Anomaly Detection in Chest X-Ray Screening based on Adversarial Screening

Binglun Wang

2290735

Supervisor

Dr Yixing Gao & Dr Hyung Jin Chang

A dissertation submitted in partial fulfillment for the degree of

Master of Science in Artificial Intelligence & Machine Learning

of

The University of Birmingham.

School of Computer Science

University of Birmingham

September 26, 2022

I, Binglun Wang, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

In Chest X-ray (CXR) screening, it is desirable for radiologists to report the test results quickly and accurately, however, due to severe workload, this is not always attainable. To reduce the workload of radiologists, this work proposes a pipeline and a novel anomaly detection algorithm that can greatly improve the work efficiency of radiologists. Due to the unpredictability and difficulty in acquiring abnormal samples, the proposed algorithm is trained by healthy images and can give anomaly probability for a test image. Compared to previous works, my method makes two important contributions. First, the proposed algorithm is more focused on pathological information due to the novel architecture and loss function. Second, the proposed algorithm uses Stein's Unbiased Risk Estimator (SURE) to make the model more robust to noise. Extensive experiments on the Montgomery dataset and Shenzhen dataset show that this work has state-of-the-art performance.

Acknowledgements

I would like to express my gratitude to all those who helped me during the project.

My deepest gratitude is first to Dr Yixing Gao, my supervisor, for her constant guidance over the past few months. Every time I get some experiments and submit my draft, I can always get her positive and comprehensive feedback and encouragement. Without her help, this project would not have been in its present form. I would also like to thank my internal supervisor, Dr Hyung Jin Chang, for his guidance. I was inspired by attending his CVPR2022 Digestion Workshop and the meetings I had with him.

Secondly, I would like to express my sincere thanks to Dr Mohan Sridharan, who was the inspector of my project. He has given me a lot of advice in my reports and presentations. Without his help, I could not have completed my report and presentation so smoothly.

Finally, I would like to thank Ping Su, my partner, for her support in life and academics. I would also like to thank my friend Dr Li Kevin Wenliang who has given me many suggestions and opinions about my experiments presentation and the academic writing of the final dissertation.

I firmly believe this project is just a fresh start to my academic path, and there must be an ocean of knowledge waiting for me to explore. Again, thank those who have given me support and care in writing this dissertation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In Chest X-ray (CXR) screening, it is desirable for radiologists to report the test results quickly and accurately, however, due to severe workload, this is not always attainable. Generally, the function of a radiologist can be thought of as anomaly detection, in which abnormal features shown on a CXR image need to be identified. Other variables, such as the shape, volume and of the lungs, should not affect the results as long as they are healthy. Therefore, I propose to use anomaly detection techniques in machine learning to help radiologists expedite the diagnosis process. Currently, deep learning provides powerful methods for diagnosis. But supervised learning algorithms have shown limited use in the field of anomaly detection due to unpredictable generalisation properties and the difficulty of acquiring abnormal samples. This work proposes a pipeline and a novel anomaly detection algorithm only trained in healthy images that can greatly improve radiologists' work efficiency.

Specifically, the pipeline firstly uses a high sensitivity filter based on the proposed algorithm to detect certainly normal samples. As for the rest images, the proposed algorithm gives anomaly probability to help radiologists. Compared to previous works, my method makes two important contributions. First, the proposed algorithm is more focused on pathological information due to the novel architecture and loss function. Second, the proposed algorithm uses Stein's Unbiased Risk Estimator (SURE) to make the model more robust to noise. Extensive experiments on the Montgomery dataset and Shenzhen dataset show that this work has state-of-the-art performance.

1.1 Motivation

In December 2019, the first known Coronavirus disease 2019 (COVID-19) case was identified in Wuhan, China [16]. Soon afterwards, COVID-19 quickly spread worldwide, resulting in the COVID-19 pandemic. As of 12 Sept 2022, 603,711,760 reported individuals were infected with COVID-19, including 53069 deaths [17].

In this pandemic, especially at the beginning, the demand for timely Chest X-ray diagnoses has increased rapidly. For example, in figure 1.1, Chen et al. show that COVID-19-related chest x-ray workload has dramatically increased at Singapore General Hospital [18].

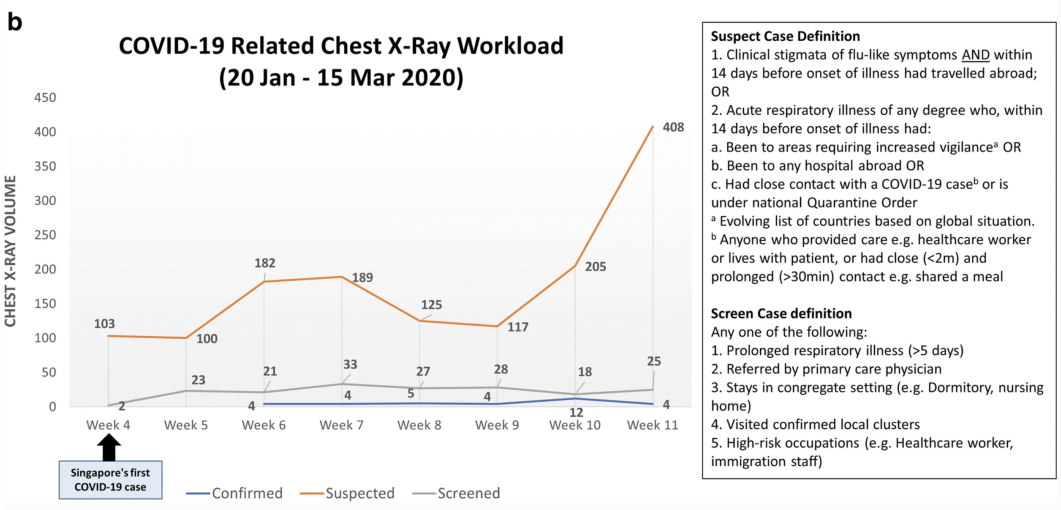


Figure 1.1: Workload related to COVID-19, 20 January $\sim$ 15 March, 2020

However, the current diagnosis efficiency does not meet needs not only in the pandemic but also in other routine diagnoses of other diseases [19]. The impact of fast CXR screening has become ever more significant in the post-COVID era. Radiologists usually have a very heavy workload and need to produce a timely report (Figure 1.2). It is problematic when the quantity of scans is large, such as during the pandemic.

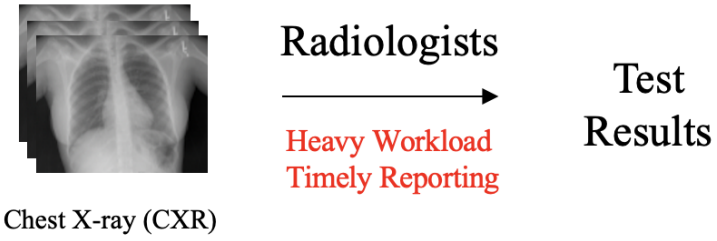


Figure 1.2: Scenario

Artificial intelligence has improved a lot of our work efficiency. It would be very helpful if we could develop an AI procedure to assist radiologists [20]. One of the applications of AI is Anomaly detection which aims to detect identifies whether an object is normal or abnormal. It can help radiologists' work.

Current deep learning research in anomaly detection provides powerful methods for diagnosis. Depending on whether the training data has a corresponding label, it is mainly divided into supervised learning and unsupervised learning. Supervised learning algorithms have shown limited use in the field of anomaly detection due to unpredictable generalisation properties and the difficulty of acquiring abnormal samples. For example, in Figure 1.3 if we train the model through normal samples and abnormal samples. The model learned the decision boundary as the red one. Now, suppose there is a scan of a new disease, coloured in yellows. The model may identify the yellow one as normal

because of the arbitrary decision boundaries. This is dangerous, as the error would lead to delayed treatment and loss of control over the spread of the disease.

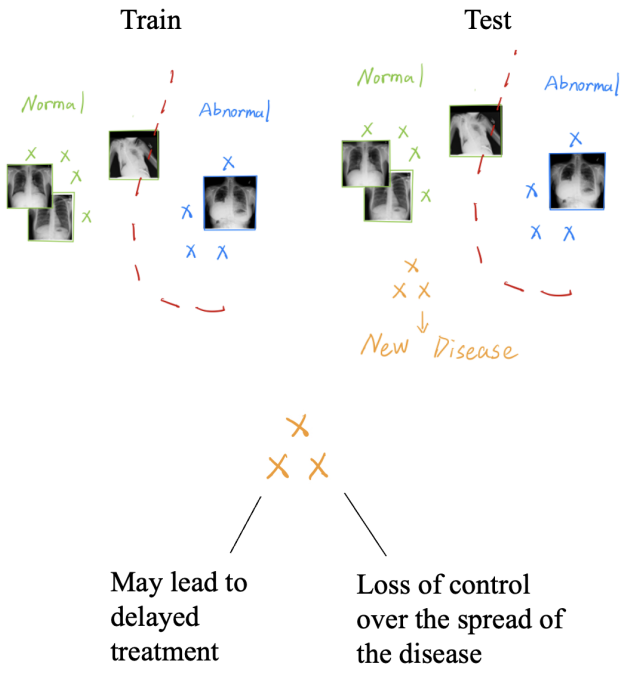


Figure 1.3: Example of Problems with Supervised Approach

As an alternative, if we can take advantage of the abundant normal and healthy scan data, we can train unsupervised or semi-supervised models to learn the distribution of this dataset. This model can then be used for anomaly detection by judging how likely a given scan belongs to the healthy scan dataset. The goal of it is to learn a distribution boundary between normal samples and all other samples trained by normal data. For example, in Figure 1.4, the greens and the yellows are normal samples, and the blues are abnormal samples. The purple circle is the model distribution boundary, and the red circle is the decision boundary we can set. If a sample is inside the decision boundary, it can be detected as a normal sample. On the contrary, if a sample is outside the boundary, it can be detected as an abnormal sample.

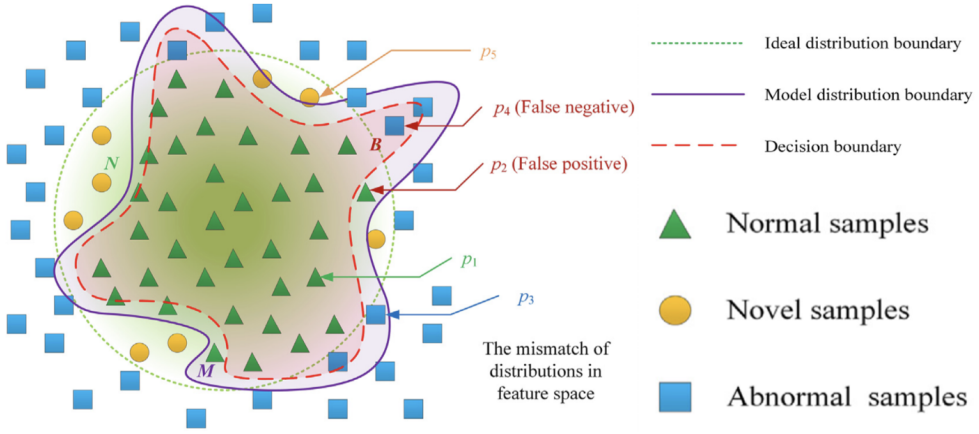


Figure 1.4: Unsupervised Anomaly Detection [1]

As a form of unsupervised learning algorithm, Generative Adversarial Networks (GAN or GANs) [11] have shown state-of-the-art performance in recent years. The outstanding capability of learning data distributions without supervision makes GAN-based methods popular in anomaly detection. GAN (including their variants) has become some of the best methods of anomaly detection. On a high level, these methods aim to learn the data distribution and feature representations in latent space by only training with normal samples. The abnormal samples can be detected due to it has the different distribution in data and feature representation space. On the other hand, the CXR dataset often has data imbalance problems due to inter-subject variability. But, GAN has a very good generalisation ability, which alleviates the problem of data imbalance to a certain extent. In addition, the CXR dataset is complex. GAN is suitable for anomaly detection tasks pertaining to the complex dataset and fits the high dimensional data distribution.

This work focuses on Anomaly Detection in Chest X-Ray Screening based on Adversarial Screening and aims to improve radiologists' work efficiency.

1.2 Challenges

Currently, unsupervised CXR anomaly detection faces several key challenges. For example, due to inter-subject variability CXR anomaly detection has data imbalance among normal samples as mentioned in the previous section. In this section, I summarise a few challenges in designing an AI pipeline for this task. Both the pipelines and algorithms I propose here collectively address the challenges faced by radiologists.

Firstly, Data imbalance is not only between normal and abnormal samples but also within normal samples. For example, in figure 1.5, some individuals have deformed bodies, but their lungs may be healthy. A classifier may detect the image as abnormal, as the dataset has fewer similar samples. How to let the model learn it?

Figure 1.5: Left: Normal samples; Right: Rare normal samples

Secondly, pathological features are confounded by inter-subject variability. For example, in figure 1.6, some people have thick lungs, and some have thin ones. Due to the variability of the human body, two healthy lung images can be very different. The difference between the lesion and healthy images is minimal in lung diseases. Under inter-subject variability of all these factors in the human body, how does the algorithm identify the abnormality in a robust and stable manner?

Figure 1.6: Left: thick lung lobes; Right: thin lung lobes

Thirdly, the pathological Region of Interest (ROI) in CXRs is the lung, which is also the ROI in this task. For example, in figure 1.7. But a Chest X-ray image contains a person's entire chest cavity, such as bones, such as body tissue. How to let the model focus on this region?

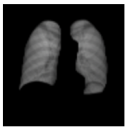


Figure 1.7: Example of pathological ROI in CXR

Lastly, medical images inevitably cover noise. How to make the model more robust to noise?

1.3 My Work and My Contributions

Currently, it is technically impossible and ethically problematic to fully replace radiologists with an AI [21], as the diagnosis by AI may not be completely accurate and, therefore, can lead to misdiagnosis for patients. So, I designed a pipeline as shown in figure 1.8. The pipeline uses the anomaly detection model to build a high sensitivity filter with a very tight decision boundary, which can detect certainly normal samples and report healthy directly. Thus, the workload of radiologists can be significantly reduced. As for the rest images, use the model to give anomaly probability to help to assist radiologists in making a diagnosis.

