HOCHSCHULE
MITTWEIDA
University of Applied Sciences

BACHELOR THESIS

Ms.
Ngoc Hien Nguyen

GANs for Numerical Simulation with Predefined Conditions on Statistical Properties of Images

Mittweida, September 2022

Faculty of **Applied Computer Sciences and Biosciences**

BACHELOR THESIS

GANs for Numerical Simulation with Predefined Conditions on Statistical Properties of Images

Author:
Ngoc Hien Nguyen

Course of Study:
Applied Mathematics

Seminar Group:
MA19w1-B

First Examiner:
Prof. Dr. rer. nat. Florian Zaussinger

Second Examiner:
Dr. rer. nat. , Marika Kaden

Submission:
Mittweida, 07.09.2022

Defense/Evaluation:
Mittweida, 21.09.2022

Faculty of **Applied Computer Sciences and Biosciences**

BACHELOR THESIS

GANs for Numerical Simulation with Predefined Conditions on Statistical Properties of Images

Author:
Ngoc Hien Nguyen

Course of Study:
Applied Mathematics

Seminar Group:
MA19w1-B

First Examiner:
Prof. Dr. rer. nat. Florian Zaussinger

Second Examiner:
Dr. rer. nat. , Marika Kaden

Submission:
Mittweida, 07.09.2022

Defense/Evaluation:
Mittweida, 21.09.2022

Bibliographic Description:

Nguyen, Ngoc Hien :
GANs for Numerical Simulation with Predefined Conditions on Statistical Properties of Images. – 2022. – 50 S.
Mittweida, Hochschule Mittweida – University of Applied Sciences, Faculty of Applied Computer Sciences and Biosciences, Bachelor Thesis, 2022.

Abstract:

Simulating complex physical systems involves solving nonlinear partial differential equations (PDEs), which can be very expensive. Generative Adversarial Networks (GAN) has recently been used to generate solutions to PDEs-governed complex systems without having to numerically solve them. However, concerns are raised that the standard GAN system cannot capture some important physical and statistical properties of a complex PDE-governed system [14], along side with other concerns for difficult and unstable training [12], the noisy appearance of generated samples [1] and lack of robust assessment methods of the sample quality apart from visual examination [13]. In this thesis, a standard GAN system is trained on a data set of Heat transfer images. We show that the generated data set can capture the true distribution of training data with respect to both visual and statistical properties, specifically the vertical statistical profile. Furthermore, we construct a GAN model which can be conditioned using variance-induced class label. We show that the variance threshold $t = 0.01$ constructs a good conditional class label, such that the generated images achieve 96% accuracy rate in complying with the given conditions.

Contents

List of Figures

List of Tables

Acknowledgment

A special thanks to Professor Florian Zaussinger, my supervisor, for introducing me to the topic, and for always being there when I needed support, reviewing my progress constantly, and guiding me through the project.

1 Introduction

1.1 Literature Review

Simulating complex physical systems involves solving nonlinear partial differential equations (PDEs) with multi-scale unresolved physics. Such simulations, despite being made possible by the advancement of high performance computing, are still very expensive.

The motivation of this bachelor thesis comes from the introduction of Generative Adversarial Networks (GAN) framework as a promising framework for generating solutions to PDEs-governed complex systems without having to numerically solve these PDEs.

GAN is different from other generative models, such as Autoencoder. The generator in generative models creates new data instances from certain codes. However, training a generator to convergence is simply training a duplicating generator, which serves the purpose of efficiently represent the training data distribution. Whereas the Generator in GAN serves to produce new and novel data instances that resemble the training samples. GAN achieves that by introducing a second player, the Discriminator, who tries to identify the correct image source, i.e. if an image comes from the training data or generated data. The presence of an opponent does two things to the Generator. On one side it pressures the Generator into creating realistic images. On another side, when the Discriminator does not force a generated sample to resemble one existing solution, the Generator is given freedom to try a wide range of new and novel samples.

Although the standard GANs can capture the true distribution of training data when global minimum of the loss function is achieved [5, 2], there are concerns that the simple GANs is unlikely to capture all the statistics of the solutions for a complex PDE-governed system, or reproduce some important physical and statistical properties of the system [14]. In additions, other challenges have been associated with the simple GAN applications, including difficult and unstable training [12], the noisy appearance of generated samples [1] and lack of robust assessment methods of the sample quality apart from visual examination [13].

1.1.1 GANs for Heat Transfer data

Considering the great potential of GANs to physical systems and the limitation of visual examination, it is worthwhile to investigate the **applicability of GANs in emulating physical systems**, and the statistical properties of generated samples. The first point of interest of this work is to build a simple GAN system, which is loosely based on the DCGAN architecture [12], as an emulator of complex PDE- governed systems. DCGAN stands for “deep, convolution GAN”. Some of the key insights of the DCGAN architecture were to: Use batch normalization [6]; Use strided convolutions (discriminator) and fractional-strided convolutions (generator), see figure 1.1 ; Use Adam optimizer.

The physical system we investigated in this work is a Heat Transfer simulation on a 2-D slab of fluid, which is bounded by two horizontal surfaces. The dataset we used for training a simple GAN model to emulate the physical system consisted of an image sequence illustrating a numerical simulation

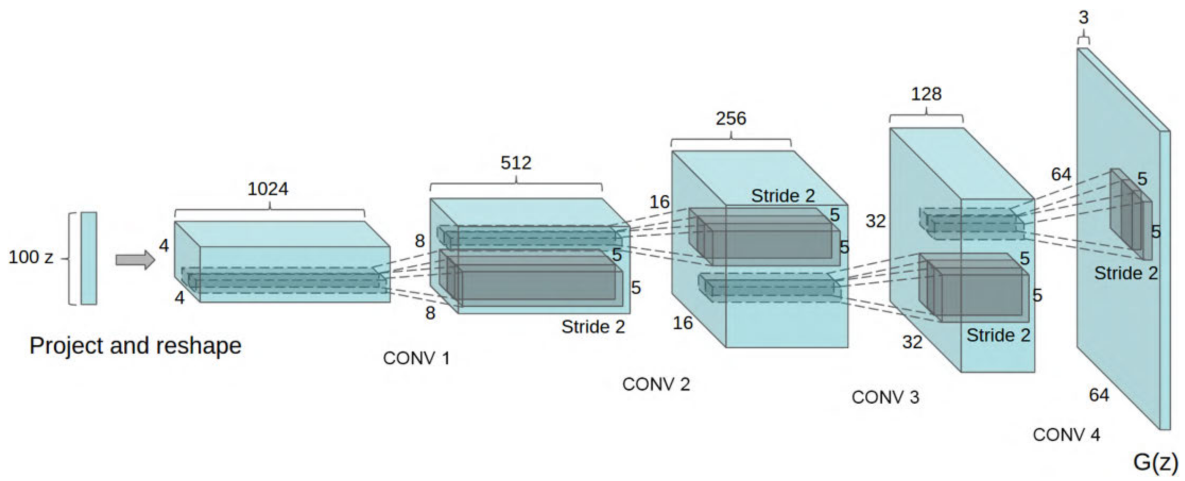


Figure 1.1: The generator network used by a DCGAN. Figure reproduced from [12]

of Heat transfer. The generated images are then subject to both visual examination and statistics examination to assess if important physical and statistical properties are captured by the simple GAN model.

1.1.2 Conditionality

Generally, conditional generative models often garner more interest from researchers compared to unconditional generative models, in most machine learning disciplines including GANs. In the context of physics systems, it is highly interesting to study the outliers. However, these samples appear at extremely low frequency, and are hard to come by, at least in an organic setting. Developing a conditional generative model that selectively generates these interesting outlier-equivalent samples, is another motivation of the thesis.

Conditional generation with GANs entails using labeled data to generate images based on a certain class. The two representative variations of GANs that employ this principle are Conditional GANs (cGAN [10]) and Aux-iliary Classifier GANs (AC-GAN [11]).

In Conditional GANs system, the discriminator and the generator are conditioned on c, which could be a class label or some data from another modality. The input and c are combined in a joint hidden representation and fed as an additional input layer in both networks. Provided with condition c, the Discriminator identifies fake and real images.

Contrary to CGANs, in Aux-iliary Classifier GANs (AC-GAN), the latent space z is conditioned on the class label. The discriminator is forced to identify fake and real images, as well as the class of the image, irrespective of whether it is fake or real.

A second point of interest for the thesis is the **conditional generation of a physical system**, based on some statistical property threshold. Applications that employed the two above condition-driven GANs define class labels based on multiple features on the image, e.g. in cGAN example of MNIST data set, the model learns to differentiate the digit 8 and 1 based on the image pixel values only. However, this thesis attempts to define class labels based on features that are not directly presented on the image, but rather on the output of some function applied on the image, e.g. the vertical variance of the image.

Depending on the context of heat transfer training data, and the construction of the statistics class labels, an appropriate variation GANs model will be chosen and applied. The accuracy rate between the predicted class labels and the ground-truth class labels of the result generated images are the basis to evaluate the performance of the chosen model.

1.2 Paper overview

The thesis is presented with the following structure: Chapter 2 describes the theoretical frameworks of necessary GAN models, followed by the method to obtain two kinds of data, the real data and the fake data generated by said GAN models. Chapter 3 presents the result fake images, and comparison against the corresponding real data, along with examples of some unsuccessful experiments. Chapter 4 discusses the results with regards to how the GAN performs compared to the expected picture painted by literature.

2 Methods

2.1 Theoretical Frameworks

This section provides the framework to tackle the two research questions raised in the Introduction. The first part discusses the statistical moment variables for the heat transfer images. These variables form a reliable method of describing a heat transfer image, and comparing any two images. The next part will deep-dive into an unconditional GAN system, referred to as the “simple GAN” system. Here we unpack the neural networks model to understand its elements, the loss functions, and the training procedure. This system is used to answer the first research question. Following that is the discussion of “conditional GAN” (or cGAN), a more sophisticated system which allows the control of the image generation, and discussion of a problem within the system which prevents the immediate application on Heat transfer images. The problem leads to the next discussion of an “adjusted cGAN” system, where some adjustments are introduced for the application of heat transfer images. This system is used to answer the second research question.

2.1.1 Statistical moments of Heat transfer images

In this section, the statistical moments are defined for Heat transfer images, which form the basis for comparing the relevant statistic properties of two arbitrary images.

Let x be an image of a heat transfer stage, represented by a *n-by-n* matrix:

$$x = (x_{i,j}) \in \mathbb{R}^{n\times n} \quad , i, j = 1, \dots, n.$$

Let $\mu_i(x)$ be the mean value of row i of x, and $M_1(x)$ be the **row-wise mean vector** of x, defined respectively:

$$\mu_i(x) = \frac{1}{n}\sum_{j=1}^{n} x_{i,j} \quad \in [0,1] \tag{2.1}$$

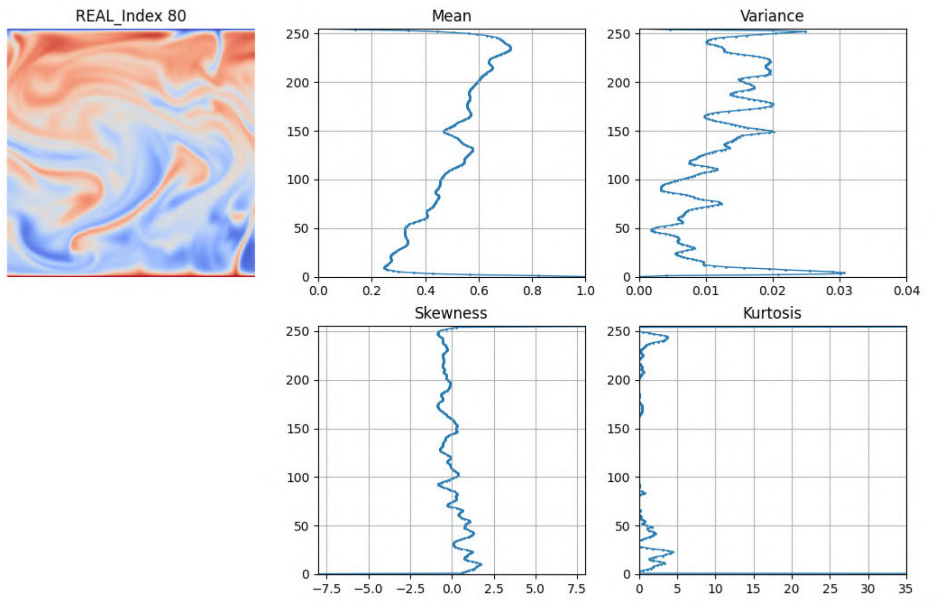
$$M_1(x) = \begin{pmatrix} \mu_1(x) \\ \vdots \\ \mu_n(x) \end{pmatrix} \in [0,1]^n \tag{2.2}$$

Let $\text{var}_i(x)$ be the variance value of row i of x, and $M_2(x)$ be the **row-wise variance vector** of x, defined respectively:

$$\text{var}_i(x) = \mathbf{E}\Big[(x - \mu_i(x))^2\Big] \quad \in \mathbb{R} \tag{2.3}$$

$$M_2(x) = \begin{pmatrix} \text{var}_1(x) \\ \vdots \\ \text{var}_n(x) \end{pmatrix} \in \mathbb{R}^n \tag{2.4}$$

Figure 2.1: *Real Index 80* is an image of Heat transfer at t=80; *Mean* plots the mean vector on the x-axis against the row indices; similarly for the *Variance* plot, *Skewness* plot and *Kurtosis* plot



Let skew$_i(x)$ be the skewness value of row i of x, and $M_3(x)$ be the **row-wise skewness vector** of x, defined respectively:

$$\text{skew}_i(x) = \mathbf{E}\Big[\Big(\frac{x - \mu_i(x)}{\sigma_i}\Big)^3\Big] \quad \in \mathbb{R} \tag{2.5}$$

$$M_3(x) = \begin{pmatrix} \text{skew}_1(x) \\ \vdots \\ \text{skew}_n(x) \end{pmatrix} \in \mathbb{R}^n \tag{2.6}$$

where $\sigma_i(x) = \sqrt{\text{var}_i(x)}$ is the standard deviation value of row i in x.

Let kurt$_i(x)$ be the kurtosis value of row i of x, and $M_3(x)$ be the **row-wise kurtosis vector** of x, defined respectively:

$$\text{kurt}_i(x) = \mathbf{E}\Big[\Big(\frac{x - \mu_i(x)}{\sigma_i}\Big)^4\Big] \quad \in \mathbb{R} \tag{2.7}$$

$$M_4(x) = \begin{pmatrix} \text{kurt}_1(x) \\ \vdots \\ \text{kurt}_n(x) \end{pmatrix} \in \mathbb{R}^n \tag{2.8}$$

where $\sigma_i(x) = \sqrt{\text{var}_i(x)}$ is the standard deviation value of row i in x.

For example, the image in figure 2.1 can be described as follows:

The first sub-figure is the image x of size 256×256. Each pixel displays heat signature between 0 (blue as cold temperature) and 1 (red as hot temperature).

The mean vector $M_1(x)$ is plotted against the row indices of the main image. Every row i of the image contributes a scalar mean value $\mu_i(x)$, as a dot on the corresponding row index of the Mean plot. Considering the image from bottom to top, the temperature goes from consistently hot, to a mixture of hot and cold in the middle rows, to consistently cold at the top row. Correspondingly in the Mean plot with ascending row indices marked on y-axis, the mean value of each row goes from 1 on the 0-th row ($\mu_0(x) = 1$), to ranging in-between and goes to 0 on the 255-th row ($\mu_{255}(x) = 0$).

Similarly, each dot of value on the Variance plot is responsible for describing the variance of the corresponding row in the main image. In this case, the row-wise variance reaches the maximum value at the 4-th row ($\text{var}_4(x) = 0.03$), indicating that the 4-th row of the image near the bottom is where the image observes the greatest difference in temperature across all columns of this row. The red swirls appearing around near this area might be responsible for this observation.

Similar explanations apply for the plots of Skewness and Kurtosis.

Using the defined row-wise statistics such as mean vector, variance vector, skewness vector and kurtosis vector, we can describe any heat transfer image x by 1-dimensional vectors, as the example above.

