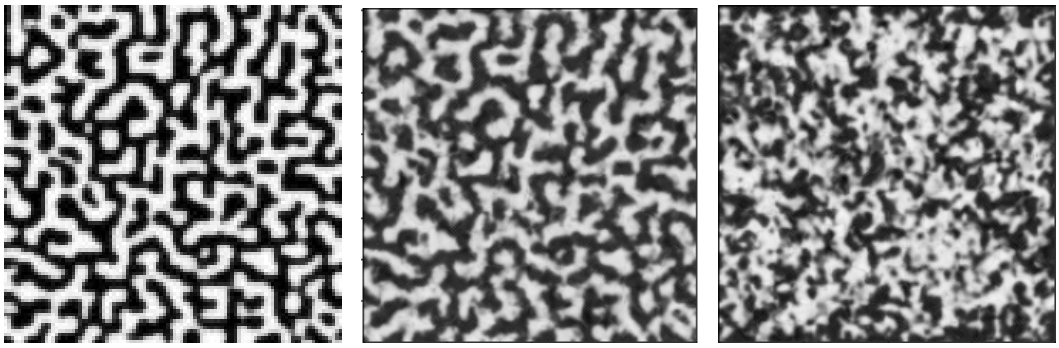
4.1 Fully Connected Neural Networks

We started our experiment with a pattern which looks somewhat similar to the steel microstructure pattern and can also be generated by a known reaction-diffusion system. Figure 4.1(a) shows a famous Turing pattern known as the stripe pattern. This pattern can be modelled with the Turing reaction-diffusion model with four different substrates. This pattern has been generated with the traditional reaction-diffusion algorithm in 10000 iterations. The reaction terms and the diffusion terms corresponding to the traditional iterative approach are provided in [31].

In order to obtain the stripe pattern with FCNN RDNet, we have generated a noise matrix of size 200 x 200 x 4 as our input. The number of iterations required inside FCNN RDNet is 25, which is much lesser than the iterative approach. The learning process was completed in 5000 epochs. Adam optimizer with a learning rate of 0.001 was used. Figure 4.1(b) shows the stripe pattern that is generated with FCNN RDNet. Since Figure 4.1(b) is very similar to Figure 4.1(a), we can safely assume that our network is working as desired for a Turing pattern with known reaction-diffusion terms. Figure 4.1(c) shows the pattern that is generated when a new noise matrix is fed as input to the trained FCNN RDNet.

Now we try to tune our parameters for generating the steel microstructure pattern shown in Figure 4.2(a). The number of substrates that would be required for generating this microstructure is completely unknown. Hence, we start with 2 substrates and depending on the visual quality of the generated image, we increase or decrease the number of substrates used. Figures 4.2(b), 4.2(c), 4.2(d), 4.2(e) and 4.2(f) show the images generated using two, three, four, five and six substrates respectively. From these figures, we can see that the images generated corresponding to four substrates is better than the others. Hence we can see that even without knowing anything related to the reaction-diffusion parameters corresponding to the

steel microstructure images, textures very similar to the steel microstructures can be generated with FCNN RDNet.



(a) (b) (c)

Figure 4.1: Turing pattern generated with FCNN RDNet (a): Original stripe pattern (b): Stripe pattern generated using FCNN RDNet (c): Image generated by the trained FCNN RDNet with a new noise matrix

4.2 Convolutional Neural Network

The size of the initial noise matrix is 200 x 200 x 4. For all the layers of the convolutional network, kernels are sized 3 x 3 and replication padding has been used. The total number of iterations required inside the CNN RDNet is 25. The training procedure required 1500 epochs. Adam optimizer is used and the learning rate is 0.001.

We have generated the images corresponding to two different images (Figure 4.3(a) and 4.3(d)) with four substrates. Each row of Figure 4.3 shows the results corresponding to a particular image. Figures 4.3(b) and 4.3(e) are the images generated from Figures 4.3(a) and 4.3(d) by training the CNN RDNet. However, when new noise matrices are fed to CNN RDNet trained with Figures 4.3(a) and 4.3(d), Figures 4.3(c) and 4.3(f) are generated. Figures 4.3(c) and 4.3(f) indicate that the network cannot generalize well.