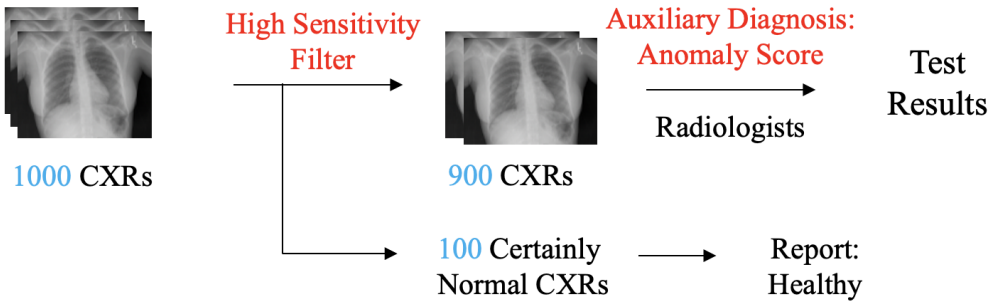


Figure 1.8: The designed application

Compared to previous works, my method makes two important contributions. First, the proposed algorithm is more focused on pathological information due to the novel architecture and loss function. Second, the proposed algorithm uses Stein's Unbiased Risk Estimator (SURE) to make the model more robust to noise. Extensive experiments on the Montgomery dataset and Shenzhen dataset show that this work has state-of-the-art performance. In summary, I made the following contributions:

1. Presented a novel anomaly detection algorithm that is only trained by normal data. It can learn to identify anomalous images and image segments.

2. Proposed a novel loss function, which can make the algorithm more focused on pathological information and more robust to noise

3. Investigated and summarised several significant modern anomaly detection methods based on GANs in CXRs.

Chapter 2

Background

The purpose of this chapter is to better understand my work for readers who do not have much background in my research field, so I will supplement the foundation of deep learning techniques and knowledge that I have designed in my work. I will introduce the principle of each knowledge and explain it from a perspective without a knowledge background. To make it easier for the reader without background, the content of this chapter will be more playful to read.

2.1 Deep Learning

Deep learning is an artificial intelligence method based on artificial neural networks (ANN). It was developed and explored by Fran Rosenblatt in 1962 [22]. Although deep learning has been around for decades, training a deep ANN needs sufficient computing resources. Therefore, deep learning is not as popular as now due to insufficient computing resources for training. With the development of the semiconductor industry, graphics processing unit (GPU) can process large blocks of data in parallel efficiently. In 2011 and 2012, deep learning-based methods achieved the best performance in many challenges, eventually leading to the deep learning revolution. For example, Krizhevsky et al. proposed that AlexNet [23] achieved the ILSVRC-2012 competition championship. At present, deep learning has shown its power in lots of applications. For example, automatic speech recognition, image recognition, and medical image analysis.

2.2 Artificial Neural Network (ANN)

Deep learning based on ANN has been mentioned before. In order to make it clear, in this section, I will introduce what ANN is.

2.2.1 Single Layer Neural Network

Before introducing a general Artificial neural network, let us learn about the simplest unit - a single-layer neural network. A single-layer neural network can be viewed as a function that maps one or more inputs x to an output $f(x)$. As shown in the figure 2.1 below, the input x_1, x_2 are in the black arrow and the *output* is in the green arrow.

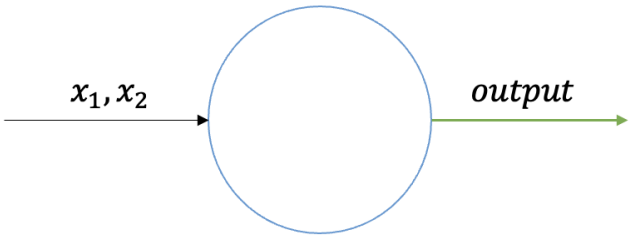


Figure 2.1: Single Artificial Neural Network

Linear transformation: A single-layer neural network also contains learn-able parameters w_1, w_2, b in the Figure 2.2. A single-layer neural network takes the input and output of the data and adjusts these parameters to map an input to the correct output value. Where w is the weight of each input, and b is the value to offset the output. If you have studied linear algebra, you will find that this is a linear transformation. For example, if we know how much Alex and Bob weigh, we want the neural network to output the combined gravity of the two people.

We actually know the formula for gravity $G = m * g$, but this single-layer neural network doesn't know it, so how can it learn this relationship? Suppose we count a lot of data, each data contains the weight of the first person in x_1 kg, the weight of the second person in x_2 kg, and their combined gravity $output$. We throw these data into the neural network to learn, and it will eventually learn two parameters w_1, w_2, approximating the value of g.

So adding parameters, our single-layer neural network can express it with this mathematical formula:

$$output = \sum_i w_i * x_i + b \tag{2.1}$$

Activation function: So far, neural networks can only do linear transformations. But in fact, in many problems in real life, the relationship between input and output is not linear.

For example, in a picture of a dog, our human eyes see it, and the brain will recognize that it is a dog. The dog picture is the input, "this is a dog" is the output, and the brain processing process is definitely not a linear transformation.

The neural network implements this nonlinear transformation through an activation function σ, in the figure 2.1, through an activation function of a nonlinear transformation. You might ask whether an activation function is so useful. Just like 0 and 1, our current computer is based on them, and it even constructs the entire virtual world. Likewise, if you have multiple single-layer neurons combined, coupled with learnable parameter tuning, it can do a lot more than you might expect.

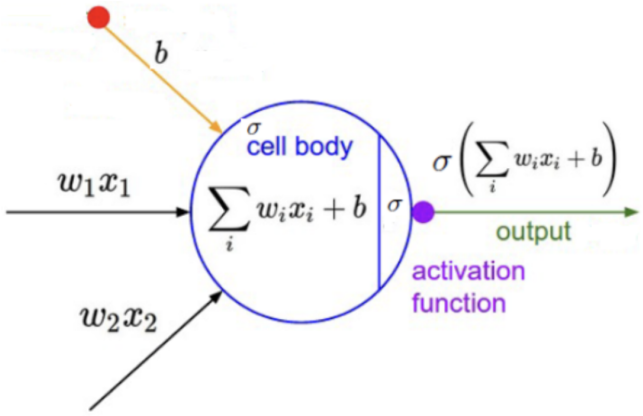


Figure 2.2: Single Artificial Neural Network [2]

Finally: $output = \sigma(\sum_i w_i * x_i + b)$.

2.2.2 Multi Layer Perceptions

From this section onwards, I will formally introduce the generic structure of neural networks, it is recommended to read the previous section first for a better understanding. Also, the purpose of this section is to understand how it works without first describing how neural networks learn.

In simple terms, a multi-layer neural network (Feed-Forward Neural Network) is built from a combination of single-layer neural networks.

A multi-layer neural network is a directed acyclic graph, where the neurons form part of the nodes of the graph, called layers, and the learnable parameters W, B form the edges and remaining nodes, respectively. The first layer of the neural network is called the input layer, the last layer is called the output layer, and the middle layer is called the hidden layer as Figure 2.3.

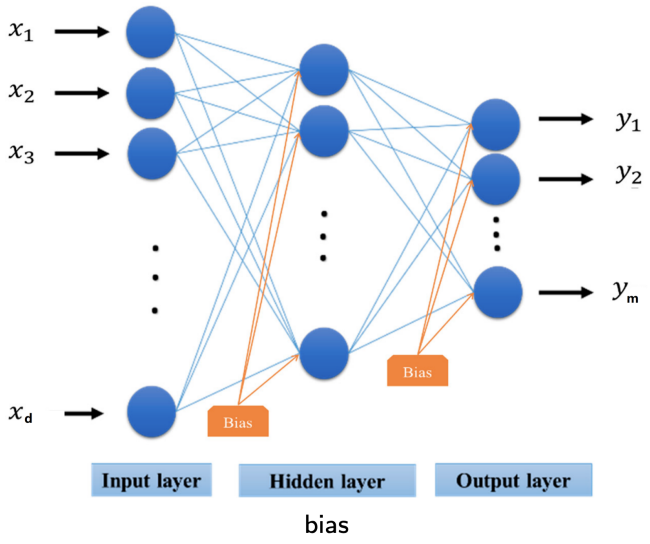


Figure 2.3: Multi-layer neural network architecture with blue circles as neurons (units), blue edges as weights (W) and yellow as biases (B) [2]

To better introduce the later sections, here we try to define the neural network mathematically (as Figure 2.4).

L: the number of layers of the neural network, the number of layers, as follows $L = 4$. Layer 1 is the input layer and layer L is the output layer.

m: the width of the neural network, $m = 4$ as shown below.

$w^l_{j,k}$: The weight between the kth neuron in layer $l-1$ and the jth neuron in layer l. This is a learnable parameter.

$b_{j,l}$: The bias of the jth neuron of the lth layer, which is also a learnable parameter.

$z^l_j = \sum_k w^l_{j,k} * a^{l-1}_k + b^l_j$: linear transformation of the input of the jth neuron in the lth layer

$a^l_j = \sigma(z^l_j)$: the final output of the jth neuron in the lth layer after the activation function

In current neural network training, parallel operations are computationally efficient, so we usually express the neural network in terms of a matrix.

$$W^l = \begin{bmatrix} w^l_{1,1} & \cdots & w^l_{1,m} \\ \vdots & \ddots & \vdots \\ w^l_{m,1} & \cdots & w^l_{m,m} \end{bmatrix}, b^l = \begin{bmatrix} b^l_1 \\ \vdots \\ b^l_m \end{bmatrix}, z^l = \begin{bmatrix} z^l_1 \\ \vdots \\ z^l_m \end{bmatrix}, a^l = \begin{bmatrix} a^l_1 \\ \vdots \\ a^l_m \end{bmatrix} \tag{2.2}$$

Eventually, we can write the equation as follows.

$$a^l = \sigma(z^l) = \sigma(W^l a^{l-1} + b^l) \tag{2.3}$$

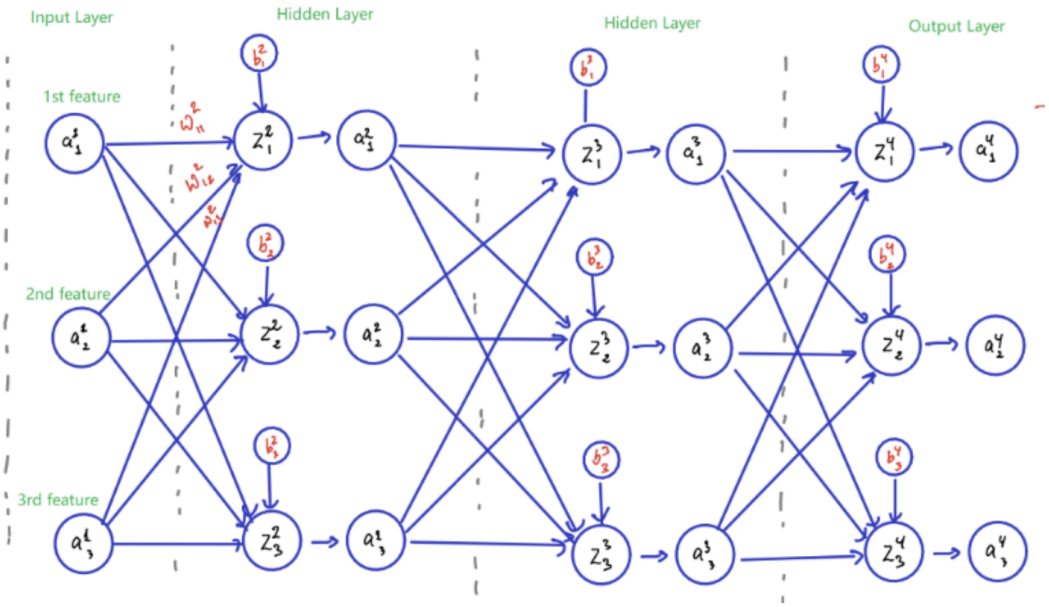


Figure 2.4: Example of Multi-layer Neural Networks [2]

2.2.3 Active Function

The activation function has been mentioned in the previous two sections, and in this section, I will explain what it is useful for from an intuitive point of view.

An activation function is a linear transformation of a neuron's input ($z^l = W^l a^{l-1} + b^l$ as mentioned in the previous section), which is then mapped to the final output by a function. Since a linear transformation has been performed earlier, many activation functions are non-linear, such as the three activation functions in figure 2.5 below.

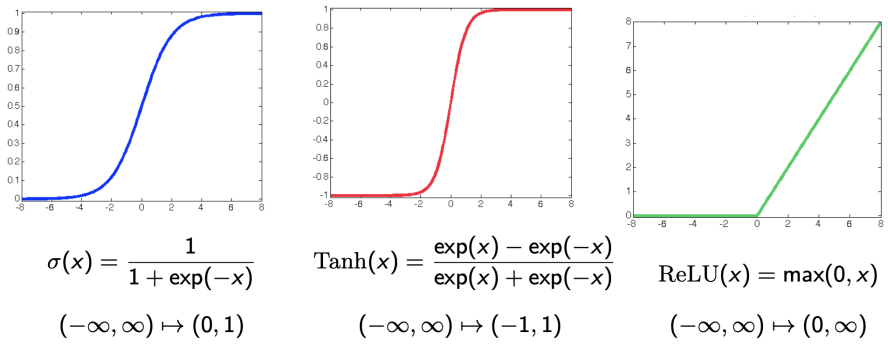


Figure 2.5: Sigmoid, Tanh, ReLU activation function [2]

As described in the previous section, it is due to the presence of non-linear activation functions that neural networks are able to solve very complex problems. So let us now understand it visually. For example, in the figure below, the linear transformation can only deflate and translate our input space.

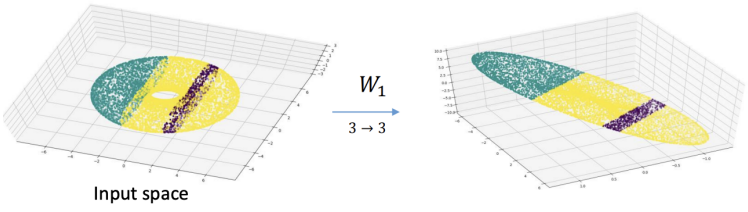


Figure 2.6: Linear transformation [3]

But if we want to map the input space to the diagram 2.7 below, it is impossible to do so with only linear transformations.

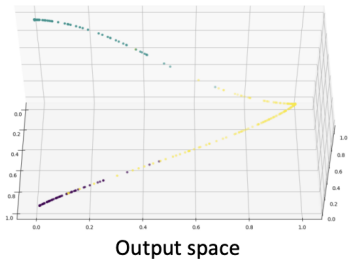


Figure 2.7: Expected output space [3]

But if we use a non-linear activation function, such as ReLU, we can ”fold” the input space like origami to the output space we want.