Furthermore, these statistical variables can be used to compare two arbitrary images x and y, more importantly compare between a real and fake image, providing a more effective evaluation method for a GAN system performance, expectantly superior to the traditional method of visually checking of the images only.

2.1.2 What is a GAN?

A GAN system comprises of three elements: two neural networks and a set of training real examples. The first neural network, outputs a fake sample $G(z, \theta_G)$ from an noise variable z, where G is a differentiable function with parameter θ_G. The second neural network outputs a single scalar $D(x, \theta_D)$, which measures the likelihood that x came from the training set, rather than the sample set from G. We train D to maximize the probability of correctly labeling the training real samples and fake samples. Simultaneously, we train G to minimize D's ability to correctly identify fake samples from G. The adversary between the two networks is the driving force for the GAN system to improve after each training iteration, and is described by the expression proposed in [5]:

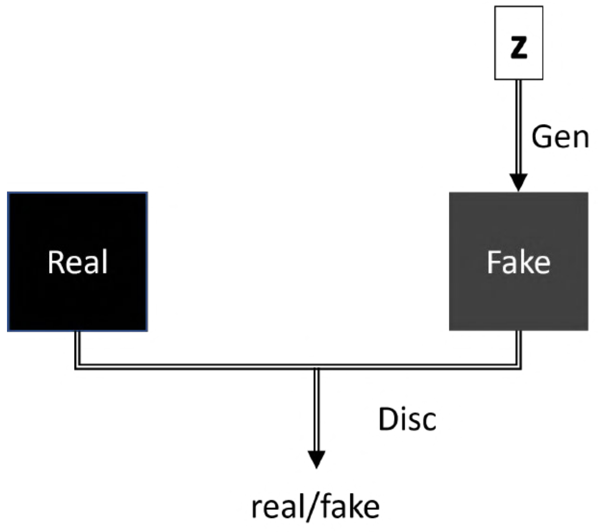
$$\min_G \max_D V(D, G) = E_{x \sim P_{data}(x)}[logD(x)] + E_{z \sim P_z(z)}[log(1 - D(G(z)))] \tag{2.9}$$

Value V sums the expected log likelihood for a sample either real or fake. In other words, value V represents the likelihood (probability) that the Discriminator correctly identify the sample origin. The Discriminator seeks to maximize this quantity V, while the Generator seeks to minimize it. In most application cases, we define separate loss functions for each neural networks.

The intuitive understanding of the loss function of each network is motivated in this section. We formally introduce the system elements, and their notation as follows:

- P_{data}: is the distribution of real training data. x is a real sample drawn from this distribution.

Figure 2.2: Diagram of a simple GAN system.



- P_z: is the distribution of noise (often a Gaussian distribution). z is a noise sample drawn from this distribution.
- G: is the Generator function. G takes a noise z as input, and returns a fake sample $G(z)$.
- $D(x)$: The Discriminator's evaluation of a real sample x.
- $D(G(z))$: The Discriminator's evaluation of a fake sample $G(z)$.
- Error(a, b): Error between a and b.

The Discriminator

The Discriminator is a differentiable function that takes input x and uses parameters θ_D. The Discriminator's goal is to correctly identify the sample source: to label a real sample as true and a generated sample as false. Following this goal, D has a loss function that calculates its evaluation errors:

$$L_D = \text{Error}(D(x), 1) + \text{Error}(D(G(z)), 0) \tag{2.10}$$

The Discriminator is trained using two batches of data. One batch coming from the training data set, where the ground-truth label is "1" for all real samples. The other batch coming from the generator, where the ground-truth label is "0" for all generated samples. The loss function represents a value that D wishes to minimize, which is error between D's predictions and the ground-truth labels. At this stage, we notate the loss function using a generic unspecific "Error" function that tells us the distance between two functions. The choices for this error function include cross-entropy or Kullback-Leibler divergence, generally depending on the applications. We will define this error function more specifically later.

The Generator

The Generator is a differentiable function G that takes noise input z and uses parameters θ_G. The Generator works similarly to how the decoder (of an auto-encoder) reconstructs a sample from a code, a latent noise z. Rather than defining the Generator's motivation as to minimize the log-probability of the Discriminator being correct like 2.9, we define that the Generator wishes to maximize the log-probability that the Discriminator being mistaken [4] with respect to the generated samples, as expressed by the loss function:

$$L_G = \text{Error}(D(G(z)), 1) \tag{2.11}$$

The Generator is trained using a batch of all generated samples, or rather the Discriminator's feedback of the generated samples. This feedback is compared to a misleading label "1" in the Generator's loss function, to express the Generator's wish that the opponent detects the batch elements as real.

Binary Cross Entropy

Since the problem is formulated to be binary classification problem, a common loss function that is used is binary cross entropy, which is defined as:

$$\text{H}(p, q) = E_{x\sim p(x)}[-\log q(x)] \tag{2.12}$$

For classification problems, the expectation of a discrete random variable can be expressed as a summation.

$$\text{H}(p, q) = -\sum_{x\in\chi} p(x) \log q(x) \tag{2.13}$$

The expression can be simplified further in the case of binary cross entropy, since there are only two labels: zero and one.

$$\text{H}(y, \hat{y}) = -\sum y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \tag{2.14}$$

This binary cross entropy function $H(y, \hat{y})$ is the Error function that we have been loosely using in the above loss functions. Binary cross entropy fulfills the GAN system's objective in that it compares the total entropy between two distributions. That means how close or far the predicted label $\hat{y}$ from the true label y. In the context of Discriminator, the loss function in 2.10 can be rewritten as:

$$\begin{aligned} L_D = &\ \text{H}(D(x), 1) + \text{H}(D(G(z)), 0) && (2.15)\\ = - &\sum_{x\in\chi, z\in\zeta} \log(D(x)) + \log(1 - D(G(z))) && (2.16) \end{aligned}$$

The Discriminator loss function 2.16 presents the error quantity to minimize. In other words, D should maximize the whole summation term with the flipped minus sign, which is exactly how the Discriminator objective is presented in the original mini-max expression 2.9.

We can do the same for 2.11, which presents the error quantity to minimize. :

$$\begin{aligned} L_G = &\ \text{H}(D(G(z)), 1) && (2.17)\\ = - &\sum_{z\in\zeta} \log(D(G(z))) && (2.18) \end{aligned}$$

After separating the loss functions of D and G, we can discuss both of their goals in terms of "minimizing" the corresponding loss output from here.

Model Optimization

Both players wish to minimize their respective loss and must do so while controlling only their own parameters. But because both players' loss functions depend on both players' outputs, the scenario is described as a game, rather than an optimization problem, and therefore can naturally be analyzed with the tools of game theory [4].

The solution to a game is a Nash equilibrium, the point at which the GAN model converges. Nash equilibrium happens when one player will not change its action regardless of what the opponent may do [8]. If both players are given sufficient training capacity, the Nash equilibrium corresponds to point where $G(z)$ comes from the same distribution as the training data, and $D(x) = \frac{1}{2}$ for all input x [4]. In other words, the generated images are so convincing that the Discriminator has a 50% chance of correctly identifying the data source.

Training Procedure

The training procedure is formally presented in the following pseudo code, modeled after [5]. The GAN system is trained on stochastic gradient descend by minibatch. The multiplier of batch size to apply to the generator, k, is a hyper-parameter. We used k = 2 in our experiment.

```
for number of training iterations do
    • Sample batch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $P_z(z)$.
    • Sample batch of $m$ real samples $\{x^{(1)}, \dots, x^{(m)}\}$ from data distribution $P_{data}(x)$.
    • Update the Discriminator by ascending its stochastic gradients:
          $\nabla_{\theta_d} \frac{1}{m} \sum_{l=1}^{m} [\, \log D(x^{(l)}) \; + \; \log(1 - D(G(z^{(l)}))) \,]$   (2.19)
    • Sample batch of $k \times m$ noise samples $\{z^{(1)}, \dots, z^{(k \times m)}\}$ from noise prior $P_z(z)$.
    • Update the Generator by ascending its stochastic gradients:
          $\nabla_{\theta_d} \frac{1}{k \times m} \sum_{l=1}^{k \times m} [\, \log(D(G(z^{(l)}))) \,]$   (2.20)
```

θ_d and θ_g are weights of the Discriminator networks and the Generator networks respectively.

The quantity 2.19 computes the average Discriminator loss of the current batch of m real and m fake samples, in which loss for each sample is defined by 2.16. Similarly, the quantity 2.20 takes the average Generator loss across all $k \times m$ fake samples, each of which is defined by 2.17.

To evaluate a fake image produced by the GAN system in the context of heat transfer application, we can compare the fake image to a real image with similar heat signature patterns found in the training data set. Further, we can compare the statistical vectors as defined in subsection 2.1.1 between the fake image and a comparable real image. For example, the row-wise mean vector of a fake image can be compared against that of a real image regarding the value ranges. This test can be done with the entire real data set against the similar-sized set of fake images, to evaluate how effective the simple GAN system can emulate the real training set.

2.1.3 Conditional GAN (cGAN)

Assuming that the training examples can be categorized into 2 classes, with class label $C(x)$ for every input example x. The second question of the thesis asks for a more sophisticated GAN framework which can generate a fake sample that is likely to originate from a specific class. In other words, this framework must take an additional condition into account everytime G creates a new fake sample. Ideally, the system of neural networks can associate the condition c given to G, with the ground-truth class label $C(x)$ of the training set.

The Conditional GAN (cGAN) system can answer part of this question by incorporating the condition variable c into its neural networks. The cGAN system although straightforward has been frequently cited as an effective framework for conditional generation of data. However for reasons which will be discussed later in this section, the tradional cGAN system has certain lacking when applied to the case of Heat transfer image data.

In this section, the traditional cGAN system is described first to highlight how a condition variable is incorporated into the neural networks. Then the thesis discusses the lacking of this traditional cGAN system, leading to an adjusted version of the cGAN system to be used for Heat transfer image data in this thesis.

The traditional cGAN system is constructed by the variables and functions:

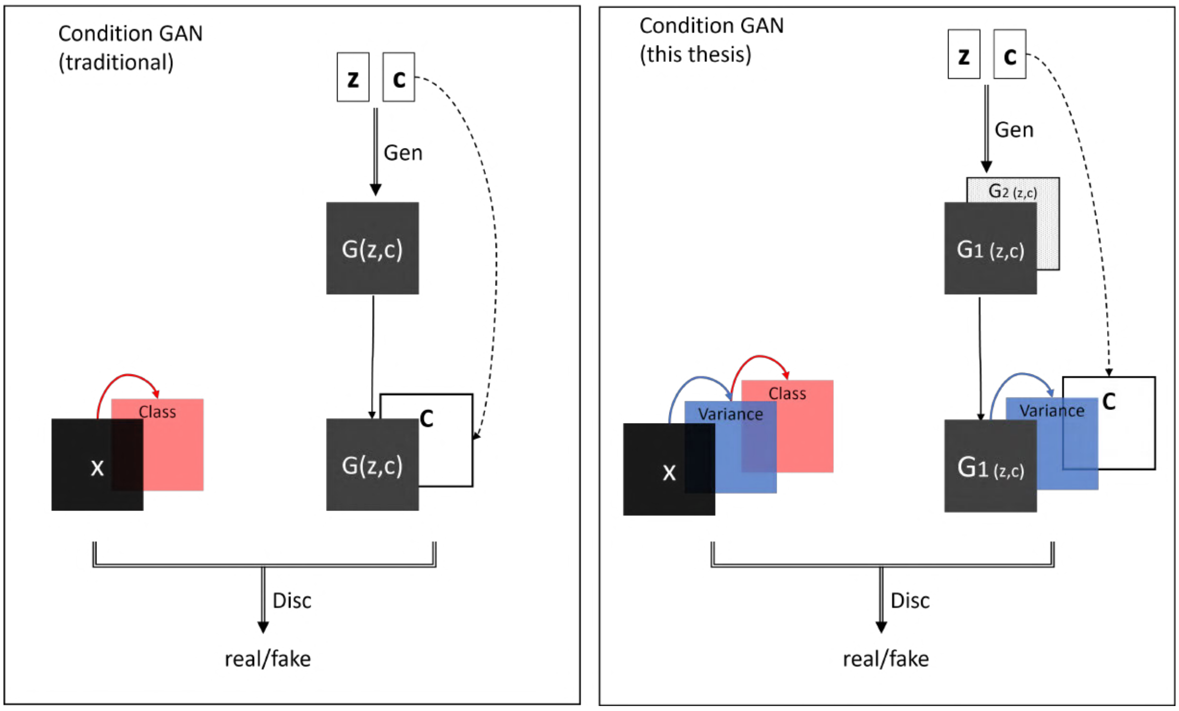
- x: an image drawn from the real training data distribution P_{data}.
- $C(x)$: the class label of an image x, defined by $C : x \mapsto \{0, 1\}$. The classification function C partitions the training set into two classes, one class of images with label 1, the other with class 0.
- z: a noise sample drawn from a noise distribution P_z.
- $c \in \{0, 1\}$: the condition that controls the Generator mode.
- G be the Generator of the traditional cGAN system. G takes two inputs, a noise vector z and a condition C, and returns a fake sample $G(z, c)$ of the same shape as a real image.
- D: the Discriminator of the traditional cGAN system, with two types of subject for evaluation. In the first case, $D\big(x, C(x)\big)$ is D's evaluation of a real image x and its ground-truth class label $C(x)$. In the second case, $D\big(G(z, c), c\big)$ is D's evaluation of a fake image $G(z, c)$ and the condition c given at its creation.
  In both cases, D returns a single scalar between 0 and 1, with 1 signifying that the image is likely to originate from the real training set, and the association between two inputs is likely to be found in the training set. In theory, $D\big(G(z, c), c\big) = 1$ would imply a high probability that $G(z, c)$ originates from the real training set, and that the condition is respected, i.e. $C(G(z, c)) = c$.
- The mini-max game of the traditional cGAN is expressed by:

$$\min_G \max_D V(D, G) = E_{x \sim P_{data}(x)}\big[logD(x, C(x))\big] + E_{z \sim P_z(z)}\big[log(1 - D\big(G(z, c), c)\big)\big] \tag{2.21}$$

2.1.4 Adjusted cGAN (Adjusted cGAN)

In the traditional cGAN system, the classification of the training image set depends only images themselves. This means, in the context of heat transfer images, the classification function $C(x)$ considers only the heat signatures presented in the image, whereas the second objective of the

Figure 2.3: Diagrams of traditional cGAN system and the adjusted cGAN system used in this thesis. In the traditional cGAN system, G generates a fake image from a noise z and condition c. D takes two cases for inputs, either a real image x and the class label derived from the image $C(x)$, or a fake image $G(z, c)$ and the condition c from before. In the adjusted cGAN system, G generates a pair of outputs, the first output is the fake image. D takes three pieces of information: the main image, the variance information, and the class label (if real image) or the condition (if fake image). Since the class label of a real image is derived from its variance information, the variance information is given to the system as part of every real or fake image.



thesis requires to classify the training images based on the output value of another function $f(x)$, making the classification function take a form $C(f(x))$. In particular, $f(x)$ represents the variances of the heat signatures. If $C(f(x))$ is applied directly into the traditional GAN system, D would be less confident in associating the condition c to the classification label $C(f(x))$, simply because the classification-decisive features are not presented on the heat signature images to learn from.

Once the problem is identified, an adjusted cGAN system is constructed, as illustrated in figure 2.3. This system centers around one key point, giving D an additional third piece of information: the output value $f(x)$ of the intermediate variance function, in combination of the existing two inputs of D in the traditional cGAN. This way, D can learn on both the heat signature image, the variance information, to make the association between them and the class label (or the condition if the image is fake).

At the same time, G must also make adjustment by returning not only the heat signature image, but a second output representing the variance information. Otherwise, if G generates the heat signature information only without a second output, the other network D has no incentive to consider its new input.

With the idea presented above, the framework of the adjusted cGAN system is formally described, in the context of heat transfer image data, to answer the second research question.

