Image-based Process Monitoring via Generative Adversarial Autoencoder with Applications to Rolling Defect Detection

by

Huai-Ming Yeh

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2019 by the
Graduate Supervisory Committee:

Hao Yan, Chair
Rong Pan
Jing Li

ARIZONA STATE UNIVERSITY

May 2019

ABSTRACT

Image-based process monitoring has recently attracted increasing attention due to the advancement of the sensing technologies. However, existing process monitoring methods fail to fully utilize the spatial information of images due to their complex characteristics including the high dimensionality and complex spatial structures. Recent advancement of the unsupervised deep models such as generative adversarial network (GAN) and generative adversarial autoencoder (AAE) have enabled to learn the complex spatial structures automatically. Inspired by this advancement, we propose an anomaly detection framework based on the AAE for unsupervised anomaly detection for images. AAE combines the power of GAN with the variational autoencoder, which serves as a nonlinear dimension reduction technique. Based on this, we propose a monitoring statistic efficiently capturing the change of the data. The performance of the proposed AAE-based anomaly detection algorithm is validated through a simulation study and real case study for rolling defect detection.

TABLE OF CONTENTS

Page

LIST OF TABLES iii

LIST OF FIGURES iv

CHAPTER

1 INTRODUCTION 1

1.1 Methodology Development 4

1.1.1 Review of Generative Adversarial Network 4

1.1.2 Adversarial Autoencoder and its Application to Anomaly Detection 5

2 RESULTS 8

2.1 Simulation Study 8

2.1.1 Simulation Setup 8

2.1.2 Simulation Results 9

2.2 Case Study 14

3 CONCLUSION 20

REFERENCES 21

APPENDIX

A ORIGINAL MODEL CODES AAE 23

B ORIGINAL MODEL CODES VAAE 25

C ORIGINAL MODEL CODES VAE 27

LIST OF TABLES

Table Page

2.1 Mean Shift Accuracy (%) Under $\delta = 0.06$ 11

2.2 Detection Power (%) 19

LIST OF FIGURES

Figure Page

1.1 Normal and Abnormal Rolling Images 2

2.1 Normal and Abnormal Images with $\delta = 0.3$ 9

2.2 Sensitivity Analysis of PCA and AAE 12

2.3 Learned 2D Manifold 13

2.4 Original and Reconstructed Images for AAE and PCA with Different Encoding Dimensions (1) 14

2.5 Original and Reconstructed Images for AAE and PCA with Different Encoding Dimensions (2) 15

2.6 Learned 2D Manifold 17

2.7 Normal and Anomaly images, PCA Reconstructed Images , AAE Reconstructed Images, and VAAE Reconstructed Images 18

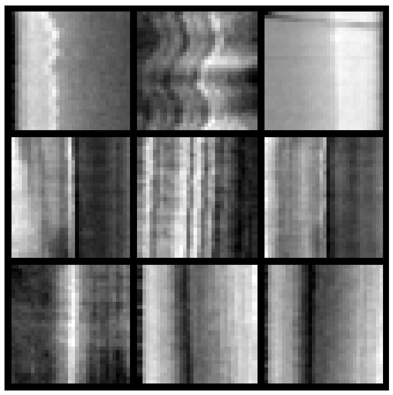
Chapter 1

INTRODUCTION

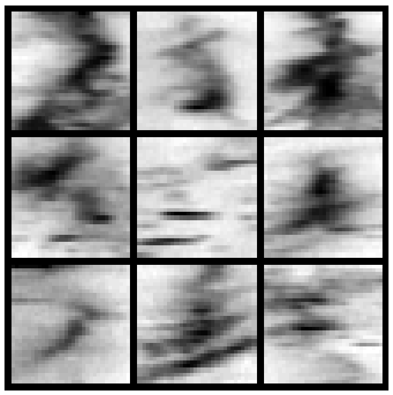
Nowadays, image data are widely used in most manufacturing processes and service systems to evaluate the process performance and product quality due to the low implementation cost and rich information it provides. Real-time image-based process monitoring and online product inspection are among the benefits that can be gained from this advancement.

For example, in the rolling inspection, a high-speed video camera is set up to monitor the surface of the rolling bars. We will then extract some areas in the rolling images, which could be the potential anomaly regions. For example, some of the detected regions have some vertical line patterns, which are minor or can be considered as normal as shown in Figure 1.1 (a). Three types of the anomalous images (i.e. denoted as anomaly 1, 2, and 3) are shown in Figure 1.1 (b), (c), (d), respectively. For example, anomaly 1 has irregular black patterns. Anomaly 2 has wider white marks. Anomaly 3 has irregular white-black-white patterns that are not shown in the normal samples. The goal of this paper is to develop an automatic anomaly detection algorithm that differs from the normal variation patterns effectively and efficiently.

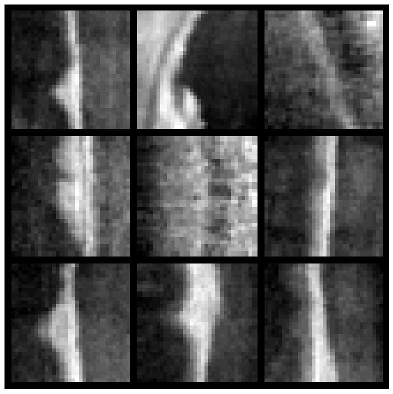
There are two major challenges regarding the image data: i) high-dimensionality: high-resolution images may have thousands or even millions of pixels. ii) complex nonlinear correlation structures. For example, these spatial dimensions are often not only locally correlated (e.g. nearby pixels normally shared similar values) but also globally correlated. Therefore, it is often very challenging to model these nonlinear variation patterns. Modeling the complex variation patterns and detecting the abnormal patterns in real time is a very challenge task.