4.3 Reaction-Diffusion GAN

The training images are not subjected to any sort of pre-processing. The model is trained in mini-batches and the batch size is 5. A batch of training images is shown

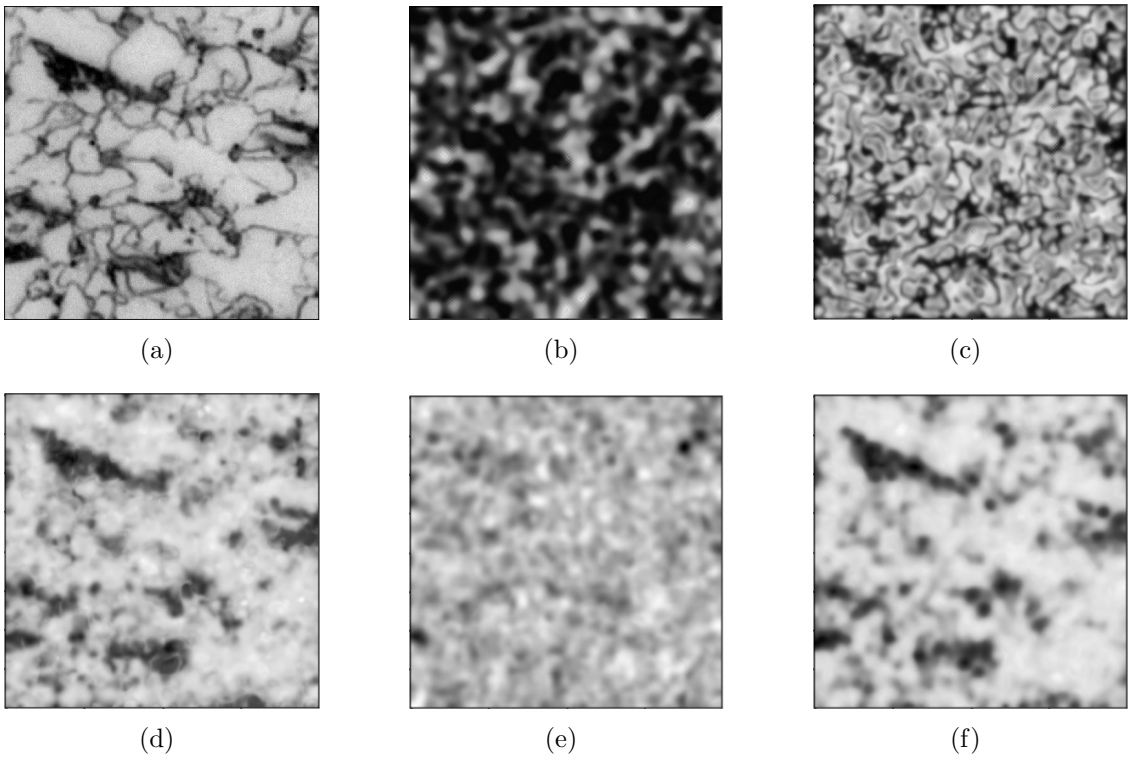


Figure 4.2: Steel microstructure images generated with FCNN RDNet (a): Original steel microstructure image (b): Image generated using 2 substrates (c): Image generated using 3 substrates (d): Image generated using 4 substrates (e): Image generated using 5 substrates (f): Image generated using 6 substrates

in Figure 4.4.

In both the generator and the discriminator networks, Leaky ReLU activation function is used having slopes of 0.002 and 0.0002 respectively. Adam optimizer has been used with a learning rate of 0.001. The model is trained in 200 epochs. The weights w_{ADV}, w_{MAP} and w_Q are set as 1, .05 and 25 respectively. The VGG-19 network is fine tuned by training the last two layers of the network for 10 epochs. The results obtained are shown in Figure 4.5. The constant c in equation 3.6 is taken as 1. Any two batches of real image is seen to have L_Q value close between 0.10 and 0.12. Hence, the objective is to be able to generate images for which L_{MAP} is close to zero and L_Q is between 0.10 and 0.12.

