

MASTER

Anomaly detection with generative models with applications to lung cancer screening

Mendoza Marin, R.A.

Award date:
2018

[Link to publication](#)

Disclaimer

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain



Department of Mathematics and Computer Science
Data Mining Research Group

Anomaly detection with generative models

with applications to lung cancer screening

Rodrigo Mendoza

Supervisors:
dr. V. Menkovski
dr. D. Mavroeidis

A thesis presented for the degree of
Master in Computer Science and Engineering

February 7, 2018

Abstract

Classification of malign and benign lung nodules is a challenging task. Existing methods are based on discriminative approaches and less research has been given to generative models. Over the last few years, there has been an increasing use of generative models for creating meaningful representations. Such representations are desirable since pixel-wise values do not share insightful information for classification tasks. We expect with this work, to open a research avenue of generative models with application to lung cancer screening.

In this work, we propose anomaly detection with generative adversarial networks and anomaly detection with variational autoencoders for the classification of malign and benign lung nodules. Additionally, we use the manifold learned from GANs as feature extractor for supervised classification. For anomaly detection, we use only benign nodules to train our model and afterwards classify both malign and benign lung nodules. When using GANs as feature extraction we employ both types of lung nodules during training. With our models we are capable of generating synthetic lung nodules.

Our experiments with NLST data showed that is possible to reconstruct and generate lung nodules never seen during training. We achieve the reconstruction by searching in the latent space of GANs the closest image in the manifold. We employ a pixel-wise and feature-wise metric to assess the closeness of a given new lung nodule to our learned representation. After the closest image in the manifold learn is obtained, we compare it against our new data point. With this comparison, we construct a residual image which contains regions that deviate from the learned representation.

We additionally include experiments with the MNIST datasets. From which we explore how unseen numbers are mapped to the learned distribution of a generative adversarial network. Here, we split all number into two groups (0-6 and from 7-9) in order to simulate the scenario of lung nodules where we have two types of lungs: malign and benign. We train our models with group 0-6 and fit numbers from the second group to the learned representation. We use the level of fitness to the learned representation as a classification of group numbers.

Preface

The thesis marks the conclusion of my Master's project and the start of a new chapter in my career. The last couple of years have been challenging and at the same time have provided an amazing cultural experience.

Firstly, I would like to thank my supervisors dr. V. Menkovskidr. and dr. D. Mavroeidis for giving the opportunity to participate in such a interesting topic in data science. For the countless weekly meetings and feedback they provided for the realization of this project.

I also thank Stefano, Nitin, Hugo, Petar, Linas and all other friends I cross during the Master's program. For the university projects and moments we shared over the course of the last two years.

Last but not least, I would like to immensely thank my parents for pushing me to achieve more. I am truly grateful for all the support and guidance they have granted. Without them, I wouldn't be standing where I am now.

Rodrigo Mendoza
February 7, 2018

Contents

1	Introduction	6
2	Background	9
2.1	Generative Adversarial Networks	10
2.1.1	Framework	11
2.1.2	DCGAN	13
2.1.3	ACGAN	15
2.1.4	SGAN	16
2.1.5	WGAN	17
2.1.6	WGAN-GP	17
2.1.7	Inverse mapping	19
2.1.8	Tips and tricks	20
2.2	Variational Autoencoders	22
2.2.1	Mathematical derivation	23
2.2.2	Reparameterisation Trick	26
2.3	Evaluation of generative models	26
3	Approach	28
3.1	Representation Learning with GAN	28
3.2	Anomaly Detection with GAN	28
3.2.1	Unsupervised manifold learning	29
3.2.2	Mapping images to the latent space	29
3.2.3	Detection of Anomalies	30
3.3	Anomaly Detection with VAE	31
4	Experimental Setup	32
4.1	Datasets	32
4.1.1	MNIST	32
4.1.2	NLST	32
4.2	Unsupervised manifold learning	34
4.3	Feature representation with GANs	36
4.4	Anomaly detection with GANs	36

4.5	Anomaly detection with VAE	37
5	Results	38
5.1	DCGAN	38
5.2	ACGAN	38
5.3	AnoDCGAN	38
5.3.1	Unsupervised Representation	38
5.3.2	Mapping images to the latent space	39
5.3.3	Anomaly Scores	39
5.4	AnoWGAN-GP	40
5.4.1	Unsupervised Representation	40
5.4.2	Images to the latent space	41
5.4.3	Anomaly Scores	42
5.5	WGAN-GP as feature extraction	44
5.5.1	Unsupervised Learning	44
5.5.2	Feature Extraction	45
5.6	AnoVAE	46
5.6.1	Unsupervised Representation	46
5.6.2	Anomaly Detection	47
6	Conclusions	62

Chapter 1

Introduction

The domain of the project is lung-cancer screening using low-dose CT. Datasets collected for this problem are taken from large screening programs where only a small subset of the screened population is diagnosed with cancer. This scenario is constantly reflected in health-care where healthy subjects represent the majority of the dataset. Each CT scan has multiple axial tomography's of the human body which scan blood vessels, bones and soft tissues.

The lungs sometimes have small masses of tissue called lung nodules. The lung nodules can be cancerous and benign. Usually a lung nodule size is around 5-30 mm and bigger lung nodules are more likely to be cancerous. The shape is also an important factor since rounded nodules tend to be benign, whereas spiculated (spikes or points) nodules are more likely to be cancerous. Usually, cancerous lung nodules increase their size over time faster than benign ones. The challenge relies in localizing and classifying lung nodules into benign or cancerous.

A challenge when using a model for classification is feature representation. Usually raw data, e.g. pixel RGB values, does not provide meaningful information for categorizing data into different classes. Instead, one might want to spot strokes and shapes from the object. We could use human annotations to describe each image and create a subset of keyword describing the characteristics of each image. Given the limitations of time and money, human annotations is not a feasible approach. One other alternative is the use of neural networks which have shown promising results for creating hidden features. For this reason, deep learning for image analysis has proven a dominant tool in the fields of health-care, text recognition, self-driven cars, automatic game-playing and image coloring.

For pattern recognition, two common approaches are supervised and unsupervised learning. In supervised learning, we extract patterns and structure from labeled data. As opposed to unsupervised learning, where there is unlabeled training examples. Labeled images are certainly more desirable since we can optimize the patterns and structures based on the labels; however, the downside is we require high volumes and sometimes the acquisition is expensive and time consuming contrary to unlabelled image data. We would like to automatically learn to classify from image data in an efficient manner without the necessity of high volumes of labeled training data.

The promise of unsupervised feature learning is to make our predictive model learn from vast amounts of unlabeled data and reuse good intermediate representations of the images to leverage the workload for supervised image classification. As mentioned in [1], generative models can be trained in a semi-supervised manner to leverage the workload of a supervised

model to generalize to unseen data points by training on a corpus of unlabeled data. For this reason, an approach is to use semi-supervised learning with generative models where the distribution from unlabelled data comes from the same domain as the labelled data.

Assuming we have training data in the form x_1, x_2, \dots, x_n with labels y_i . A generative model would learn the joint distribution $P(x, y)$ with the ability to also generate images $P(x|y)$. While a discriminative model, would learn to categorize observations into one of the classes $P(y|x)$. There are different task for which we rather have a generative approach. For example, to mention a few of them: image to image translation [2], is able to transform gray-scale to color images and change daytime images to night time; super resolution [3], generating higher pixel density images from a low resolution photo. Imagine using a discriminative model to transform an image to another image in this scenario. How many image to image mappings we would need? What if the position of objects change from daytime to night-time?

There are different approaches to generative models. As mentioned in [1], we can group them into two categories if we assume they use maximum likelihood estimation: explicit and implicit density estimation. Explicit density estimation either use the chain rule of probability to disentangle $P(x)$ which explicitly calculates the distribution or fall in the category of approximation inference which sacrifices accuracy for computational speed by creating a lower bound over the log-likelihood (intractable density functions). On the other hand, implicit density estimation instead rely on density estimation by comparison, i.e. compare generated samples against training data and then update parameters of the model.

There are currently two dominant generative models: generative adversarial networks [4] and variational autoencoders [5]. Generative adversarial networks belong to the category of implicit density estimation. While VAE use explicit density estimation by approximating with a lower bound the log-likelihood. Our work will explore how both generative models can assist with unsupervised feature representation in the context of medical image analysis when dealing with high-dimensional spaces. Most of the interesting problem, such as weather forecast or health-care diagnose, contain high number of variables. Nevertheless, as the number of variables increases we face with the curse of dimensionality.

Generative models can be compared qualitatively by comparing data samples generated. The work from [6, 2] have used human trials where the people had to differentiate between real and fake images. Additionally [6] proposed the inception score metric as an alternative to human annotations, which uses the pre-trained inception model to classify the generated images from a generative model into one of the classes of the inception model. If the generated images are real enough, then the model will classify them with high confidence to one of the classes. There are other approaches which rely on the same idea such as [2] with FCN-score. Different generative models do not share the same loss function, therefore we couldn't just compare them by relying on the loss. The ideal scenario would be to find a way to score the realness of the images while tracking the variability of the images. As a result, comparing different generative models is still a hard problem.

Generative adversarial networks were recently developed and have had a huge popularity for unsupervised learning representation, generation of synthetic data, conversion image to image, and avoiding large sums of labeled data, to name a few. Currently, there are above 180 generative adversarial network and for a complete list of the GANs zoo refer to [7, 8]. For this reason, there is not only one model we can plug and play. In chapter 2, we cover the main ingredients of a GAN architecture which we will refer as the vanilla GAN model. Afterwards, we cover specific variations of GANs which we consider useful for avoiding large

sums of labeled data.

Generative adversarial networks in the medical image analysis have been used for diverse applications such as monitor of disease progression [9], brain lesion segmentation [10], detection of prostate cancer [11], predicting CT images from MRI images [12], translation from a tree image to a retinal image [13] and medical image segmentation [14]. In the medical field, we have highly unbalanced data where the majority of the dataset belongs to healthy subjects. As mentioned by Andrew Ng. in his coursera course [15], one approach for highly unbalanced data is to change a classification problem into an anomaly detection problem. AnoGAN [9] has explored a mixture between anomaly detection and generative adversarial network with applications to retina image data.