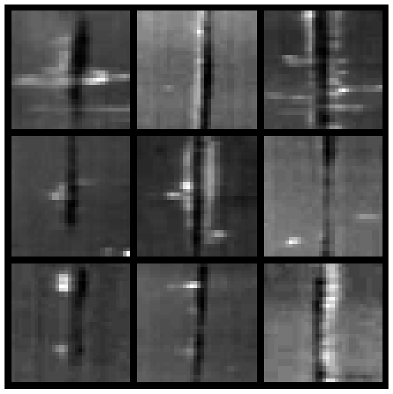
(a) Normal



(b) Anomaly 1



(c) Anomaly 2



(d) Anomaly 3

Figure 1.1: Normal and Abnormal Rolling Images

Most of the literature on process monitoring techniques for image data can be divided into the following three categories: i) Linear dimension reduction techniques: they treat the image data as a high-dimensional data and utilizing the dimension reduction techniques such as principal component analysis (PCA) [17] and independent component analysis [4] to reduce the dimensionality to a lower dimension. Furthermore, some variants of the dimension reduction techniques are developed including functional PCA methods [9, 19], and tensor decomposition methods [14, 8]). However, these techniques are mostly linear methods and therefore don't perform well dealing with complex nonlinear patterns. ii) Functional data analysis: These methods treat image as continuous functions and analyze the functional features by transforming the data into certain feature spaces for feature extraction, such as wavelet transformation [7, 20], B-spline approximation [2], and functional decomposition-based techniques [15]. Finally, process monitoring techniques focus on monitoring the feature coefficients or residuals [16]. These techniques typically have very strong assumptions on the image to be applied. For example, B-spline and kernel methods are designed for images with smooth background and wavelet is only designed for wave-form types of spatial structures. Both cases cannot be applied to more complex variation patterns. iii) Manifold learning techniques: kernel PCA [13] and maximum variance unfolding projections [12] are proposed to learn the representation of the nonlinear profile for process monitoring. However, manifold learning is designed for modeling or Phase-I monitoring. How to efficiently construct the monitoring statistics for different types of changes remains a nontrivial problem.

Recently, unsupervised deep learning methods such as generative adversarial network (GAN) [5] has been proposed and demonstrated that it can generate very realistic images. GAN-based approaches can capture the complex spatial correlation of the images data recently, therefore has been applied for anomaly detection in medical imaging [11, 18, 1]. However, these methods lack the ability to directly map the data into the feature space, which hinder its use for the efficient and real-time procedure. Recently, generative adversarial autoencoder (AAE) has been proposed [6], which combines the power of GAN and the variational autoencoder to ensure that the encoded features follow the normal distribution and the features can be directly computed by encoders. In this paper, we will propose how to use AAE methods for efficient and real-time process monitoring.

1.1 Methodology Development

Before we dive into the proposed methodology for anomaly detection, we would like to first briefly review the generative adversarial network (GAN), generative adversarial autoencoder (AAE), and how it can be used for efficient anomaly detection and process monitoring.

1.1.1 Review of Generative Adversarial Network

Generative adversarial networks (GAN) [5] has recently gained much attention due to its ability to learn the complex high-dimensional distribution by jointly train a generator G and discriminator D in a zero-sum game. The discriminator $D(x)$ is normally a trained neural network that computes the probability that a point x is from the generator $G(z)$ or the set of real samples. The generator uses a function $G(z)$, which is typically another neural network, to generate the samples from the prior $p(z)$ (i.e. usually a multivariate normal distribution with identity covariance matrix) to generate the data. The goal is to maximally confuse the discriminator $D(x)$ so that it cannot tell the differences between whether the data is generated from the generator $G(z)$ or from the real samples. In this case, the generator $G(z)$ is identical to the real data distribution $p(x)$.

To train GAN, the following mini-max problem function is used to train D and G simultaneously to minimize $\log(1 - D(G(z))$. [5]

$$\min_G \max_D E_{x \sim p_d(x)}[\log D(x)] + E_{z \sim p(z)}[\log(1 - D(G(z))] \tag{1.1}$$

For given G fixed, the optimal discriminator D can be written as

$$D^*_G(x) = \frac{P_{data}(x)}{P_{data}(x) + P_g(x)} \tag{1.2}$$

For estimating the conditional probability P(Y = y—x), D is trained to maximizing the log-likelihood, where Y indicates whether x comes from p_{data} or from p_g. Then, reform the objective function1.1 as

$$E_{x\sim p_d(x)}[\frac{P_{data}(x)}{P_{data}(x)+P_g(x)}]+E_{z\sim p(z)}[\frac{P_g(x)}{P_{data}(x)+P_g(x)}] \tag{1.3}$$

The global optimum of the training is achieved if and only if $p_g = p_{data}$. If G and D have enough capacity, as the discriminator D achieved optimum given G, and p_g is updated to approach p_{data}.

Typically, the training of GAN happens in two stages: (a) train the discriminator D to distinguish the true samples and the fake samples. (b) train the generator G to fool the discriminator.

However, the major limitation of GAN is that it lacks an efficient encoder to match the original data back to its feature. GAN only match the G(z) to the p(x), and use D to prevent over-fitting the images. We will discuss how generative adversarial autoencoder (AAE)[6] is able to solve this problem and how it can be used for anomaly detection in the next subsection.

1.1.2 Adversarial Autoencoder and its Application to Anomaly Detection

Adversarial Autoencoder

In this section, we will discuss how AAE[6] achieve an efficient encoder to match the original data back to its encoded feature.

Here, we will still use x to represent the high-dimensional data to be monitored (e.g. signals, images, functional data from different sensors) and introduce z to be the low-dimensional features that not observable but are inferred x.

$$p(z)=\int_x p(z|x)p(x)dx \tag{1.4}$$

AAE is designed by introducing another autoencoder function $q(z|x)$ by taking the input data x to compute the latent code z. The goal of AAE is to match the aggregated posterior distribution

$$q(z) = \int_x q(z|x) p_d(x) dx \tag{1.5}$$

to a certain prior distribution $p(z)$, where $p_d(x)$ is the data distribution. This can also act like a regularization so that the method would not over-fit. In order to do so, an adversarial network is trained to guide the $q(z)$ to match the $p(z)$ which is the same as how the adversarial training in GAN is used to match the generated data distribution to the sample data distribution.

In the original papers, the authors propose three different types of encoders for $q(z|x)$. In this paper, we will just use the deterministic posterior where the $q(z|x)$ is assumed to be a deterministic function of x since we have not found clear differences in terms of performance in other types of encoders.