- x: an image drawn from the real training data distribution P_{data}, being represented by a *n-by-n* matrix carrying the heat signature information in its entries:

$$x = \begin{pmatrix} x_{1,1} & \dots & x_{1,j} & \dots & x_{1,n} \\ & & \vdots & & \\ x_{i,1} & \dots & x_{i,j} & \dots & x_{i,n} \\ & & \vdots & & \\ x_{n,1} & \dots & x_{n,j} & \dots & x_{n,n} \end{pmatrix} \in \mathbb{R}^{n \times n} \tag{2.22}$$

- $V(x)$: a *n-by-n* matrix, carrying the row-wise variance information in its entries:

$$V(x) = \begin{pmatrix} \text{var}_1(x) & \dots & \text{var}_1(x) & \dots & \text{var}_1(x) \\ & & \vdots & & \\ \text{var}_i(x) & \dots & \text{var}_i(x) & \dots & \text{var}_i(x) \\ & & \vdots & & \\ \text{var}_n(x) & \dots & \text{var}_n(x) & \dots & \text{var}_n(x) \end{pmatrix} \in \mathbb{R}^{n \times n} \tag{2.23}$$

  Each entry var$_i(x)$ is defined by function 2.3, $i = 1, \dots, n$. All entries on the same row in $V(x)$ hold the same value, the variance of the corresponding row in x.
- $C_t(V(x))$: the class label of an image x, defined with the output value of $V(x)$ and a threshold $t \in \mathbb{R}$:

$$C_t(V(x)) = \begin{cases} 1, & \text{if} \quad \forall\, \text{var}_i(x) \in V(x) : \ \text{var}_i(x) \leq t \\ 0, & \text{otherwise} \end{cases} \tag{2.24}$$

  A real image x has the class label 1 if all row-wise variances of the heat signature do not exceed the threshold t. This function C_t partitions the training image set into two classes, depending on the variance information instead of depending on the heat signature.
- z: a noise sample drawn from a noise distribution P_z.
- $c \in \{0, 1\}$: the condition that controls the Generator mode.
- $G : (z, c) \longmapsto (G_1(z, c), G_2(z, c))$: the Generator of the adjusted cGAN system. G takes two inputs, a noise vector z and a condition c, and returns a pair of matrices. The first output matrix contains the heat signature information. This is the fake image that will be used for later steps. The second output matrix contains what G thinks to be the variance information. Since the function V can extract the variance information accurately from the fake image, this second output matrix of G is not useful for the result. Nevertheless its presence is extremely important for the system to work properly.
- D: the Discriminator of the adjusted cGAN system, with two types of subject for evaluation.
  In the first case, D takes three inputs, a real image x, the variance-induced matrix $V(x)$, and the ground-truth class label $C_t(V(x))$. Then, $D(x, V(x), C_t(V(x)))$ is D's evaluation of this real-tuple.
  In the second case, D takes three inputs, a fake image $G_1(z, c)$ (which is the first matrix of G output), the variance-induced matrix $V(G_1(z, c))$ and the condition c given at the fake sample's creation. Then, $D(G_1(z, c), V(G_1(z, c)), c)$ is D's evaluation of this fake-tuple.
  In both cases, D returns a single scalar between 0 and 1, with 1 signifying that the image is likely to originate from the real training set, and the association between the three inputs is likely to be found in the training set.

In theory, $D(G_1(z,c), V(G_1(z,c)), c) = 1$ would imply a high probability that the fake image of heat signature originates from the real training set, and that the condition is respected, i.e. $C_t(V(G_1(z,c))) = c$.

- The mini-max game of the adjusted cGAN is expressed by:

$$\min_G \max_D V(D,G) = E_{x\sim P_{data}(x)}[\log D(x, V(x), C_t(V(x)))] \tag{2.25}$$

$$+ E_{z\sim P_z(z)}[\log(1 - D(G_1(z,c), V(G_1(z,c)), c))] \tag{2.26}$$

The training procedure is formally presented in the following pseudo code. The adjusted cGAN system is trained on stochastic gradient descend by minibatch. The multiplier of batch size to apply to the generator, k, is a hyperparameter. We used k = 1 in our experiment.

```
for number of training iterations do
    • Sample batch of $m$ real samples $\{x^{(1)}, \dots, x^{(m)}\}$ from data distribution $P_{data}(x)$.
    • Compute the corresponding variance-induced matrices $\{V(x^{(1)}), \dots, V(x^{(m)})\}$.
    • Compute the corresponding class labels $\{C_t(V(x^{(1)})), \dots, C_t(V(x^{(m)}))\}$.
    • Sample batch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $P_z(z)$.
    • Sample batch of $m$ conditional labels $\{c^{(1)}, \dots, c^{(m)}\}$, borrowing the labels from the real
      sample batch, i.e. $c^{(l)} = C_t(V(x^{(l)}))$
    • Generate batch of $m$ fake images, using noise samples and class labels:
      $\{\tilde{x}^{(1)}, \dots, \tilde{x}^{(m)}\}$, where $\tilde{x}^{(l)} = G_1(z^{(l)}, c^{(l)})$.
    • Update the Discriminator by ascending its stochastic gradients:
```

$$\nabla_{\theta_d} \frac{1}{m} \sum_{l=1}^{m} [\ \log D(x^{(l)}, V(x^{(l)}), C_t(V(x^{(l)}))) \tag{2.27}$$

$$+\ \log(1 - D(\tilde{x}^{(l)}, V(\tilde{x}^{(l)}), c^{(l)}))\ ] \tag{2.28}$$

```
    • Update the Generator by ascending its stochastic gradients:
```

$$\nabla_{\theta_d} \frac{1}{k \times m} \sum_{l=1}^{k\times m} [\ \log D(\tilde{x}^{(l)}, V(\tilde{x}^{(l)}), c^{(l)})\ ] \tag{2.29}$$

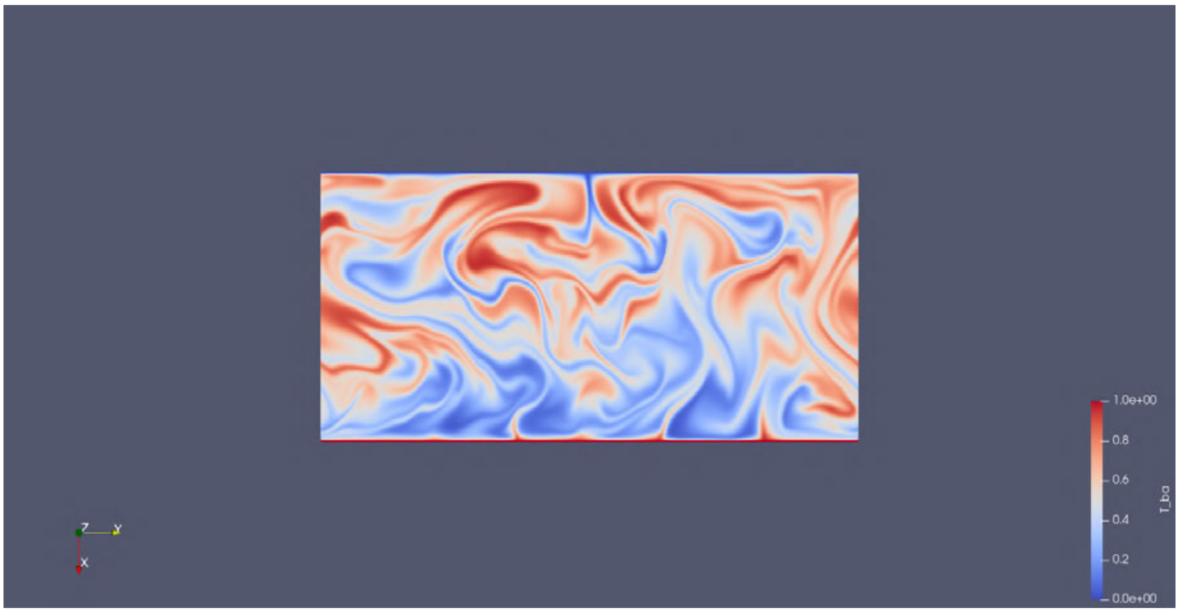
θ_d and θ_g are weights of the Discriminator networks and the Generator networks respectively.

To evaluate how effectively the condition is used by the Generator in this adjusted cGAN system, we can apply the classification function C_t on the fake images, and compare the outcome (ground-truth labels) to the condition (predicted labels) used at their generation. A high accuracy rate between the truth and prediction indicates that G is effectively using the condition to generate images, and the adjusted cGAN works as intended.

2.2 Real Data

To test the hypotheses of the unconditional and conditional generation of convincing fake Heat Transfer images, databases of real simulations will be necessary. The word "real" in this context is used not because the data is collected from the a real-world event, but to indicate that data of a

Figure 2.4: Database A - An image illustration of data sample at time $t = 65$, before pre-processing



numerical simulation is obtained through computation and is indeed physics- and mathematically-correct. These databases are used to train the GAN and condition GAN systems that try to imitate instances of them.

There are two databases discussed in the thesis, both depict Heat transfer simulations, which are described in the next part, followed by the pre-processing steps to construct new data sets for GAN training. To prevent confusion, the term “original sample” refers to an instance of the unprocessed database. The term “training sample” refers to an instance of the post-processed data set, ready to be applied for GAN training.

Database A is an ordered collection of matrices. Each matrix, with 256 columns and 512 rows of entry values between 0 and 1, represents the heat signature of a 2D rectangular disc at a time t. The collection of these raw-data matrices describes a continuous heat transfer simulation on a 2D rectangular dics through 945 time stamps.

The database is pre-processed as follows:

- Step 1: Convert to grayscale images.
  Output samples: 945.
- Step 2: Crop into square images of size 256-by-256, at n different positions.
  Output samples: $945 \times n$.

Step 1 administers the conversion to grayscale images. This step is a result of the application of statistical analysis as an evaluation method of the GAN systems. Temperature data needs to be extracted accurately from an image in order to give reliable evaluation. Setting the image to gray-scale with distinction between two boundary conditions (hot versus cold) means the pixel value can be used directly as heat signature. On the contrary, a multiple-channel colored image makes for a more complicated and less confident conversion between a set of color values and a scalar heat value.

Figure 2.5: Samples from Training Dataset A-1

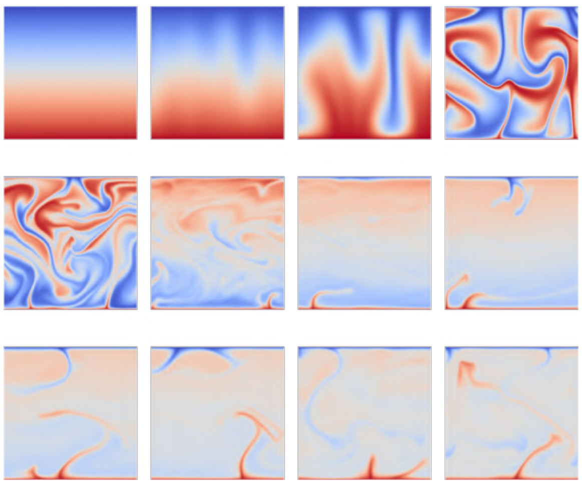
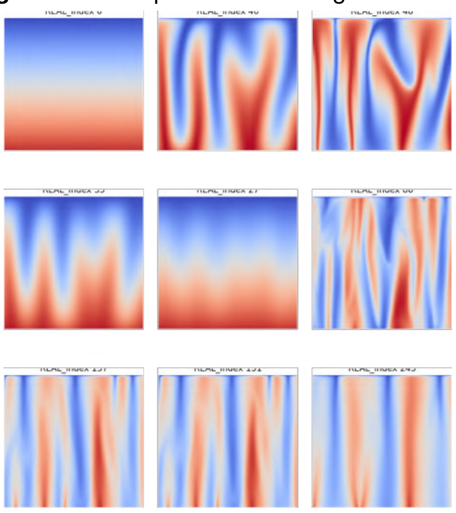


Figure 2.6: Samples from Training Dataset B-1



Step 2 is performed not out of necessity, but to add benefits to the training experiments. Setting a training data set with squared training samples will standardize the experiment for future applications. On the other hand, from an original sample of 1:2 width-to-length ratio, extracting two non-overlapping squares from it, is equivalent to applying the pre-processing step 2 with $n = 2$. These cropped squares can be compiled into a new set, a training data set suitable for training a GAN system. This choice $(n = 2)$ benefits the experiment by preserving all information of the database, compared to an alternative $n = 1$. Alternatively, choosing $n = 3$ where three squares are cropped from an original sample, brings the same benefits, eventhough some features of the extraction will be repeated from the overlapping areas. One can apply even larger number, or a mixed choice, of cropping positions, depending on how extensively they want to populate the training data set.

This feature is unique to the application of Heat transfer images, since the feature of interest (the heat signature dynamic) lie on each vertical strip of the original sample. Additionally, populating a fixed number of training samples from every original sample will not distort the pattern distribution of the original database, unless manipulation is necessary, in which case a mixture of n can be applied. The choice of n and whether to apply this step to some or all available samples depends on each application of the GAN system type.

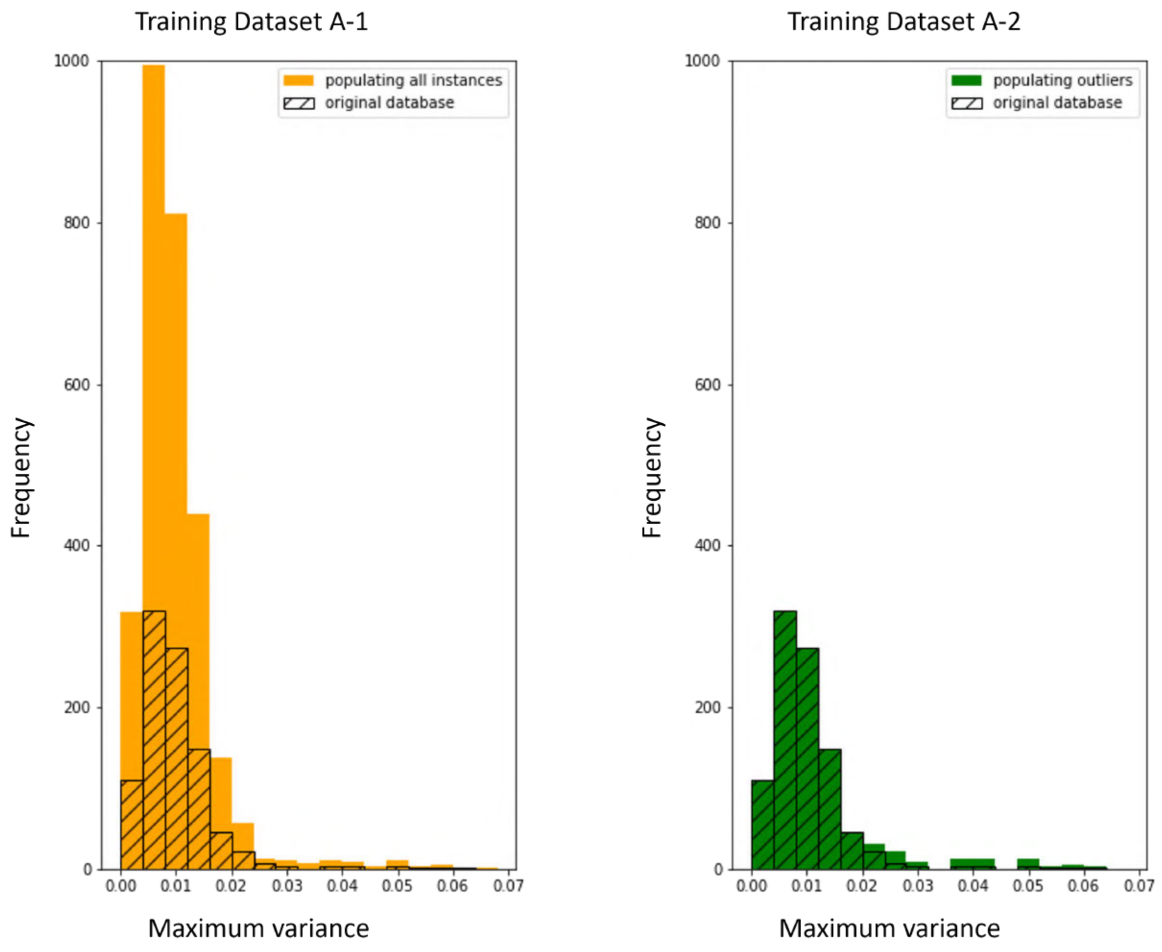
2.2.1 Training Dataset for Simple GAN

To prepare the training dataset for simple GAN experiment, the pre-processing steps, including step 2 with parameter $n = 3$, are applied to all instances in the original database A. The croppings are taken at the far left square, far right square, and centered square of each original sample. In this case, a dataset A-1 is obtained, containting 2835 training samples, suitable for a simple GAN application.