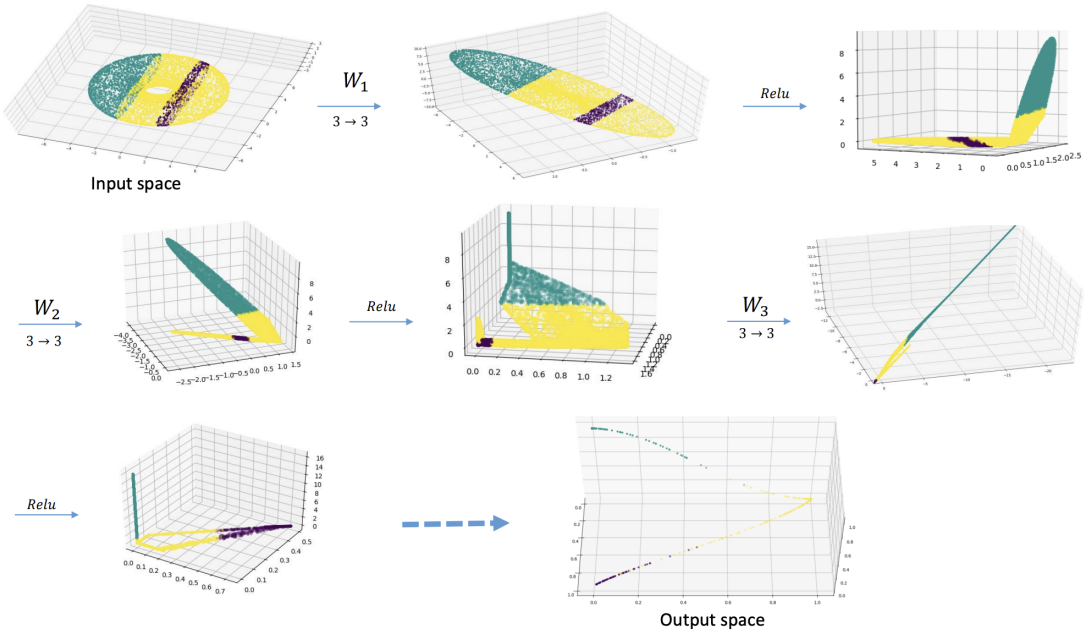


Figure 2.8: ReLU function [3]

Isn’t it very shocking that a very simple ReLU can be so useful?

After understanding the role of the activation function, the next step is to introduce a few commonly used activation functions. Prior to 2012, the main activation functions used in neural networks were the Sigmoid and Tanh activation functions. One of the disadvantages of these two activation functions is that they can suffer from ”gradient disappearance”.

The gradient is described in more detail in a later section, but it is not a bad idea to introduce it now. The training principle of neural networks is to use the gradient to approximate the global optimum and thus fit a suitable parametric model. It can be seen that when the input is particularly small or large, the derivatives in the sigmoid and Tanh are almost zero, which can make it difficult to transform the parameters after training, which is the ”gradient disappearance” problem mentioned earlier.

Krizhevsky et al.'s work on AlexNet [23] uses the ReLU function, which is currently the most used loss function in the structure of the paper due to the stability of the gradient on the one hand, and the follow-up of AlexNet's work on the other. Even though Sigmoid, the problem of vanishing Tanh gradients has been solved to some extent in subsequent research work, most network designs are still based on the ReLU activation function and its variants due to its simplicity properties. For example, Leaky ReLU is a variant based on ReLU, which has a relatively flat slope when the input is negative. It is very much used for sparse tasks, such as work on generative models related to GAN.

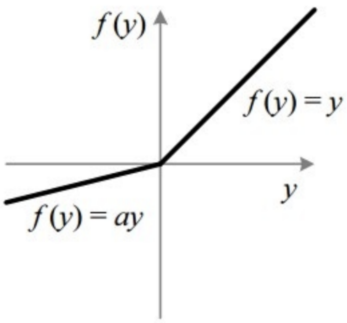


Figure 2.9: Leaky ReLU function [4]

Also since Sigmoid can map the output to the $[0, 1]$ interval, many networks add Sigmoid to their last layer and design the output as a probabilistic output.

Method	Year	Papers
ReLU	2000	7505
Sigmoid Activation	2000	5013
GELU Gaussian Error Linear Units (GELUs)	2016	4671
Tanh Activation	2000	4623
Leaky ReLU	2014	822
GLU Language Modeling with Gated Convolutional Networks	2016	268

Figure 2.10: Activation functions in papers top6 [4]

2.3 Back Propagation

2.3.1 Loss Function

Neural networks have been introduced in the previous sections, and from this chapter onwards, it is time to describe how neural networks learn.

Before talking about how to teach neural networks to learn, let's consider how people can learn. We can learn by adjusting our behaviour through feedback on the results of an example. For example, checking answers after d problem is a very effective way to improve the correctness of even doing a

problem.

This is also true for teaching neural networks, where we need a loss function to evaluate the results of the network. For example, if we want to train a neural network that estimates the price of a London pizza based on its size. Then its input is the size of the pizza x and its output is the price $output$. Assuming that the real pizza size and price data are x, y, then the loss function can be designed as $loss = (output - y)^2$

Here is the $L1, L2$ loss function:

$$L_1(y, \hat{y}) = \alpha \|y - \hat{y}\|_1 = \alpha |y - \hat{y}| \tag{2.4}$$

$$L_2(y, \hat{y}) = \alpha \|y - \hat{y}\|_2 = \alpha (y - \hat{y})^2 \tag{2.5}$$

Here w is the weight, y is the output value and $\hat{y}$ is the true value.

When $\alpha = 1$, the $L1, L2$ loss functions are also known as Mean Absolute Error (MAE) and Mean Square Error (MSE).

The neural network is actually trained according to the gradient, so we find after the derivation:

$$\frac{\partial L_1}{\partial y} = 1 \ when \ y >= \hat{y} \tag{2.6}$$

$$= -1 \ when \ y < \hat{y} \tag{2.7}$$

$$\frac{\partial L_2}{\partial y} = 2\alpha (y - \hat{y}) \tag{2.8}$$

$L1$ is more stable because its derivative value is fixed, and the speed of training is more stable. $L2$ depends on the difference with the true value, so the training speed will be faster when the difference is larger.

2.3.2 Gradient Descent & Computation Graph

Let the loss function be y and the learnable parameters be w . Evaluating the learning effectiveness of a neural network is a matter of tuning w so that its loss function on the dataset is as small as possible. As a very simple example, the figure below shows the loss function $y = (w - 5)^2 + 5$ for only one parameter. The value of the loss function is minimised when the model learns $w = 5$.

Figure 2.11: Examples of loss functions and parameters

It is easy to see that for a neural network, it is actually possible to find the global minimum by computing the system of equations by directly deriving the loss function for all parameters so that the derivative is equal to zero, and this gives the best results for the neural network. But in fact, in deep learning, because the neural network is very large, very deep, and has a lot of parameters if we go to compute it directly, it is impossible to solve it at the right time with the current computational resources.

But we can use gradient descent to find the global minimum. For example, when w is at 7.9, we can calculate the derivative and find a better solution than the current value by following it in the opposite direction (yellow arrow). The biggest problem with gradient descent is that a function may have multiple local minima, and the gradient descent method is likely to 'trap' the parameters in the local minima. This is the fundamental reason why so much research is focused on neural network training today.

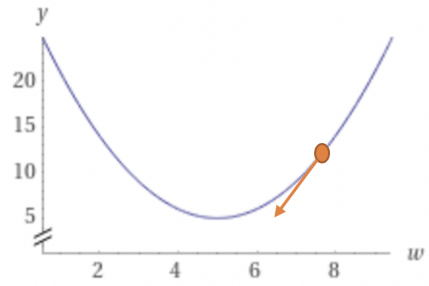


Figure 2.12: Examples of loss functions and parameters

To better understand backpropagation, the concept of a computational graph is introduced here: any formula can be written as a directed acyclic graph, e.g. $f(a,b,c) = (a+b)(b+c)$ can be written in the form of the following graph.

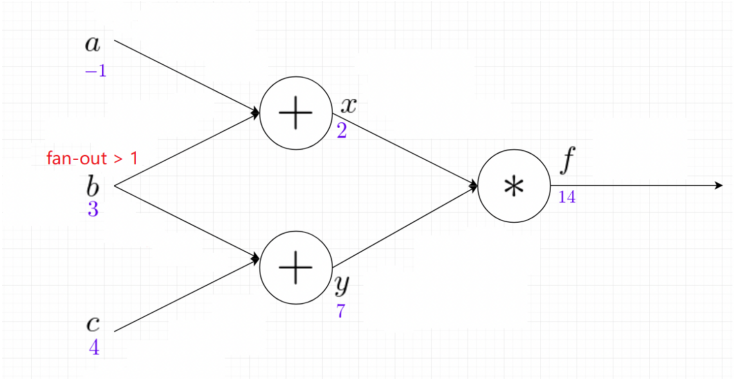


Figure 2.13: Example of a computational graph[2]

Then when $a=-1, b=3, c=4$ we get $x=2, y=7, f=14$ respectively. If we want to calculate the partial derivative of f for each parameter, we can do so by backpropagation. We calculate them in order from right to left, so that the calculation of each new partial derivative only needs to consider the relationship with the right layer.

like Figure 2.14 $\frac{\partial f}{\partial a} = \frac{\partial f}{\partial f} * \frac{\partial f}{\partial x} * \frac{\partial x}{\partial a}$, the order from left to right on the right-hand side of the equation is the order in which the graph backpropagation operations are calculated.

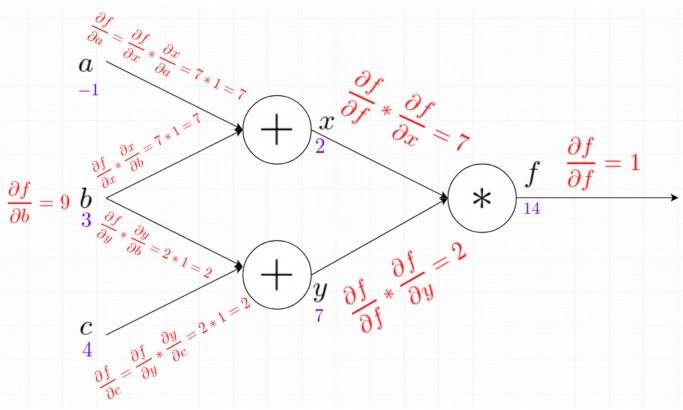


Figure 2.14: Example of a computational graph[2]

2.3.3 Back Propagation in ANN

Backpropagation is the theory underlying the training of neural networks. As mentioned before w and b are the two parameters to be learned by the neural network and the purpose of backpropagation is to make the loss function as small as possible. Backpropagation updates w, b backward on the computational graph of the neural network by means of gradient descent so that the two parameters are as close to the global minimum as possible.

In mathematical terms, let C be the loss function, and update w, b.

$w = w - \alpha \cdot \frac{\partial C}{\partial w}, b = b - \alpha \cdot \frac{\partial C}{\partial b}$ as soon as the gradient has been computed

The following section is on how to derive the $\frac{\partial C}{\partial w}, \frac{\partial C}{\partial b}$

For ease of reading, here again, are the definitions that have been used in previous articles.

L: the number of layers of the neural network, the number of layers, as follows $L = 4$. Layer 1 is the input layer and layer L is the output layer.

m: the width of the neural network, $m = 4$ as shown below.

$w^l_{j,k}$: The weight between the kth neuron in layer $l-1$ and the jth neuron in layer l. This is a learnable parameter.

$b_{j,l}$: The bias of the jth neuron of the lth layer, which is also a learnable parameter.

$z^l_j = \sum_k w^l_{j,k} * a^{l-1}_k + b^l_j$: linear transformation of the input of the jth neuron in the lth layer

$a^l_j = \sigma(z^l_j)$: the final output of the jth neuron in the lth layer after the activation function

In current neural network training, parallel operations are computationally efficient, so we usually express the neural network in terms of a matrix.

$$W^l = \begin{bmatrix} w^l_{1,1} & \cdots & w^l_{1,m} \\ \vdots & \ddots & \vdots \\ w^l_{m,1} & \cdots & w^l_{m,m} \end{bmatrix}, b^l = \begin{bmatrix} b^l_1 \\ \vdots \\ b^l_m \end{bmatrix}, z^l = \begin{bmatrix} z^l_1 \\ \vdots \\ z^l_m \end{bmatrix}, a^l = \begin{bmatrix} a^l_1 \\ \vdots \\ a^l_m \end{bmatrix}$$

So eventually, $a^l = \sigma(z^l) = \sigma(W^l a^{l-1} + b^l)$

Let the loss function be C, and $z^l_j = \sum_k w^l_{j,k} * a^{l-1}_k + b^l_j$

$\frac{\partial C}{\partial w^l_{j,k}} = \frac{\partial C}{\partial z^l_j} \cdot \frac{\partial z^l_j}{\partial w^l_{j,k}} = \frac{\partial C}{\partial z^l_j} \cdot a^{l-1}_k$

$\frac{\partial C}{\partial b^l_j} = \frac{\partial C}{\partial z^l_j} \cdot \frac{\partial z^l_j}{\partial b^l_j} = \frac{\partial C}{\partial z^l_j}$

Let $\delta^l_j = \frac{\partial C}{\partial z^l_j}, \delta^l = \begin{bmatrix} \delta^l_1 \\ \vdots \\ \delta^l_m \end{bmatrix}$

So we end up with: $\frac{\partial C}{\partial W^l} = \delta^l (a^{l-1})^T, \frac{\partial C}{\partial b^l} = \delta^l$

When $l = L$, (the last layer):

$\delta^L_j = \frac{\partial C}{\partial a^L_j} \cdot \sigma'(z^L_j)$

Otherwise, (Hidden layer): $\delta^l_j = \frac{\partial C}{\partial a^l_j} \cdot \sigma'(z^l_j) = \sum_k \frac{\partial C}{\partial z^{l+1}_k} \frac{\partial z^{l+1}_k}{\partial a^l_j} \cdot \sigma'(z^l_j) = \sigma'(z^l_j) \cdot \sum_k \delta^{l+1}_k \cdot w^{l+1}_{k,j}$

So ultimately the following can be concluded:

$$\frac{\partial C}{\partial W^l} = \delta^l (a^{l-1})^T, \frac{\partial C}{\partial b^l} = \delta^l \tag{2.9}$$

Where, $\delta^l = \begin{cases} \nabla_{a^L} \odot \sigma'(z^L), & \text{if } l = L \text{ (output layer)} \\ \sigma'(z^l) \odot ((W^{l+1})^T \delta^{l+1}), & \text{otherwise (hidden layer)} \end{cases}$

2.4 Optimization

2.4.1 SGD & Mini-Batch

The computational resources required to compute the gradient are large, so often in practice, a random portion of the data can be selected for training. Suppose there are a total of n data, and one is

randomly selected for training at a time. This is the SGD algorithm (Stochastic Gradient Descent)

If a portion of the data is drawn from it, this is the Mini-batch gradient descent algorithm

$\frac{\partial C}{\partial W^l} \approx \frac{1}{s} \sum_i \frac{\partial C}{\partial W_i^l}$

$1 < s < n$, Mini-batch gradient descent

$s = 1$, Stochastic gradient descent

$s = n$, Batch gradient descent

The gradient descent algorithm mentioned previously has the potential to make the parameters trapped to local optima. However, SGD and Minibatch GD can solve this problem to some extent. Since each training is sampled, the gradient is more random due to the sampling of the data and can "escape" even if it is trapped at a local optimum.