Furthermore, in the AAE method, adversarial training is used to ensure the encoded features can match the Gaussian prior distribution by introducing the discriminator.

$$\min_Q \min_G l_{recon} + \min_Q \max_D l_{adversarial} \tag{1.6}$$

Finally, the loss function consists of two components: i) The reconstruction error: The cost function of the reconstruction error is l_{recon}. Here, the reconstruction error l_{recon} is defined as

$$l_{recon} = E_{x \sim p_d(x)} l(x, G(Q(z|x)) \tag{1.7}$$

It worth noting that the data likelihood is derived from the distribution of the data. ii) The discrimination error: The discriminative error is designed based on the adversarial training, which leads to the loss function $l_{adversarial}$ similar to (1.1) to match the $p(z)$ and $q(z) = \int_x q(z|x) p_d(x) dx$. Here the adversarial loss $l_{adversarial}$ is defined as:

$$\begin{aligned} l_{adversarial} &= E_{z\sim p(z)}[\log D(z)] \\ &+ E_{x\sim p_d(x)} E_{z\sim Q(z|x)}[\log(1 - D(z))] \end{aligned} \tag{1.8}$$

The adversarial loss $l_{adversarial}$ is to ensure that the latent code would match the aggregated posterior distribution and prior distribution $p(z)$. After the generator G and the discriminator D are trained, we would like to discuss the way to monitor the data in the next subsection.

Phase I and Phase II Analysis

We propose to use the reconstruction error $l_{recon}(x) = l(x, G(Q(z|x))$ as the monitoring statistics. Here, $l(x, G(Q(z|x))$ is the likelihood function, which depends on the probability distribution of the original data x. For example, if the data is Gaussian distributed, we can use the sum of squared error between the input image and the reconstruction image $\|x - G(Q(z|x))\|^2$. If the data is Bernoulli distributed, the cross entropy loss can be used, which is defined as $\|x \log(G(Q(z|x))) + (1 - x) \log(1 - G(Q(z|x)))\|_1$.

In the phase I analysis, we would like to sample a set of validation data set which belongs to the normal (in-control) samples but not in the training data set to estimate the distribution of the $l_{recon}(x)$ and the control limit c_0. If the number of the normal samples is large, using the quantile is normally good enough of one validation data set is often good enough. However, if the number of the normal samples is limited, we can also use the cross-validation to compute the quantile of multiple validation data set as the control limit c_0. Finally, c_0 is often selected such that the type I error is as a certain number. In this paper, we will set up the control limit c_0 based on the 5% false positive rate. Finally, in the Phase II analysis, we propose to use the $l_{recon}(x) > c_0$ to detect the anomalies.

Chapter 2

RESULTS

2.1 Simulation Study

2.1.1 Simulation Setup

In this section, we generate 32×32 2D images containing a circle with different locations and shapes for normal samples from the following formula:

$$y = \sqrt{1 - ((x_1 - x_{10})/a)^2 - ((x_2 - x_{20})/b)^2},$$

where $x_{10}, x_{20} \sim N(0.5, 0.05^2)$, $a, b \sim N(0.2, 0.05^2)$ denote pixel locations on an image.

To validate the Phase II process monitoring performance, we generate four types of abnormal data for performance evaluation by the following formula:

- Mean Shift:

$$y = \delta + \sqrt{1 - ((x_1 - x_{10})/a)^2 - ((x_2 - x_{20})/b)^2},$$

- Magnitude Change:

$$y = (1+\delta)\sqrt{1 - ((x_1 - x_{10})/a)^2 - ((x_2 - x_{20})/b)^2},$$

- Width Change:

$$y = \sqrt{1 - ((x_1 - x_{10})/(a+\delta))^2 - ((x_2 - x_{20})/b)^2},$$

- Location Change:

$$y = \sqrt{1 - ((x_1 - x_{10})/a)^2 - ((x_2 - x_{20})/(b+\delta))^2}.$$

Here δ represent the change magnitude, x_{10}, x_{20} denote pixel locations on an image, and a, b denote the widths of the circle. Examples of these generated normal images and abnormal images ($\delta = 0.3$) are shown in Figure 2.1.

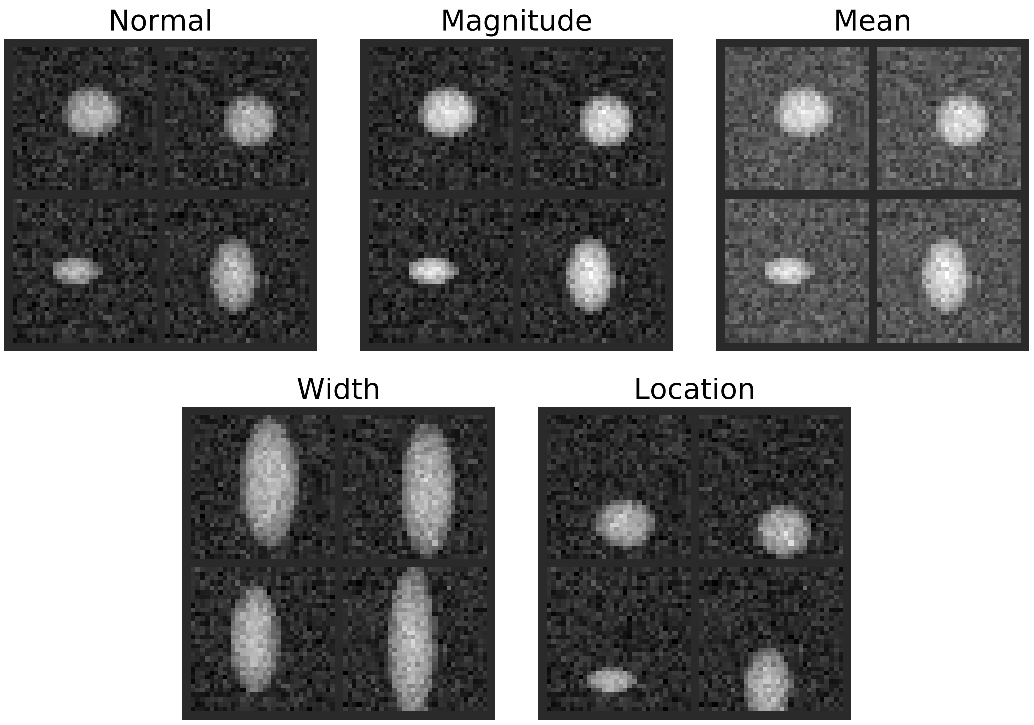


Figure 2.1: Normal and Abnormal Images with $\delta = 0.3$

2.1.2 Simulation Results

For the proposed AAE anomaly detection method, we proposed to use the latest DCGAN architectures [10] for the encoder and decoder, which combined a set of convolutional layers, batch normalization, and ReLU activation. We will compare the different latent dimensions such as 6 and 10 for the AAE method. Furthermore, in order to balance the generator and discriminator, the generator learning rate is set to 0.002 whereas the discriminator learning rate is 0.0002. Both use Adam optimizer for the best result. For the benchmark methods, we propose to compare with the widely used PCA-based methods with different principal components (PCs) and the unsupervised methods combination of AAE and VAE (VAAE).