The pattern distribution of heat transfer stages should be almost identical between the original database and the emerging training data set A-1, only differing in sample set sizes, evidently in the left histogram of figure 2.7. The maximum variance value is chosen as the basis of comparison, the left histogram describes the distribution of maximum variances obtained from the original database (orange), and that of the training set A-1 (hatch-textured). Every orange filled bar is merely a tripple-stacked version of the corresponding hatched bar.

Database B is similar to Database A in most format aspects, except with only 336 data matrices. Applying the same data pre-processing steps with $n = 3$ yields a dataset B with 1009 images suitable for a simple GAN application.

Figure 2.7: Sample Distribution of Training Dataset for Simple GAN system, and for Adjusted cGAN system. The Database A samples are broadcasted three times to populate the dataset A-1 for training the simple GAN system. The Database A outliers are broadcasted three times, the rest is broadcasted once only, to populate the training dataset A-2 for the adjusted cGAN system, to elevate the unbalance.



2.2.2 Training Dataset for Adjusted cGAN

As part of premise, the training data set for the adjusted Conditional GAN systems are partitioned into two classes. Hence, it is of better benefits to use a set with relatively balance between the classes.

Under the same comparison method (via maximum variance) the Database A shows a small subset of outliers, which represent the heat transfer stages with intense turbulent structures; and a large subset of the dissipated stages. These outliers carry valuable information for the experiment, since the thesis is motivated by the generation of outliers. Therefore, the original outlier samples are cropped at $n = 3$ positions (left, right, center), while the other samples are cropped at $n = 1$ position only, discarding the remaining areas. In this context, the outliers are defined to be sample values greater than 0.023, regarding the scale of maximum variance. In this context, outliers are data points that are above $Q3 + 1.5 \times IQR$, where $Q1$ and $Q3$ are the 25th and 75th percentile of the dataset respectively, and IQR represents the inter-quartile range and given by $Q3 - Q1$.

While this method of selectively populating the training data set admits a drawback regarding the missed opportunity of using all available information from the database, it is out-weighted by the benefit. That said, the discarded information is already sufficiently represented in the training set.

Following the above pre-processing conditions, a dataset A-2 of 1009 images is obtained, suitable for the adjusted cGAN application. The set A-2 has slightly different pattern distribution compared to that of the original database A, as shown in the right histogram in figure 2.7. The right histogram describes the distribution of maximum variances obtained from the original database (hatch-textured), and that of the training set A-2 (green). The two distributions are almost exactly shaped, except for their right tails. The green right-tail is thicker than the hatch-textured tail, signifying that more samples in this range are presented in the training data set A-2.

2.2.3 Classification of Training Data

The adjusted cGAN system runs in companion with a classification function $C_t(V(x))$, constructed with a threshold t, which partitions the training data set into two classes. The threshold t is chosen later in this part.

The classification function C_t outputs a scalar class label for an arbitrary image, signifying that the function C_t defines the ground-truth label. On the other hand, the Generator is given a condition label c as one of its inputs to generate a fake sample. The conditional label c tied with this fake sample acts as a label predicted by the adjusted cGAN model. The performance of the adjusted cGAN model is evaluated by the accuracy rate between the predicted labels c and the ground-truth labels C_t of a fake sample set.

We explore two cases. The first case uses the threshold $t_1 = 0.01$ to divide the training set A-2 into to two classes, using the classification function $C_{t_1=0.01}$. This threshold t_1 sits near the 70-th percentile of the data set. A sample with all row variances less or equal to $t_1 = 0.01$ is labeled “1”, otherwise the sample is labeled “0”. This classification label is incorporated in an adjusted cGAN system to train and evaluate.

The second case uses the the threshold $t_2 = 0.006$, which sits near the 50-th percentile of the data set. Similarly, this classification label is incorporated in an adjusted cGAN system to train and evaluate.

2.3 Fake Data Generation

In this chapter, the implementation of a GAN system and a cGAN system in a specific environment will be explained, as the method of obtaining the Fake data set. Different sections will be explained referencing their row numbers in the corresponding script in the Annex. Two scripts will implemented to carry out the experiments, a script for the unconditional GAN system, and a second script for the adjusted conditional GAN system. It is important to note that the structure of the scripts has been developed based on various works [3, 7, 9].

The environment chosen to carry out this project has been:

Execution environment: Google Colab
Code language: Python 3.6
Neural network library: Tensorflow

Google Colab offers cloud processing capacity with dedicated graphics cards for training and interference tasks. Python has been chosen because of the previous knowledge of the language and Tensorflow due to its high level configuration.

2.3.1 Simple GAN system

The simple GAN system is implemeneted using the pseudo code in Appendix 5.1.

Dataset

After the required libraries have been imported into the environment, the Dataset class from Tensorflow is implemented to generate a Dataset of third-order tensor objetcs from image files in the specified directory (line 16-18). Besides that, the Map function normalizes all element tensors to range between -1 and 1 (line 19).

For the simple GAN system, calling a Dataset yields batches of 32 tensor objects, each of which has shape [256,256,1] - a notation method for tensor shapes in Tensorflow module, representing a square and grayscale image of 256 rows and 256 columns worth of pixels. Literature have suggested good performance of GAN systems when using batch-size of 32, equal row-to-column ratio, and image resolution of less or equal 256. The use of grayscale, or 1-channel, image is specific to this thesis, to enable accurate extraction of temperature information for statistic analysis, as explained in section 2.2.

Discriminator and Generator Models

The Discriminator and the Generator model are constructed between line 23 - 137. The network architect is loosely based on [3]. The simple GAN system is compatible with a 1-channel tensor, specifically a [256,256,1] tensor. The Discriminator takes a 1-channel tensor as the input, and a single scalar output between 0 and 1, with 1 signifying that the tensor is likely to come from the real dataset. The Generator starts with a noise vector and outputs a 1-channel tensor. It must be ensured that two objects admit the same shape: the Discriminator's input, and the Generator's ouput.

The Discriminator restricts the output to a scalar between 0 and 1 via the Sigmoid activation function in its last layer. The Generator uses hyperbolic tangent activation function to produce a tensor with pixel values between -1 and 1 (as the original tensors in the dataset).

Training Steps and Optimizer Models

Included in this code section (line 151 - 207) is the training loop customized for a GAN system. This is the translation of the training procedure presented in section 2.19. The two networks D and G are trained using the loss function Binary Cross-Entropy to calculate errors between the truth versus prediction real/fake labels. It is important to note that when updating the weights of the Generator through gradient descend, all weights of the Discriminator should be manually frozen. Otherwise, the Discriminator will be "persuaded" by the Generator, which may lead to mode collapse, a failure of GAN model where its Generator keeps producing one or very few examples. The behaviour of updating all trainable weights once unless otherwise stated is native to Tensorflow Keras in custom training mode.

Between line 210 - 212 is where the GAN model instance is initialized and compiled with choice of Optimizer for respective sub-models. Both neural networks are optimized using Adam optimization at respective learning rate of $2e^{-4}$ for the Discriminator and $5e^{-4}$ for the Generator.

In between line 221 - 224, the GAN model instance is fitted with the dataset. For the first training session, a new GAN model instance can be initialized with random weight values and trained for a number of training iterations (epochs). To enable the GAN model instance to be trained in multiple sessions, a callback object of Tensorflow Keras is used to periodically save the weights of the model (line 217-218). When training is resumed in a future session, a new GAN model instance can be initialized and loaded with the latest weight values (line 215) and continued to be trained for some additional epochs.

Generating Final Results

Two simple GAN models are trained. They differ in the training data and the length of training. Each of these models give a corresponding set of fake images.

The first simple GAN model is trained on the training set A-1. After 1,500 epochs, the trained weights are saved, and used to generate 100 fake images (line 229). This set of fake images, named “F1”, will be compared against the training set A-1 for evaluation, regarding visual profile and statistical properties (line 232-235).

Similarly, the second simple GAN model is trained on the training set B-1. After 1,500 epochs, the trained model is used to generate a set of 100 fake images, named “F2”, which will be compared against the training set B-1 for evaluation.

2.3.2 Adjusted cGAN system

The adjusted cGAN system is implemeneted using the pseudo code in Appendix 5.2.

Dataset

For the adjusted cGAN system, calling a Dataset yields batches of 10 tensor objects of shape [64,64,1]. As training the adjusted cGAN system is expected to take longer than the simple GAN before a presentable result is produced, the thesis reduces the tensor size to put less stress on computing performance. This would also allows for flexible testing of the adjusted cGAN system with various conditional statements.

Discriminator and Generator Models

The adjusted cGAN Discriminator and Generator model are constructed between line 23 - 116. The network architecture is loosely based on [7, 9, 3].

There are two key differences to the adjusted cGAN system. Firstly, the Generator starts with two inputs: a noise vector, and a scalar conditional class label, then outputs a 2-channel tensor, which represents the fake Heat signature on the first channel, and the fake variance-like channel. Secondly, the Discriminator takes a 3-channel tensor as input. The input carries three informative channels in order: the image-induced channel, the variance-induced channel, and the conditional-induced channel, the last of which is simply a channel that has been filled with a scalar class label. The Discriminator outputs a single scalar between 0 and 1, with 1 signifying that the tensor is likely to be induced by a real image and the tensor corresponds to its conditional class label.

Similar to the simple system, the Discriminator uses the Sigmoid activation function in its last layer, and The Generator uses hyperbolic tangent activation function in its last layer.

Training Steps and Optimizer Models

The customized training loop for an adjusted cGAN system is translated from the training procedure in section 2.27, implemented in line 157 - 217. Similar remark is applied when training this cGAN system, the Discriminator should be frozen in all its weights when the Generating is being updated.

Similar configurations from the simple GAN system case are applied for the choice of Optimizers, learning rates, and method for periodically saving model's weights.

Generating Final Results

Three versions of adjusted cGAN models are trained. All of them are trained on the data set A-2. Each of these models generates a corresponding set of 1,000 fake images, half of which is initiated with label “1”, the other half label “0”. Three versions differ in the conditional threshold t, and the initializing weights.

The first adjusted cGAN model is trained using the threshold $t_1 = 0.01$, and initialized by random weights, i.e. the model has no pre-trained information. After 2,000 epochs, the model generates a fake image set F3. The accuracy rate between the initiated labels and the ground truth classification labels is the effectiveness indicator for this model.

The second and third adjusted cGAN models are both trained using another threshold $t_2 = 0.006$. However, the second model is trained for 2,000 epochs on top of the first model. In other words, the second model has pre-trained information from the previous threshold t_1, while the third model is trained from scratch, also for 2,000 epochs. The comparison of the corresponding fake image set, F4 and F5, can indicate if using a pre-trained model gives the undesired effect of “sticky memory”, i.e. the model stills remembers the previous condition, and thus affects present accuracy. Furthermore, a comparison between F3 and F5, two models trained on different thresholds, may indicate which threshold works better than the other, regarding how much the conditions are observed by each model.

3 Results

3.1 Results 1 - Simple GAN

This section looks at two generated image sets, both generated by the same simple GAN framework, but trained on different training data. Set F1 is generated by a GAN system trained on set A-1. Set F2 is generated by a GAN system trained on set B-1.

3.1.1 Fake image set F1

Visual Inspection

After training, the simple GAN model is capable of generating a variety of image examples. Figure 3.1 displays 12 best-looking and most representative result, out of 100 generated images in set F1. Each sub-image represents a group of images with similar patterns. The patterns displayed in the first columns shows up in the generated set F1 with low frequencies and with less smooth details. Each patterns from the second, third, and forth columns in the figure share some similar characteristics but are distinct patterns, regarding the number of hot and cold swirls, their locations, sizes and intensity.

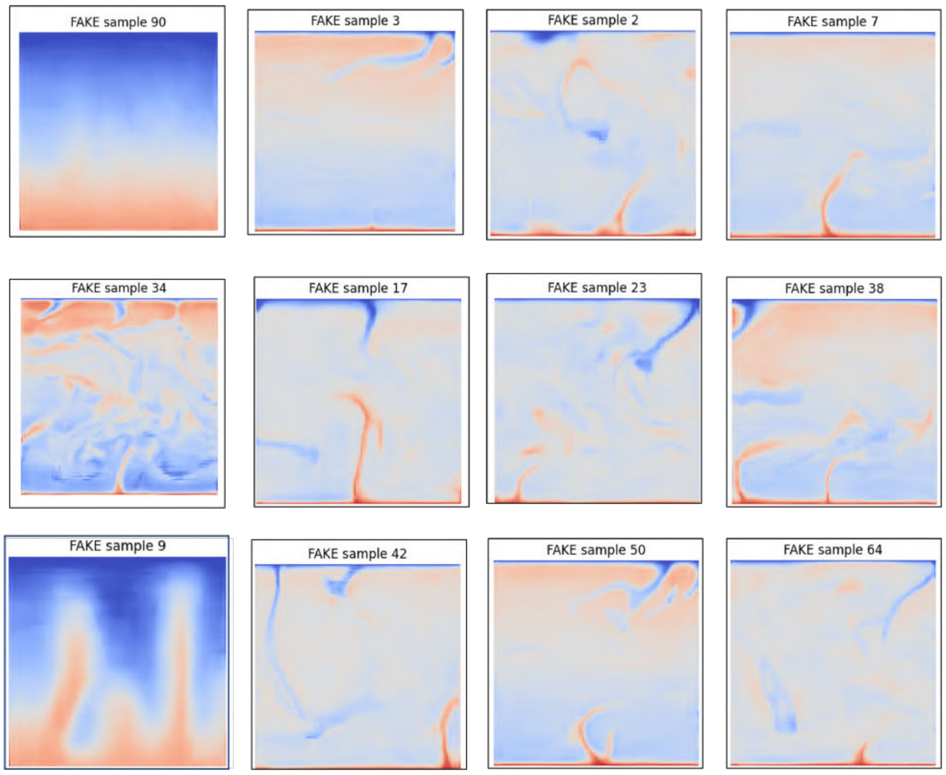
Statistical Inspection

Figure 3.2 and 3.3 show a good example of a real and a fake image, chosen based on their similar heat structures (top left illustrations). Four statistical vectors are plotted from the temperature data of each image. This pattern is a commonly found pattern in both the training set and the generated set. The real and fake example show more similarities in their mean and variance vectors, fewer in their skewness and kurtosis vectors.

Similarly, figure 3.4 and 3.5 show a good example of real and fake image. The pattern observed here is one of the most uncommon patterns found in the training set and generated set. Despite this, the fake image has similar vector of row mean, variance and skewness. Figure 3.6 contains 4 sub-plots. The top-left plot is obtained by aligning mean vectors of 100 real images in the training set A-1. These images are chosen such that they can represent the entire set. Similarly, the lower-left plot is obtained by aligning mean vectors of 100 generated images in set F1. It can be seen that the generated set F1 mimics the training set, except for some minor blank space not filled with data points.

The right half of figure 3.6 shows the alignment of row variance vectors to represents each data set, training set on top-right, generate set on lower-right. The generated set F1 mimics the majority of training examples, and contains few to none examples with variance exceeding 0.04. It must be noted that training examples exceeding 0.04 only accounts for 1% of the entire A-1 set, therefore the generated images represent already the majority of the training set.

Figure 3.1: 12 most representative fake samples from generated set F1. All fake samples demonstrate realistic heat transfer activity.