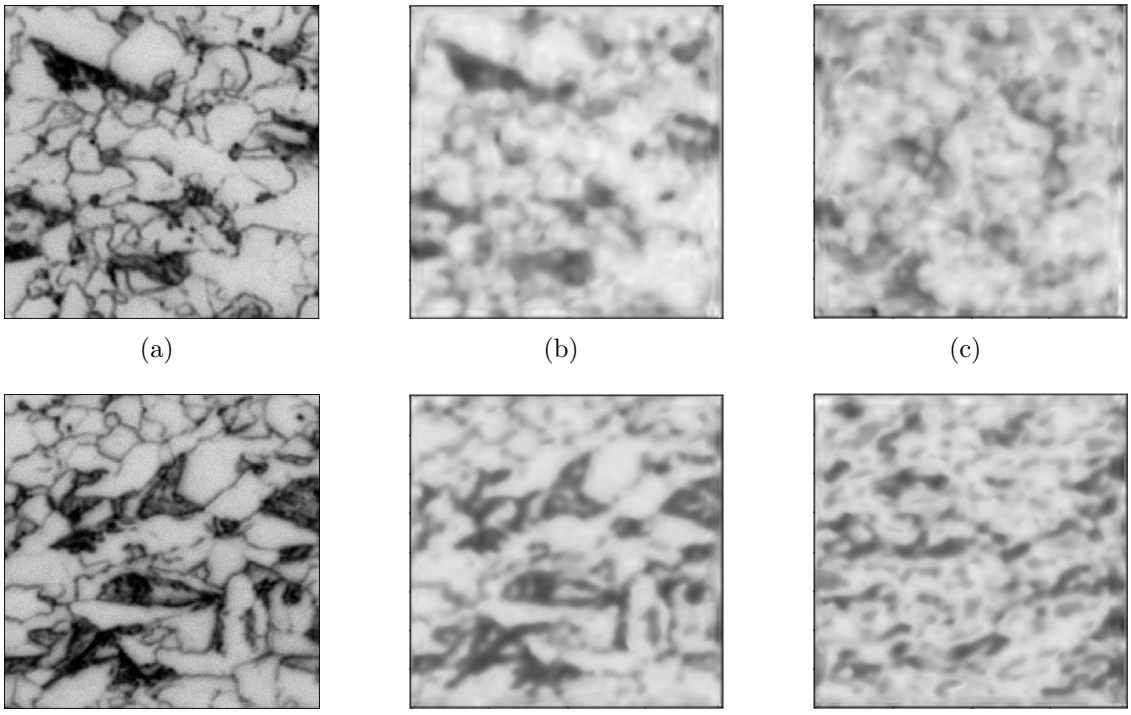


Figure 4.3: Steel microstructure images generated with CNN RDNet (a) A steel microstructure image from the training dataset (b): Image generated by training CNN RDNet using image **(a)** (c): Image generated when a new noise matrix is fed as input to CNN RDNet trained with image **(a)** (d) A steel microstructure image from the training dataset (e): Image generated by training CNN RDNet using image **(d)** (f): Image generated when a new noise matrix is fed as input to CNN RDNet trained with image **(d)**