For AAE, we will use the reconstruction error as the monitoring statistics. For PCA methods, we propose to combine the Q-chart and T2 chart for better performance [3]. Furthermore, to identify the anomaly, we would like to use the 5% false positive rate for all methods. For the combination model VAAE, we adopted Q-chart and T2 chart for monitoring statistics.

For the evaluation, we propose to use the detection power of Phase II analysis, defined as the percentage of samples that are correctly detected as anomalies by the algorithms over the number of anomalous samples. Finally, the detection power of AAE, VAAE, and PCA with different encoding dimensions and PC dimensions are shown in Table 2.1, respectively. From Table 2.1, we can conclude that AAE is more powerful than PCA and VAAE in all cases that we tested.

For examples, in the non-linear change such as width and location change, AAE has a very clear advantage over PCA since PCA is not able to represent these non-linear change patterns. In the linear change such as mean-shift, AAE still has some advantage over PCA. We can also conclude that simply adding the number of PCs only helps on the linear change patterns such as mean shift, but doesn't help too much on the complex nonlinear change patterns such as the width change and location change. Also, VAAE able to detect well in linear change patterns but not in nonlinear change patterns.

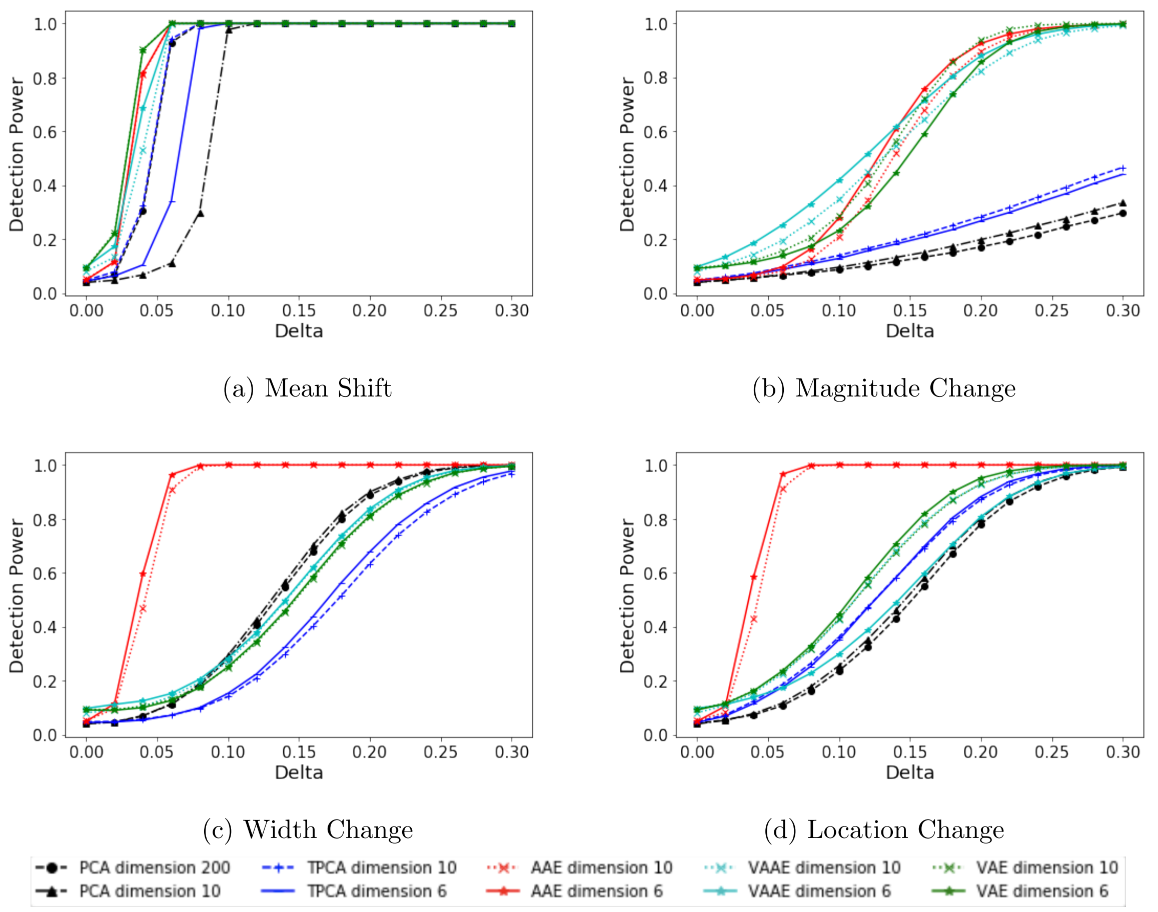
Furthermore, adding the number of PCs from 50 to 200 may also reduce the performance of PCA methods on the Width change (i.e. decrease from 18.78 to 18.35). However, AAE is much more robust to the selection of the latent dimensions. For example, the performance of AAE with 6 or 10 latent dimensions are similar, and AAE with 6 latent dimensions works slightly better.

Furthermore, we also plot the detection power with different change magnitudes δ in Figure 2.2. From Figure 2.2, we can observe that PCA is sensitive to the number of PCs chosen. Magnitude change is challenging to both models since it doesn't change the shape of the original images. We found PCA can hardly detect the change even when $\delta = 0.3$ whereas, AAE is able to reach about 100% accuracy. For the nonlinear changes such location change and width change, using AAE with 6 dimensions outperforms PCA with even up to 200 dimensions.