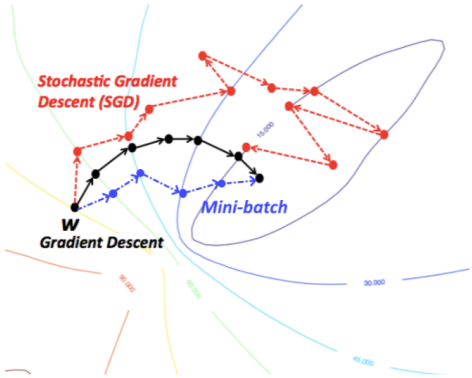


Figure: Search paths of standard gradient descent, SGD and mini-batch gradient descent. Figure modified from https://wikidocs.net/3413

Figure 2.15: Example of SGD algorithm gradient descent, with the global optimum in red[2]

2.4.2 Solving the Magnitude Problems (Learning Rate Decay & Adaptive Learning Rate Methods

When training a neural network, updating the parameters requires setting a learning rate ∇.

For example, $w = w - \eta \cdot \nabla w$

Finding a suitable learning rate is difficult and important [24]; if it is too small, it will lead to slow convergence. If it is too large, it will cause the training to oscillate or even diverge around the optimal parameter points.

A good solution to this problem is to set a learning rate decay [25]: set a very large learning rate at the beginning of the training and then gradually decrease the learning rate during the training process.

The intuitive explanation is that a large learning rate at the beginning makes it easy for the model to escape from local minima and speeds up training. And later on, when the global optimum is found, reducing the learning rate avoids the problem of oscillating back and forth between local minima.

On the other hand, the explanation from a machine learning perspective is that initially larger learning rates suppress memory for noisy data, while lowering the learning rate improves learning of complex patterns [25].

Another amplitude problem is that the training gradient decreases dramatically when approaching a local optimum or saddle point, making it difficult to train. A solution to this is to normalise the gradient, as shown in the figure

$$\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) - \eta \frac{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}(t))}{\|\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}(t))\|_2}$$

Figure 2.16: normalization [5]

In addition, the learning rate decay method is not flexible enough. It requires advanced planning of the learning rate variation method and with more hyperparameters to tune, making training more difficult. Adaptive Learning Rate Method, such as AdaGrad[26], RMSProp[6]. The RMSProp algorithm improves on the AdaGrad algorithm to obtain the following algorithm. On the one hand, the algorithm normalises the gradient; on the other hand, if the previous gradient is large, the algorithm makes the gradient smaller and, on the contrary, makes it larger, achieving the purpose of adaptivity.

$$\theta_j(t+1) = \theta_j(t) - \eta \frac{\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}(t))}{\sqrt{S_j(t+1) + \epsilon}}$$

where

$$S_j(t+1) = \beta S_j(t) + (1-\beta) \left(\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}(t)) \right)^2$$

and usually constant $\beta = 0.9$

Figure 2.17: RMSProp algorithm [6]

2.4.3 Solving the Direction Problems (Momentum & ADAM)

The inverse direction of the gradient is perpendicular to the contour (contour) of the function. As shown in the diagram, you can see that the global optimum is at the point in the direction of the gradient.

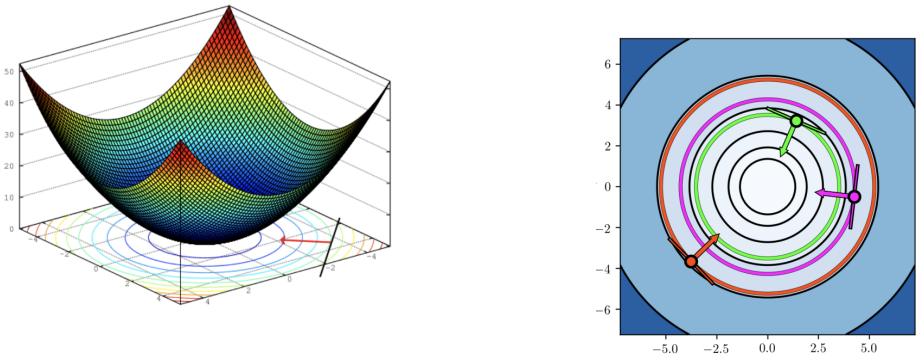


Figure 2.18: The sphere function and its contour [7]

However, the global optimum is not necessarily in the gradient direction for many functions. In this case, performing a gradient descent algorithm leads to problems of direction (zig-zagging).

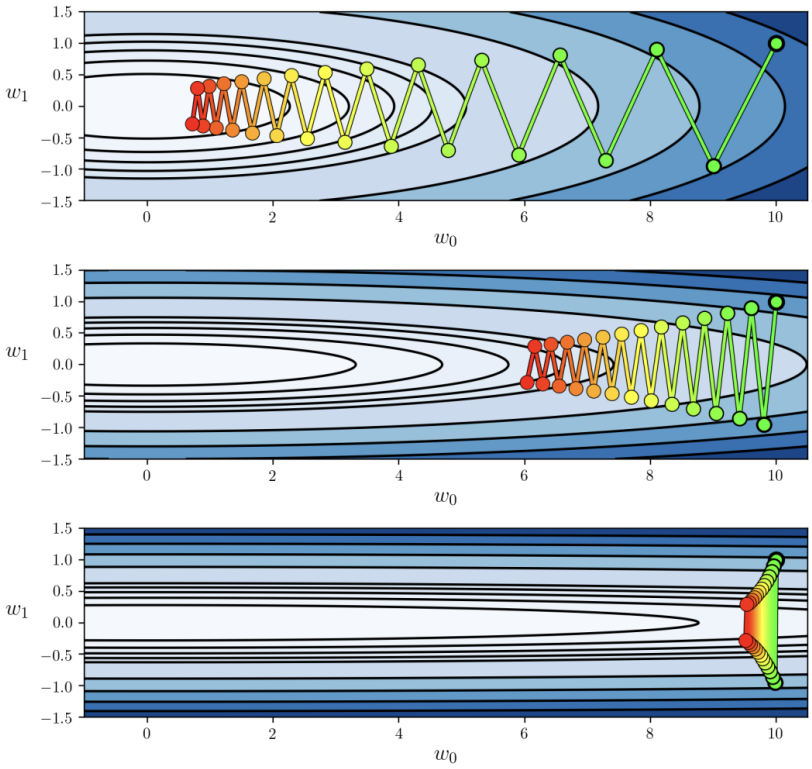


Figure 2.19: zig-zagging problem [7]

The negative gradient direction constantly points towards a contour perpendicular to the function. As the contours become increasingly parallel, the direction of the negative gradient changes rapidly during each run, resulting in a jagged training change.

One solution is to use Momentum, which combines the current gradient direction with the previous direction linearly, thus alleviating the zig-zagging problem.

$$\boldsymbol{v}(t+1) = \beta \boldsymbol{v}(t) - (1-\beta)\nabla_{\boldsymbol{\theta}} J(\hat{\boldsymbol{\theta}}),$$

where

$$\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}(t) + \beta \boldsymbol{v}(t)$$

Figure 2.20: Blue: Momentum [7]

An example is the NAG algorithm (Nesterov Accelerated Gradient) [27]:

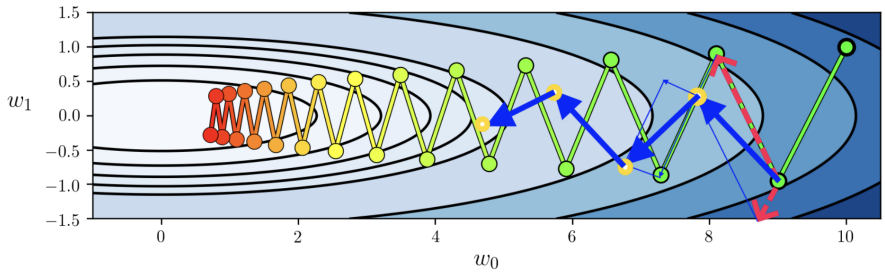


Figure 2.21: NAG [7]

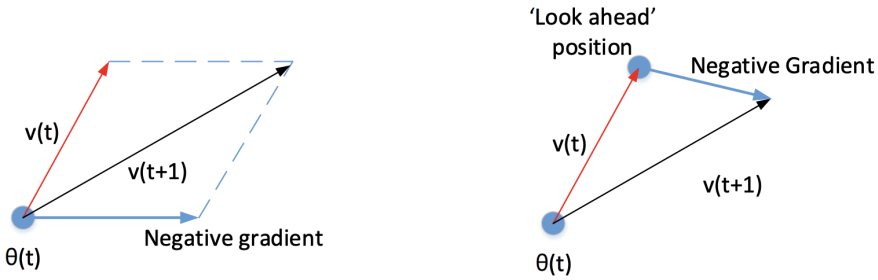


Figure 2.22: NAG [7]

Adaptive moment estimation [8], ADAM, is a very popular optimisation algorithm for deep learning today. The principle of ADAM is a combination of the previously described Moment and RMSProp algorithms, while largely addressing the direction and amplitude of training. The algorithm is shown in the figure.

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Figure 2.23: ADAM Algorithm [8]

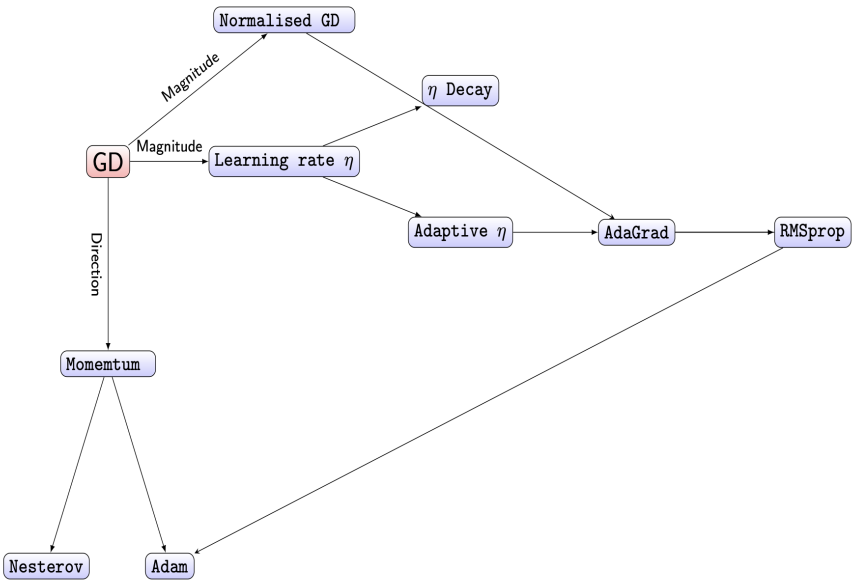


Figure 2.24: Conclusion of optimisation algorithms [7]

2.5 Convolution Neural network

It is easy to face memory problems when operating directly on images with neural networks. For example, a black and white image with $200 \cdot 200$ pixel points would equate to $40,000$ units in the first input layer, and if the second layer had the same number of units as the first layer, there would be $3.2e9$ learnable parameters. One solution is the convolutional neural network, which uses convolutional and pooling layers to extract image information.

Inspired by biometric process [28, 29, 30, 31], the Convolution Neural Network (CNN) was

proposed by Kunihiko et al. in 1989 [32]. The CNN consists of an input layer, a hidden layer, and an output layer, where the hidden layer contains a convolutional layer and a pooling layer.

Convolutional layer: The convolutional layer performs a matrix dot product of the input with a convolutional kernel, which is subsequently used as the feature output, similar to how neurons in the visual cortex respond to a particular stimulus.

Specifically, if the size of an input image is H, W and the size of the convolution kernel is k_h, k_w. Then the convolution operation is performed and the size of the image is $H - k_h, W - k_w$. where each pixel value of the output is: $output_{i,j} = \sum_{dx=-k_h}^{k_h} \sum_{dx=-k_w}^{k_w} k(dx + k_h, dy + k_w) \cdot I_{i+dx, j+dy}$

However, the generic convolution also includes the operations of padding, stride, and dilation. By far, the most commonly used are stride and padding.

stride: the step size of each move of the convolution kernel operation

padding: the expansion of the original image (by default adding 0 pixels to the image perimeter)

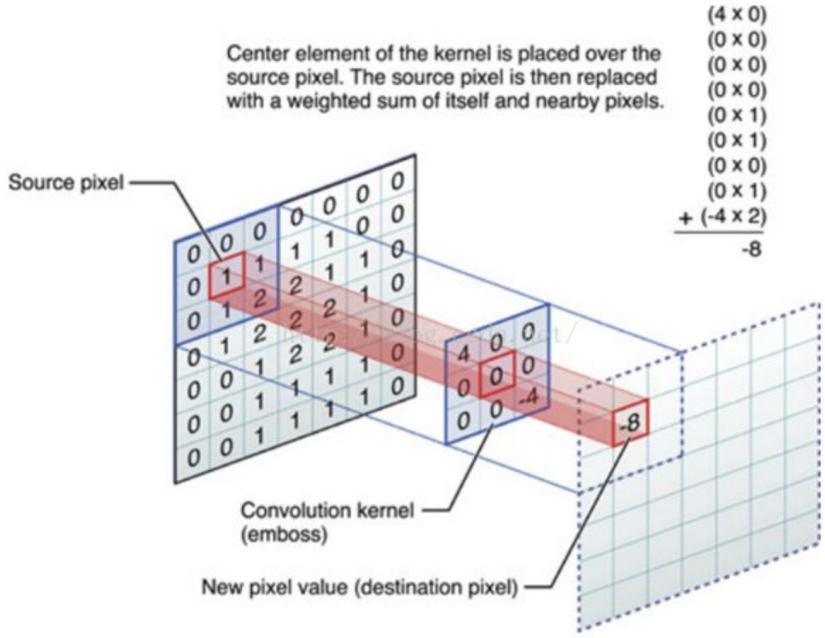


Figure 2.25: Example: Convolutional process

Pooling layers. Pooling layers are often used to reduce the dimensionality of an image, such as average pooling layers and maximum pooling layers.