Anomaly detection learns a distribution over the training set \mathcal{X}_{train} and determines how much a new data point deviates from the learned representation. We would like to setup a model which creates a boundary between healthy and disease lung nodules and at the same time visualize which regions are considered abnormal. The approach is to learn the manifold of healthy subjects and whenever a new data sample is presented decide if it belongs to the learned representation of healthiness. The challenge relies first in creating a *good* representation, which usually converts a high dimensional space, e.g. images, into a lower dimensional representation. Once we have such representation, we would need a technique to map new data samples into the representation, which we will refer as mapping to the latent space. Finally, a metric is applied in the latent space to determine the closeness of unseen data to the learned manifold of healthy subjects.

A common approach for anomaly detection is to use one-class SVM [16] to learn a frontier delimitation around some subspace. Other techniques that rely on neural networks are based on reconstruction thresholds as presented in [17]. Additionally, convolutional autoencoders have been explored in [18] for medical image data. Other methods, use multivariate Gaussians to estimate μ and σ for each of the variables and then determine if a new data point is an outlier. Most of the methods presented use a discriminative approach which infer the target values based on the inputs without being able to generate unseen data.

In this work, we create a pipeline for anomaly detection based on two generative models (generative adversarial networks and variational autoencoders), explore the latent space using pixel-wise with semantic-wise metric and construct a GAN architecture which is able to generate diverse synthetic lung nodules samples. With GANs we explore two approaches: (1) We implement based on the ideas from AnoGAN[9] three steps: unsupervised representation of *normal* data, mapping images to the latent space and scoring the level of abnormality when applied to new data samples. (2) Unsupervised-supervised, which contains two steps: learn a unsupervised representation model trained with both healthy and unhealthy lung nodules; afterwards, train a supervised classifier using the representation transformation learned in the previous step. With VAE we train a model only with healthy training samples and use the reconstruction loss as a measure of abnormality.

First, we introduce the framework of GANs alongside VAE and cover the evaluation of generative models in chapter 2. Our approach, presented in chapter 3, covers manifold learning, mapping images to the latent space and detection of abnormalities. For our experiments, in chapter 4, we choose a benchmark dataset MNIST and the national lung screening trial (NLST) dataset. Chapter 5 presents the output of our experiments and compares different results. We draw conclusions from our results in chapter 6 and additionally include future improvements and avenues worth researching.

Chapter 2

Background

We have a dataset in the form x_1, x_2, \dots, x_n under certain distribution P_{Data} with labels y_i . We would like our model P_{model} to output the most likely class y_i given a new data point. One approach, is learning to discriminate between data points for different classes by using some boundary representation and given the most likely class $P(y|x)$. The other approach, instead learns how the data was generated $P(x, y)$ and uses Bayes rule to obtain $P(y|x)$.

For example, suppose we would like to distinguish rock and jazz audio. We could spend 10 years practicing music until we learn how to produce jazz and rock songs. Then, when we hear a new track we would match precisely to our knowledge of what makes a track being jazz or rock. The other approach we could use is start hearing thousands of tracks (jazz and rock) until we can classify a new track. The first approach is learning directly $P(x, y)$ and at the same time is able to create jazz or rock with $P(X|y)$ where X is the track and y is the gender. The later approach focuses on choosing the music gender given an audio track $P(y|x)$.

Both approaches are commonly used in the machine learning community. When working directly with $P(y|x)$, we call it a discriminative model. Opposed to generative models which give an estimate of P_{Data} . There are different types of generative models. The work from [19] categorizes generative models in prescribed and implicit models.

Prescribed models create an explicit estimate of the log-likelihood and most of the machine learning models fall in this category. Implicit models instead define a strategy for assessing the generation of data by using likelihood-free inference. The likelihood-free scenario learns by comparing sample from the model against samples of the real distribution. This process is called density estimation by comparison.

As mentioned in [1], when using explicit density estimation we can group generative models in tractable density or approximation density. Tractable models use chain rule over a set of variables using the conditional probability. Models based on density approximation, create a lower bound to maximize the likelihood.

Having a log-likelihood allows us to select the parameters which maximize the log probability of our model using training samples

$$\hat{\theta} = \operatorname{argmax}_{\theta} \mathbb{E}_{x \sim P_{data}} \log P_{model}(x; \theta)$$

Opposed to likelihood-free, which trains the parameters based on model sampling. There are four avenues for training in likelihood-free scenario covered in [19]: class-probability

estimation, divergence matching, ratio matching and moment matching. Class-probability estimation creates a classifier to distinguish samples from the generator model against observations from the training set. Divergence matching works with density dissimilarity between $D_f(P_{data}||P_{model})$ and uses one of the f-divergence (e.g. KL-divergence or total variation) to measure the divergence between two probability distributions. Ratio matching estimates the ratio of densities using Bregman divergence. Moment matching equates samples moments against theoretical moments, it minimizes the divergence of both moments.

There are different techniques for estimating P_{data} as previously mentioned. Generative models sample data from P_{model} and have the option of defining an explicit or implicit density estimation. For example, a generative adversarial network [4], which uses implicit density estimation can be converted into prescribed model by using f-divergences as shown in [20]. One of the main advantages of implicit density estimation is the absence of inferring latent variables, i.e. hidden variables inferred from observations.

Currently there are two dominant generative models: variational autoencoders [5] and generative adversarial networks [4]. Variational autoencoders are explicit density estimation based on log-likelihood approximation which uses latent variables for the generator (decoder). Generative adversarial networks are part of implicit density estimation and are trained using class-probability estimation. In this chapter, we cover the framework of both generative models and finalize on how to evaluate such generative models.

2.1 Generative Adversarial Networks

Generative adversarial networks [4], a.k.a. GANs, is a structured probabilistic generative model based on a counterfeit game between two players: discriminator and generator. The generator creates samples which try to fool the discriminator. While the discriminator, decides if the sample are real or fake. Suppose, we train a generator to learn a family of cosine and sine functions; then, the discriminators job would be to decide if such artificial waves are cosine/sine.

Usually, two neural networks take the place of the discriminator D_θ and generator G_ϕ . The loss of discriminator \mathcal{L}_D uses both players parameters and only updates θ_D . Reciprocally, generator loss \mathcal{L}_G only updates parameters ϕ_G while using both players parameters. The stopping criteria is the Nash equilibrium where \mathcal{L}_D has a local minimum with respect to θ_D and \mathcal{L}_G with respect to ϕ_G .

For the **generator**, we define input vector z with a pre-defined distribution $p(z)$ and pass it through a differentiable function $G_\theta : z \rightarrow X$. As mentioned in [1], there is no restriction on the positioning of z , i.e. not necessarily need to be as input all the variables of the vector z . We can split z into z_1 and z_2 and use each z_i as input for certain layer of G_θ by using merge techniques, such as, multiplicative or additive.

The **discriminator**, is a normal classifier with input $X \leftarrow X_{Real} + X_{Fake}$ sampled from real data $X_{Real} \sim P_{Data}$ and fake data $X_{Fake} \sim G(z)$ which is trained to distinguish real and fake samples. There are variants for the input during training, some consider using first only a batch of real images X_{Real} and then using a batch containing only X_{Fake} ; others, keep a mixture of real and fake images.

The process for training has two steps: (1) We sample real and fake data x and pass through $D(x)$, where the goal of the discriminator is to learn what is real. The real images

labels have 1s and fake images have 0s. The discriminator wants to be near 1. (2) Sample a batch of images from the generator $G(z)$ where $z \sim p(z)$ and label all images with 1s. We know the images are fake (first step had fake images with label 0), but we want to fool the discriminator. The goal of the generator is to stay close to real images (1s), while discriminator has the option to answer fake/real (0s/1s).

2.1.1 Framework

For the generator, let $G_\phi : z \rightarrow X$ be a differentiable function with prior distribution $z \sim p(z)$ with parameters ϕ . The discriminator is a differentiable function $D_\theta(x)$ which returns the probability that x comes from the true distribution P_{Data} . The interaction between generator $G(z)$ and discriminator $D(x)$ is a minimax game defined in [4] as

$$\min_G \max_D \mathbb{E}_{x \sim \mathbb{P}_r} \log(D(x)) + \mathbb{E}_{z \sim \mathbb{P}_z(z)} \log(1 - D(G(x))) \quad (2.1)$$

There are different approaches to train the game. One approach as stated by [4], is to take several steps to train the discriminator D and then train with low learning rate the generator G in order to avoid overfitting the discriminator. A discriminator trained fully first would reject mostly all samples from the generator. In early stages of training, $G(z)$ creates noise images which discriminator $D(G(z))$ can reject with high confidence. For this reason, some GAN architecture have $-\mathbb{E}_{z \sim \mathbb{P}_z(z)} \log(D(G(x)))$ instead of $\mathbb{E}_{z \sim \mathbb{P}_z(z)} \log(1 - D(G(x)))$ for the second term in equation 2.1.

Transposed Convolution

One common layer in the architecture is fractional strided convolution, also known as transposed convolution, and referred wrongly as "deconvolution". We can find the layer in Keras [21] as 'Conv2DTranspose'. The goal is to go in the opposite direction of a normal convolution. With a normal convolution layer we reduce the spatial resolution, opposed to fractional strided convolution which increases the spatial resolution:

- Suppose we have an input image (1 channel i.e. grayscale) with shape 8x8 and if we apply a 5x5 filter with stride 1, the output would be a 4x4. How would be able to go the opposite direction? One approach would be to add padding with zeros to our image of 4x4 and apply a filter to get back the 8x8 shape. In our example, if we add 4 zero padding around the image and apply a filter of 5x5 with a stride of 1 we get an image of 8x8. The problem with this approach is the high count of zeros.
- A small example on how to revert the process of a convolution is shown in Figure 2.2 which is based on the work from [22].
- An insightful visualization animation can be found at [23] which shows how a transposed convolution can be generated using a normal convolution.