Methods	Dimension	Mean	Magnitude	Width	Location
PCA	6	18.30	37.07	19.54	19.72
	10	22.54	34.85	19.75	19.95
	50	96.90	32.77	18.78	19.51
	200	97.66	32.31	18.53	19.77
Tucker PCA	6	34.02	33.48	7.17	17.52
	10	94.28	35.5	7.21	18.54
Parafac PCA	6	11.24	29.38	11.99	16.88
	10	9.84	17.1	9.57	11.73
VAE	6	100.0	97.05	12.65	23.57
	10	100.0	99.34	13.10	23.01
AAE	6	100.0	97.97	96.44	96.58
	10	100.0	97.47	90.98	91.31
VAAE	6	99.99	95.99	15.25	17.28
	10	99.7	93.83	14.15	22.5

Table 2.1: Mean Shift Accuracy (%) Under $\delta = 0.06$

To understand how AAE detect the change, we would like to plot the real and



(a) Mean Shift

(b) Magnitude Change

(c) Width Change

(d) Location Change

Figure 2.2: Sensitivity Analysis of PCA and AAE

reconstructed images in Figure 2.5. From this figure, we can conclude that AAE is able to generate accurate reconstruction images for the normal samples in Figure 2.5 (a). PCA with 10 PCs (i.e. PCA 10) cannot get the clear reconstruction of the original images. Furthermore, PCA with 200 PCs (i.e. PCA 200) can reconstruct the original images, but it generates a much noisy result. For the mean shift in Figure 2.5 (b), AAE is able to recover the darker circle and background. The reconstructed images of PCA with 10 and 200 PCs is largely affected by the background mean shift. For the magnitude change in Figure 2.5 (c), the AAE generates much darker reconstruction images than PCA, which shows that AAE actually learned the right magnitude for

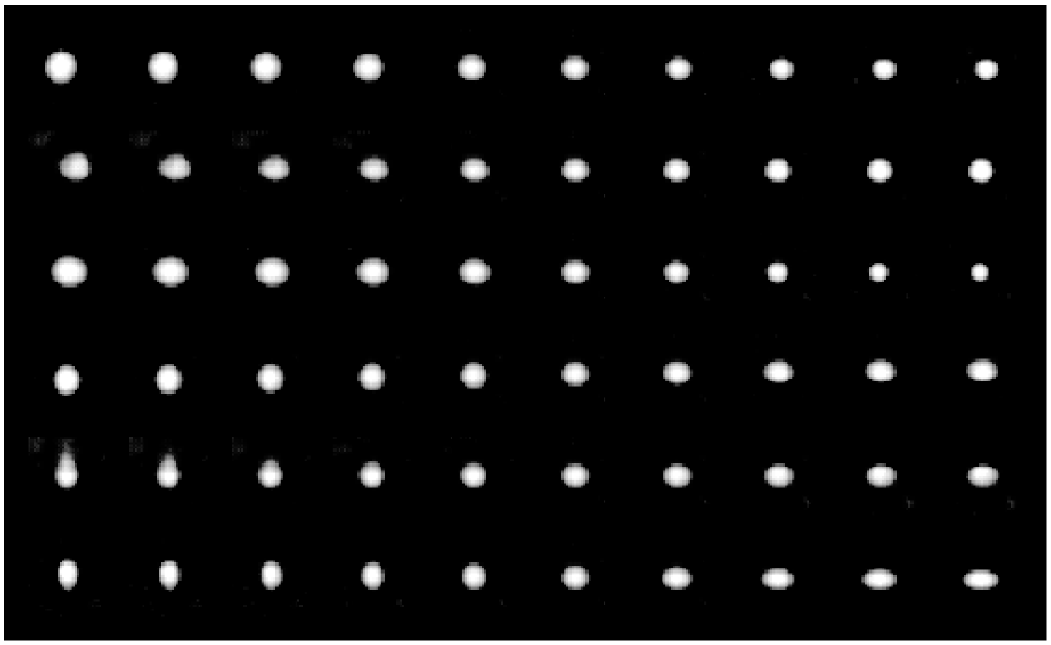
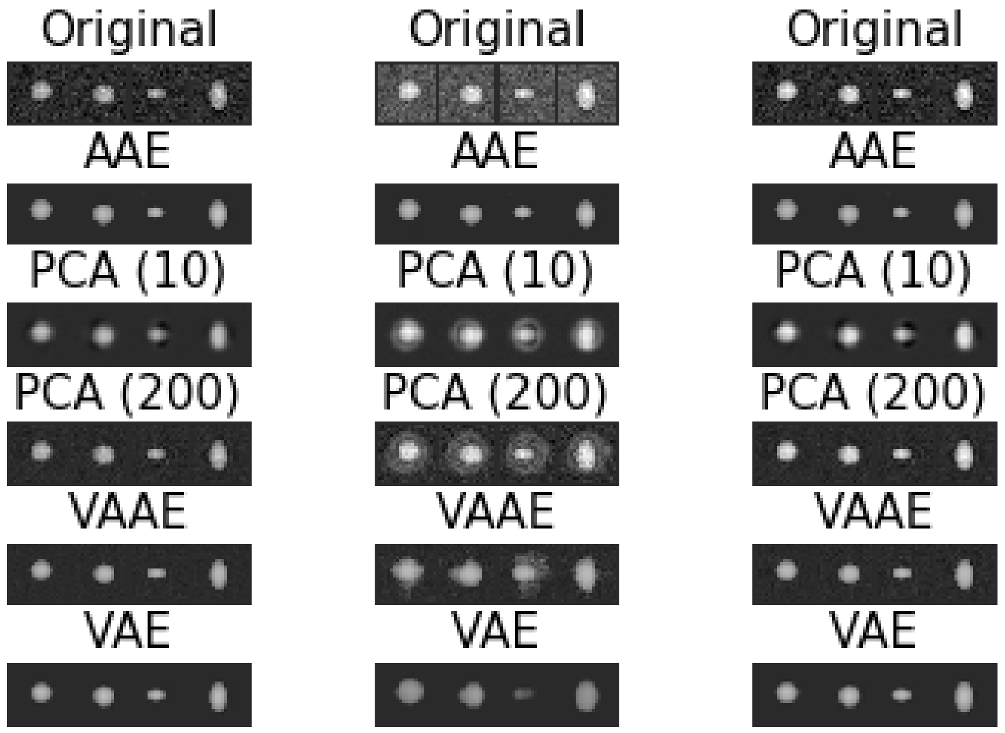


Figure 2.3: Learned 2D Manifold

the normal samples, compared to the PCA methods. Finally, for width change and location change, the AAE reconstructed images are not affected by the out-of-control samples, which make the reconstruction error a great monitoring statistics. However, PCA with 10 PCs generates very blurry images and PCA with 200 PCs tend to reconstruct the original images, which makes the Q-chart (i.e. control chard designed based on residual) not as effective.

Finally, to understand how AAE represent the complex nonlinear variational patterns, we also plot the 2D manifold reconstructed from the 2D lattice from a slice of 6-D latent space z in Figure 2.3. Each row of the image represent a dimension corresponding to a latent space. As shown in the figure, first and third rows represent the magnitude change, second row shows width change, forth and fifth rows reveal the vertical location change, and the six correspond to the horizontal shape change. This demonstrates latent variables learned the feature space distribution and the smoothness of the generated samples from the latent feature z.