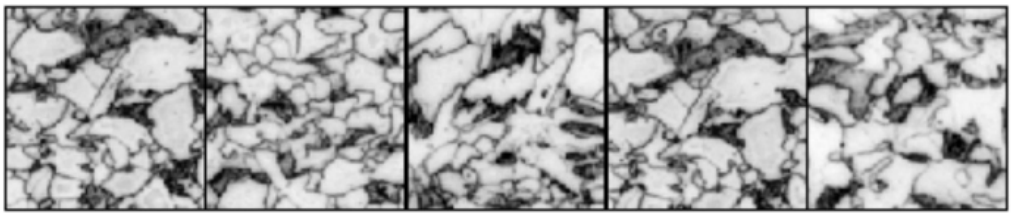
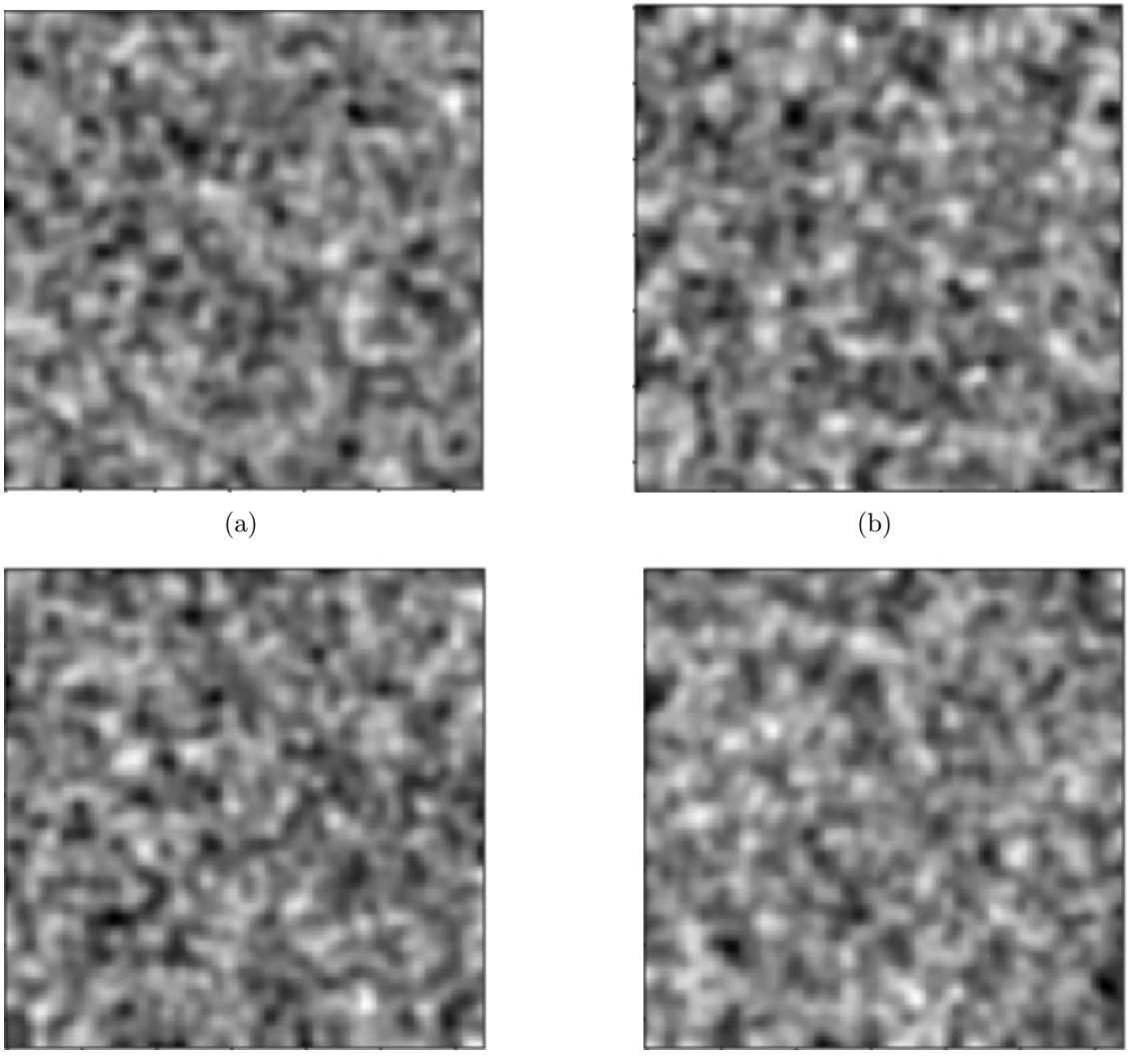


Figure 4.4: A batch of images used for training Reaction-Diffusion GAN



(a) (b)

(c) (d)

Figure 4.5: Steel microstructure images generated with RDGAN

4.4 Discussion

Method	Calculated Error
FCNN RDNet	0.0311
CNN RDNet	**0.0237**
RDGAN	0.0539
DCGAN	0.0241
PatchMatch	0.0302

Table 4.1: Comparison between the different methods. Details of error calculation are given in Section 4.4

Table 4.1 shows the errors that are obtained with the different methods. For FCNN RDNet, CNN RDNet and RDGAN, the images that have been generated with four substrates have been used for calculating the errors. The error is calculated for each method separately. The training set contains 300 images. Considering each method, let *Generated_Images* be the set of all images generated by that method. For a particular method, for every *imageG* in the *Generated_Images* set for that method, the MSE loss is calculated with all the images in the training dataset. So, for any method, if nG be the total number of images generated by it, then $nG \times 300$ number of losses are calculated for that method. The minimum loss among all the calculated losses is reported for each method. Table 4.1 shows that CNN RDNet has the lowest error followed by DCGAN.

The results shown in Figure 4.1 suggests that FCNN RDNet can generate the **stripe** pattern quite well. Figure 4.2 indicates that the images generated by FCNN RDNet are close to the steel microstructure images though it is not capable of generating the detailed structures that are present in the microstructures. However, Figure 4.1(c) suggests that FCNN RDNet cannot generalize well, i.e, given a new noise matrix, the trained network cannot generate a similar stripe pattern. Figures 4.3(b) and 4.3(e) indicates that CNN RDNet can generate images which are very close to the desired steel microstructure images. Unlike FCNN RDNet, CNN RDNet is capable of reproducing the intricate structures present in the steel microstructure images. But Figures 4.3(c) and 4.3(f) suggest that even this network cannot generalize.