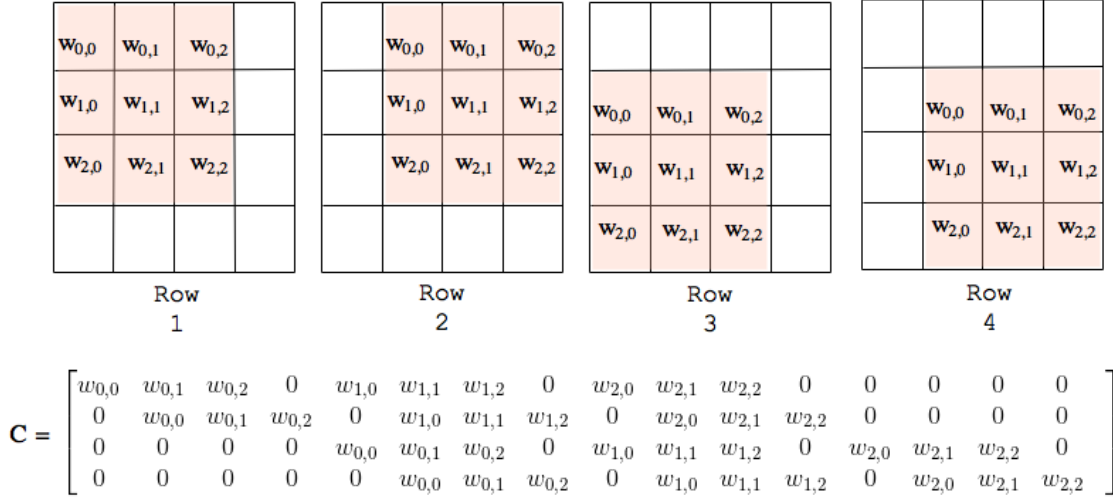


Figure 2.1: The top four blocks animate the process of applying convolution to an image of 4x4 with kernel 3x3 over a stride of 1. This process can be represented as a sparse matrix shown in the bottom row. Each row of the sparse matrix is a reflection of each convolution step. As we see, the rows are a flattened version of each convolution step. The size of the sparse matrix is 16x4. The convolution process creates an output image of size 2x2 which we can flatten into 1x4 representation. Our goal is to revert the process by going from 2x2 into 4x4. In order to recover the original image of size 4x4, we need to multiply C^T by the flattened output (16x4 x 4x1) to produce a matrix of size 16x1. The output matrix of shape 16x1 can be reshaped into 4x4.

Architecture

The generator from a GAN architecture requires to produce images of the same shape as the ones from P_{Data} . Common techniques such as grid search in deep learning would stack and remove convolution layers for tuning the parameters of a model. However, when using up-sampling techniques in the context of GANs, we have to stack a specific number of upsample layers in order to guarantee a desirable output size. The work from [24] has presented some architecture topology guidelines from which we use as baseline for construction our models. In this section, we present an approach for determining the number of up-samples required in the generator $G(z)$ when using transposed convolutions.

The idea is to think the generator has 6 sections as shown in Figure 2.2. We design the model from right to left:

1. Choose size of image, lets assume we use shape 3x32x32 (channels x width x height)
2. Define number of transposed convolutions, assuming a stride of 2 then one transposed convolution before the image would be $f_i \times 14 \times 14$ where f represents the filters for the layer i .
3. Optionally add more transposed convolution layers, an additional previous layer with stride 2 would make a shape of $f_{i-1} \times 7 \times 7$.
4. Reshape step, create a reshape layer with target shape $f_{i-1} \times 7 \times 7$. Which is using the same 7x7 as the next feature layer.

5. Dense layer before reshape, the number of filter must be $f_{i-1} * 7 * 7$.
6. Dense layers after input, we stack zero or more dense layers.
7. $P(z)$, prior distribution over z .

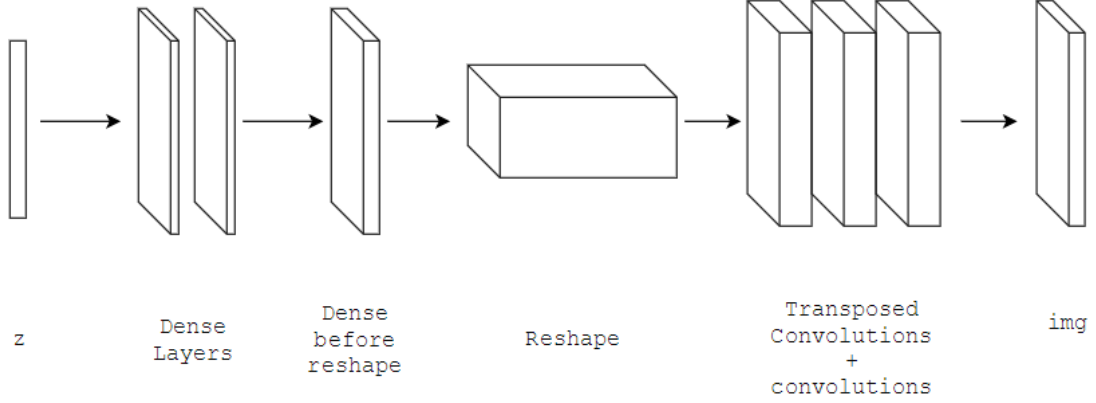


Figure 2.2: A topology architecture of a generative model which uses transposed convolutional layers. There are 6 sections labeled below each stack of layers.

Batch Normalization

Batch normalization [25] is a method for improving the stability of training neural networks by avoiding covariate shifts and is presented after each convolution layer. The BN creates unit Gaussian inputs for other layers by normalizing the features across the batch. The idea is to calculate batch mean μ_β , and variance σ_β^2 , then normalize using

$$x_i = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}}$$

where ϵ is a small constant. Afterwards, we scale and shift with $y_i = \gamma x_i + \beta$. The model can learn to undo the batch normalization if needed by setting $\gamma = \sqrt{\sigma_\beta^2 + \epsilon}$ and $\beta = \mu_\beta$. As result, even when adding batch normalization, the model can undo the process by learning an inverse transformation. As a side note, during testing phase, we do not want μ_β , σ_β^2 to change over time in order to have a deterministic output.

2.1.2 DCGAN

Deep convolutional generative adversarial networks, DCGAN [24], is an adversarial network based on unsupervised learning. The model creates from high-dimensional data a representation in lower-dimensions. The features learned are used to represent more meaningful variables of images rather than using pixels. Once the representations are created, we can train a classifier using the features. The work from [24] covers three main aspects: supervised learning with unsupervised pre-training, stable architecture using convolution which does not

collapse the generator into producing noisy images and perform arithmetics over the latent space.

The generator from the model uses vector z as input, then they project to a stack of dense layers followed by a reshape. The reshape is done in order to prepare the input for the convolution layers. From this point, they stack fractionally-strided convolutions until obtaining the shape of the image. The model follows the same vanilla GAN architecture as presented in Figure 2.3 with the difference of fractional-strided convolutions.

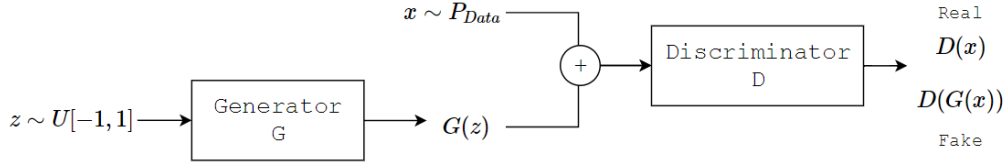


Figure 2.3: DCGAN graph structure with two neural networks: generator and discriminator. The loss is binary cross-entropy.

For completeness, we include the recommendations from DCGAN [24]:

1. Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator). The fractional-strided is a convolution followed by an upsampling.
2. Use batch normalization in both the generator and the discriminator.
3. Remove fully connected hidden layers for deeper architectures. Probably the authors were suggesting the use of convolution layers with dropout.
4. Use ReLU activation in generator for all layers except for the output, which uses Tanh.
5. Use LeakyReLU activation in the discriminator for all layers.
6. Apply adam optimizer with learning rate 0.0002 and momentum 0.5.
7. Initialize weights from a zero-centered Normal distribution with standard deviation 0.02.
8. Slope of LeakyReLU 0.2 for all models.

One approach for supervised learning from an unsupervised model is to use the unsupervised model as feature representation. The feature representations are obtained by passing training data x_1, x_2, \dots, x_n until one or more layer of the discriminator f_i . As a result, we obtain one or more vectors in the form $f_1(x_i), f_2(x_i), \dots, f_n(x_i)$ which we concatenate to obtain a single vector representation $f_R(x)$. We can now train a classifier with features $f_R(x)$ and labels y .

The unsupervised-supervised approach was explored in [24] using DCGAN+L2-SVM. Where the unsupervised model is DCGAN and the supervised model is L2-SVM. The authors from [24] compared the model against state of the art discriminative classifiers $P(y|X)$; however, the model did not outperform state of the art. The benefit from such experiment

was creating a classifier with only 512 features compared to exemplar CNN which is using 1024 and creating a framework which leverages the necessity of labeled images.

The latent space is also explored by doing interpolations using the generator and visualizing the features of the discriminator. The interpolation is done by selecting two seeds z_1, z_2 constructing the images $G(z_1), G(z_2)$ and doing interpolation, e.g. linear, between z_1 and z_2 . The interpolation generates $z_1, z_i, z_{i+2}, \dots, z_{i+n}, z_2$ where n represents that total of evenly spaced vectors over a specified interval. For each of the interpolated vectors, we pass them by the generator $G(z_i)$ and visualize the transitions. If we are able to do interpolation in the latent space, then we can speculate the model has learn some meaningful representations. On the other hand, the visualization of the learned discriminator features are done by using guided back-propagation to highlight activations from different layers.