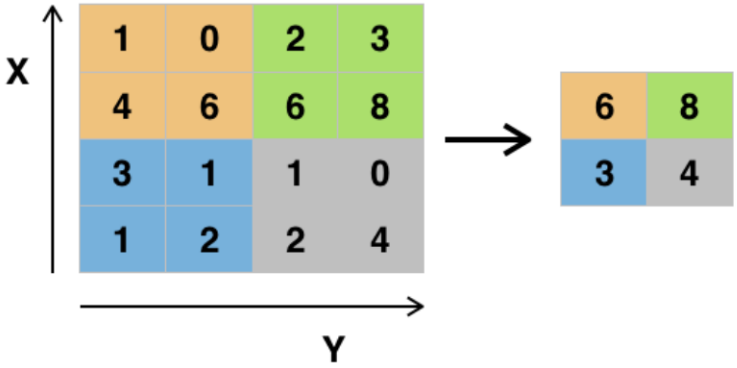


Figure 2.26: Example: max-pooling process

2.6 Auto-Encoder

The Autoencoder (AE) was first proposed by David et al. [33]. in 1985 and formalised as a neural network structure by Lecun et al. [34]. in 1987. AE is a completely unsupervised model in which the encoder part of the model extracts features z from the input, and the decoder part of the model uses features z to generate data. Today AE and AE-based models have become an important member of the neural network architecture and are used in areas such as generative tasks, dimensionality reduction, anomaly detection, etc.

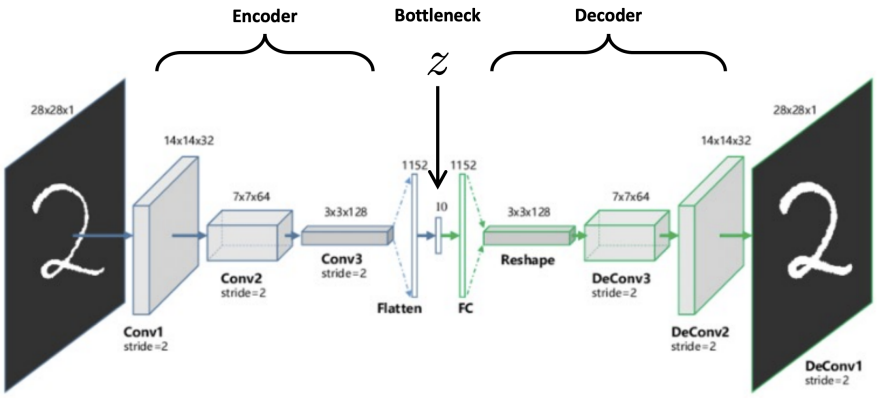


Figure 2.27: Example: Convolutional Auto-Encoder

Let the data be X, Encoder a function E and Decoder a function D. In training, the loss function:

$$l = \|X - D(E(X))\|_2$$

Because the final generation part uses only the feature layer information to decoder, the model can be trained to generate images using the feature latitude.

2.7 Batch Normalization

In training a neural network, it has a problem – Internal Covariate Shift, which is the change in the distribution of network activations due to the change in network parameters during training. Because of this problem, early neural network training is not very good at setting large learning rates and is prone to gradient explosion and vanishing problems.

Sergey et al. proposed the Batch Normalisation [35] algorithm, which is based on the principles of Momentum and normalisation, and largely solves this problem. The algorithm is as follows:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 2.28: Batch normalization

Chapter 3

Related Works

3.1 Generative Adversarial Network (GAN) and Common variants

3.1.1 GAN Framework

If the most important research in the field of artificial intelligence in recent years is to be found, Generative Adversarial Network (GAN) [11] is definitely one of them.

GAN is an unsupervised model that is trained by adversarially matching two sub-models. The input to the model is sampled from uniformly distributed noise, mapping the noise distribution to the data distribution. During training, the GAN throws the generated data and the real data to the discriminator for classification. The generative model is like a counterfeiter, whose aim is to create as many images as possible that are identical to the real data; the discriminator is like a tester, whose aim is to distinguish those things that are real as far as possible. The relationship between them is like playing a min-max game of game theory, where both sides adopt the optimal strategy and eventually reach a Nash equilibrium, where the counterfeiter makes things look like the real thing, and the tester is 50% accurate in distinguishing between the real and the fake.

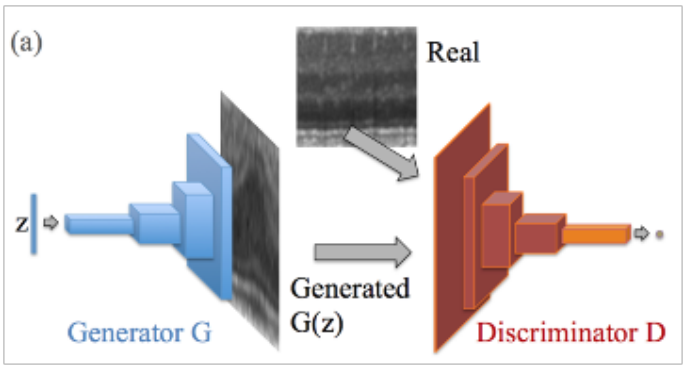


Figure 3.1: Example, a convolutional GAN [9, 10, 11]

We denote the data as x, the random input noise as z, the generating function as G and the

discriminating function as D.

The discriminator model outputs a value of 0 or 1. As mentioned above, the purpose of the discriminator is to distinguish as much as possible between what is true and what is false, so that:

For real data: its purpose is to identify the real data as 1 so that

$$\max_D E_{x\sim p_{\text{data}}(x)}[\log D(x)]$$

For generated false data: it aims to identify the generated data as 0 if possible, and such that:

$$\max_D E_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

The generation model aims to generate as much real data as possible so that the discriminator recognizes it as a real image, and such that:

$$\min_G E_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

In summary, the loss function of GAN can be defined as:

$$\min_G \max_D E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

The ability of GAN to do generative models stems from the conclusion that when a model satisfies the loss function optimum. If we find a global optimum, it will eventually make $p_g = p_{data}$ and the distribution of the generated data will be equal to the distribution of the training data.

Proof.

Let the loss function be:

$$V(D,G) = E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1-D(G(z)))] \tag{3.1}$$

$$V(D,G) = E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{x\sim p_g(x)}[\log(1-D(x))] \tag{3.2}$$

Then fix G, Find the optimal D:

$$\max_D V(D,G) = \max_D \int (p_{data}\log D + p_g log(1-D))\mathrm{d}x \tag{3.3}$$

$$\frac{\partial V(D,G)}{\partial D} = \frac{p_{data}}{D} + \frac{p_g}{1-D} \tag{3.4}$$

For $V(D,G)$, the optimal value must be at the point where the derivative is 0:

$$\frac{\partial V(D,G)}{\partial D} = 0, \tag{3.5}$$

$$D_G^* = \frac{p_{data}}{p_{data} + p_g} \tag{3.6}$$

Fix D_G^*, find optimal G:

$$\min_G V(D,G) = \min_G E_{x\sim p_{\text{data}}(x)}[\log \frac{p_{data}}{p_{data} + p_g}] + E_{x\sim p_g(x)}[\log \frac{p_g}{p_{data} + p_g}] \tag{3.7}$$

Converting this equation to KL, JS divergence (in fact, this step can also be used to directly infer what the optimal value is, but to link it more closely to later content, the equation is converted here)

$$KL(p_1||p_2) = E_{x\sim p_1} \log \frac{p_1}{p_2} \tag{3.8}$$

$$JS(p_1||p_2) = \frac{1}{2}(KL(p_1||\frac{p_1+p_2}{2}) + \frac{1}{2}(KL(p_2||\frac{p_1+p_2}{2}) \tag{3.9}$$

Since p_1, p_2 in KL divergence is a probability distribution, p1,p2 must lie between [0,1]. So $min_G(V,G)$ can be written as:

$$\min_G V(D,G) = \min_G E_{x\sim p_{\text{data}}(x)}[\log \frac{p_{data}}{(p_{data} + p_g)/2} * \frac{1}{2}] + E_{x\sim p_g(x)}[\log \frac{p_g}{(p_{data} + p_g)/2} * \frac{1}{2}] \tag{3.10}$$

$$\min_G V(D,G) = \min_G KL(p_{data}||\frac{p_{data}+p_g}{2}) - \log 2 + KL(p_g||\frac{p_{data}+p_g}{2}) - \log 2 \tag{3.11}$$

$$\min_G V(D,G) = JS(p_{data}||p_g) - \log 4 \tag{3.12}$$

Observe that the equation shows that $\min_G(V,G) >= -\log 4$

The minimum value $\log 4$ d is obtained only when $p_{data} = p_g$

3.1.2 Training Problems with GAN

After GAN[11] was proposed in 2014, many works followed it, such as the famous DCGAN[9]. But GAN is extremely unstable to train in applications, mainly because: if the discriminator is trained too well when GAN is trained, it can lead to difficult training of the generator[36], just like the tester is too good, which leads to the counterfeiter having difficulty finding positive feedback and not knowing what direction to work towards to improve skills.

This problem was also demonstrated mathematically by Arjovsky et al. in [36], when D reaches optimality, at which point the loss function is

$JS(p_{data}||p_g) - \log 4$

It can be seen that the training gradient arises from the JS divergence.

Now to analyse the JS divergence:

$$KL(p_1||p_2) = E_{x\sim p_1} \log \frac{p_1}{p_2} \tag{3.13}$$

$$JS(p_1||p_2) = \frac{1}{2}(KL(p_1||\frac{p_1+p_2}{2}) + \frac{1}{2}(KL(p_2||\frac{p_1+p_2}{2}) \tag{3.14}$$

$$JS(p_1||p_2) = \frac{1}{2}[\int (p_1 \log \frac{2p_1}{p_1+p_2} + p_2 log \frac{2p_2}{p_1+p_2})\mathrm{d}x] \tag{3.15}$$

$$JS(p_1||p_2) = \log 2 + \frac{1}{2}[\int (p_1 \log \frac{p_1}{p_1+p_2} + p_2 log \frac{p_2}{p_1+p_2})\mathrm{d}x] \tag{3.16}$$

Analysing this equation, if the p_1, p_2 probability distributions do not overlap, then we have the following situation when $p_1 = 0, p_2 = 0$, no significance When $p_1 = 0$, it is found that $p_1 \log \frac{p_1}{p_1+p_2} = 0; \log \frac{p_2}{p_1+p_2} = \log 1 = 0$ When $p_2 = 0$, same as situation $p_1 = 0$ So the two probability distributions do not overlap, which leads to $JS(p_1||p_2) = 0$

So when there is no overlap between p_g, p_{data}, there will be no gradient in the JS divergence.

Whereas in the real case, it is easy to have no gradient for the following reasons.

When the generative model is not well trained, pg and pdata are not particularly close to each other. The generated data is a sparse space of high latitude, which is difficult to overlap with the real data.

Even if the true distribution of the data overlaps, it is easy to assume that there is no overlap in the distribution of the sampled data, as the data is sampled each time the model is trained.

3.1.3 Two Tricks & WGAN-GP

The mathematical principles of GAN training instability were analysed previously. When D is trained too well, G is difficult to train because the generated data and the real data distribution hardly overlap, resulting in a loss function with almost no gradient in the generated model. There are two different angles of training techniques for these situations.

The first trick adds a large variance of noise to the real data and the generated data at the beginning. It gradually reduces the noise during the training process, so that the generated data and the real data distribution can overlap. This method solves the GAN training problem to a certain extent, but it also causes the model to learn about the noise.

Another trick is to reinitialise the parameters of D when the value is small when D is updated, so that it is trained from scratch. When G has no training gradient, it is easy for D to find the boundary line for classification, resulting in D has no gradient. With this in mind, it is possible to simply reinitialise D, resulting in D not being well trained, thus allowing G to continue to train. This method also solves the GAN training problem to some extent, but the intermediate training process is extremely unstable, and an additional hyperparameter needs to be adjusted, making training more

difficult.

The training instability of GAN stems from the unreasonable setting of JS divergence. Arjovsky et al. propose the Wasserstein distance instead of JS scatter based on GAN [36], and this loss function can avoid the problem of no gradient of JS.

Wasserstein distance:

$$W(p_{data}, p_g) = \inf_{\gamma \sim \pi(p_{data}, p_g)} E_{(x,y)\sim\gamma}[\|x-y\|] \tag{3.17}$$

where $\pi(p_{data}, p_g)$ refers to the joint distribution of p_{data}, p_g

In 2009 Villani proved [37] that the Wasserstein distance is equivalent to the following equation:

$$W(p_{data}, p_g) = \frac{1}{K} \sup_{\|f\|_{L<=K}} E_{x\sim p_{data}}[f(x)] - E_{x\sim p_g}[f(x)] \tag{3.18}$$

where $\|f\|_{L<=K}$ is the *Lipschitz* continuity such that the function satisfies $|f(x_1) - f(x_2)| \leq K|x_1 - x_2|$

Based on this, the WGAN[12] work proposes the technique of using clip to restrict all parameters to a small range. The final loss function is: $\min_G \max_D E_{x\sim p_{data}}[D(x)] - E_{x\sim p_g}[D(x)]$

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of priors.
    $g_w \leftarrow \nabla_w [\frac{1}{m} \sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Figure 3.2: Algorithm of WGAN [12]

However, WGAN has problems in difficulties in training and slow convergence.

In a subsequent work, Gulrajani et al. proposed WGAN-GP [3], which improved it. $\|f\|_{L<=K}$. When $K = 1$ *Lipschitz* continuity can be equated to $\|\nabla x D(x)\| \leq 1$

However, $\nabla x D(x)$ is also impractical to compute directly, so the authors decided to sample data distributed between p_g, p_{data} to compute the gradient. Let $\hat{x}$ be the randomly sampled data distributed between p_g, p_{data}, x being the real data and $\tilde{x}$ being the generated data. Then it can be sampled like this: $\hat{x} = \alpha x + (1-\alpha)\tilde{x}$,where α is a randomly generated number in $[0,1]$.

Based on these points, the authors designed the final loss function using a penalty term of:

$$L = E_{x\sim p_g}[D(x)] - E_{x\sim p_{data}}[D(x)] + \lambda E_{\hat{x}\sim p_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2] \tag{3.19}$$

Notice that in the penalty term, the gradient less than 1 also has a penalty value, the reason being that the authors wanted the gradient to be not too low during training. So the final algorithm is obtained as follows.