Hence we modify CNN RDNet and build the RDGAN architecture. The generator of RDGAN is the CNN RDNet. Since Figures 4.3(b) and 4.3(e) are very similar to Figures 4.3(a) and 4.3(d) respectively, it was hoped that RDGAN would be able to generate images very similar to the steel microstructure images. However, the images in Figure 4.5 indicate something different. Even after tuning the hyper parameters of RDGAN extensively, the generated images are quite different from the desired steel microstructure images. This might be due to the objective functions that we have considered for the generator and the discriminator networks. The inaccurate results might also be due to the selection of the incorrect optimizer or incorrect loss functions.

Method	Total Epochs	GPU	Time Taken
PatchMatch	2	**4 GB RAM Intel Core i3-4005U CPU 1.7 GHz, 3MB L3 cache**	145 minutes
FCNN RDNet	5000	**Google Colab**	10 minutes
CNN RDNet	1500	**Google Colab**	25 minutes

Table 4.2: Time taken by PatchMatch, FCNN RDNet and CNN RDNet for generating steel microstructure images

Type of GAN	Total Epochs	GPU	Time Taken
DCGAN	2000	**64 GB RAM Intel Core i7-7700K CPU @ 4.2 GHz x 8 TITAN XP GPU**	1540 minutes
RDGAN	200	**94 GB RAM Intel Core i9-9820X CPU @ 3.3 GHz x 20 TITAN XP GPU**	490 minutes

Table 4.3: Time comparison between DCGAN and proposed RDGAN

Table 4.2 shows the time taken by PatchMatch, FCNN RDNet and CNN RDNet. Table 4.3 compares the time taken for generation of steel microstructure images by DCGAN and the proposed RDGAN. PatchMatch is a traditional algorithm. Since we have repeated the entire PatchMatch method twice, the total number of epochs taken as 2. The time taken by PatchMatch is much more than FCNN RDNet and CNN RDNet though all three of them generate a single texture with a single reference image. For FCNN RDNet and CNN RDNet, the training process involves only one image. Hence they converge pretty fast. On the other hand, in case of RDGAN and DCGAN, the training involves the entire dataset having 300 images. The time required for training RDGAN is much less than the time required for training DCGAN. Also, the total number of epochs required for training DCGAN is much more than what is required for training RDGAN.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this work, we have explored different methods for generating the steel microstructure images. The steel microstructure images generated by PatchMatch depends largely on the neighbourhood selection. Moreover, the time required by PatchMatch is much more than the other methods. Compared to FCNN RDNet and CNN RDNet, PatchMatch takes much more time to generate the desired images even though all the three methods generate a texture image using a single input reference image. The images generated by DCGAN are very good, but it suffers from mode collapse and needs to be trained for a large number of epochs. FCNN RDNet can generate the stripe pattern very well without any knowledge about the reaction-diffusion parameters of the pattern. It can also generate images similar to the steel microstructure images. However, FCNN RDNet cannot generate the intricate details that are present in the steel microstructure images. CNN RDNet, on the other hand, can generate images that are very close to the steel microstructure images. However, it cannot generalize. RDGAN architecture is more complex than it seems to be apparently and the coefficients of the losses involved have to be tuned very minutely for generating the desired images.

5.2 Future Work

In future, we would like to analyse the RDGAN architecture and try to understand why it cannot generate the desired images. The reason might be the choice of irrelevant loss functions, incorrect optimizer or inaccurate learning rates. Each of these aspects needs to be checked minutely before arriving at a conclusion. Since CNN RDNet can generate quite satisfactory results, RDGAN should also be able to do it. This is because the generator of RDNet is CNN RDNet. But since GANs are

extremely unstable, the hyper parameter tuning process is a very important step. Due to the incorrect choice of hyper parameters, everything might collapse. We hope to analyse all these aspects in future and modify the RDGAN architecture to suit our requirements. We would also like to experiment with other microstructure textures and different kinds of natural textures and propose an architecture that would be able to generate them.

Bibliography

[1] L.-Y. Wei and M. Levoy, "Fast texture synthesis using tree-structured vector quantization," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000, pp. 479–488.