2.1.3 ACGAN

Auxiliary Conditional Generative Adversarial Network, ACGAN [26], is a supervised generative adversarial model which introduces two modifications: (1) the generator input vector z has labels included (2) Discriminator has two losses: one to discern between fake/real and the other for classification with respect to the labels of the images. Since the class is a discrete variable and the vector z is a continuous vector, we can use embedding layers as the input for the generator. The model architecture is shown in Figure 2.4.

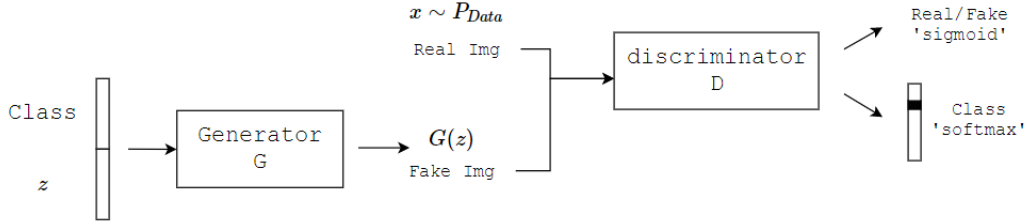


Figure 2.4: ACGAN architecture.

As mentioned in ACGAN [26], the vanilla GAN discriminator is maximizing the log-likelihood the discriminator attributes to real and fake data, e.g. binary cross-entropy

$$L = E[\log P(S = \text{real}|X_{\text{real}})] + E[\log P(S = \text{fake}|X_{\text{fake}})]$$

while the generator is minimizing the second term $E[\log P(S = \text{fake}|X_{\text{real}})]$ to reduce the number of fake samples. Whereas in ACGAN [26], the model has two losses: fake/real loss L_s , which deals with distinguishing real from fake and classification loss L_c which works as a categorical classifier. The loss for each component follows,

$$\begin{aligned} L_s &= E[\log P(S = \text{real}|X_{\text{real}})] + E[\log P(S = \text{fake}|X_{\text{fake}})] \\ L_c &= E[\log P(C = c|X_{\text{real}})] + E[\log P(C = c|X_{\text{fake}})] \end{aligned}$$

where discriminator maximizes $L_s + L_c$ and generator is maximizing $L_s - L_c$. Since the model has two losses, we need true labels for distinguishing real from fake and for classification. This changes the pipeline of only using X for training as in DCGAN.

For training the discriminator, we construct a triplet $X, Y_{F/R}, Y_{aux}$ which has data, fake/real labels and classification labels respectively. Let X represent the subset composed by real $X_{real} \sim P_{Data}$ and fake data samples $X_{fake} \sim G(z)$. Fake/real labels $Y_{F/R}$ contains 1s for real and 0s for fake. Classification labels Y_{aux} has class labels C_i for each X_{real} and a randomly uniform class set for each X_{fake} .

For training the generator, we construct triplet X, Y_R, Y_{aux} which has data, real labels and classification labels. Let X be sampled from $z \sim P(z)$. The fake/real labels Y_R contain only 1s. The classification labels are uniform random classes C_i . When training on batch, is important to maintain the same size X for both generator and discriminator; for example, the batch of real images was 25 and fake images was 25, then the generator should be trained with z_1, z_2, \dots, z_{50} where $z_i \sim P(z)$.

The authors from [26] ran experiments using the ImageNet dataset. There are around 1000 classes in such dataset. The assumption was that a GAN train over a few classes would be able to increase the quality of images. For this reason, the author created multiple ACGAN trained with subsets of only a few classes. The results showed that training a model with higher count of classes repercussions on the quality of images produced by the generator.

2.1.4 SGAN

Semi-supervised generative adversarial networks, SGAN [27], is an middle point between fully unsupervised (e.g. DCGAN [24]) and supervised generative models (e.g. ACGAN [26]). The idea is to transform the output of the discriminator $[Real, Fake]$ into $[C_1, C_2, \dots, C_n, Fake]$ where C_i represents a class label. Real Images $X_{Real} \sim P_{Data}$ have labels C_i , while images from the generator $X_{Fake} \sim G(z)$ have labels $Fake$. The architecture of the models is shown in Figure 2.5.

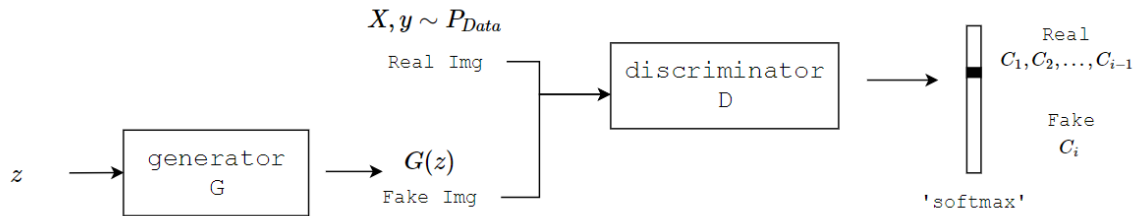


Figure 2.5: SGAN architecture

The work from SGAN [27] compares an isolated classifier $P(C_i|X)$ against a SGAN model. While leaving the weights of the generator static, as less data is used, SGAN outperforms the isolated classifier. The inclusion of the generator is actually helping the role of the discriminator in the SGAN model. The discriminator of the SGAN is doing classification of labels and discerning fake from real. This experiment shows that merging classification with discrimination of fake/real actually improves the model.

The authors also propose two insightful avenues to explore: (1) convert the discriminator output to $2N$ outcomes with the form of $[F_1, F_2, \dots, F_i, R_1, R_2, \dots, R_j]$ where F are consider fake classes and R real classes. (2) instead of fully sharing the weights between discriminator and classifier, find some middle ground where two nets share some weights but not all of

them. This partial sharing would allow some neurons to specialize in discriminating fake/real and other into classifying.

2.1.5 WGAN

Wasserstein generative adversarial network, WGAN [28], is a variation of GANs which explores different divergences when comparing two distributions $D(P_r, P_g)$. There are different divergence we can use for comparing two distributions such as Kullback-Leibler divergence, total variation distance, Jensen-Shannon divergence or Earth Mover distance. All the group of divergences can be categorized as f -divergence or as $D_f(P_r||P_g)$. The work from WGAN [28] focuses on the earth mover distance or 1-Wassertein.

The earth mover distance is inspired in moving piles from one place to another times the distance. By moving horizontally with earth distance we are comparing two distributions on axis-x (comparing vertically would be the difference over two probability functions). The EM distance can be formulated as a transportation plan with a cost function and constrains. The authors from [28] showed that there are sequences of distributions for which the EM-distance finds a solution and the other sequences, such as KL or JS divergence, do not.

The architecture of the model stays the same as shown in Figure 2.3, however there are three main changes to the original framework of a generative adversarial network to convert a vanilla GAN into WGAN:

- Loss function: instead of using the earth moving distance directly due being intractable, the authors use the Kantorovich-Rubinstein duality. The loss function for the discriminator becomes $\frac{1}{m} \sum_{i=1}^m D(x) - \frac{1}{m} \sum_{i=1}^m D(G(z))$ and the loss for the generator is now $-\frac{1}{m} \sum_{i=1}^m D(G(z))$ where $\{x\}_{i=1}^m \sim P_{Data}$ and $\{z\}_{i=1}^m \sim P(z)$. Both losses reduce the mean without having logarithm.
- The output from the discriminator does not have activation due absence of the log in the loss, e.g. remove sigmoid of last layer. Because it is no longer a probability output, the authors find it more suitable to call the discriminator a critic.
- Weight clipping: enforce 1-Lipschitz over the weights of the discriminator by clipping the values in a range $[-c, c]$ after being updated using RMSProp.
- Training steps: the discriminator trains n_{critic} steps for one of the generator.

The main contribution from WGAN [28] is the comparison between divergences and showing why the earth moving distance is a strong candidate. By using this type of distance and enforcing weight clipping in the optimizing method, the model shows empirical results of correlation between better images as the loss tends to drop. Some of the problems of this model are addressed at [29], which focuses on the problems of weight clipping to enforce 1-Lipschitz showing a pathological pattern, because the weights start pushing in one direction and clipping them back creates loops.

2.1.6 WGAN-GP

WGAN-GP [29] stands for Wasserstein generative adversarial network with gradient penalty which is an improved version of WGAN [28]. Wasserstein is a synonymous to the earth mover

distance, a metric to compare divergences between two distribution by using distance and mass into account, e.g. in a discrete scenario the movement between two bins is the distance and moving x elements from that bin is the mass. The improved version removes weight clipping in the discriminator and introduces gradient penalty as an alternative to enforce 1-Lipschitz. The gradient penalty penalizes the gradient of the critic (discriminator) with respect to the input instead of using weight clipping as in [28].

The earth mover distance

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y) \sim \gamma} \|x - y\|$$

is over the infimum over all joint distributions $\Pi(P_r, P_g)$ containing real P_r and generated P_g distributions. For being intractable, we use the Kantorovich-Rubinstein dual form

$$W(P_r, P_g) = \sup_{\|f\|_{L \leq 1}} \mathbb{E}_{x \sim P_r} f(x) - \mathbb{E}_{x \sim P_g} f(x)$$

which seeks the smallest difference between the expectation of two distributions with a 1-Lipschitz function constrain $\|f\|_{L \leq 1}$. A K-Lipschitz function enforces the slope, i.e. the first derivative, to be bounded by k such that

$$f(x_1) - f(x_2) \leq K(x_1 - x_2)$$

Suppose we have $f(x) = 7x$ and we want to enforce 1-Lipschitz. We can create another function $g(x) = \frac{1}{7}f(x)$ to enforce a slope of 1 by dividing by $\frac{1}{k}$. We can think of the constant $\frac{1}{7} * 7$ as a trainable parameter θ which absorbs the value of k . In practice, we only care of the value $\frac{1}{7} * 7$ and not the exact value for k . From this idea, the smallest difference between the expectation of P_r, P_g over k -Lipschitz is a factor of K :

$$KW(P_r, P_g) = \sup_{\|f\|_{L \leq k}} \mathbb{E}_{x \sim P_r} f(x) - \mathbb{E}_{x \sim P_g} f(x)$$

Previously, the work from [28] used weight clipping to constrain 1-Lipschitz function. We have some parameter θ from a function which we update $f(x) = \theta x$ and one hotfix to ensure 1-Lipschitz is to clamp the weight θ back within a box $[-c, c]$. The weight clipping approach was discouraging for presenting pathological patterns, i.e. pushing weights to the clipping values, and vanishing/exploding of gradient norms as addressed in [29].