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.
while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0,1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1-\epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^m L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^m \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^m -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Figure 3.3: Algorithm of WGAN [12]

3.2 GANomaly

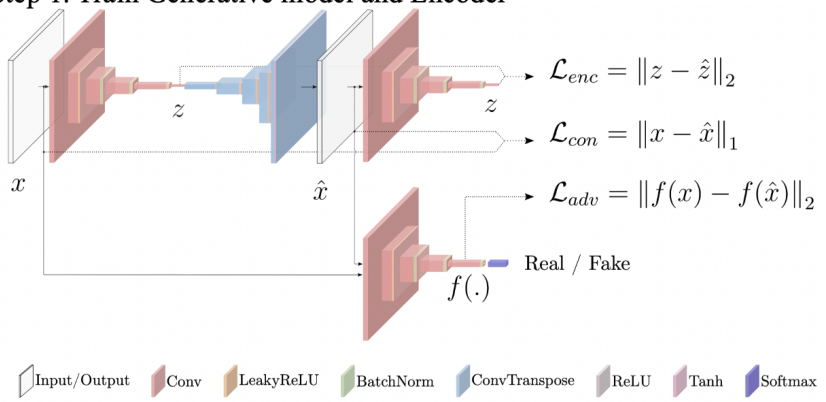
Akcay et al. proposed GANomaly [13] in 2018, which is an influential work that has now been cited in 764 articles. This model is now performing at a Stae-of-the-art level in many anomaly detection tasks, so much so that many current anomaly detection tasks are compared to it, such as the ProxyAno work in the field of medical imaging [38].

GANomaly is trained on normal images only, with the aim of allowing the network to learn the distribution of normal data and the distribution of its features. As shown in the figure below, the model extracts the feature representation space of an image through the first Encoder and subsequently maps the feature representation space to the normal data distribution through the Decoder. The purpose of the second encoder of the model is to learn the inverse function of the decoder to map the generated data into the feature representation space. In addition, the model is trained against the generative model via discriminator, so that the model learns how to map the feature representation space distribution to the data space distribution.

For a test image, the network defines the distance between the feature representations extracted by the two encoders as the anomaly score. Since its network only learns how to generate the spatial distribution of a normal image, the feature representation for an abnormal image will be very different from that of the generated normal image feature representation, so the abnormal image will have a higher abnormal score.

- GANomaly

Step 1. Train Generative model and Encoder



Step 2. Inference

Test *sample $\hat{x}$ within the test dataset D.* $\mathcal{A}(\hat{x}) = \|G_E(\hat{x}) - E(G(\hat{x}))\|_1$

for a test sample $\hat{x}$, anomaly score:

Anomaly scores $S = \{s_i : A(\hat{x}), \hat{x} \in D\}$

To evaluate the overall anomaly performance, apply feature scaling to have the anomaly scores within the probabilistic range of [0, 1]

$$s'_i = \frac{s_i - min(\mathcal{S})}{max(\mathcal{S}) - min(\mathcal{S})}$$

Figure 3.4: Algorithm of GANomaly[13]

3.3 Self-supervised Denoising via Stein's Unbiased Risk Estimator

In medical imaging, most of the observed images contain noise. For example, an image contains Gaussian noise.

$$X \sim \mathcal{N}(\mu, I\sigma^2).$$

Where u is the image without noise, sigma is the standard variance of the Gaussian noise and I is the unit matrix. Let $h(x)$ be an estimator of u from x. This can be written as $h(x) = x + g(x)$, where $g(x)$ is weakly differentiable. So, we expect to make the following equation be as optimal as possible:

$$MSE(h) = E_u \|h(x) - \mu\|^2$$

But where u is not known, and it is not possible to make direct use of this loss function. In practice, however, if the noise distribution is Gaussian, it is possible to use SURE-based loss to achieve the same effect as the desired loss function [14].

$$E_u(SURE(h)) = MSE(h), \tag{3.20}$$

$$SURE(h) = -d\sigma^2 + \|g(x)\|^2 + 2\sigma^2 \nabla h(x) \tag{3.21}$$

Proof:

We wish to show that

$$\mathrm{E}_\mu \|h(x) - \mu\|^2 = \mathrm{E}_\mu\{\mathrm{SURE}(h)\}.$$

We start by expanding the MSE as

$$\begin{aligned}\mathrm{E}_\mu \|h(x) - \mu\|^2 &= \mathrm{E}_\mu \|g(x) + x - \mu\|^2 \\ &= \mathrm{E}_\mu \|g(x)\|^2 + \mathrm{E}_\mu \|x - \mu\|^2 + 2\,\mathrm{E}_\mu\, g(x)^T (x - \mu) \\ &= \mathrm{E}_\mu \|g(x)\|^2 + d\sigma^2 + 2\,\mathrm{E}_\mu\, g(x)^T (x - \mu).\end{aligned}$$

Now we use integration by parts to rewrite the last term:

$$\begin{aligned}\mathrm{E}_\mu\, g(x)^T (x - \mu) &= \int_{\mathbb{R}^d} \frac{1}{\sqrt{2\pi\sigma^{2d}}} \exp\left(-\frac{\|x - \mu\|^2}{2\sigma^2}\right) \sum_{i=1}^{d} g_i(x)(x_i - \mu_i) d^d x \\ &= \sigma^2 \sum_{i=1}^{d} \int_{\mathbb{R}^d} \frac{1}{\sqrt{2\pi\sigma^{2d}}} \exp\left(-\frac{\|x - \mu\|^2}{2\sigma^2}\right) \frac{dg_i}{dx_i} d^d x \\ &= \sigma^2 \sum_{i=1}^{d} \mathrm{E}_\mu \frac{dg_i}{dx_i}.\end{aligned}$$

Substituting this into the expression for the MSE, we arrive at

$$\mathrm{E}_\mu \|h(x) - \mu\|^2 = \mathrm{E}_\mu \left(d\sigma^2 + \|g(x)\|^2 + 2\sigma^2 \sum_{i=1}^{d} \frac{dg_i}{dx_i} \right).$$

Figure 3.5: Proof [14]

So, Define h_θ as a denoiser network.

$$L_{SURE}(\theta) = \|x - h_\theta(x)\|^2 - \sigma^2 + 2\sigma^2 \nabla \cdot h_\theta(x) \tag{3.22}$$

It is impractical to calculate $\nabla \cdot h_\theta(x)$ directly. However, Ramani et al. proposed a method [39] to estimate the gradient.

$$\nabla \cdot h_\theta(x) = \lim_{\tau \to 0} E_b\left(\frac{1}{\tau} b^T (h_\theta(x + \tau b) - h_\theta(x))\right) \tag{3.23}$$

where b is a zero-mean $i.i.d.$ random vector with unit variance and bounded higher order moments.

3.4 Unsupervised Anomaly Detection

Unsupervised Anomaly Detection (UAD) has Representation-based, Statistical and Deep Learning Methods. Among them, deep learning is mainly divided into Discriminative, Generative and Knowledge Distillation-based methods. In the Discriminative method, Oza et al. propose a deep learning-based classifier inspired by One Class SVM[40]. Ruff et al. propose a deep SVDD framework [41] trains a deep representation with the goal of containing positively labelled data embeddings with the smallest hypersphere possible. One of the most prominent strategies for UAD is generative models. Denoising auto-encoders, generative adversarial networks, auto-regressive models, adversarial auto-encoders, and other methods are used in these models [10, 42, 43, 44, 45, 46, 47, 48]. A few contemporary strategies have combined knowledge distillation with student-teacher model training [49, 50].

In UAD on medical imaging, the use of adversarial learning for anomaly detection was first proposed by Schlegel et al. in AnoGAN [10], but their method required several iterations to find the feature layer z during detection. With the theoretical development of GAN, Schlegel et al. continued to carry out improvements to this work and proposed F-AnoGAN [10] in 2019. On the one hand, based on the WGAN-GP approach, the network training is more stable, and on the other hand, the work makes use of an encoder to act as an inverse function of the generative model, which greatly improves its detection efficiency. However, its residual values based on the original image and the real image are not suitable for anomaly detection on some medical images, such as anomalies in lung images, for example, αGAN [51] proposed by Nakao et al. in 2021 tried to use this method for anomaly detection, but the results obtained were not particularly good. Also, in 2021, the RANDGAN [48] work was based on a GAN model for anomaly detection, and in this work, they found that if the information from the lung segmentation was used for anomaly detection, it would be better than not. However, the model in this work is based on a semi-supervised approach, and its training data contains anomalous samples.

Chapter 4

The Proposed Algorithm

Inspired by GANomaly [13] proposed by Akcay et al., which is an influential work cited by 751 papers, I proposed a new novel model and novel loss.

During training, my model aims to learn how to generate a healthy normal lung image from an input image by an ROI generator. It is only trained on healthy and normal samples and learning the feature representations in latent space. During the test given new images, for the abnormal samples, the abnormal features will be removed by the first stage decoder, and we can detect anomalies by comparing the corresponding latent features.

4.1 Definition

We denote the training dataset by X are normal images, M is the corresponding lung segmentation mask, and only the ROI part of the training dataset is retained as Y, $(Y = X \cdot M)$. As shown in the figure 4.1, E_1 is the first Encoder, which takes the input image and extracts the feature representation Z^{full}. De in the model is the Decoder, which takes the extracted feature representation Z^{full} and generates $\hat{Y}$ containing only the ROI part, which is trained so that the generated data distribution is only as close as possible to the data distribution of Y, or lie on manifold Y. E_2 in the model is the second Encoder. Dc is the Discriminator, which distinguishes whether the input image is a real image or a generated image.

The test dataset $\hat{X}$ contains both normal and abnormal images with pathology. The aim of the trained model is to distinguish between normal and abnormal images in the test dataset in a reasonable way.

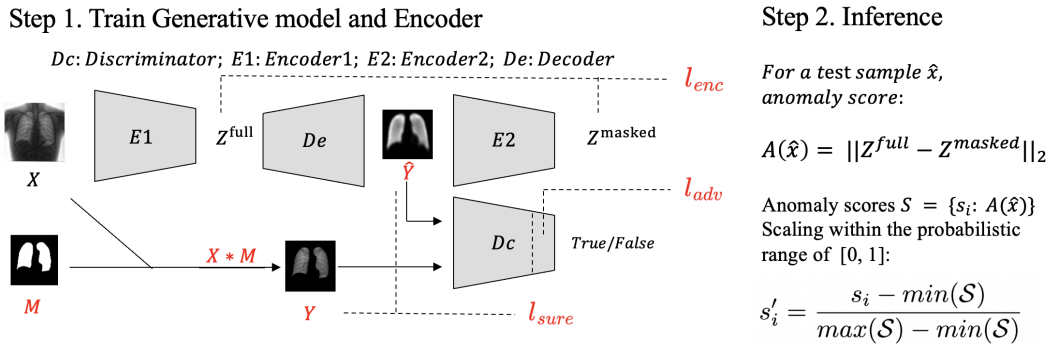


Figure 4.1: The proposed algorithm

4.2 Architecture

The model is divided into three parts, the first part is the ROI Generator, where the Encoder extracts the features representation space of the healthy image and the Decoder generates the ROI of the image. The second part is the Inverse model. It uses the encoder to learn the inverse process of generating images to feature representation space and to ensure that the encoder in the first part focuses only on the ROI. The third part is the discriminator, it helps the ROI Generator to generate data lie on the learned normal data manifold.

4.2.1 ROI Generator

The ROI generator consists of two parts: the autoencoder and the data processing. The autoencoder consists of an encoder E_1 and a decoder De to extract the feature representation space of the image and generate the image from the feature representation space with only the ROI in the image retained, respectively. Inspired by the structure of DCGAN [9], Encoder uses a similar structure to DCGAN, consisting of a number of blocks, the first of which consists of a convolutional layer and a Leaky ReLU activation function, and each of the intermediate blocks consists of a convolutional layer, Batch normalization layer and Leaky ReLU activation function, with the last layer being output as a bottleneck feature through the convolutional layer. The decoder uses a completely symmetrical structure to Encoder, except for the last block, where each block consists of a deconvolution layer, Batch normalisation layer and Leaky ReLU layer, and the last block consists of a deconvolution layer and Tanh activation function.

The data processing part makes use of the information from the lung segmentation mask, the image X with mask M for dot product preserving ROI from the image as a result $Y = X \cdot M$. The purpose of this part is to use the processed data Y as ground truth for training the autoencoder.

4.2.2 Inverse Model

The inverse function model consists of an Encoder E_2, designed to extract the feature representation space of the generated image and to help train the Encoder E_1 of the Generator to focus more on

the ROI region of the image. Encoder E_2 has exactly the same structure as Encoder E_1, but the parameters are not shared so the two encoders are separate and independent models.

4.2.3 Discriminator

The discriminator consists of a classifier that identifies whether an image is real or generated, and based on the theoretical basis of adversarial training, its purpose is to help train the generator and make it generate an image $\hat{Y}$ lie on manifold Y that has the same data distribution as Y which are healthy images. Except for the last layer of the discriminator, the structure of it is exactly the same as the encoder. The discriminator has an addition of a Sigmoid as the activation function in the last layer, which maps the final output to the probability of the image being real or fake.

4.3 Loss function

The generative loss function is weighted by the l_{adv} , l_{enc} and l_{sure} components:

$$l_{gen} = l_{adv} * w_{adv} + l_{enc} * w_{enc} + l_{sure} * w_{sure} \tag{4.1}$$

Where w_{adv}, w_{enc} and w_{sure} are three hyperparameters.

In order to use the discriminator to assist in the training of the ROI generator, the discriminator is also trained at the same times as the other components of the model, so that the loss function of the discriminator is l_{Dc}.

4.3.1 Adversarial Learning

l_{adv} enforces the generated image to lie on the learned manifold Y, which are the healthy normal lung images.

$$l_{adv} = \|f(Y) - f(\hat{Y})\|_2 \tag{4.2}$$

l_{adv} calculates feature matching loss for adversarial learning. Feature matching has been shown by Salimans et al. [52] to reduce the instability of GAN training.

As mentioned in section 3.1 before, l_{Dc} proposed by Goodfellow et al. [11].

$$l_{Dc} = 0.5 \cdot BCE(Dc(\hat{Y}), 0) + 0.5 \cdot BCE(Dc(Y), 1). \tag{4.3}$$

Where, $BCE(a,b) = -[b \cdot \log a + (1-b) \cdot \log(1-a)]$.

4.3.2 More focus on Pathological ROI

This part aims the l_{enc} aims to learn the inverses function of $De(Z^{full})$ in normal data. It enforces the model to focus on the lung part of the images, which is pathological ROI as mentioned.

It calculates the $l2$ distance between Z^{full} and Z^{masked}.

$l_{enc} = \|Z^{full} - Z^{masked}\|_2$

4.3.3 SURE-based Consistency

Inspired by SURE framework [14] as mentioned in section 3.3, l_{sure} enforces the visual similarity between Y and $\hat{Y}$. And make the model more robust to noise.

$$l_{sure} = \|Y - \hat{Y}\|_2 - \sigma^2 + \frac{2\sigma^2}{m} b * (De(E_1(X + \tau bM)) - \hat{Y}) \tag{4.4}$$

Where $b \sim \mathcal{N}(0, I)$, τ is a small fixed number.