[2] A. Paul, A. Gangopadhyay, A. R. Chintha, D. P. Mukherjee, P. Das, and S. Kundu, "Calculation of phase fraction in steel microstructure images using random forest classifier," *IET Image Processing*, vol. 12, no. 8, pp. 1370–1377, 2018.

[3] M. Ashikhmin, "Synthesizing natural textures," in *Proceedings of the 2001 symposium on Interactive 3D graphics*, 2001, pp. 217–226.

[4] Y. Wexler, E. Shechtman, and M. Irani, "Space-time completion of video," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 29, no. 3, pp. 463–476, 2007.

[5] J. Kopf, C.-W. Fu, D. Cohen-Or, O. Deussen, D. Lischinski, and T.-T. Wong, "Solid texture synthesis from 2d exemplars," in *ACM SIGGRAPH 2007*, 2007, pp. 2–es.

[6] L.-Y. Wei, K. Zhou, J. Han, B. Guo, and H.-Y. Shum, "Inverse texture synthesis," Jul. 5 2011, uS Patent 7,973,798.

[7] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra, "Texture optimization for example-based synthesis," in *ACM SIGGRAPH 2005*, 2005, pp. 795–802.

[8] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001, pp. 327–340.

[9] S. Lefebvre and H. Hoppe, "Parallel controllable texture synthesis," in *ACM SIGGRAPH 2005*, 2005, pp. 777–786.

[10] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, "Patchmatch: A randomized correspondence algorithm for structural image editing," *ACM Trans. Graph.*, vol. 28, no. 3, p. 24, 2009.

[11] A. M. Turing, "The chemical basis of morphogenesis," *Bulletin of mathematical biology*, vol. 52, no. 1-2, pp. 153–197, 1990.

[12] J. B. Bard, "A model for generating aspects of zebra and other mammalian coat patterns," *Journal of Theoretical Biology*, vol. 93, no. 2, pp. 363–385, 1981.

[13] J. D. Murray, "On pattern formation mechanisms for lepidopteran wing patterns and mammalian coat markings," *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, vol. 295, no. 1078, pp. 473–496, 1981.

[14] H. Meinhardt, "Models of biological pattern formation," *New York*, p. 118, 1982.

[15] G. Turk, "Generating textures on arbitrary surfaces using reaction-diffusion," *Acm Siggraph Computer Graphics*, vol. 25, no. 4, pp. 289–298, 1991.

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[17] I. Goodfellow, "Nips 2016 tutorial: Generative adversarial networks," *arXiv preprint arXiv:1701.00160*, 2016.

[18] Y. Bengio, E. Laufer, G. Alain, and J. Yosinski, "Deep generative stochastic networks trainable by backprop," in *International Conference on Machine Learning*, 2014, pp. 226–234.

[19] Y. Bengio, L. Yao, G. Alain, and P. Vincent, "Generalized denoising auto-encoders as generative models," in *Advances in neural information processing systems*, 2013, pp. 899–907.

[20] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[21] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[22] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," *arXiv preprint arXiv:1802.05957*, 2018.

[23] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[24] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *International Conference on Machine Learning*, 2019, pp. 7354–7363.

[25] Y. Liu, P. Kothari, and A. Alahi, "Collaborative sampling in generative adversarial networks," *arXiv preprint arXiv:1902.00813*, 2019.

[26] S. T. Acton, D. P. Mukherjee, J. P. Havlicek, and A. C. Bovik, "Oriented texture completion by am-fm reaction-diffusion," *IEEE Transactions on Image Processing*, vol. 10, no. 6, pp. 885–896, 2001.

[27] T. T. Marquez-Lago and P. Padilla, "A selection criterion for patterns in reaction–diffusion systems," *Theoretical Biology and Medical Modelling*, vol. 11, no. 1, pp. 1–17, 2014.

[28] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[29] W. Xian, P. Sangkloy, V. Agrawal, A. Raj, J. Lu, C. Fang, F. Yu, and J. Hays, "Texturegan: Controlling deep image synthesis with texture patches," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8456–8465.

[30] X. Zhu, L. Zhang, L. Zhang, X. Liu, Y. Shen, and S. Zhao, "Gan-based image super-resolution with a novel quality loss," *Mathematical Problems in Engineering*, vol. 2020, 2020.

[31] W. Wang, Y. Lin, F. Rao, L. Zhang, and Y. Tan, "Pattern selection in a ratio-dependent predator–prey model," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2010, no. 11, p. P11036, 2010.