To avoid the problems presented with weight clipping, the authors from [29] present an alternative called gradient penalty. The idea is to calculate the gradient norm of critic (discriminator) and penalize if it's not 1. The reasoning behind GP is that a differentiable function is 1-Lipschitz if and only if it has gradient norm ≤ 1 as proven in [29] showing

$$\mathbb{P}_{(x,y) \sim \Pi} \left[\|\nabla f^*(x_t)\| = \frac{\|y - x_t\|}{\|y - x_t\|} \right] = 1$$

where f^* is a differentiable function, $x_t = tx + (1-t)y$ with $0 \leq t \leq 1$, $y \sim P_g$, $x \sim P_r$ and the assumption of $\pi(x = y) = 0$ to avoid having exact overlap when x is itself x . Evaluating over all values for t is intractable. For this reason, the authors from [29] propose a soft version where they choose a random cut $\epsilon \sim U[0, 1]$ as a replacement for t .

Now that we have a technique to enforce 1-Lipschitz, we can move to the construction of the gradient penalty. The gradient penalty $\lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}} [(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]$ has parameter λ to control how much weight is given and in [29] they use a value of $\lambda = 10$. The real images $x \sim \mathbb{P}_r$ and generated images $\tilde{x} = G(z)$ form the input $\hat{x} = \epsilon x + (1 - \epsilon)\tilde{x}$ to the gradient penalty. L_2 norm is applied to the gradient $\|\nabla_{\hat{x}} D(\hat{x})\|_2 = \sqrt{\sum_{i=1}^n \nabla_{\hat{x}} D(\hat{x})^2}$; and rather than a fixed value for ϵ , the algorithm uses a uniform distribution $\epsilon \sim U[0, 1]$.

For the calculation of the gradient penalty, we require only two batches of real and generated samples and $\epsilon \sim U[0, 1]$. As mentioned in [29], the gradient penalizes each input by itself; therefore, we should not use batch normalization and the authors encourage rather using layer normalization as presented in [30]. Figure 2.6 presents the overview architecture of the WGAN with gradient penalty.

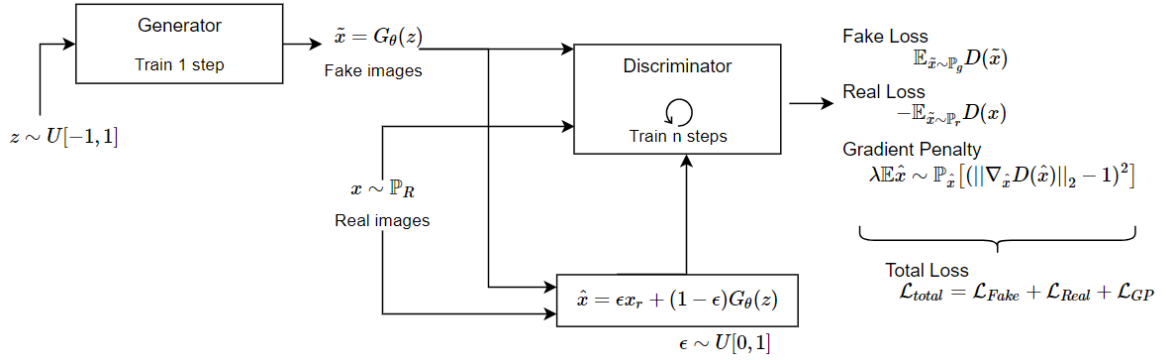


Figure 2.6: WGAN-GP architecture where generator and discriminator are two neural networks. The generator receives a vector from the distribution of $P(z)$ to construct fake images $G(z)$. The discriminators input are fake images $G(z)$, real images $x \sim P_{Data}$ and random interpolated images \hat{x} . The total loss is made of real/fake loss, which is common across GANs, and the gradient penalty loss.

2.1.7 Inverse mapping

So far, the vanilla GAN has two components: generator and discriminator. The generator creates from a vector representing the latent space a collection of images. We can sample from $P(z)$ and pass it through G_θ to produce images. The capacity of the generator is almost always limited to one direction $z \rightarrow G(z)$. The inverse process $G(z) \leftarrow z$, which we will call mapping images to the latent space, is desirable because: (1) we can explore what is similar in the spatial representation, e.g. train a GAN with faces, construct the vector representation of our face $G(z) \rightarrow z$ and do small modification to z in order to find a double of a living person (2) the latent space is a compressed representation of our data (3) small modification of the latent vector representation allows us to generate more synthetic data.

Mapping images to the latent space is a hard problem in the generative adversarial framework $G(z) \rightarrow z$. There are two approaches for solving this problem: either we include in the training pipeline the learning process of $G(z) \rightarrow z$ or we modify the input z w.r.t a loss function and search in the manifold \mathcal{X} the closest image. We will refer to the first approach as explicit mapping and the later as implicit mapping.

Examples of implicit mapping: inpainting [31], where the authors are able to reconstruct

missing parts of the images by blending the closest image in the latent space and the image with missing parts; another example, in the medical field, AnoGAN [9] where the authors are able to trace disease progression by mapping abnormal images back to the latent space of a model trained with only one-class data. Both approaches use a different loss for learning an unsupervised manifold and mapping images back to the latent space. With implicit mapping technique, we can build on top on a pre-trained GAN models. The mapping is stochastic because of the initial random seed z in the learned space and the back-propagation method.

With explicit mapping, models learn $G(z) \leftarrow z$ as part of the training process, some examples are: merging VAE and GANs as shown in [32], adversarial learned inference AliGAN [33] and bidirectional BiGAN [34]. With this type of model, we include as part of the training the process of learning $E(z|X)$. Mapping images to the latent space is faster with explicit mapping compared to implicit and have more deterministic mapping to the latent space since we only pass the image x through some encoding function $E(z|X)$.

We present in Figure 2.7 an explicit mapping BiGAN[34] and implicit mapping in-painting [31]. Both architectures are different approaches to produce the inverse mapping $G(z) \rightarrow z$.

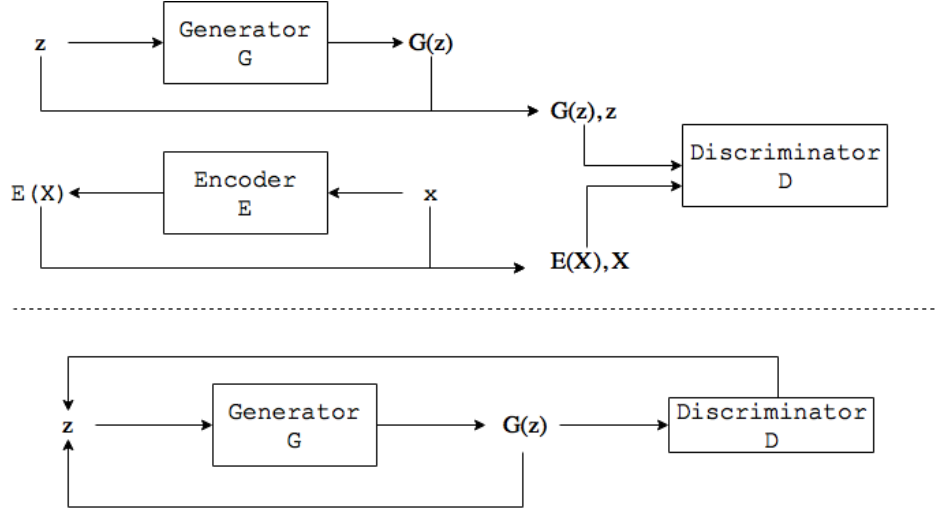


Figure 2.7: The model on the top is a bidirectional gan [34] which has trained a mapping to the latent space. The bottom model is an example of the architecture used for in-painting [31], which updates z using back-propagation with the feedback from the generator and discriminator.

2.1.8 Tips and tricks

A collection of techniques from different resources for improving the stability of training. We encourage to keep track of [35], which contains a list of tips and trick for training a GAN.

How to avoid mode collapse?

Mode collapse is the effect produced by the generator G_θ to produce a single point to fool the discriminator with high confidence. There is also partial collapse which creates a constant repetition of an image subset. The goal is to increase diversity of images produced by the

generator and avoid the generator from overtraining on the discriminator. In figure 2.8, we present the effect of partial collapse.



Figure 2.8: The left image shows more variability, while the right images presents partial collapse on different numbers. Both sets of images where generated using different ACGAN models.

When training on batch the generator, we use tuple $\{X, y\}_{i=1}^m$ where $X \sim G(z)$ and $y = \{1\}_{i=1}^m$. The predictions $\hat{y} = D(G(z))$ are either 1s or 0s. If $\hat{y} = 0$, then the generator has not fooled the discriminator which allows the generator improve. However, when $\hat{y} = 1$ the generator has fooled the discriminator making the generator preserves its state. One possible solution, is to use **label smoothing** as presented in [35, 6], which replaces the target values of 0s and 1s for smoothed values such as 0.92 or .03. There are other two methods presented in the work from [6] that encourage diversity such as feature matching and mini-batch discrimination.