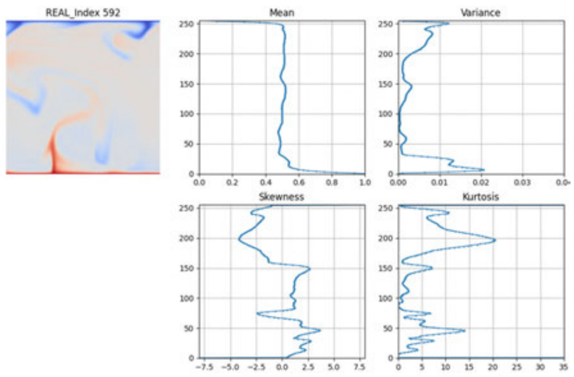


Figure 3.2: Visual and vertical statistics profile of a real heat transfer stage at time stamp 592

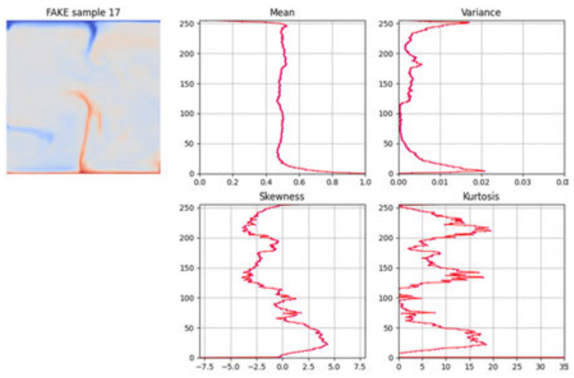


Figure 3.3: A fake image of heat transfer shows similar visual and vertical statistics profile to a real image, especially regarding vertical mean temperature, and variance temperature.

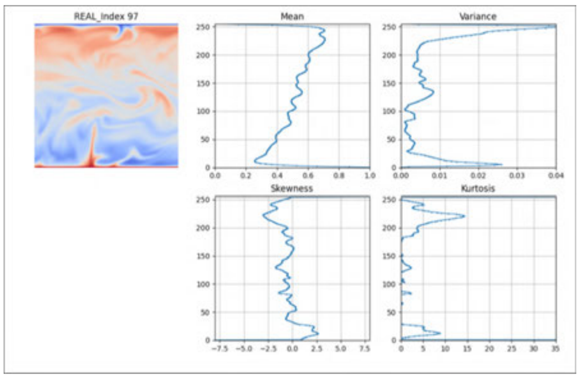


Figure 3.4: Visual and vertical statistics profile of a real heat transfer stage at time stamp 97, which occurs in the data set at low frequency.

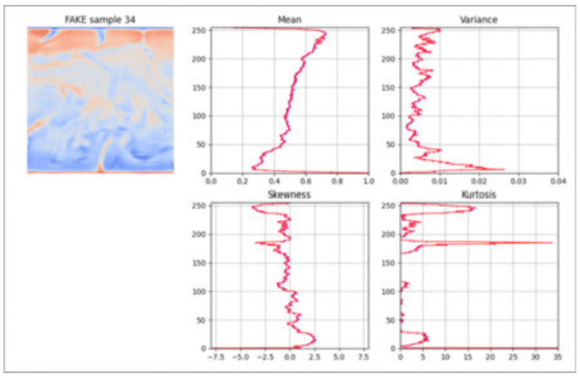
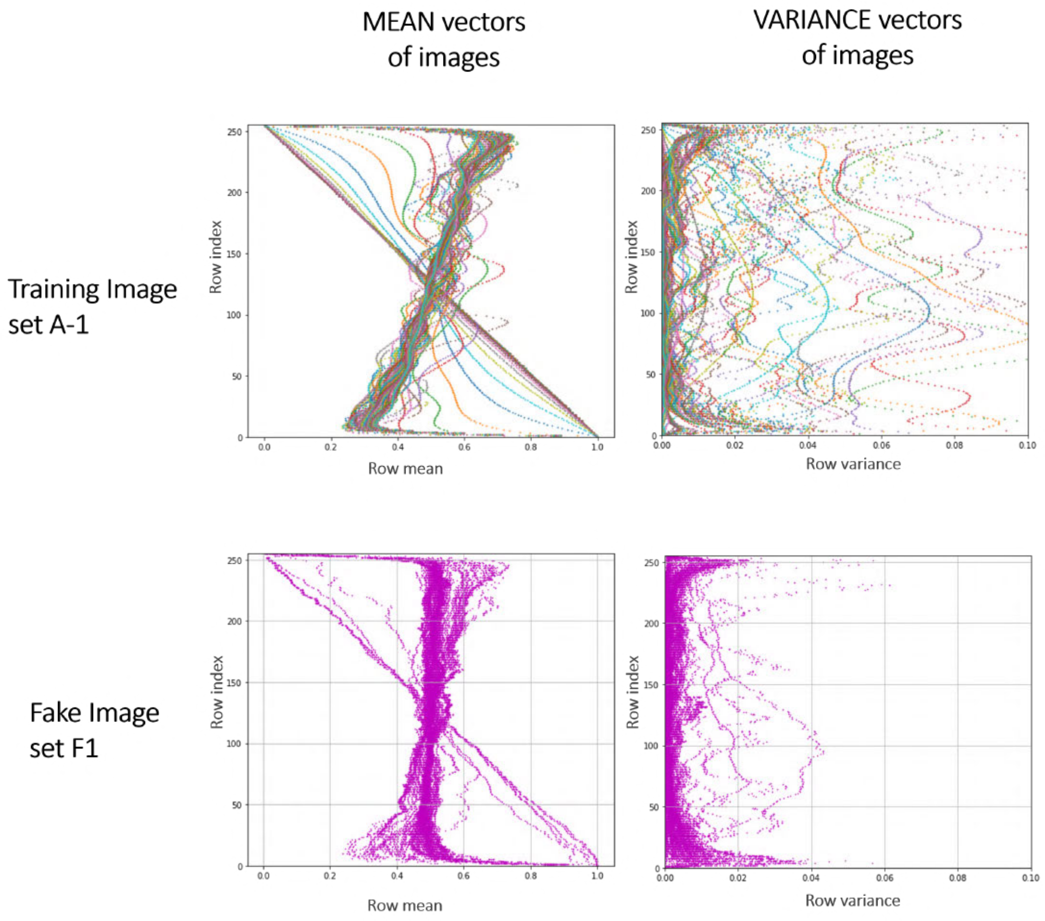


Figure 3.5: A fake image of heat transfer shows similar visual and vertical statistics profile to a real image, despite limited learning materials for this pattern in the training set.

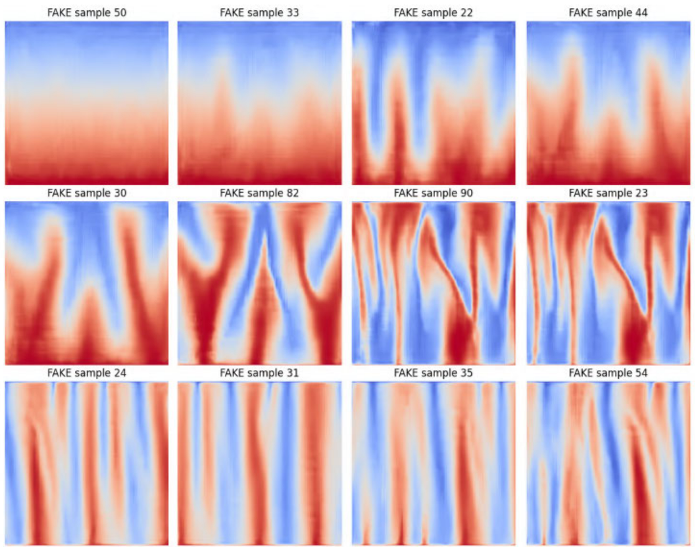
Figure 3.6: Distribution of vertical statistics profiles, between the training data set and the generated set. The simple GAN can capture upto the 99-th percentiles of the real training data distribution, shown by the agreement in coverage of multicolor- and magenta-data spaces. Some outlier patterns from the training data are not generated by the GAN system, shown by the occupied area by multicolor data space versus the empty area in the magenta data space.



3.1.2 Fake image set F2

Visual Inspection

Figure 3.7: 12 representative fake images generated by the simple GAN system.



After training on the set B-1, the simple GAN model is capable of generating a variety of image examples. Figure 3.7 displays 12 results, out of 100 generated images in set F2. All generated images show visually acceptable details on both the global and local level. The patterns displayed on the bottom row accounts for 50% in the generated set, the other half was shared between the different patterns on the top two rows.

Statistical Inspection

Figure 3.8: Visual and vertical statistics profile of a real heat transfer stage at time stamp 41, which occurs in the data set at low frequency.

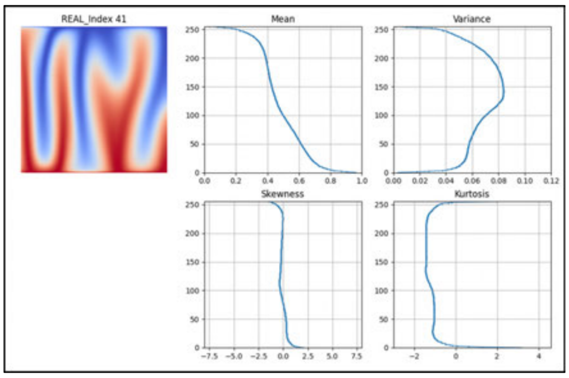
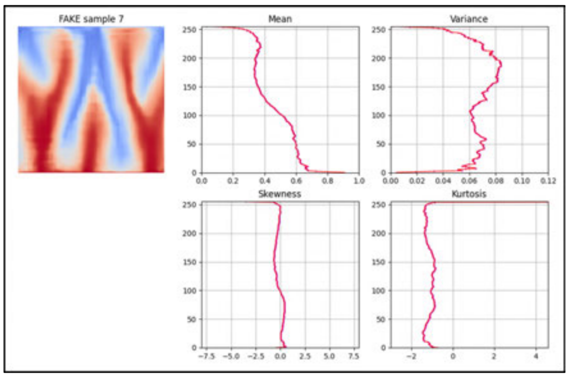
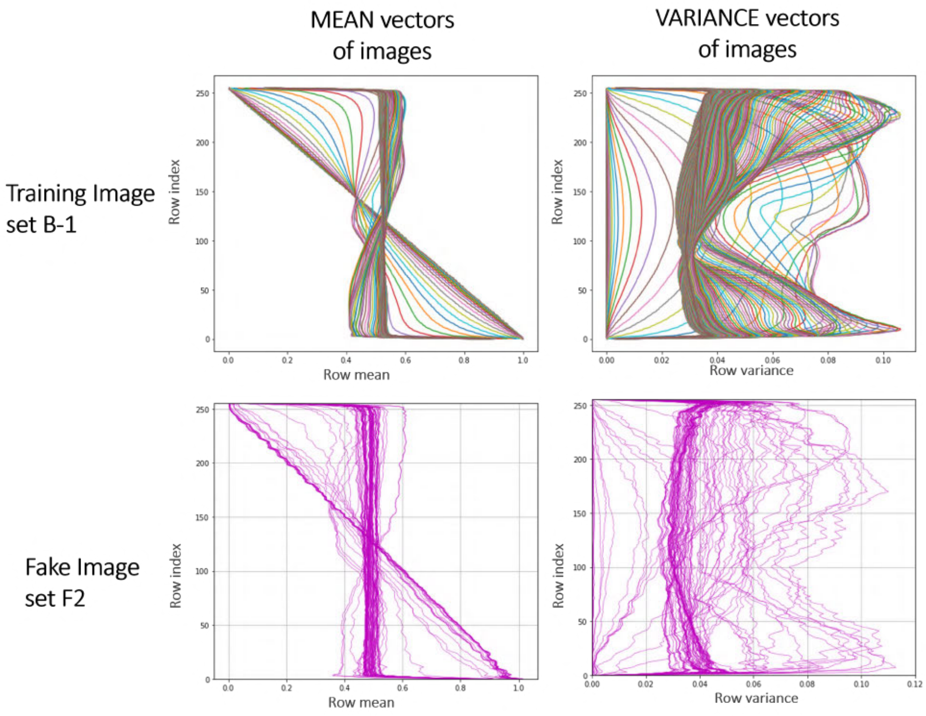


Figure 3.9: A fake image of heat transfer shows similar visual and vertical statistics profile to a real image, despite limited learning materials for this pattern in the training set.



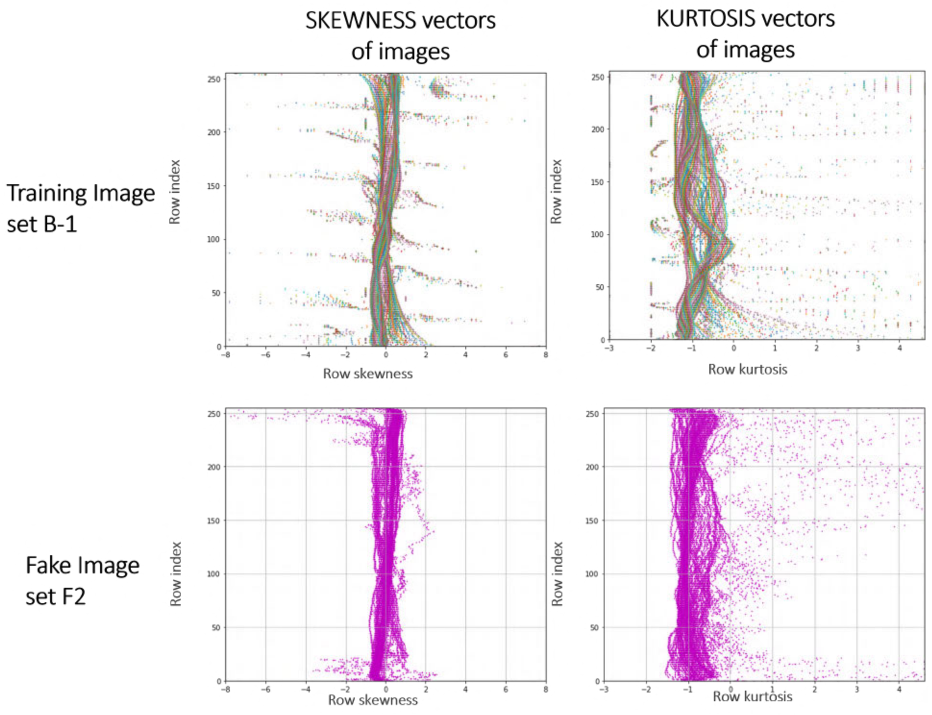
The real-versus-fake comparison with a pair images 3.8 3.9 shows that the fake image display extremely similar physical and statistical properties to the real image, eventhough the fake image is not as smooth.

Figure 3.10: Distribution of vertical statistics profiles for mean and variance temperature, between the training data set and the generated set. The simple GAN can capture upto the 99-th percentiles of the real training data distribution, shown by the agreement in coverage of multicolor- and magenta-data spaces. Very few outlier patterns from the training data are not generated by the GAN system.



The figure 3.10 is obtained by aligning many vectors on top of each other to demonstrate the value range captured by a group of images. The multi-colored vectors are obtained from 100 training examples in the set B-1. The magenta vectors are obtained from the generated images in set F2. On both accounts of row mean and variance vectors, this generated set mimics the training set quite fully. Nevertheless, the simple GAN model fails to generate the outlier samples that are available in the real image set, illustrated better in the comparison of skewness vectors or kurtosis vectors in figure 3.11. There are few to none fake images with row skewness in the range outside the interval $[-2, 2]$. This observation corresponds with the lack of fake images with kurtosis vectors that reach below -1.5.

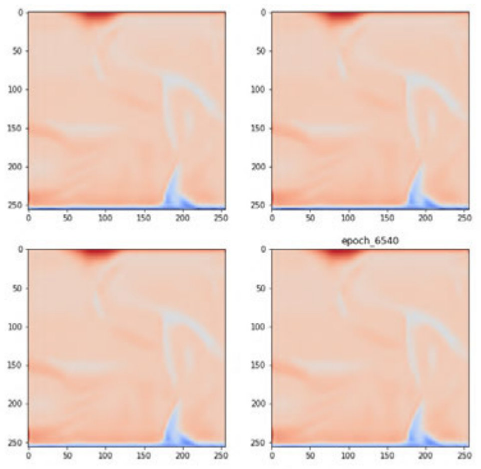
Figure 3.11: Distribution of vertical statistics profiles for mean and variance temperature, between the training data set and the generated set. The simple GAN can capture upto the 99-th percentiles of the real training data distribution, shown by the agreement in coverage of multicolor- and magenta-data spaces. Very few outlier patterns from the training data are not generated by the GAN system.