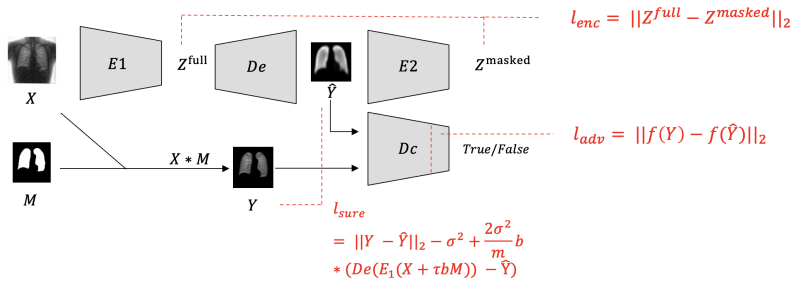


Figure 4.2: The proposed algorithm

4.4 Inference

After training, the generator is constrained to generate healthy images, even though the original scan is abnormal.

For abnormal pathological images, the data distribution and feature space distribution are very different from the normal image. The Decoder in ROI Generator has learned how to generate the normal data distribution from feature representation space, thus it can remove the pathological feature and generate normal images. Because the input of Encoder in Inverse Model are the images from ROI Generator, which are normal healthy images. Therefore, the Encoder in Inverse Model will only have normal feature representations. When the model detects anomaly images, the distance between z^{full} and z^{masked} will be larger than detecting normal images. Finally, the anomaly score will be larger than the detection normal.

Based on this, the model uses the L2 distance of the difference between the two features to define the anomaly detection score.

$$A(\hat{X}) = \|Z^{full} - Z^{masked}\|_2 \tag{4.5}$$

To facilitate the evaluation, I have normalised the anomaly score to the interval $[0,1]$ as the probability score.

Anomaly scores $S = \{s_i : A(\hat{x})\}$.

$$S'_i = \frac{s_i - \min(S)}{\max(S) - \min(S)} \tag{4.6}$$

Chapter 5

Experiments & Results

In this Chapter, I compared the proposed algorithm with three different learning strategies: vanilla GANomaly given by network and loss in section 3.2, the proposed algorithm and an oracle proposed algorithm trained with noiseless loss that replaces l_{sure} with $l_2 = \|Y - \hat{Y}\|_2$. Both of these aim to ensure visual similarity on the generated images, but l2 loss is not an unbiased estimation and is more sensitive to noise. The oracle-proposed algorithm provides a way to evaluate the separate contributions of network and loss.

I emphasize that the experiments here are not intended to demonstrate the need to achieve state-of-the-art levels. The experiments are a proof of concept, and the results are obtained without heavy hyperparameter tuning.

5.1 Dataset

I choose the M&S dataset in my experiment, which is a small dataset. A small dataset is very useful for a proof of concept study, like this work. In this dataset, all images are well aligned. It also has a manually segmented lung mask, which is useful for humans to judge but rarely for training anomaly detection models.

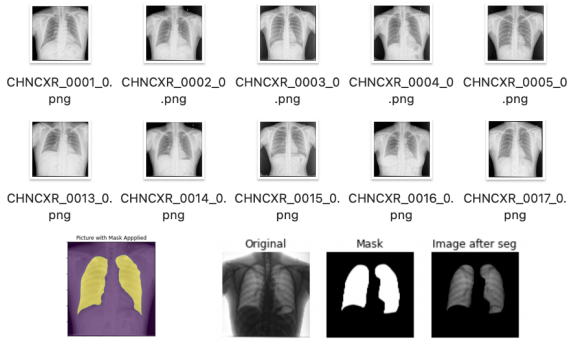


Figure 5.1: M&S dataset

The M&S dataset [53, 15] have been collected in USA and China. It contains 800 images in total, of which 408 are normal cases, and 394 are cases with manifestations of TB, including

manually segmented lung masks.

In the experiment, the normal images were randomly split to 9:1 for training and validation. I also randomly choose the same number of abnormal images as the validation dataset. Consequently, my dataset in experiment as table 5.1

Label	Train	Val
Normal	311	35
Abnormal	0	35

Table 5.1: Experiment dataset

5.2 Setup and Implementation

For the proposed algorithm and the proposed network structure, the encoder, decoder and discriminator used the structure mentioned in 4.2. All convolution layer kernel size is 4. In encoder and discriminator, if the convolutional layer is last, the stride is 1, and padding is 0. Otherwise, stride is 2, padding is 1. Depending on the size of the image, the number of the hidden block would change accordingly. For GANomaly I used the same network in [13]. All convolutional layers in the model were initialised to have mean 0 and variance 0.02, while the Batch normalisation layers were initialised to have mean 1 and variance 0.02.

In training, all training and test images were resized to $(256, 256)$ without data augmentation. In each training session, I first update the parameters of the training generator and subsequently update the parameters of the training discriminator. To avoid the problem of difficult GAN training (as mentioned in section 3.1), I used the reinitialisation trick, and reinitialised the discriminator when its weights were small. Following the work of DCGAN [9] and GANomaly [13], $w_{enc}, w_{sure}, w_{adv}$ were set $1, 50, 1$ respectively, and the generated models and discriminators were trained using the Adam optimiser.

5.3 Results

In this experiment, I used Receiver Operating Characteristics (ROC) curve to report the performance of models and residual images to analyse models' generative performance and anomalies. The ROC curve represents the relationship between the false positive rate (FPR) and the true positive rate (TPR) obtained by a classifier. They are defined as:TPR = (Number of correctly classified positive samples)/(Number of positive samples), FPR = (Number of misclassified negative samples)/(Number of negative samples). And the area under the ROC (AUC-ROC rate) is a commonly used metric to evaluate anomaly detection. Consequently, I use it to compare different models.

During training, GANomaly typically achieves its best performance on the validation dataset AUROC at around 20 epochs. Through experimental observation and analysis, GANomaly overfitting occurs as the number of training rounds increases, and learns a situation that we do not want to happen: when the network generates as many images as possible with whatever input it has. Also

for the two encoders in GANomaly, in this case, it is easy for them to learn the same parameters, which allows the network to have a very low loss function. However, such a case is not suitable for anomaly detection, and we would prefer that GANomaly would only learn to map out the distribution of normal images based on the features.

My proposed algorithm, on the other hand, does not allow the above situation to occur in my network due to the fact that the images to be generated are different from the network input images, and the purpose of the $loss_{sure}$ is to make the visual similarity between the network input images and the generated images. On the other hand, in my proposed algorithm the first encoder input is an image with all the information, and the second encoder input is a generated image with only the ROI retained, during the training process $loss_{enc}$ wants the features extracted by both encoders to be similar, so the first encoder needs to have the ability to ignore the non-ROI region information features. Therefore, the parameters of the two encoders must not be similar.

Table 5.2 shows that my method is close to the performance of GANomaly. In addition, my method performs better in the case of unbiased estimation loss. This also confirms that the method performs better against noise.

Model	Best AUROC
GANomaly	0.9061
Mywork$_{Oracle}$	0.8755
Mywork	0.8987

Table 5.2: Experiment dataset

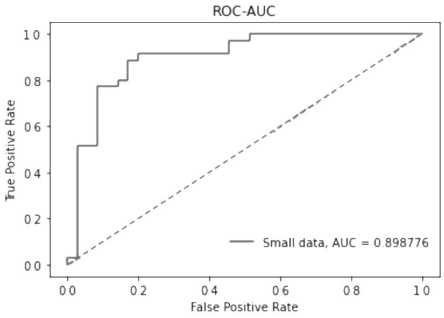


Figure 5.2: The proposed algorithm best AUROC

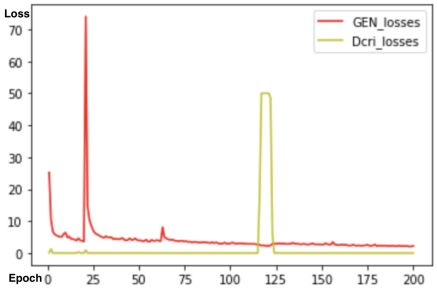


Figure 5.3: The proposed algorithm loss function of the generator model and discriminator with the number of training epoch

Chapter 6

Critical & Reflective Thinking

In this section, I would like to discuss a number of issues, as well as a critique of current work and some future perspectives

6.1 Discussion

Since the proposed only trains on “normal” images, will any image that is sufficiently different from the training set automatically be considered ”abnormal”?

As data imbalance mentioned in the introduction, this is a very important challenge for unsupervised anomaly detection as a whole. If the normal data is not collected sufficiently, it can affect the performance of the model.

However, the algorithm I designed is motivated by this pipeline Figure 6.1. We can use the algorithm to set up a highly sensitive filter to filter out the absolutely normal images, thus already relieving the burden on the radiologist. As for the images that are not filtered out, they are still given to the radiologist. If the size of normal data is not sufficiently large, although it will affect the filtering efficiency, it will still relieve the radiologist’s burden as the unfiltered images are still handed over to the radiologist, so there is no problem of AI misdiagnosis due to insufficient data collection.

Of course the more adequate the data collection, the more helpful it will be for model training. But we don’t need to completely solve the work problems faced by radiologists all at once.

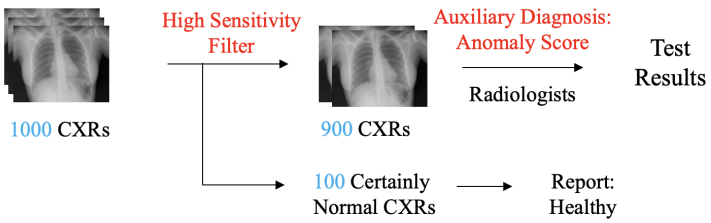


Figure 6.1: The proposed pipeline

6.2 Critical thinking

6.2.1 More Robust for Poisson-Gaussian Noise

Another common noise in CXRs is Poisson noise, and there are works on Poisson, Gauss-Poisson hybrid noise in the SURE framework [54, 55, 56] in recent years. Further work can be done to improve the work on mixed Gauss-Poisson noise.

6.2.2 Proof for GANomaly

As mentioned in Chapter 3, GAN can learn the global minima $p_{data} = p_g$ with a condition p_g generate from a fixed gaussian noise which is an Independent and identically distributed random variables (i.i.d). But GANomaly generates images from a features representation in latent space from the encoder. If the input is out-of-distribution, it is unknown why this method works. It can do some proof study in future.

6.2.3 Another Dataset

The proposed algorithm did not perform well on other datasets. Not only my method, but also GANomaly and F-AnoGAN (see the appendix for their related experiments). My initial guess is that the experimental dataset was not properly calibrated, resulting in no aligned images and ultimately making the model perform poorly. There is still much to be explored in future.

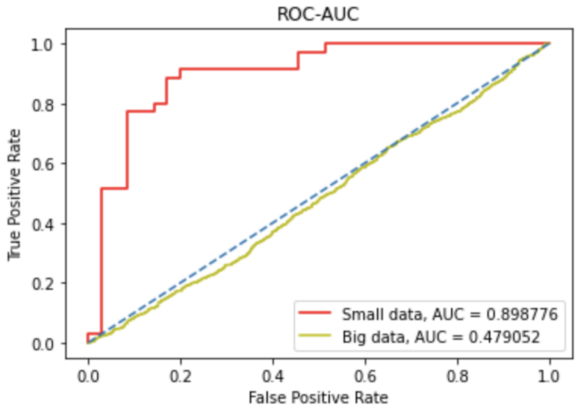


Figure 6.2: Low AUROC in NIH dataset [15]

Chapter 7

Project Management

The project started out based on the proposed pipeline, with plans to compare several different anomaly detection networks and discuss them, making a GUI (graphical user interface) to help radiologists at the end of the project. Almost all of these goals were accomplished by the middle of the project. And in an unexpected moment of reflection in early August, I came up with the prototype of the proposed algorithm and communicated with my supervisor to eventually decide to change the project to a full research-based project.

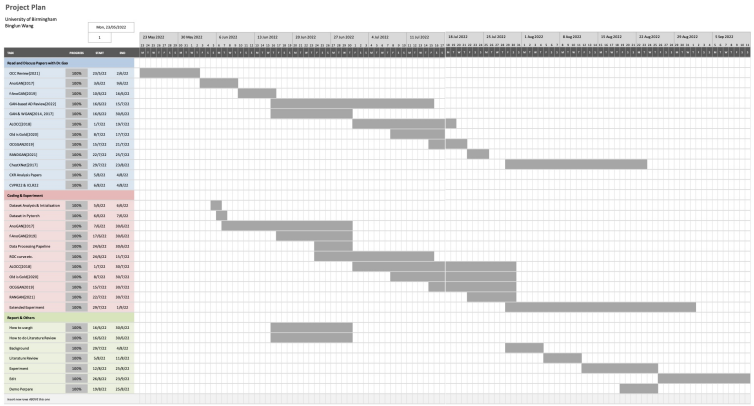


Figure 7.1: Project Management

I met with my supervisor Dr Gao once a week for up to an hour to discuss one paper in depth at a time and to report on the week's progress and subsequent plans. In addition to the meetings, I also kept in close contact with my supervisor. On the other hand, I spend an average of about 40 hours a week on the project, with an average of 15 hours on experiments.

All my experiments are carried out in Colab and on my own rented server. The main experiments are on Colab and are recorded as follows.

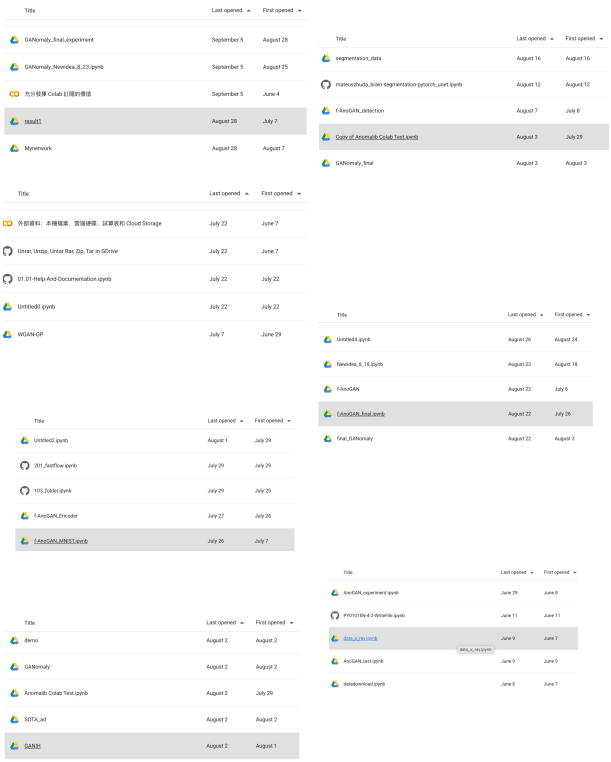


Figure 7.2: Recording of Colab

Chapter 8

Conclusions

This work proposes a pipeline and a novel anomaly detection algorithm that can greatly improve radiologists' work efficiency. Due to the unpredictability and difficulty in acquiring abnormal samples, the proposed algorithm is only trained by healthy images and can give anomaly probability for a test image. Compared to previous works, my method makes two important contributions. First, the proposed algorithm is more focused on pathological information due to the novel architecture and loss function. Second, the proposed algorithm uses Stein's Unbiased Risk Estimator (SURE) to make the model more robust to noise. Extensive experiments on the Montgomery dataset and Shenzhen dataset show that this work has state-of-the-art performance.