Feature matching is a new objective function for the generator which matches statistics of a layer from the discriminator across batches of real and fake images. To create feature matching, we pass the output of the generator $G(z)$ through the discriminator until some layer f obtaining $f(G(z))$ and compare with a distant metric against $f(X_{Real})$. The distant metric used in the paper is L2-loss squared:

$$\|\mathbb{E}_{x \sim p_{data}} f(x) - \mathbb{E}_{x \sim p_z} f(G(z))\|_2^2 = \sqrt{\sum_{i=1}^N (\mathbb{E}_{x \sim p_{data}} f(x) - \mathbb{E}_{x \sim p_z} f(G(z)))^2}^2 \quad (2.2)$$

$$= \sum_{n=1}^N (\mathbb{E}_{x \sim p_{data}} f(x) - \mathbb{E}_{x \sim p_z} f(G(z)))^2 \quad (2.3)$$

$$= \sum_{n=1}^N \left(\frac{1}{n} \sum_{i=1}^N f(x_i) - \frac{1}{n} \sum_{i=1}^N f(G(z_i)) \right)^2 \quad (2.4)$$

Now, we can reduce the mean from equation 2.4 as the new cost function for the generator.

Minibatch discrimination is a loss for the generator which encourages samples to be more dissimilar. The idea is to have features f_i from an intermediate layer from the discriminator for each sample x_i across the batch. Then, use closeness batch $c_b(f(x_i), f(x_j))$ as presented in [6] to measure how likely two samples are. The total cost for each image x_i

is the sum against all other images $\sum_{j=1}^n c_b(f(x_i), f(x_j))$. As mentioned in [6], minibatch discrimination has had more success when the goal is to create appealing images. Opposed to feature matching, which works much better when the goal is classification.

Checkerboard effect

The checkerboard effect is a repetitive grid of colors presented on the synthetic images from the generator $G(z)$. The pattern is more notorious when zooming over the image or when seeing the images generated during the first epochs of training. This effect is not specifically related to the GAN architecture as shown in [36].

The generator usually goes from a low dimensional to a higher dimensional space, such as images, by using upsampling techniques. For the case of transposed convolution, when the kernel is not divisible by the stride some pixels are covered more than once when moving the kernel. In practice, the neural network could learn in theory to undo the overlap painting. Nevertheless, this is one of the reasons for checkerboard effect.

The work from [36] suggest 3 approaches to deal with the checkerboard effect: (1) Use transposed convolution with kernel size divisible by the stride to avoid overlap. (2) Allow the neural network to learn an evenly balanced output (3) Change upsampling technique, e.g. nearest-neighbor interpolation or bi-linear interpolation, and use convolution afterwards.

2.2 Variational Autoencoders

A probabilistic model works with the joint distributions of variables x and hidden variables z , $p(x, z)$. When doing inference about the hidden variables $p(z|x)$, we use the posterior $p(z|x) = p(x, z)/p(x)$. Usually the denominator $p(x)$ is intractable in high dimensional space, for this reason we use a method called approximation inference.

The idea behind approximation inference of the posterior is to start at some point in space with a family of distributions over the hidden variables and move the parameters of such distributions to get as close as possible to the posterior. How close are both distributions is measured using f-divergence, such as kullback leibler divergence. Since we do not have knowledge of the posterior itself to calculate f-divergence, variational autoencoders use the evidence lower bound. In short, approximation inference is transforming the problem of inference into an optimization problem. Section 2.2.1 presents a more elaborate formulation of the evidence lower bound (ELBO).

Variational autoencoders, also know as VAE, are generative models that fall in the category of explicit density estimation which use approximation inference. VAE are capable of generating images, however their main criticism is the generation of blurry images. The model uses latent variables z , i.e. variables that make possible x , $P_\theta(X|Z)$. Where P is usually represented by a neural network parameterized by θ and called decoder. The latent variables are hidden variables parameterized by a predefined distribution which are constructed usually also by a neural network called encoder $Q_\phi(Z|X)$.

The VAE architecture has similarity with autoencoders, however autoencoders are not generative models. The main difference between both models are: (1) Vanilla autoencoders use only the pixel reconstruction without using any f-divergence to measure latent loss. (2) VAE use the reparametrization trick to allow the gradient to flow through the network, which

is not required in autoencoders. (3) To sample data, VAE decoder uses $z \sim Q(Z|X)$ where $Z = \mu(x) + \sigma^2 \cdot \epsilon$ with $\epsilon \sim \mathcal{N}(0, I)$; whereas autoencoder, need to use encoder-decoder to generate an output.

2.2.1 Mathematical derivation

There are hidden variables z which allows the generation of data X . We would like to learn those hidden variables $P(Z|X)$ from a set of training data $\{x\}_{i=1}^m$ in order to have a meaningful representation of our data. The idea is to start at some point in space with a family of distributions over the latent variables $Q(Z)$ and get as close as possible to the exact posterior $P(Z|X)$. We can measure closeness of both distributions in multiple ways. For now, we will focus on KL-divergence metric which is presented in the original VAE work found at [5].

$$KL(Q(Z)||P(Z|X)) = \mathbb{E}_{z \sim Q}[\log Q(z) - \log P(Z|X)] \quad (2.5a)$$

What do we know from $P(Z|X)$? Applying Bayes rule we obtain:

$$P(Z|X) = \frac{P(X|Z)P(Z)}{P(X)}$$

$$posterior = \frac{likelihood * prior}{evidence}$$

Using the logarithm version,

$$\log P(Z|X) = \log\left(\frac{P(X|Z)P(Z)}{P(X)}\right) \quad (2.6a)$$

$$= \log[P(X|Z)P(Z)] - \log P(X) \quad \log(x/y) = \log(x) - \log(y) \quad (2.6b)$$

$$= \log P(X|Z) + \log P(Z) - \log P(X) \quad \log(xy) = \log(x) + \log(y) \quad (2.6c)$$

The metric from equation (2.5a) can be rewritten using (2.6c) as,

$$KL(Q(Z)||P(Z|X)) = \mathbb{E}_{z \sim Q}[\log Q(z) - (\log P(X|Z) + \log P(Z) - \log P(X))] \quad (2.7a)$$

$$= \mathbb{E}_{z \sim Q}[\log Q(z) - \log P(X|Z) - \log P(Z)] - \log P(X) \quad (2.7b)$$

Step (2.7b) takes out any term not dependant of z . If we rearrange the terms in (2.7):

$$\log P(X) - KL(Q(Z)||P(Z|X)) = -\mathbb{E}_{z \sim Q}[\log Q(z) - \log P(X|Z) - \log P(Z)] \quad (2.8a)$$

$$= \mathbb{E}_{z \sim Q}[\log P(X|Z) + \log P(Z) - \log Q(Z)] \quad (2.8b)$$

$$= \mathbb{E}_{z \sim Q}[\log P(X|Z)] + \mathbb{E}_{z \sim Q}[\log P(Z) - \log Q(Z)] \quad (2.8c)$$

Before proceeding, lets recap the definition of KL-divergence to write it as the expectation and use it in equation (2.8c). The KL-divergence between two distributions $Q(Z)$ and $P(Z)$

is

$$KL(Q(Z)||P(Z)) = \sum_z Q(Z) \log \frac{Q(Z)}{P(Z)} \quad (2.9a)$$

$$= \mathbb{E}_{z \sim Q} \left[\log \frac{Q(Z)}{P(Z)} \right] \quad (2.9b)$$

$$= \mathbb{E}_{z \sim Q} [\log Q(Z) - \log P(Z)] \quad (2.9c)$$

Now, if we continue from equation 2.8c and rewrite the term $\mathbb{E}_{z \sim Q} [\log P(Z) - \log Q(Z)]$:

$$\mathbb{E}_{z \sim Q} [\log P(Z) - \log Q(Z)] = -\mathbb{E}_{z \sim Q} [\log Q(Z) - \log P(Z)] \quad (2.10a)$$

$$= -\mathbb{E}_{z \sim Q} \left[\log \frac{P(Z)}{Q(Z)} \right] \quad (2.10b)$$

$$= -KL[Q(Z)||P(Z)] \quad (2.10c)$$

We can plug 2.10c transformation into 2.8c:

$$\log P(X) - KL(Q(Z)||P(Z|X)) = \mathbb{E}_{z \sim Q} [\log P(X|Z)] - KL[Q(Z)||P(Z)] \quad (2.11a)$$

Here $Q(Z)$ is a distribution we choose that makes $KL(Q(Z)||P(Z|X))$ small. A good option is to choose $Q(Z|X)$, because we want to create X which is dependant of z and to construct z it would be wise to be dependant on X . Plugging $Q(Z|X)$ into equation 2.12a:

$$\underbrace{\log P(X)}_{\text{Model our data}} - \underbrace{KL(Q(Z|X)||P(Z|X))}_{\text{under some error}} = \underbrace{\mathbb{E}_{z \sim Q} [\log P(X|Z)]}_{\substack{\text{Reconstruction loss} \\ \text{Pixel Difference}}} - \underbrace{KL[Q(Z|X)||P(Z)]}_{\substack{\text{Latent loss} \\ \text{Distribution of } z \text{ to be} \\ \text{as close to the prior imposed}}} \quad (2.12a)$$

evidence lower bound (ELBO)

Recall the original goal is to move $Q(Z)$ as close as possible to the true posterior $P(Z|X)$ based on the metric KL-divergence. The problem is you need the true posterior $P(Z|X)$ to compute the divergence. How do you do variational inference if you do not have the posterior itself? Use evidence lower bound, since maximizing the ELBO is the same as minimizing the KL-divergence. The evidence lower bound follows,

$$\log P(X) \geq \mathbb{E}_{z \sim Q} [\log P(X|Z)] - KL[Q(Z|X)||P(Z)] \quad (2.13)$$

Up to now, we want to model our data under some error by optimizing the ELBO. How do we compute the ELBO loss? There are two steps involved: reconstruction, measure the discrepancy between your images; latent loss, divergence between the distribution of our latent space imposed by our model $Q(Z|X)$ and some prior $P(Z)$.