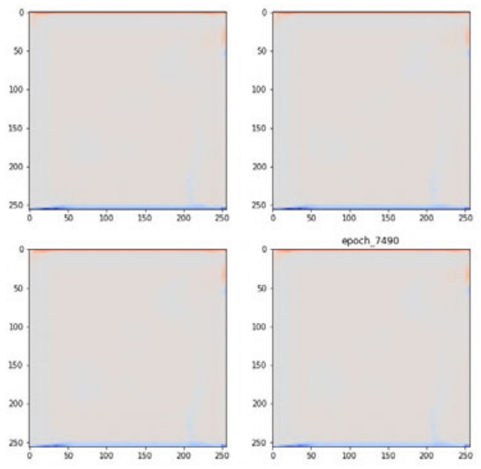


3.1.3 Mode collapse

Figure 3.12: Failed model at epoch 6540

Figure 3.13: Failed model at epoch 7490





This part shows results from a failed version of the simple GAN model which was trained on the real image set A-1, for an extensive amount of iterations (more than 10,000 epochs). The model experienced a mode collapse, a situation where G could only produce one mode of image pattern, no matter what noise z was given.

The figure 3.12 contains 4 images which were generated by different seeded noise z, yet having the same appearances. A different figure 3.13 shows another 4 examples, from a different epoch stamp. This failed model was caused by a critical error in the coding, where D was allowed to be updated while G was training, breaking the adversary between them. This causes D to be “persuaded” by the generated images and assigning them a high probability score. After a while, G focuses on generating only one mode that scores the highest point. At many points during the training, the model broke entirely where G generated blank images with all 0 values.

3.2 Results 2 - Adjusted cGAN

Figure 3.14: Generated images given conditional label “1” - all images should have vertical variances less or equal 0.01.

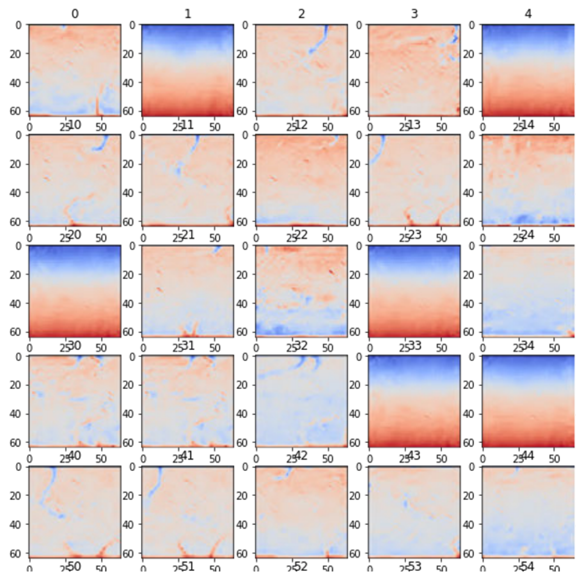
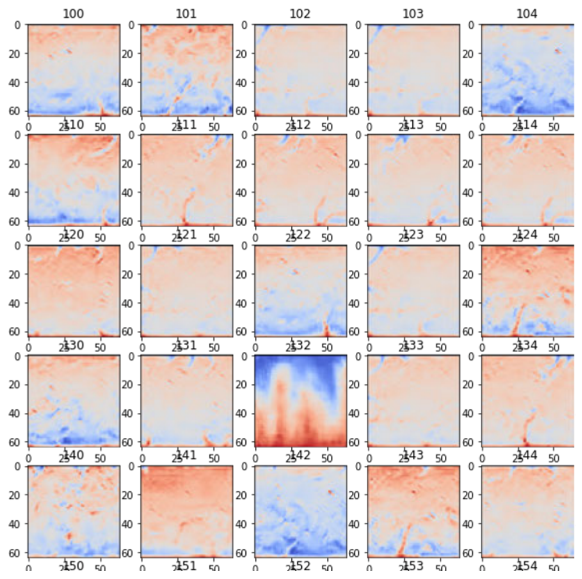


Figure 3.15: Generated images given conditional label “0” - all images should have vertical variances greater than 0.01



After training for 2000 epochs, the adjusted cGAN is capable of generating a variety of patterns. Furthermore, the model takes into account a conditional label and generates images accordingly. The figure 3.14 compiles 25 random images generated under the conditional label “1”. The figure 3.15 compiles 25 random images generated under the conditional label “0”.

Table 3.1: Accuracy rate of Fake image set F3 under threshold $t_1 = 0.01$. Predict label is given to G for image generation. Truth label is computed for the generated images with the classification function C_{t_1}. High accuracy rate implies G can effectively control the outputs' variances.

		Predict label	
		1	**0**
	1	476	17
Truth label	**0**	24	483

Table 3.2: Accuracy rate of Fake image set F4 under threshold $t_2 = 0.006$ is 64.6% for conditional label "1", 97% for "0". G was initialised with an pre-trained weights from another conditional threshold.

		Predict label	
		1	**0**
	1	323	15
Truth label	**0**	177	485

Table 3.3: Accuracy rate of Fake image set F5 under threshold $t_2 = 0.006$ is 66.2% for conditional label "1", 97.6% for "0". G was initialised with random weights and had no pre-learned knowledge.

		Predict label	
		1	**0**
	1	331	12
Truth label	**0**	169	488

3.2.1 Conditional threshold $t_1 = 0.01$

Under the classification of image based on the maximum variance threshold $t_1 = 0.01$, the adjusted cGAN model is capable of generating fake images that comply with a given conditional label, with an average accuracy of 95.9 %, as illustrated by table 3.1. Specifically, when G is instructed to generate 500 images of maximum variance no greater than t_1, G can comply with 95.2% confidence; with the opposite instruction, G can comply with 96.6% confidence.

3.2.2 Conditional threshold $t_2 = 0.006$

The table 3.2 and 3.3 are obtained from two versions of adjusted cGAN models. The two versions both use the conditional threshold t_2, but differ in how the model weights are initialized. The former uses an existing model that was previously trained on a different threshold, the latter is trained from scratch by using random weights at creation. The generated set F4 and F5 show some similarities. The accuracy rate for each label is similar between the two model versions. Both models observe the pattern of having lower accuracy rate for label "1", and higher rate for label "0".

3.2.3 A failed model

This part shows results from a failed version of the adjusted cGAN model. The model did not strictly follow the loss function for G presented in 2.29, but with an additional penalty term to encourage the G to generate images that satisfy the condition, by minimizing the difference in the predict label and ground-truth label. More specifically, G is trained using the gradients as follows:

$$\nabla_{\theta_d} \frac{1}{k \times m} \sum_{l=1}^{k \times m} [\text{ log } D(\tilde{x}^{(l)}, c) \ + \text{log } ||C_t(V(\tilde{x}^{(l)})) - c||\ ] \tag{3.1}$$

Figure 3.16: Label accuracy of fake image set F3, generated by an adjusted cGAN model using threshold $t_1 = 0.01$.

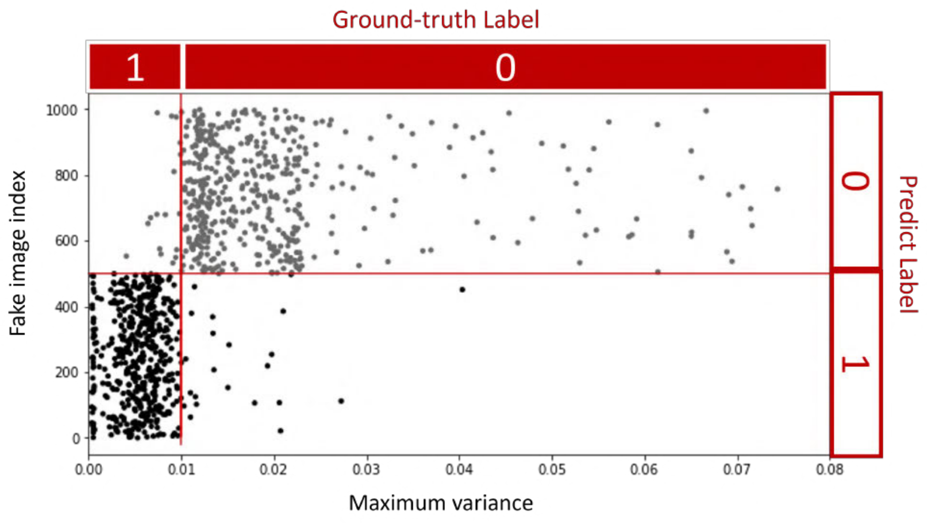


Figure 3.17: Label accuracy of fake image set F5, generated by an adjusted cGAN model using threshold $t_2 = 0.006$.

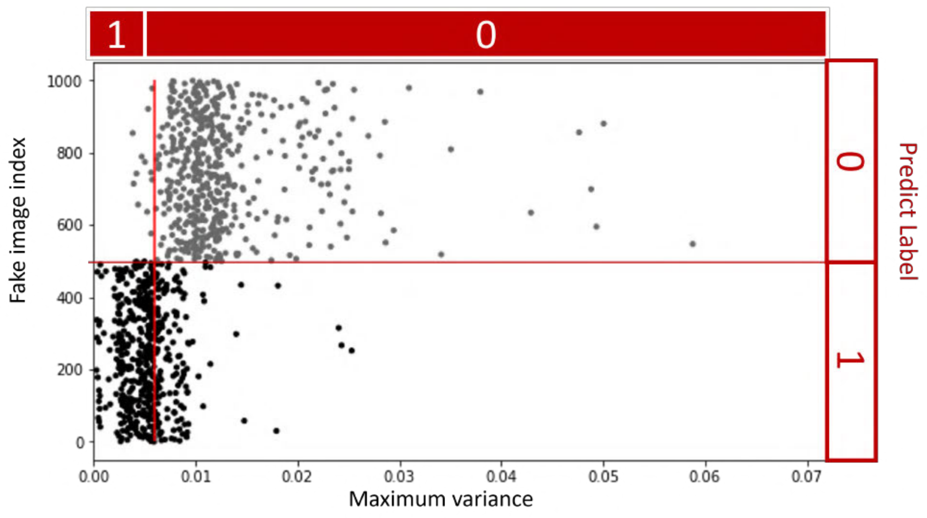
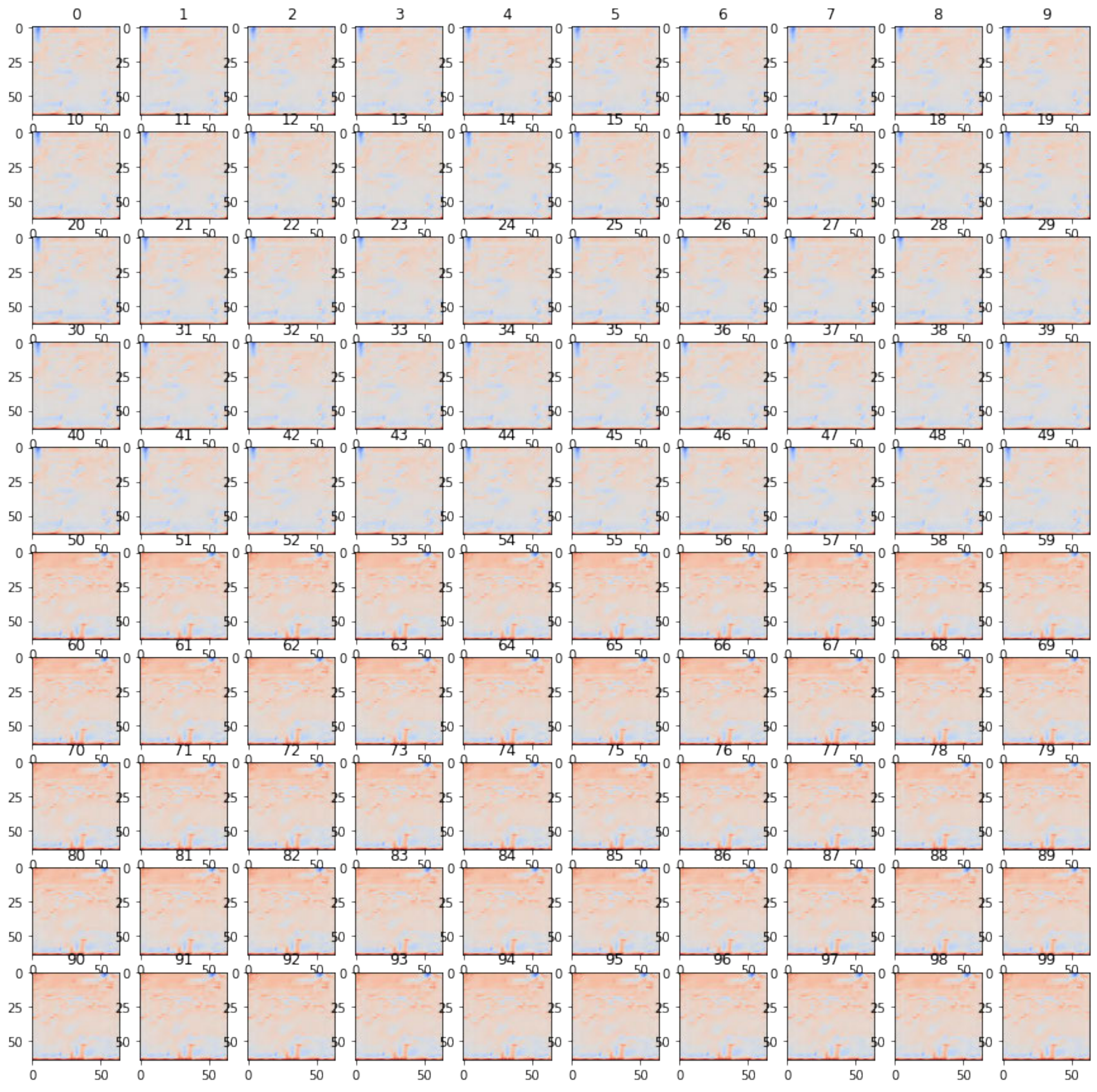


Figure 3.18: Generated images from a failed model. Top 5 rows were given conditional label “1”, bottom 5 rows label “0”. Each label observes mode collapse failure.



where $\tilde{x} = G_1(z^{(l)}, C_t(V(x^{(l)})))$ is the fake images of interest. Figure 3.18 shows fake images generated by this model. The top 5 rows were generated under the conditional label "1", the 5 bottom rows under label "0". This model experienced mode collapse, i.e. one mode was repeated generated within one label group, but for reasons different from the situation described in 3.1.3.

4 Discussion

This chapter discusses the experiment results in their mission to answer the research questions, first regarding the feasibility of a simple GAN model to generate acceptable Heat transfer images, and to what extent the model can capture the most important physical and statistical properties of the training images. Secondly, the results of the adjusted cGAN can answer if the additional variance information is effective, and which variance threshold contributes to a better performing model. Finally, the challenges along the process are mentioned, followed by some suggestions for future directions.

The ***conclusion*** can be drawn for the ***simple*** GAN system as follows. The simple GAN system is capable of generating Heat Transfer images in a variety of patterns, with important physical and statistical properties similar to the training set. However, the similarities in statistical properties are more pronounced with respect to the mean and variance profile, and less for higher order moments. Similar to how certain outliers occur at significantly low frequency in the real data set, such outliers observe the same under-representation in the generated data set.

The ***conclusion*** can be drawn for the ***conditional*** GAN system as follows. The addition of variance-data into each heat-data image, in form of a second channel adds a benefit that the Generator can be now conditioned based on the variance-data. The result suggests that in order to construct a good performing model, the conditional threshold should be chosen such that the training examples have distinct features between the classes with respect to both the main information of heat signature, and the auxiliary information. A good choice for the conditional variance threshold is $t = 0.01$ for the training data A-2, such that the Generator can achieve 95.9% confidence that the generated images comply with the conditional labels.