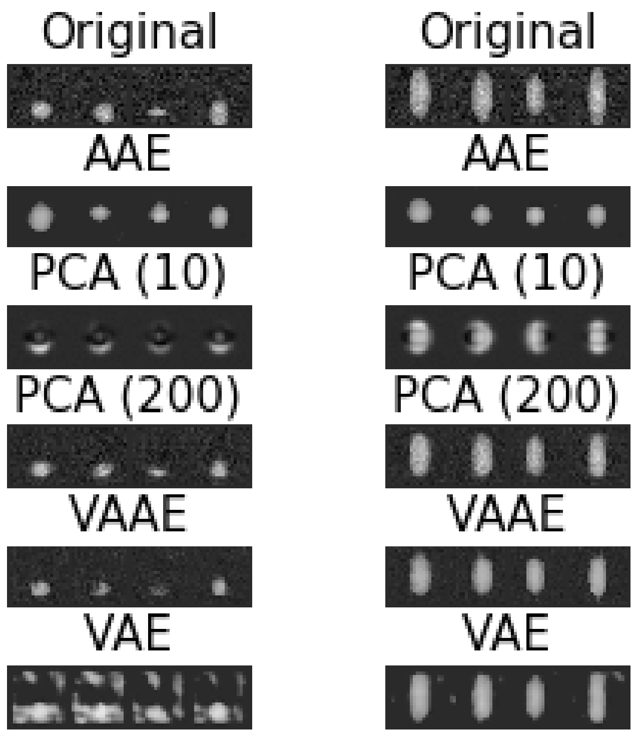


(a) Normal Images (b) Mean Shift (c) Magnitude Change

Figure 2.4: Original and Reconstructed Images for AAE and PCA with Different Encoding Dimensions (1)

2.2 Case Study

In this section, we will use real images from the quality inspection in the rolling manufacturing to illustrate the performance of the proposed AAE anomaly detection procedure. The dataset is made of metal rolling inspection images that are potentially defect. The domain engineers have labeled the images as normal or abnormal samples. Training data is made of 879 normal images and 294 abnormal images in 3 different



(a) Width Change (b) Location Change

Figure 2.5: Original and Reconstructed Images for AAE and PCA with Different Encoding Dimensions (2)

types of abnormal conditions: Anomaly 1, Anomaly 2, and Anomaly 3. The regular images feature vertical texture or some minor overfills with black or white lines. The examples of normal images, Anomaly 1, Anomaly 2, and Anomaly 3 are shown in Figure 2.7 (a), (e), (i), (m), respectively.

We will compare AAE with PCA and VAAE on the detection power (percentage of detected samples) with a fixed 5% false positive rate. We will use the same architecture and optimizer for the case study. We will use 6 or 10 latent space dimensions for the AAE method. For PCA, we will investigate to use $2, 10, 50, 200$ PCs for anomaly detection.

As shown in table 2.2, PCA with 2 PCs works the best for *Anomaly 1.* The reason is that the change of *Anomaly 1* is quite obvious and the shape looks completely different than the normal vertical texture and can be fully detected via PCA with 6 PCs and AAE with 6 latent dimensions. However, for *Anomaly 2* and *Anomaly 3*, both are much harder to detect since the anomaly patterns also feature vertical textures. For AAE, in Anomaly 2 and 3, increasing the latent dimension to 10 will increase the detection accuracy to 100%. This is because AAE put the adversarial regularization so it can control the model complexity. However, for *Anomaly 2*, PCA with 50 PCs is optimal and achieves detection accuracy around 60%. For *Anomaly 3*, PCA with 2 PCs works the best. Further increasing the number of PCs decreases the detection power. This shows that simply increasing the number of PCs may lead to over-fitting problems. VAAE shows descent power of detection, this is because VAAE also has strong adversarial regularization enable the model control the model complexity.

Finally, we also plot the normal images, anomaly images, AAE-reconstructed images, and PCA reconstructed images with 6 PCs (i.e. PCA 6) and 200 PCs (i.e. PCA 200) in Figure 2.7. From Figure 2.7, we can conclude that AAE can reconstruct very realistic images with texture details. PCA with 6 PCs tends to create blurry images without enough details. On the other hand, PCA with 200 PCs will overfit and generate exactly the original images. VAAE can reconstruct clearance and details of images; however, comparing to AAE, it seems affected by the input data, which show the model could overfit the data.

Finally, we also plot the AAE learned manifold in Figure2.6. This figure is generated in a 2D lattice slice of a 10-D latent space z by the decoder $q(z|x)$. This map shows that AAE actually learns a smooth manifold of all the rolling images. Each row of the image represent a dimension corresponding to a latent space. As shown in the figure, first row corresponded to the background marks, second and third rows represented diagonal patterns, forth and fifth reveal the marks horizontal position, and sixth row is for the marks vertical changes. his demonstrates latent variables learned the feature space distribution .