The reconstruction loss $\mathbb{E}_{z \sim Q} [\log P(X|Z)]$ is maximum likelihood estimation. The input z is passed over function Q to obtain the prediction $Q(z)$ and we compare the result against the

true value X in order to maximize the conditional probability $P(X|Z)$ by moving the parameters of Q . For the latent loss $KL[Q(Z|X)||P(Z)]$ we need to choose the output distribution of our model and the prior distribution:

1. Output distribution: we choose a Gaussian distribution with mean and variance dependant on X , $\mathcal{N}(\mu(X), \Sigma(X))$. Both of the values are outputs from the encoder. We can choose other continuous distributions. The variance of a multivariate Gaussian is the covariance matrix represented as $\Sigma(X)$.
2. Prior distribution: usually selected as $\mathcal{N}(0, 1)$ for univariate or $\mathcal{N}(0, I)$ for multivariate Gaussian where I is the identity matrix. The reasoning is because any distribution in high dimensional space can be modeled by taking k normally distributed values and passing them by an elaborated function, which in our case is a neural network.

Given the selection of the two previous distributions, compute the KL-divergence between them:

$$\begin{aligned} KL[Q(Z|X)||P(Z)] &= KL[\mathcal{N}(\mu(X), \Sigma(X))||\mathcal{N}(0, 1)] \\ &= \frac{1}{2} \sum_k (\exp(\Sigma(X)) + \mu^2(X) - 1 - \Sigma(X)) \end{aligned}$$

The visualization of the previous equations is shown in Figure 2.9. There is still one concept we need to introduce called the reparameterisation trick (center box of Figure 2.9), which transforms a non-uniform random variable into a deterministic function $z = \mu(X) + \sigma^2 \circ \epsilon$ where another variable ϵ has its randomness $\epsilon \sim \mathcal{N}(0, I)$. The reparameterisation trick is a technique referred as one-liners [37, 38]. Why we need this? Because, when doing back-propagation we can not have stochastic nodes floating in the middle: all nodes must be deterministic. By making ϵ an input, the back-propagation can flow through the neural network. For more detail on the reparameterisation trick, refer to section 2.2.2.

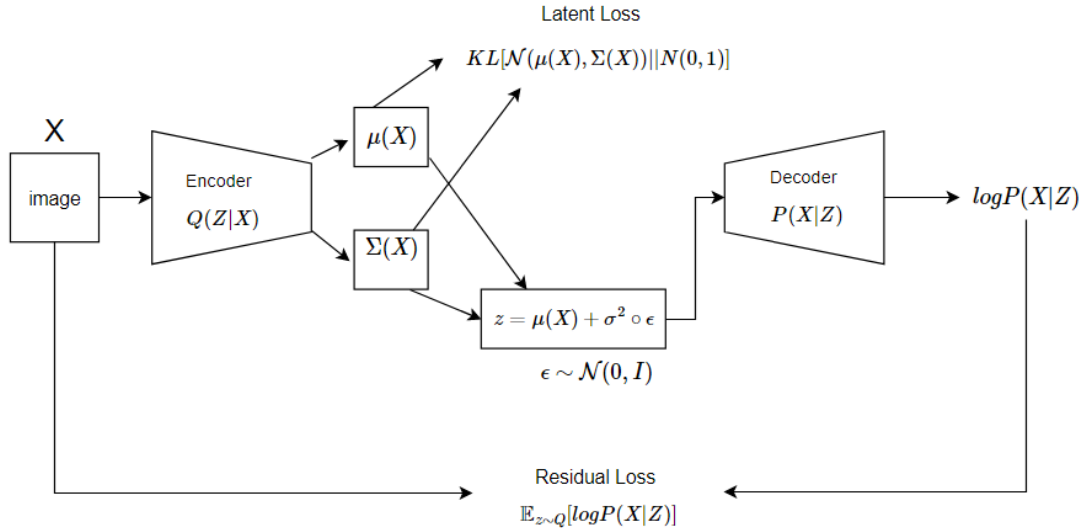


Figure 2.9: Variational Autoencoder

2.2.2 Reparameterisation Trick

A random variable is transformed into a deterministic function and gives the random dependency to another variable. The technique is based on the method referred as one-liners [37, 38]. There are different application for which we would like a stochastic function to be deterministic, the one we are concerned primary is back-propagation in stochastic functions. In order to push the gradient through the neural network, we need to reparametrizise and make each node deterministic except for the input.

There are different techniques for doing reparameterisation such as polar methods, co-ordinate transformation methods or inverse methods. The name of one-liner comes from the simple implementation in one line of code. For example, the Gaussian $\mathcal{N}(\mu; \sigma^2)$ can be written as $\mu + \sigma^2 \epsilon$ where $\epsilon \sim \mathcal{N}(0, 1)$; another example, take a discrete probability distribution such as Rademacher $f(k) = \frac{1}{2}$ if $k \in \{-1, 1\}$ and 0 otherwise, we can rewrite in one line as $2\epsilon - 1$ where $\epsilon \sim \text{Bernoulli}(\frac{1}{2})$.

Reparametrization of random variables is a useful tool when dealing with stochastic back-propagation. Instead of writing the full probability density function, we can use one line of code making it appealing for implementation. The value of the technique relies on removing the stochastic part of a function and pushing the random part into a new variable as an input to the function.

2.3 Evaluation of generative models

Quality of samples generated are uncorrelated about the likelihood. Models can have poor image diversity and have good likelihood, which makes density models hard to compare quantitatively in the generative model framework as shown in [39]. One avenue, for assessing quantitatively generative adversarial networks in the unsupervised learning framework, is using inception score proposed by [6] or training classifiers on top of the learned features as presented in DCGAN[24]. Other models such as inpainting [31] uses modifications of the inception score.

The inception score, which is based on the pre-trained Inception model, has two goals: (1) recognize meaningful objects by having low entropy (low measurement of uncertainty) on $P(y|G(z))$ where P is the inception model; (2) assessing variety of images by obtaining high entropy over the marginal $\int p(y|G(z))dz$. Example for entropy over the marginal: suppose we have $y_1 = [a, a, a, a, b, b, b, b]$ and $y_2 = [a, a, a, a, a, a, b, b]$, using entropy $E = -\sum P(y) \log P(y)$ we have $E(y_1) = 1.0$, $E(y_2) = 0.81$ and clearly $E(y_1)$ has more diversity. With the entropy over the marginals, we are only using the number of elements chosen for each class; while the entropy over the conditional $p(y|G(z))$, uses probability vectors. To merge classification and diversity, the authors from [6] propose inception score

$$\exp(\mathbb{E}_x KL(p(y|G(z))|p(y)))$$

where they use the Kullback-Leibler divergence (relative entropy) to measure the divergence from both probabilities. As results, the approach is based on having the same classifier as a common denominator across different generative models.

When the goal is to produce high quality images, we can use a qualitative measurement by visually assesing the images. The ultimate goal would be to have human annotators not being

able to discern between generated images and real images. The work from [6] used human annotators to assess the quality of a generative model and the results from such experiment showed that trails depend heavily on the setup and motivation of the annotators. For this reason, the research has been pushed towards quantification.

The evaluation of generative models can be done from a quantitatively or qualitatively perspective. The context of the problem, such as, generating images or classification indicates what kind of evaluation is more appropriate. Clearly there is not a one for all loss across multiple generative models. As result, evaluation of generative models is an active research area.

Comparing GANs and VAE

1. VAE have a elaborated probabilistic formulation over the latent variables which allows us to use our desired distribution (tractable likelihood).
2. GANs have had better image quality generation, however finding an equilibrium between discriminator-generator is harder than training a VAE.
3. Both are graphical models which use a latent variable z to produce data points X

$$f : z \rightarrow X$$

4. VAE use a lower bound on the likelihood, rather than working with the likelihood itself.
5. With GANs it is hard to generate discrete data such as text. The first attempt was presented in [29].
6. VAE can easily produce $P(z|X)$, whereas the vanilla GAN needs to deal with the inverse mapping.

Chapter 3

Approach

Previous sections gave an overview of generative models with focus on generative adversarial networks and variational autoencoders. The chapter covers representation learning with GANs and how to transform the problem of classification using only one-class with GANs and VAE. Switching a generative model to learn over one-class, as we will see in sections 3.2 and 3.3, allows us to do anomaly detection. In a nutshell, anomaly detection learns a "normal" representation and afterwards uses the learnt representation to decide if new data points belong to the representation.

3.1 Representation Learning with GAN

After learning an unsupervised representation using a generative adversarial network, we use parts of the convolution layers of the discriminator as feature representations. The feature representations are obtained by passing data points through the discriminator until certain layer f . The features matrices f are flatten and afterwards trained with a classifier that uses the learnt features as input. The approach is illustrated in Figure 3.1

3.2 Anomaly Detection with GAN

We explore anomaly detection with generative adversarial networks by re-implementing the pipeline from AnoGAN [9]. The problem we are trying to solve is the detection of anomalies including the detection of anomaly regions, i.e. regions of images which were not able to reconstruct by exploring the latent space. Detection of anomalies can be approached as a classification problem, while detection of regions is a regression problem. The authors from AnoGAN[9] use a perceptual and visual component to solve anomaly detection.

The method presents three steps: first, learn the manifold \mathcal{X} in an unsupervised way of a corpus of "normal" images; secondly, mapping images to the latent space; finally, detecting abnormalities by using a visual and perceptual component. In the following subsections, we provide a more comprehensive description of each step.

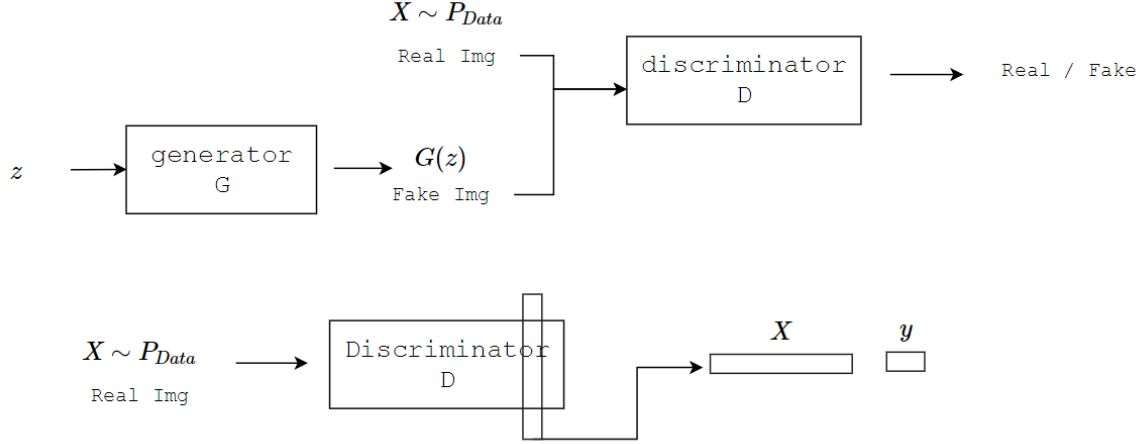


Figure 3.1: Feature representation model. The top row model is an unsupervised generative adversarial network. The bottom model is in charge of extracting features and training a classifier with the representations.