4.1 Simple GAN results

4.1.1 Successful Model

The generated images show that the simple GAN model does reliably learn on the heat signature information of a real heat transfer simulation. Most features of interests specific to an arbitrary heat transfer stage are present in the generated images. Two boundaries at the bottom and top consistent shows as the hottest and coldest area in each image, in the image body shows a transition between the hot and cold fluids, where hot fluid rises and cold fluid sinks.

While the blurriness of most generated images may be a side effect of using GAN approach [1], the disproportion between patterns within the generated set may be the direct result from the training data itself. For patterns considered as outliers in the training set, the a small number of such patterns could only provide limited learning material for the simple GAN model. On the other hand, the abundance of learning materials for commonly found patterns contained in the training set may be responsible for the fact that the simple GAN model can learn to generate their heat activity in a more natural and organic manner.

On image-to-image comparison, the simple GAN system is capable of imitate the heat activity and the statistics properties of a real image, more successfully regarding the overall heat activity, compared to the local area. This can be observed from visual inspection, where the generate overall heat activity are imitated well but local heat activity is quite choppy and less natural. The same observation can be inferred from the statistical inspection provided in 2.1.1. The fake and real vectors agree in their description of the overall heat activities with the same shape and value range, with the greatest similarity observed in mean vectors and variance vectors, and smaller similarity in skewness and kurtosis vectors. This may be because the fake image is reconstructed pixel-by-pixel. The local heat activity is more difficult to imitate on a pixel level, compared to the global story of the image. This is why the higher order moments, as skewness and kurtosis, are more subject to the local imperfect details of a fake image.

This result indicates that the standard GAN can reproduce the most important physical and statistical properties of a complex PDE-goverened system, despite being otherwise suggested by [14]. Although arguments can be made that this particular simple GAN application uses vertical statistical profile, represented by a lower-dimensional vector which describes an image structure on a global level, instead of a full covariance matrices, which focus more on the local level of statistical properties.

On set-to-set comparison, the simple GAN system is capable of imitate a variety of patterns almost as fully as the training data set, except for real outliers, as illustrated in figure 3.6 regarding mean and variance vectors, or figure 3.11 regarding skewness and kurtosis vectors. All types of statistics show that the generated images have comparable variety in patterns, evidently the densely populated areas occur in the same value ranges. However, the thinly populated areas, where only outliers are present in the training data, are filled with few to none samples in the generated set. This is the direct result of the limited learning materials of the training set for these types of pattern.

The results from the simple GAN implementations suggest that given sufficient training material and training time, a simple GAN system can generate synthetic images which are likely to come from the real data distribution. However, similar to how certain outliers occur at significantly low frequency in the real data set, such outliers observe the same under-representation in the generated data set.

4.1.2 Mode Collapse Model

The failed version of a simple GAN system shows that building proper programming steps is often more beneficial to the experiment than configuring the model parameters (network architecture, learning rates, etc.). Unless the parameters are too extreme or unreasonable, a proper training procedure will eventually give good results.

Generally, the reason for Mode Collapse is not specific, a variety of things can lead to its manifestation. This failed model shows that a training error is one reason that leads to Mode Collapse. The failed model was caused by a critical error in the coding, where D was allowed to be updated while G was training, breaking the adversarial competition between them. This causes D to be “persuaded” by the generated images and assigning them a high probability score. After a while, G focuses on generating only one mode that scores the highest point.

4.2 Adjusted cGAN

4.2.1 Additional variance channel

The generated images by the adjusted cGAN model are visually similar to the unconditional simple GAN results, with regards to both individual heat signature image and variety of patterns. This result suggests that the addition of variance-data into each heat-data image, in form of a second channel, does not cause disruption to the traditional cGAN framework. This proof of concept opens to a plethora of possibilities for conditional GAN. One can construct a GAN system such that the image generation can be conditioned on some sub-sequential information, instead of directly on the main image features. The key is to include the necessary auxiliary information as the additional channel(s) into the main image before giving to the GAN model. In fact, this is not different to how the GAN approaches to generating colored images, e.g. RGB images.

4.2.2 Reasonable Conditional threshold

The adjusted cGAN model performs better with the conditional threshold $t_1 = 0.01$ than with $t_2 = 0.006$, illustrated by figure 3.16 and 3.17 respectively. This might be due to how the conditional threshold t_2 deals with certain characteristics of the training set A-2. The range around this threshold t_2 is where a large number of data points gather, implying that a large number of training images have the maximum variance value around the range of t_2. It also happens that these images depicts very similar patterns of heat structure, because the image pattern of one image is almost identical to the next in a continuous sequence. The Discriminator receives mix signal when D is exposed to images in this range, where two images have similar heat structures, yet carry different labels. As a result, D is less accurate for images containing these patterns, and the mirror network G becomes less accurate when generating these patterns.

The performance comparison between different thresholds suggests that the performance of an adjusted GAN system is affected by the threshold chosen to partition the training examples into classes. The system performs better when the conditional threshold is chosen in such a way that the training examples have distinct features between the classes with respect to both the main information of heat signature, and the auxiliary information. However, if a threshold is chosen such that the two classes have some overlapping heat-data features in the training set, then the generated set may not comply with the conditional labels entirely.

4.2.3 Disruptive Penalty Term

A failed model, with corresponding result in figure 3.12, was brought upon by the inclusion of the penalty term which was added to G's loss function to maximize the likelihood of correct label [11], without matching this term in D's loss function. What may happen in the presence of this penalty term is, its effect is so strong that it encourages the last several layers of G to alter the details on the image in order to reduce the error of image labeling, since the neural network is trained using back propagation. After some training time, G finds that generating a pattern with minimal turbulent structure in the image is enough to satisfy the Discriminator, while adjusting the contrast of some area is enough to satisfy the conditional label. Unfortunately, D is allowing this behaviors because the same term is not included in D's optimization strategy, breaking the adversarial dynamic.

To recover the adversary, both G and D must take on the same penalty term, should this strategy be applied. However, the previous results suggest that this penalty term is not necessary, at least in answering the research questions. This unsuccessful model is brought up to highlight that while various strategies may be applied to formulate a conditional GANs model, the adversarial relationship between D and G must be prioritized, for the strategy to have a chance at working correctly.

4.3 Challenges

One of the first challenges in this work has been constructing the training data set. A high-quality training set is as important as the working programming script that runs the model. The application on Heat transfer simulation images requires that the temperature data be extracted accurately for the sub-sequential statistical analysis. Gray-scale image must be the object of the GAN model, with distinction values between hot and cold region. An 3-channel color image (RGB) which is converted into gray-scale does not have distinct value for hot and cold region. In a case where RGB is the only available data base format, one can convert to CIELUV color space before converting to grayscale as temperature data value. However, the accuracy is not ensured.

One of the main challenges in this thesis has been to achieve the adjusted cGAN system framework and to ensure that the system takes the conditional label into account. Published works done on the conditional generation of data have extensively explored applications where the label-deciding features are only on the main channel, but not on any other related channel, like the variance. Different GAN-derived frameworks in literature have been tested to achieve a good method of incorporating the variance information into G and D, resulting in the use of an additional channel pasted into the main image channel.

However, giving the additional information to the Discriminator was the first logical method to employ, but enforcing the Generator to produce the corresponding channel was a key point which was discovered later in the process. In hind sight, it makes perfect logic that the two networks need to mirror each other to maintain the adversarial competition. Otherwise, if G is instructed to produce 1-channel image, then D only considers that first channel of the inputs, no matter how many additional channels are given.

Another interesting point is the development of the training configuration for G. In each batch, G needs to be initiated with conditional labels for fake image generation. Choices for label source included uniform or random distribution in most applications, or the labels borrowed from the real image in current batch. In extreme cases where each real batch has approximately 1:9 ratio of class labels, while generated image batch has 9:1 ratio of labels in the opposite direction, the model risks not learning much about either classes. To prevent this, borrowing labels from the real batch is a more sensible choice. In another aspect, when batch normalization layers are used for stabilizing training, the generated images from the same batch will take the form of the most general samples appearing in the batch. For a 9:1 label ratio scenario, the minority labels risk not being learned at all. Reducing the batch size to 5 or 10, which is not common in literature applications, seemed to somewhat combat the problem.

The large limitation in this thesis work has been the lack of published works done on training heat transfer images or conditioning on statistics properties. The majority of knowledge learned by the deep learning community which signals the success or failure of a model does not apply for this application. For some time, InfoGAN has been the main inspiration for adding the conditional inputs since it seemed most promising for the task, with mixed results. When applying variance in form of continuous condition, the generated images varies only in color intensity, but not variances, which rejects the model. When applying variance in form of class label condition, it gives mixed results. After some test done to isolate the possible cause, the penalty term for labeling error has been found to manipulate the generated images to the point of homogenized images, and subsequently mode collapse. Removing this element essential left a model akin to the traditional cGAN framework.

Programming skill has been a source of many obstructions through out the project. While Tensorflow and Keras offer various default structures to build and run popular deep learning models, GANs are not included. Constructing a GAN model with two neural networks requires that users delve into lower-level API protocols, especially when a network model uses multiple inputs and outputs. Furthermore, customizing training loops is almost always necessary for the derivative models of GAN. These factors complicate the process between idea and execution.

4.4 Future work

The adjusted cGAN system takes into account the variance information when making the association with the conditional labels, but not solely depends on the variance. This explains why images with similar overall patterns but different variance labels cause confusion to the model. This may be due to D's single output, representing the combined likelihood for image source and image label. If the latter likelihood term is isolated as a second scalar output from D, the cGAN system can optimize this likelihood to make the association with class labels based solely on the variance information. Taken from the lesson of disruptive penalty term, the above likelihood for image label should be included in both D and G loss functions.

5 Annex

5.1 Pseudocode for Simple GAN system

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import tensorflow_probability as tfp
import matplotlib.pyplot as plt
import os
import gdown
from tqdm import tqdm
from PIL import Image

path_a1 = f'dataset_a1'
path_a2 = f'dataset_a2'
path_b1 = f'dataset_b1'

dataset = keras.preprocessing.image_dataset_from_directory(
    path_a1, label_mode=None, image_size=(256,256),
    batch_size=32,shuffle =True,color_mode="grayscale")
dataset = dataset.map(lambda x: (x/255) *2 - 1)

latent_dim = 4096

def generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Reshape(target_shape = [1, 1, 4096], input_shape = [4096]))
    assert model.output_shape == (None, 1, 1, 4096)

    model.add(layers.Conv2DTranspose(filters = 256, kernel_size = 4))
    model.add(layers.Activation('relu'))
    assert model.output_shape == (None, 4, 4, 256)

    model.add(layers.Conv2D(filters = 256, kernel_size = 4, padding = 'same'))
    model.add(layers.BatchNormalization(momentum = 0.7))
    model.add(layers.Activation('relu'))
    model.add(layers.UpSampling2D())
    assert model.output_shape == (None, 8, 8, 256)

    model.add(layers.Conv2D(filters = 128, kernel_size = 4, padding = 'same'))
    model.add(layers.BatchNormalization(momentum = 0.7))
    model.add(layers.Activation('relu'))
    model.add(layers.UpSampling2D())
    assert model.output_shape == (None, 16, 16, 128)

    model.add(layers.Conv2D(filters = 64, kernel_size = 3, padding = 'same'))
    model.add(layers.BatchNormalization(momentum = 0.7))
    model.add(layers.Activation('relu'))
    model.add(layers.UpSampling2D())
    assert model.output_shape == (None, 32, 32, 64)

    model.add(layers.Conv2D(filters = 32, kernel_size = 3, padding = 'same'))
    model.add(layers.BatchNormalization(momentum = 0.7))
    model.add(layers.Activation('relu'))
    model.add(layers.UpSampling2D())
    assert model.output_shape == (None, 64, 64, 32)

    model.add(layers.Conv2D(filters = 16, kernel_size = 3, padding = 'same'))
    model.add(layers.BatchNormalization(momentum = 0.7))
    model.add(layers.Activation('relu'))
    model.add(layers.UpSampling2D())
    assert model.output_shape == (None, 128, 128, 16)

    model.add(layers.Conv2D(filters = 8, kernel_size = 3, padding = 'same'))
    model.add(layers.Activation('relu'))
    model.add(layers.UpSampling2D())
    assert model.output_shape == (None, 256, 256, 8)

    model.add(layers.Conv2D(filters = 1, kernel_size = 3, padding = 'same'))
    model.add(layers.Activation('tanh'))
    assert model.output_shape == (None, 256, 256, 1)

    return model

# declares the generator
generator = generator_model()
generator.summary()

def discriminator_model():
    model = tf.keras.Sequential()

    #add Gaussian noise to prevent Discriminator overfitting
    model.add(layers.GaussianNoise(0.2, input_shape = [256, 256, 1]))

    #256x256x3 Image
    model.add(layers.Conv2D(filters = 8, kernel_size = 3, padding = 'same'))
    model.add(layers.LeakyReLU(0.2))
    model.add(layers.Dropout(0.25))
    model.add(layers.AveragePooling2D())

    #128x128x8
    model.add(layers.Conv2D(filters = 16, kernel_size = 3, padding = 'same'))
    model.add(layers.BatchNormalization(momentum = 0.7))
    model.add(layers.LeakyReLU(0.2))
    model.add(layers.Dropout(0.25))
    model.add(layers.AveragePooling2D())

    #64x64x16
    model.add(layers.Conv2D(filters = 32, kernel_size = 3, padding = 'same'))
    model.add(layers.BatchNormalization(momentum = 0.7))
    model.add(layers.LeakyReLU(0.2))
    model.add(layers.Dropout(0.25))
    model.add(layers.AveragePooling2D())

    #32x32x32
    model.add(layers.Conv2D(filters = 64, kernel_size = 3, padding = 'same'))
    model.add(layers.BatchNormalization(momentum = 0.7))
    model.add(layers.LeakyReLU(0.2))
    model.add(layers.Dropout(0.25))
    model.add(layers.AveragePooling2D())

    #16x16x64
    model.add(layers.Conv2D(filters = 128, kernel_size = 3, padding = 'same'))
    model.add(layers.BatchNormalization(momentum = 0.7))
    model.add(layers.LeakyReLU(0.2))
    model.add(layers.Dropout(0.25))
    model.add(layers.AveragePooling2D())

    #8x8x128
    model.add(layers.Conv2D(filters = 256, kernel_size = 3, padding = 'same'))
    model.add(layers.BatchNormalization(momentum = 0.7))
    model.add(layers.LeakyReLU(0.2))
    model.add(layers.Dropout(0.25))
    model.add(layers.AveragePooling2D())

    #4x4x256
    model.add(layers.Flatten())

    #256
    model.add(layers.Dense(128))
    model.add(layers.LeakyReLU(0.2))

    model.add(layers.Dense(1, activation = 'sigmoid'))

    return model

# Declares the discriminator
discriminator = discriminator_model()
discriminator.summary()

class simpleGAN(keras.Model):
    def __init__(self, d_model, g_model, latent_dim):
        super(simpleGAN, self).__init__()
        self.d_model = d_model
        self.g_model = g_model
        self.latent_dim = latent_dim

    def compile(self, d_optimizer, g_optimizer):
        super(simpleGAN, self).compile()
        self.d_optimizer = d_optimizer
        self.g_optimizer = g_optimizer

    def train_step(self, real_image_batch):
        # Define loss functions
        binary_loss = keras.losses.BinaryCrossentropy()
        # Half-batch for training discriminator and batch for training generator
        # and auxiliary model
        batch = tf.shape(real_image_batch)[0]
        # Create generator input
        random_latent_vectors = tf.random.normal(shape=(batch, self.latent_dim))

        with tf.GradientTape() as d_tape:
            self.d_model.trainable = True
            d_tape.watch(self.d_model.trainable_variables)

            # Train discriminator using half batch real images
            y_disc_real = tf.ones((batch, 1))
            d_real_output = self.d_model(real_image_batch, training=True)
            d_loss_real = binary_loss(y_disc_real, d_real_output)