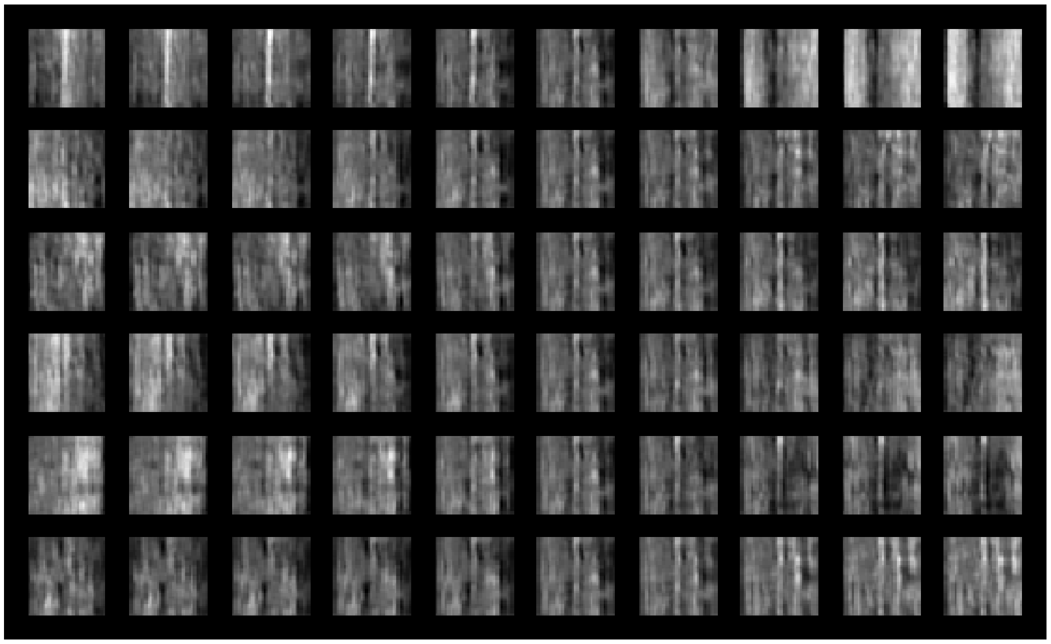
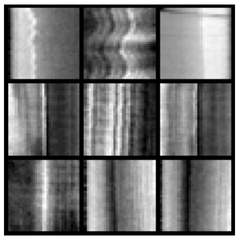
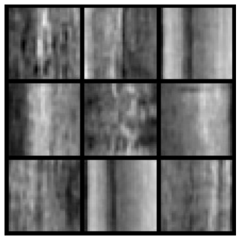
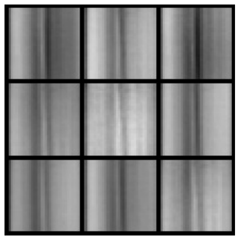
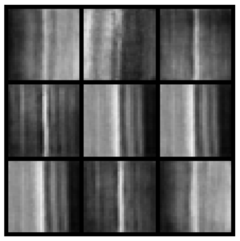
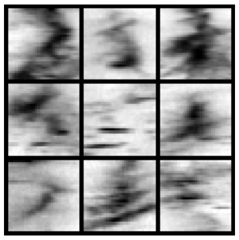
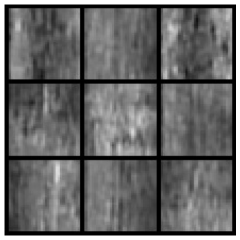
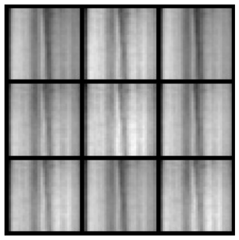
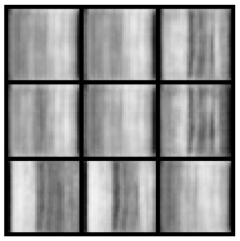
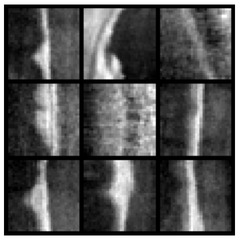
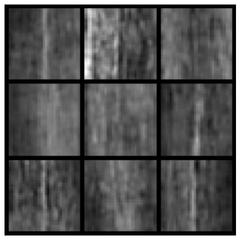
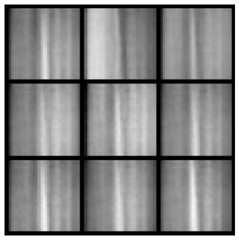
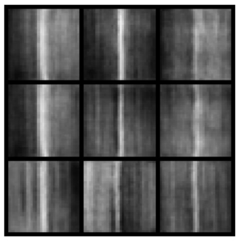


Figure 2.6: Learned 2D Manifold


(a) Normal

(b) AAE

(c) PCA 6

(d) VAAE


(e) Change 1

(f) AAE

(g) PCA 6

(h) VAAE


(i) Change 2

(j) AAE

(k) PCA 6

(l) VAAE

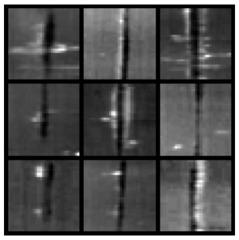
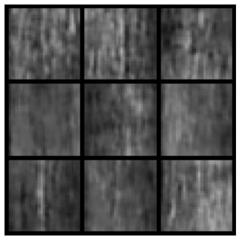
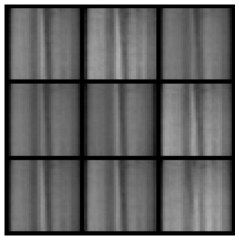
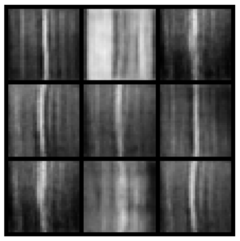

(m) Change 3

(n) AAE

(o) PCA 6

(p) VAAE

Figure 2.7: Normal and Anomaly images, PCA Reconstructed Images , AAE Reconstructed Images, and VAAE Reconstructed Images

Methods	Dimension	Change 1	Change 2	Change 3
PCA	6	100.0	43.05	20.22
	50	99.25	59.72	20.22
	200	99.25	47.22	23.59
Tucker PCA	2	94.22	47.22	14.60
	6	97.76	2.78	12.36
	10	97.76	2.78	12.36
Parafac PCA	2	100.0	62.5	17.97
	6	36.56	2.78	1.15
	10	12.68	1.39	0.00
VAE	2	35.07	45.83	33.70
	6	62.68	58.33	33.70
	10	96.26	68.05	30.33
AAE	2	94.02	54.16	51.68
	6	100.0	86.11	78.65
	10	100.0	100.0	100.0
VAAE	2	99.25	37.50	35.95
	6	100.0	63.88	33.71
	10	100.0	54.16	29.21

Table 2.2: Detection Power (%)

Chapter 3

CONCLUSION

Image-based process monitoring and anomaly detection often deal with complex nonlinear spatial correlation structures. In this paper, we propose a nonlinear anomaly detection algorithm based on generative adversarial autoencoder. The proposed method has shown a largely improved accuracy over the traditional linear process monitoring method such as PCA. A simulation study and a real case study from rolling manufacturing have been added to the original paper to demonstrate the advantage of the proposed AAE-based anomaly detection methods.

REFERENCES

[1] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training. *arXiv preprint arXiv:1805.06725*, 2018.

[2] Shing I Chang and Srikanth Yadama. Statistical process control for monitoring non-linear profiles using wavelet filtering and b-spline approximation. *International Journal of Production Research*, 48(4):1049–1068, 2010.

[3] Q Chen, U Kruger, M Meronk, and AYT Leung. Synthesis of t2 and q statistics for process monitoring. *Control Engineering Practice*, 12(6):745–755, 2004.

[4] Yu Ding, Li Zeng, and Shiyu Zhou. Phase i analysis for monitoring nonlinear profiles in manufacturing processes. *Journal of Quality Technology*, 38(3):199, 2006.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[6] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[7] Kamran Paynabar and Jionghua Jin. Characterization of non-linear profiles variations using mixed-effect models and wavelets. *Iie transactions*, 43(4):275–290, 2011.