Appendix A

Other Works

I compared F-AnoGAN [47], GANomaly [13], and a pipeline in my experiments using UNET[57] to segment the images and then used the Anomaly Detection model to detect the remaining images.

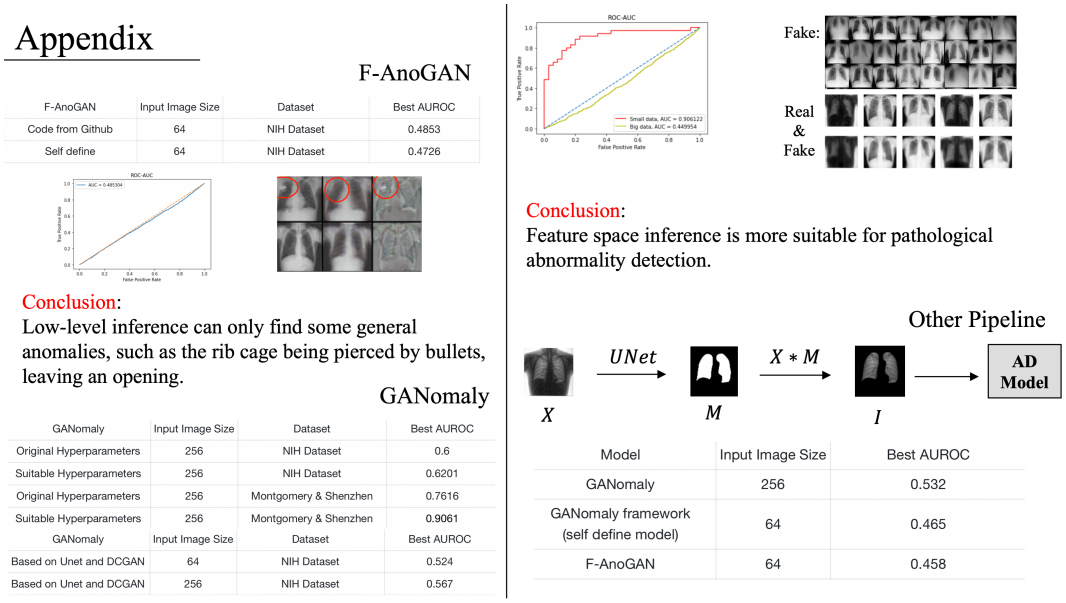


Figure A.1: Other works

Some experiments were based on another dataset, the NIH dataset [58], in which I randomly selected 13203 images. The training dataset has 10803 healthy images. And The test set consisted of 1200 healthy images and 1200 abnormal images.

In F-AnoGAN as mentioned in Chapter 3 Related Works, I used the improved WGAN training procedure [59] for stable GAN training, where the generator and the discriminator as the same as DCGAN network [9]. The encoder is implemented by an architecture similar to the generator's architecture. I find the image inference can find general anomalies, like the red circle in the figure. But it is not suitable for detecting pathological abnormalities. And training F-AnoGAN requires a lot of data.

In GANomaly, I set the first convolutional layer channels as 64 in the encoder and decoder

cause channels in CXRs are 1, it does not need as much expansion as 3 channel images in the Original network. Following Akcay et al. (2018) [3], I set z-space with 100 dimensions. The $l_{enc}, l_{con}, l_{adv}$ fractional weight are 1:50:1. I used the re-initialisation as mentioned for stable GAN training. It can find Feature space inference more suitable for pathological abnormality detection. And detected images will achieve better results if they have registration (aligned).

And I also replace the first encoder and decoder with a residual UNet and used the improved WGAN training procedure [59] for stable GAN training. However, the experimental results are not good enough compared to the original network. After around 30 epochs the AUROC are still equal to 0.5. Because UNET is residual, it can learn to output what image is input. In this way, the loss of con and adv will be reduced to a very low level, and the second encoder will also learn the same parameters as the first encoder, so that the loss of enc will be the lowest. So, after long-term training, the model can easily learn that the two encoders have similar weights. This result helped me to find the problem of GANomaly as mentioned in Chapter 3.

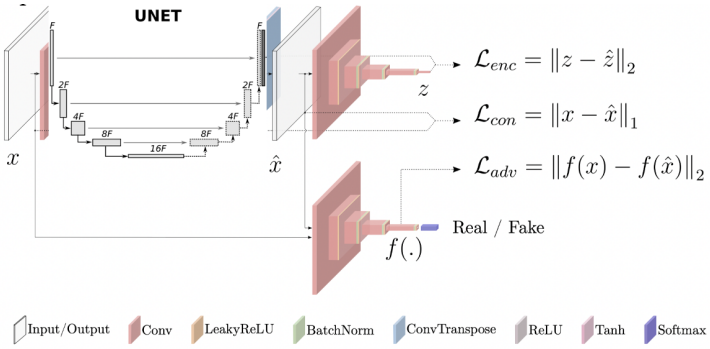


Figure A.2: The residual unet GANomaly

In the pipeline experiments, I suspect that it is not appropriate to use this top-down approach directly for anomaly detection, but the relevant experiments are not sufficient, and subsequent work can continue to verify this.

Appendix B

How to use code

To run the experiments which were developed in this research, access the GitLab repository using the below link:

`https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2021/bxw135`

All the data has been processed and uploaded to Kaggle. If you want to try the proposed algorithm code, just run Final_code.ipynb in the Final_code folder and do the following.

1. Run the first line of code uncommented and download the dataset. To download the dataset you will need to provide the Kaggle platform account and Kaggle key on the command line. If you don't have one, go to `https://www.kaggle.com`.

2. As shown in the Figure below B.1, change the address of the dataset in Preprocess.

3. As shown in the Figure below B.2, modify all hyperparameters in the first block of code in TRAIN.

```
NORMAL_IMAGE_PATH = '/root/ms-dataset/content/segdata/normal/image'
NORMAL_MASK_PATH = '/root/ms-dataset/content/segdata/normal/mask'
NORMAL_I_M_PATH = '/root/ms-dataset/content/segdata/normal/segimage'
ABNORMAL_IMAGE_PATH = '/root/ms-dataset/content/segdata/abnormal/image'
ABNORMAL_MASK_PATH = '/root/ms-dataset/content/segdata/abnormal/mask'
ABNORMAL_I_M_PATH = '/root/ms-dataset/content/segdata/abnormal/segimage'
```

Figure B.1: dataset address

```
# Hyperparameters etc.
device = "cuda" if torch.cuda.is_available() else "cpu"
LEARNING_RATE = 0.0002
BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS_IMG = 1
Z_DIM = 100
NUM_EPOCHS = 70
FEATURES_DCRI = 64
FEATURES_GEN = 64
PRE_PATH = ['result/GANomaly/', 'result/Myidea/', 'result/GANomaly_sure/', 'result/Myidea_sure/']
PATH = ['GANomaly.pth', 'Myidea.pth', 'GANomaly_sure.pth', 'Myidea_sure.pth']
W_ADV = 1
W_CON = 50
W_ENC = 1
```

Figure B.2: hyperparameters

References

[1] Xuan Xia, Xizhou Pan, Nan Li, Xing He, Lin Ma, Xiaoguang Zhang, and Ning Ding. Gan-based anomaly detection: A review. *Neurocomputing*, 2022.

[2] University of Birmingham. Neural computation, 2021.

[3] Mustafa Hajij and Kyle Istvan. A topological framework for deep learning. *arXiv preprint arXiv:2008.13697*, 2020.

[4] Paperswithcode.com. Papers with code - an overview of activation functions. [online] available at: ¡papers with code - an overview of activation functions¿ [accessed 15 september 2022]., 2022.

[5] Ryan Murray, Brian Swenson, and Soummya Kar. Revisiting normalized gradient descent: Fast evasion of saddle points. *IEEE Transactions on Automatic Control*, 64(11):4818–4824, 2019.

[6] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

[7] University of Birmingham. Mathematical foundations in artificial intelligence and machine learning, 2021.

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[10] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International conference on information processing in medical imaging*, pages 146–157. Springer, 2017.

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[12] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[13] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Asian conference on computer vision*, pages 622–637. Springer, 2018.

[14] Charles M Stein. Estimation of the mean of a multivariate normal distribution. *The annals of Statistics*, pages 1135–1151, 1981.

[15] Sergii Stirenko, Yuriy Kochura, Oleg Alienin, Oleksandr Rokovyi, Yuri Gordienko, Peng Gang, and Wei Zeng. Chest x-ray analysis of tuberculosis by deep learning with segmentation and augmentation. In *2018 IEEE 38th International Conference on Electronics and Nanotechnology (ELNANO)*, pages 422–428. IEEE, 2018.

[16] Jeremy Page, Drew Hinshaw, and Betsy McKay. In hunt for covid-19 origin, patient zero points to second wuhan market–the man with the first confirmed infection of the new coronavirus told the who team that his parents had shopped there. *The Wall Street Journal*, 2021.

[17] World Health Organization. Coronavirus (covid-19) dashboard, 2022.

[18] Robert Chun Chen, Thuan Tong Tan, and Lai Peng Chan. Adapting to a new normal? 5 key operational principles for a radiology service facing the covid-19 pandemic, 2020.

[19] RJM Bruls and RM Kwee. Workload for radiologists during on-call hours: dramatic increase in the past 15 years. *Insights into imaging*, 11(1):1–7, 2020.

[20] Federico Caobelli. Artificial intelligence in medical imaging: Game over for radiologists? *European journal of radiology*, 126, 2020.

[21] Curtis P Langlotz. Will artificial intelligence replace radiologists? *Radiology. Artificial intelligence*, 1(3), 2019.

[22] Charles C. Tappert. Who is the father of deep learning? In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 343–348, 2019.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[24] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[25] Kaichao You, Mingsheng Long, Jianmin Wang, and Michael I Jordan. How does learning rate decay help modern neural networks? *arXiv preprint arXiv:1908.01878*, 2019.

[26] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[27] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[28] Kunihiko Fukushima. Neocognitron. *Scholarpedia*, 2(1):1717, 2007.

[29] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[30] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[31] Masakazu Matsugu, Katsuhiko Mori, Yusuke Mitari, and Yuji Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5-6):555–559, 2003.

[32] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[33] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[34] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4):291–294, 1988.

[35] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[36] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[37] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer, 2009.

[38] Kang Zhou, Jing Li, Weixin Luo, Zhengxin Li, Jianlong Yang, Huazhu Fu, Jun Cheng, Jiang Liu, and Shenghua Gao. Proxy-bridged image reconstruction network for anomaly detection in medical images. *IEEE Transactions on Medical Imaging*, 41(3):582–594, 2021.

[39] Sathish Ramani, Thierry Blu, and Michael Unser. Monte-carlo sure: A black-box optimization of regularization parameters for general denoising algorithms. *IEEE Transactions on image processing*, 17(9):1540–1554, 2008.

[40] Poojan Oza and Vishal M Patel. One-class convolutional neural network. *IEEE Signal Processing Letters*, 26(2):277–281, 2018.

[41] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *International conference on machine learning*, pages 4393–4402. PMLR, 2018.

[42] Mohammad Sabokrou, Mohammad Khalooei, Mahmood Fathy, and Ehsan Adeli. Adversarially learned one-class classifier for novelty detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3379–3388, 2018.

[43] Pramuditha Perera, Ramesh Nallapati, and Bing Xiang. Ocgan: One-class novelty detection using gans with constrained latent representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2898–2906, 2019.

[44] Stanislav Pidhorskyi, Ranya Almohsen, and Gianfranco Doretto. Generative probabilistic novelty detection with adversarial autoencoders. *Advances in neural information processing systems*, 31, 2018.

[45] Patrick Schlachter, Yiwen Liao, and Bin Yang. One-class feature learning using intra-class splitting. In *2019 27th European Signal Processing Conference (EUSIPCO)*, pages 1–5. IEEE, 2019.

[46] Davide Abati, Angelo Porrello, Simone Calderara, and Rita Cucchiara. Latent space autoregression for novelty detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 481–490, 2019.

[47] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Georg Langs, and Ursula Schmidt-Erfurth. f-anogan: Fast unsupervised anomaly detection with generative adversarial networks. *Medical image analysis*, 54:30–44, 2019.

[48] Saman Motamed, Patrik Rogalla, and Farzad Khalvati. Randgan: randomized generative adversarial network for detection of covid-19 in chest x-ray. *Scientific Reports*, 11(1):1–10, 2021.

[49] Zhiwei Zhang, Shifeng Chen, and Lei Sun. P-kdgan: Progressive knowledge distillation with gans for one-class novelty detection. *arXiv preprint arXiv:2007.06963*, 2020.

[50] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Uninformed students: Student-teacher anomaly detection with discriminative latent embeddings. In *Proceedings of*

the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 4183–4192, 2020.

[51] Takahiro Nakao, Shouhei Hanaoka, Yukihiro Nomura, Masaki Murata, Tomomi Takenaga, Soichiro Miki, Takeyuki Watadani, Takeharu Yoshikawa, Naoto Hayashi, and Osamu Abe. Unsupervised deep anomaly detection in chest radiographs. *Journal of Digital Imaging*, 34(2):418–427, 2021.

[52] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

[53] Seyed Hamid Safiabadi Tali, Jason J LeBlanc, Zubi Sadiq, Oyejide Damilola Oyewunmi, Carolina Camargo, Bahareh Nikpour, Narges Armanfard, Selena M Sagan, and Sana Jahanshahi-Anbuhi. Tools and techniques for severe acute respiratory syndrome coronavirus 2 (sars-cov-2)/covid-19 detection. *Clinical microbiology reviews*, 34(3):e00228–20, 2021.

[54] Yonina C Eldar. Generalized sure for exponential families: Applications to regularization. *IEEE Transactions on Signal Processing*, 57(2):471–481, 2008.

[55] Florian Luisier, Thierry Blu, and Michael Unser. Image denoising in mixed poisson–gaussian noise. *IEEE Transactions on image processing*, 20(3):696–708, 2010.

[56] Martin Raphan and Eero P Simoncelli. Least squares estimation without priors or supervision. *Neural computation*, 23(2):374–420, 2011.

[57] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[58] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2097–2106, 2017.

[59] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.