            # Train discriminator using half batch fake images
            y_disc_fake = tf.zeros((batch, 1))
            # Create fake image batch
            fake_image_batch = self.g_model(random_latent_vectors, training=True)
            d_fake_output = self.d_model(fake_image_batch, training=True)
            d_loss_fake = binary_loss(y_disc_fake, d_fake_output)

            d_loss = d_loss_real + d_loss_fake
        # Calculate gradients
        d_gradients = d_tape.gradient(d_loss, self.d_model.trainable_variables)
        # Optimize
        self.d_optimizer.apply_gradients(zip(d_gradients,
                        self.d_model.trainable_variables))


        with tf.GradientTape() as g_tape:
            # Create generator input
            random_latent_vectors = tf.random.normal(shape=(batch*2,
                        self.latent_dim))
            g_tape.watch(self.g_model.trainable_variables)

            # Create fake image batch
            fake_image_batch = self.g_model(random_latent_vectors, training=True)
            d_fake_output = self.d_model(fake_image_batch, training=True)
            y_gen_fake = tf.ones((batch*2, 1))
            g_img_loss = binary_loss(y_gen_fake, d_fake_output)

            g_loss = g_img_loss
        # Calculate gradients
        # We do not want to modify the neurons in the discriminator when
        # training the generator and the auxiliary model
        self.d_model.trainable=False
        g_gradients = g_tape.gradient(g_loss, self.g_model.trainable_variables)
        # Optimize
        self.g_optimizer.apply_gradients(zip(g_gradients,
                        self.g_model.trainable_variables))

        return {"d_loss_real": d_loss_real,
            "d_loss_fake": d_loss_fake, "g_img_loss": g_img_loss}

# Create a new model instance
simple_gan = simpleGAN(discriminator, generator,  latent_dim=4096)
simple_gan.compile(d_optimizer=keras.optimizers.Adam(learning_rate=2e-4),
                g_optimizer=keras.optimizers.Adam(learning_rate=5e-4))

## Restore the weights
#simple_gan.load_weights('save_path')

cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath='save_path',
            save_freq = 32*50,save_weights_only=True)

# Resume training
first = 0
last = 1
simple_gan.fit(dataset, epochs=last, initial_epoch = first,
                callbacks=[cp_callback])

# Generate fake images
tf.random.set_seed(100)
random_latent_vectors = tf.random.normal(shape=(200, 4096), seed = 100)
F1 = np.squeeze(simple_gan.g_model(random_latent_vectors))

# Compute statistics vectors
M_1 = np.mean(F1,axis= 2)
M_2 = np.var(F1,axis= 2)
M_3 = scipy.stats.skew(F1,axis= 2)
M_4 = scipy.stats.kurtosis(F1,axis= 2)
```

5.2 Pseudocode for Adjusted cGAN system

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import tensorflow_probability as tfp
import matplotlib.pyplot as plt
import os
import gdown
from tqdm import tqdm
from PIL import Image

path_a1 = f'dataset_a1'
path_a2 = f'dataset_a2'
path_b1 = f'dataset_b1'

dataset = keras.preprocessing.image_dataset_from_directory(
    path_a1, label_mode=None, image_size=(64,64),
    batch_size=10,shuffle =True,color_mode="grayscale")
dataset = dataset.map(lambda x:  (x/255) *2 - 1)
```

```

latent_dim = 62+1

def generator_model():
    model = tf.keras.Sequential()

    model.add(layers.Dense(4096, input_shape = [latent_dim]))
    model.add(layers.BatchNormalization())
    model.add(layers.Activation('relu'))
    model.add(layers.Reshape(target_shape=(2, 2, 1024)))

    model.add(layers.Conv2DTranspose(filters = 256, kernel_size = 4,
                                     padding = 'same'))
    model.add(layers.Activation('relu'))
    assert model.output_shape == (None, 2, 2, 256)

    model.add(layers.Conv2D(filters = 128, kernel_size = 4, padding = 'same'))
    model.add(layers.BatchNormalization(momentum = 0.7))
    model.add(layers.Activation('relu'))
    model.add(layers.UpSampling2D())
    assert model.output_shape == (None, 4, 4, 128)

    model.add(layers.Conv2D(filters = 64, kernel_size = 4, padding = 'same'))
    model.add(layers.BatchNormalization(momentum = 0.7))
    model.add(layers.Activation('relu'))
    model.add(layers.UpSampling2D())
    assert model.output_shape == (None, 8, 8, 64)

    model.add(layers.Conv2D(filters = 32, kernel_size = 4, padding = 'same'))
    model.add(layers.BatchNormalization(momentum = 0.7))
    model.add(layers.Activation('relu'))
    model.add(layers.UpSampling2D())
    assert model.output_shape == (None, 16, 16, 32)

    model.add(layers.Conv2D(filters = 16, kernel_size = 3, padding = 'same'))
    model.add(layers.BatchNormalization(momentum = 0.7))
    model.add(layers.Activation('relu'))
    model.add(layers.UpSampling2D())
    assert model.output_shape == (None, 32, 32, 16)

    model.add(layers.Conv2D(filters = 8, kernel_size = 3, padding = 'same'))
    model.add(layers.Activation('relu'))
    model.add(layers.UpSampling2D())
    assert model.output_shape == (None, 64, 64, 8)

    model.add(layers.Conv2D(filters = 2, kernel_size = 3, padding = 'same'))
    model.add(layers.Activation('tanh'))
    assert model.output_shape == (None, 64, 64, 2)

    return model
# declares the generator
generator = generator_model()
generator.summary()
```

```

def discriminator_model():
    image_input = keras.layers.Input(shape=[64, 64, 3])

    #add Gaussian noise to prevent Discriminator overfitting
    gau = layers.GaussianNoise(0.2, input_shape = [64, 64, 3])(image_input)

    c3 = layers.Conv2D(filters = 32, kernel_size = 3, padding = 'same')(gau)
    bnorm3 = layers.BatchNormalization(momentum = 0.7)(c3)
    act3 = layers.LeakyReLU(0.2)(bnorm3)
    drop3 = layers.Dropout(0.25)(act3)
    pool3 = layers.AveragePooling2D()(drop3)

    c4 = layers.Conv2D(filters = 64, kernel_size = 3, padding = 'same')(pool3)
    bnorm4 = layers.BatchNormalization(momentum = 0.7)(c4)
    act4 = layers.LeakyReLU(0.2)(bnorm4)
    drop4 = layers.Dropout(0.25)(act4)
    pool4 = layers.AveragePooling2D()(drop4)

    c5 = layers.Conv2D(filters = 128, kernel_size = 3, padding = 'same')(pool4)
    bnorm5 = layers.BatchNormalization(momentum = 0.7)(c5)
    act5 = layers.LeakyReLU(0.2)(bnorm5)
    drop5 = layers.Dropout(0.25)(act5)
    pool5 = layers.AveragePooling2D()(drop5)

    c6 = layers.Conv2D(filters = 256, kernel_size = 3, padding = 'same')(pool5)
    bnorm6 = layers.BatchNormalization(momentum = 0.7)(c6)
    act6 = layers.LeakyReLU(0.2)(bnorm6)
    drop6 = layers.Dropout(0.25)(act6)
    pool6 = layers.AveragePooling2D()(drop6)

    flat = layers.Flatten()(pool6)

    dense128 = layers.Dense(128)(flat)
    act7 = layers.LeakyReLU(0.2)(dense128)

    d_output = layers.Dense(1,activation='sigmoid')(act7)
    d_model = keras.models.Model(inputs=image_input, outputs=d_output)

    return d_model

# Declares the discriminator
discriminator = discriminator_model()
discriminator.summary()

class Adj_cGAN(keras.Model):
    def __init__(self, d_model, g_model, noise_size, num_classes):
        super(Adj_cGAN, self).__init__()
        self.d_model = d_model
        self.g_model = g_model
        self.noise_size = noise_size
        self.num_classes = num_classes

    def compile(self, d_optimizer, g_optimizer):
        super(Adj_cGAN, self).compile()
        self.d_optimizer = d_optimizer
        self.g_optimizer = g_optimizer

    def get_channel_var(self,real_image_batch):
        '''returns a batch of variance-channels
        '''
        variance = tf.math.reduce_variance(real_image_batch,axis =2,keepdims=True)
        var_full = tf.repeat(variance, repeats = 64, axis = 2)
        return var_full # output shape (batch,64,64,1)

    def get_label(self,real_image_batch, threshold = 0.01):
        '''returns a batch of ground-truth labels
        '''
        batch = tf.shape(real_image_batch)[0]
        var_full = self.get_channel_var(real_image_batch)
        r1 = tf.reduce_all(var_full<=threshold, axis = 1)
        r2 = tf.reduce_all(r1,axis = 1)
        r2 = tf.cast(r2,tf.float32)
        return r2 # output shape (batch,1)

    def get_channel_one_hot(self,real_image_batch):
        '''returns a batch of label-channels
        '''
        batch = tf.shape(real_image_batch)[0]
        label = self.get_label(real_image_batch)
        one_hot = tf.repeat(label,repeats = 64*64)
        one_hot = tf.reshape(one_hot,[batch,64,64,1])
        return one_hot # output shape (batch,64,64,1)

    def train_step(self, real_image_batch):
        # Define loss functions
        binary_loss = keras.losses.BinaryCrossentropy(reduction="auto")
        # Half-batch for training discriminator
        batch = tf.shape(real_image_batch)[0]

        # Prepare for Discriminator
        # Concatenate 3 channels of real batch
        real_image_channel_1 = self.get_channel_var(real_image_batch)
        real_image_channel_2 = self.get_channel_one_hot(real_image_batch)
        real_images_0_1_2 = keras.layers.Concatenate()([real_image_batch,
                                                        real_image_channel_1,
                                                        real_image_channel_2])
        # Generate a fake batch, borrow conditions from real batch
        noise = tf.random.normal([batch, self.noise_size], seed=None)
        label = self.get_label(real_image_batch, threshold = 0.01)
        g_input = keras.layers.Concatenate()([noise, label])
        fake_images_0_1 = self.g_model(g_input, training =True)
        fake_images_0_1_2 = keras.layers.Concatenate()([fake_images_0_1,
                                                        real_image_channel_2])
        # Train Discriminator
        with tf.GradientTape() as d_tape:
            self.d_model.trainable = True
            d_tape.watch(self.d_model.trainable_variables)
            d_real_output = self.d_model(real_images_0_1_2, training=True)
            d_loss_real = binary_loss(tf.ones((batch, 1)), d_real_output)
            d_fake = self.d_model(fake_images_0_1_2, training=True)
            d_loss_fake = binary_loss(tf.zeros((batch, 1)), d_fake)
            d_loss =  d_loss_real + d_loss_fake
        # Calculate gradients
        d_gradients = d_tape.gradient(d_loss, self.d_model.trainable_variables)
        # Optimize
        self.d_optimizer.apply_gradients(zip(d_gradients,
                                             self.d_model.trainable_variables))

        # Prepare for Generator
        noise = tf.random.normal([batch, self.noise_size], seed=None)
        label = self.get_label(real_image_batch, threshold = 0.01)
        g_input = keras.layers.Concatenate()([noise, label])
        # Train Generator
        with tf.GradientTape() as g_tape:
            self.g_model.trainable = True
            g_tape.watch(self.g_model.trainable_variables)
            # Create conditional fake image batch
            fake_images_0_1 = self.g_model(g_input, training =True)
            fake_images_0_1_2 = keras.layers.Concatenate()([fake_images_0_1,
                                                    real_image_channel_2])
            d_fake = self.d_model(fake_images_0_1_2, training=True)
            g_img_loss = binary_loss(tf.ones((batch, 1)),d_fake)
            g_loss = g_img_loss

        # Calculate gradients
        # We do not want to modify the neurons in the discriminator
        # when training the generator
        self.d_model.trainable=False
        g_gradients = g_tape.gradient(g_loss, self.g_model.trainable_variables)
        # Optimize
        self.g_optimizer.apply_gradients(zip(g_gradients,
                                             self.g_model.trainable_variables))

        return {"d_loss": d_loss, "g_img_loss": g_img_loss}


# Create a new model instance
adj_cgan = Adj_cGAN(discriminator, generator, noise_size=62,
                               num_classes=1)
adj_cgan.compile(d_optimizer=keras.optimizers.Adam(learning_rate=2e-4),
                g_optimizer=keras.optimizers.Adam(learning_rate=5e-4))

## Restore the weights
adj_cgan.load_weights('save_path')

cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath='save_path',
                                                 save_freq = 101*50,
                                                 save_weights_only=True)
```

```

# Resume training
first = 0
last = 1
adj_cgan.fit(dataset, epochs=last, initial_epoch = first,
              callbacks=[cp_callback])



# Prepare inputs for G
batch = 200
halfbatch = 100
c_batch = tf.concat([tf.ones([halfbatch]),tf.zeros([halfbatch])],axis = 0)
c_batch = tf.reshape(c_batch,[batch,1])
z_batch = tf.random.normal([batch, adj_cgan.noise_size], seed=None)
zc = keras.layers.Concatenate()([z_batch, c_batch])

# Generate fake images (2-channel)
fake_images_0_1 = adj_cgan.g_model(zc, training =False)

# Extract first channel
FAKE = np.array(fake_images_0_1)[:,:,:,0].reshape((batch,64,64,1))

# Compute maximum variances
VAR = np.var(FAKE,axis = 2).reshape((batch,64))
```

Bibliography

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. DOI: 10.48550/ARXIV.1701.07875. URL: https://arxiv.org/abs/1701.07875.

[2] Konstantinos Bousmalis et al. "Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks". In: *CoRR* abs/1612.05424 (2016). arXiv: 1612.05424. URL: http://arxiv.org/abs/1612.05424.

[3] François Chollet. *DCGAN to generate face images*. https://keras.io/examples/generative/dcgan_overriding_train_step. Accessed: 2022-08-01. 2021.

[4] Ian J. Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks". In: *CoRR* abs/1701.00160 (2017). arXiv: 1701.00160. URL: http://arxiv.org/abs/1701.00160.

[5] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. DOI: 10.48550/ARXIV.1406.2661. URL: https://arxiv.org/abs/1406.2661.

[6] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[7] Jason Brownlee. *How to Develop an Information Maximizing GAN (InfoGAN) in Keras*. https://machinelearningmastery.com/how-to-develop-an-information-maximizing-generative-adversarial-network-infogan-in-keras/. Accessed: 2022-08-01. 2019.

[8] Jonathan Hui. *GAN — Why it is so hard to train Generative Adversarial Networks!* https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b. Accessed: 2022-08-01. 2018.

[9] Kuan Wei. *Build InfoGAN From Scratch*. https://towardsdatascience.com/build-infogan-from-scratch-f20ee85cba03. Accessed: 2022-08-01. 2020.

[10] Mehdi Mirza and Simon Osindero. "Conditional Generative Adversarial Nets". In: *CoRR* abs/1411.1784 (2014). arXiv: 1411.1784. URL: http://arxiv.org/abs/1411.1784.

[11] Augustus Odena, Christopher Olah, and Jonathon Shlens. *Conditional Image Synthesis With Auxiliary Classifier GANs*. 2016. DOI: 10.48550/ARXIV.1610.09585. URL: https://arxiv.org/abs/1610.09585.

[12] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2015. DOI: 10.48550/ARXIV.1511.06434. URL: https://arxiv.org/abs/1511.06434.

[13] Tim Salimans et al. "Improved Techniques for Training GANs". In: *CoRR* abs/1606.03498 (2016). arXiv: 1606.03498. URL: http://arxiv.org/abs/1606.03498.

[14] Jin-Long Wu et al. "Enforcing statistical constraints in generative adversarial networks for modeling chaotic dynamical systems". In: *Journal of Computational Physics* 406 (Apr. 2020), p. 109209. DOI: 10.1016/j.jcp.2019.109209. URL: https://doi.org/10.1016%5C%2Fj.jcp.2019.109209.

Eidesstattliche Erklärung

Hiermit versichere ich – Ngoc Hien Nguyen – an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt oder anderweitig veröffentlicht.

[redacted]

Mittweida, 06. September 2022

Ort, Datum

Ngoc Hien Nguyen