[8] Kamran Paynabar, Jionghua Jin, and Massimo Pacella. Monitoring and diagnosis of multichannel nonlinear profile variations using uncorrelated multilinear principal component analysis. *IIE Transactions*, 45(11):1235–1247, 2013.

[9] Kamran Paynabar, Changliang Zou, and Peihua Qiu. A change-point approach for phase-i analysis in multivariate profile monitoring and diagnosis. *Technometrics*, 58(2):191–204, 2016.

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

[11] 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.

[12] Ji-Dong Shao and Gang Rong. Nonlinear process monitoring based on maximum variance unfolding projections. *Expert Systems with Applications*, 36(8):11332–11340, 2009.

[13] Zhenyu Shi, Daniel W Apley, and George C Runger. Discovering the nature of variation in nonlinear profile data. *Technometrics*, 58(3):371–382, 2016.

[14] Hao Yan, Kamran Paynabar, and Jianjun Shi. Image-based process monitoring using low-rank tensor decomposition. *IEEE Transactions on Automation Science and Engineering*, 12(1):216–227, 2015.

[15] Hao Yan, Kamran Paynabar, and Jianjun Shi. Anomaly detection in images with smooth background via smooth-sparse decomposition. *Technometrics*, 59(1):102–114, 2017.

[16] Hao Yan, Kamran Paynabar, and Jianjun Shi. Real-time monitoring of high-dimensional functional data streams via spatio-temporal smooth sparse decomposition. *Technometrics*, (just-accepted), 2017.

[17] Guan Yu, Changliang Zou, and Zhaojun Wang. Outlier detection in functional observations with applications to profile monitoring. *Technometrics*, 54(3):308–318, 2012.

[18] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. Efficient gan-based anomaly detection. *arXiv preprint arXiv:1802.06222*, 2018.

[19] Chen Zhang, Hao Yan, Seungho Lee, and Jianjun Shi. Weakly correlated profile monitoring based on sparse multi-channel functional principal component analysis. *IISE Transactions*, 50(10):878–891, 2018.

[20] Shiyu Zhou, Baocheng Sun, and Jianjun Shi. An spc monitoring system for cycle-based waveform signals using haar transform. *IEEE Transactions on Automation Science and Engineering*, 3(1):60–72, 2006.

APPENDIX A

ORIGINAL MODEL CODES AAE

The AAE model Architecture:

```
import numpy as np

def createnetwork(encoding_dims):
    network = {
            'generator': {
                'name': AdversarialAutoencoderGenerator,
                'args': {
                    'encoding_dims': encoding_dims,
                    'input_size': 32,
                    'input_channels': 1
                },
                'optimizer': {
                    'name': Adam,
                    'args': {
                        'lr': 0.002,
                        'betas': (0.5, 0.999)
                    }
                }
            },
            'discriminator': {
                'name': AdversarialAutoencoderDiscriminator,
                'args': {
                    'input_dims': encoding_dims,
                },
                'optimizer': {
                    'name': Adam,
                    'args': {
                        'lr': 0.0002,
                        'betas': (0.5, 0.999)
                    }
                }
            }
    }
    return network
```

APPENDIX B

ORIGINAL MODEL CODES VAAE

The VAAE model Architecture:

```
VAAEAutoencoder(
  (encoder): Sequential(
    (0): Conv2d(1, 32, kernel_size=(4, 4), stride=(2, 2),
    padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace)
    (3): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2)
    , padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True
    , track_running_stats=True)
    (5): LeakyReLU(negative_slope=0.2, inplace)
    (6): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2)
    , padding=(1, 1))
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True
    , track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.2, inplace)
    (9): View())
  (dense_mu): Linear(in_features=2048, out_features=10)
  (dense_logvar): Linear(in_features=2048, out_features=10)
  (decoder): Sequential(
    (0): Linear(in_features=10, out_features=2048
    , bias=True)
    (1): BatchNorm1d(2048, eps=1e-05, momentum=0.1
    , affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace)
    (3): View()
    (4): ConvTranspose2d(128, 64, kernel_size=(4, 4),
    stride=(2, 2), padding=(1, 1))
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
    (6): LeakyReLU(negative_slope=0.2, inplace)
    (7): ConvTranspose2d(64, 32, kernel_size=(4, 4),
    stride=(2, 2), padding=(1, 1))
    (8): BatchNorm2d(32, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
    (9): LeakyReLU(negative_slope=0.2, inplace)
    (10): ConvTranspose2d(32, 1, kernel_size=(4, 4),
    stride=(2, 2), padding=(1, 1))
    (11): Tanh()))
```

APPENDIX C

ORIGINAL MODEL CODES VAE

The VAE model Architecture:

```
VAE(
  (vaeencode): Sequential(
    (0): Conv2d(1, 4, kernel_size=(3, 3), stride=(1, 1),
    padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(4, 4, kernel_size=(3, 3), stride=(2, 2),
    padding=(1, 1))
    (3): ReLU(inplace)
    (4): Conv2d(4, 8, kernel_size=(3, 3), stride=(1, 1),
    padding=(1, 1))
    (5): ReLU(inplace)
    (6): Conv2d(8, 8, kernel_size=(4, 4), stride=(2, 2),
    padding=(1, 1))
    (7): ReLU(inplace)
  )
  (conv_mu): Conv2d(8, 6, kernel_size=(8, 8),
  stride=(1, 1))
  (conv_logvar): Conv2d(8, 6, kernel_size=(8, 8),
  stride=(1, 1))
  (vaedecode): Sequential(
    (0): ConvTranspose2d(6, 8, kernel_size=(8, 8),
    stride=(1, 1))
    (1): ReLU(inplace)
    (2): ConvTranspose2d(8, 8, kernel_size=(4, 4),
    stride=(2, 2), padding=(1, 1))
    (3): ReLU(inplace)
    (4): ConvTranspose2d(8, 4, kernel_size=(3, 3),
    stride=(1, 1), padding=(1, 1))
    (5): ReLU(inplace)
    (6): ConvTranspose2d(4, 4, kernel_size=(3, 3),
    stride=(2, 2), padding=(1, 1))
    (7): ReLU(inplace)
    (8): ConvTranspose2d(4, 1, kernel_size=(4, 4),
    stride=(1, 1), padding=(1, 1))))
```