3.2.1 Unsupervised manifold learning

For the MNIST dataset we use the DCGAN [24] architecture as presented in AnoGAN [9]. For the NLST dataset, we did not use the same architecture since the model did not capture the variability of the images presenting high mode collapse. We propose a different architecture using the work from WGAN-GP [29].

3.2.2 Mapping images to the latent space

In manifold learning, we constructed an unsupervised model to learn the manifold of normal images by using a generative model. With this model, we can generate images $G(z)$ from a random seed z and discriminate images belonging to the representation using $D(x)$. What we do not have is the inverse process $G(z) \rightarrow z$ which would allow us to map new images to our learned representation. If we could map an image back, then the new data point would belong to the distribution of our model P_m . The task relies on the process of mapping images back to the latent space, while giving a guarantee it belongs to the distribution of our model P_m . Our inverse process $G(z) \rightarrow z$ approach is closely related to the work of [9, 31].

The approach is to start at some random point in the latent space z_1 which can generate an image $G(z_1)$ and walk in the direction of our target image x . If the target image x belongs to the learned representation, we would be able to reach or recover the target image after $z_1, z_2, \dots, z_\gamma$ steps in our walk. The final image $G(z_\gamma)$ can be compared against the target image x under some metric. First, we will define a metric as our transport and then we will cover how we can use the metric with back-propagation to walk in the latent space.

The metric or loss is formed by a residual and discriminator component as presented in AnoGAN [9] which is based on the work of [6, 31]. The residual loss compares similarity of images on pixel level using the generator G . While the discriminator loss uses a perceptual component by comparing the statistics learn by the discriminator D .

The residual loss is defined by

$$\mathcal{L}_R(z_\gamma) = \sum |x - G(z_\gamma)| \quad (3.1)$$

where x is the target image and $G(z_\gamma)$ is the last step and possibly the closest image after $1, 2, \dots, \gamma$ steps. If the images are visually equal then $\mathcal{L}_R(z_\gamma)$ would be zero. The loss is comparing images at a pixel-wise level.

The discriminator loss is defined by

$$\mathcal{L}_D(z_\gamma) = \sum |f(x) - f(G(z_\gamma))| \quad (3.2)$$

where f is a layer from the discriminator. The statistics of the target image $f(x)$ are compared against the statistics of the closest image in the latent space $f(G(z_\gamma))$. The loss is based on feature matching [6] and compares images at a perception level.

The overall loss is the sum of the residual and discriminator loss with parameter λ for controlling to what extent we use each loss defined by

$$\mathcal{L}(z_\gamma) = (1 - \lambda)\mathcal{L}_R(z_\gamma) + \lambda\mathcal{L}_D(z_\gamma) \quad (3.3)$$

Using λ with value of 0 will completely use the generator (residual loss); while a value of 1, will use the discriminator (discriminator loss). The residual loss measures the difference between the images and the perceptual loss measures the divergence between the statistics of the images.

How do we update z_1 to get z_γ ? We update z w.r.t. the loss function using stochastic gradient descent with momentum,

$$\begin{aligned} \Delta z_i &= -\alpha \frac{\partial \mathcal{L}(z_i)}{\partial z_i} - \beta \Delta z_{i-1} \\ z_{i+1} &= z_i + \Delta z_i \end{aligned}$$

where α is the learning rate, β is momentum. There are different approach for updating the parameter z once we have the gradients such as Adagrad, RMSprop or Adam. For our approach, we only explore SGD with momentum.

3.2.3 Detection of Anomalies

From the previous step, mapping images to the latent space, we have the closest image $G(z_\gamma)$ and the loss value $\mathcal{L}(z_\gamma)$. We can use the loss value from equation 3.3 to set a threshold for anomaly detection. Images that belong to the learned representation would have a lower score compared to images that do not belong to the learned manifold \mathcal{X} . We use the same anomaly equation as AnoGAN [9] defined as

$$A(x) = (1 - \lambda)\mathcal{L}_R(z_\gamma) + \lambda\mathcal{L}_D(z_\gamma)$$

The anomaly score values $A(x)$ are in the range $[0, 1]$ and λ defines how much does each loss weights. Let ϵ define the threshold over $A(x)$, then we can define the final model as

$$f(x) = \begin{cases} 1, & \text{if } A(x) \geq \epsilon \\ 0, & \text{otherwise} \end{cases}$$

where 1 is abnormal and 0 is normal.

3.3 Anomaly Detection with VAE

We propose anomaly detection with variational autoencoders based on the reconstruction probability $\mathbb{E}_{z \sim Q(z|X)} P(X|z)$. First, we train a VAE on a corpus of *normal* data points which correspond to only one class. Then, once we have learned a semi-supervised representation model, we use ℓ_1 -norm between the original image and its reconstruction to obtain the discrepancy between the images (abnormal score). The abnormal score can then be filtered by a threshold ϵ to determine if a new data point belong to the learned representation. The abnormal scores are continuous values and we can compare them against the true values (discrete values), the value of ϵ is an hyperparameter we can tune by maximizing the number of true and false positives. We expect the model to have high reconstruction error with data outside the learned manifold. For this reason, data points with high reconstruction errors are considered abnormal. The proposed method is shown in Algorithm 1.

Algorithm 1 Anomaly detection with VAE

Input: Normal data X_n , Mix normal/abnormal X_m , threshold ϵ

Output: Abnormal Score S

```

1:  $Q(z|X), P(X|z) \leftarrow$  Train VAE with  $X_n$ 
2:  $A_{1 \leq i \leq m}, S_{1 \leq i \leq m}$  empty arrays
3: for  $x_i \in X_{1 \leq i \leq m}$  do
4:    $x_r \leftarrow \mathbb{E}_{z \sim Q(z|x_i)} P(X|z)$ 
5:    $A_i \leftarrow \frac{1}{n} \sum |x_i - x_r|$ 
6: end for
7: for  $a_i \in A$  do
8:   if  $a_i < \epsilon$  then
9:      $S_i \leftarrow 0$  {normal}
10:  else
11:     $S_i \leftarrow 1$  {abnormal}
12:  end if
13: end for
14: return  $S$ 

```

How to choose a proper threshold ϵ ? The Algorithm 1 has two important arrays: scores before threshold A , scores after threshold S . We can use A to calculate AUC and determine the best cutoff point for ϵ . There are different approaches for determining the best cutoff point, such as, minimizing a cost function to balance true positive and false positive rates.

Chapter 4

Experimental Setup

The chapter presents the experiments based on our approach presented in Section 3. First, we will introduce the type of datasets we used for our experiments. Then, we cover the setup for anomaly detection with generative models. The implementation was using python with keras [21] and tensorflow [40] as the back-end.

4.1 Datasets

4.1.1 MNIST

The MNIST is a greyscale handwritten number dataset, where the numbers are centered and with a fixed size of 28x28. For anomaly detection with MNIST we split the numbers into two groups. The first group which we will call from now on *normal* contains only numbers from 0-6. The second group, referred as *abnormal*, contains digits within the range 7-9. The goal is to train a model with only one-group and be able to distinguish a new data entry as being *normal* or *abnormal*.

For training we use 41,935 digits containing only *normal* digits. For testing we use 10,000 digits containing 6989 *normal* and 3011 *abnormal* digits. We left the group of abnormal to be relative smaller than the normal group to simulate unbalanced datasets. For each number, we set label '0' if it belongs to *abnormal* group and '1' otherwise.

4.1.2 NLST

The National Lung Screening Trial (NLST) contains low-dose computed tomography (CT) scans of lungs. The scans are slices of the human body and within the lungs there are sometimes lung nodules presented. The lung nodules are small oval-shaped masses of tissue in the lung with a size around 6-30 mm. For each patient, a nodule detector was used in order to identify the nodules location and consequently we extracted multiple slices around the lung nodule. In figure 4.1, we present 5 slices of a lung nodule of shape 28x28.

a nodule detector was used in order to identify the nodules location and consequently we extracted a

Each lung nodule has been assigned a label '0' for benign or a label of '1' representing malignant lung nodule. The database is quite unbalanced with most of the lung nodules

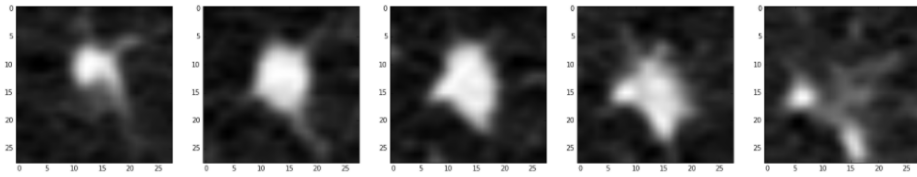


Figure 4.1: Five slices presented of a lung nodule.

belonging to healthy patients and presents a relative small dataset. The dataset was split into training, validation and test dataset as shown in 4.1.

Table 4.1: NLST lung nodule dataset count

	Label 0	Label 1	Total
Training	1722	460	2182
Validation	431	115	546
Testing	539	143	682

Instead of taking parallel slices of the lung nodule, our models used axial, coronal and sagittal view of the lung nodules. The spacing is isotropic with 1 mm per voxel. The size and labels maintain the same values, but the perspective is different. We opt for this transformation since it presented better results when using convolutional neural networks for classification. Figures 4.2 presents examples of healthy and unhealthy lung nodules.

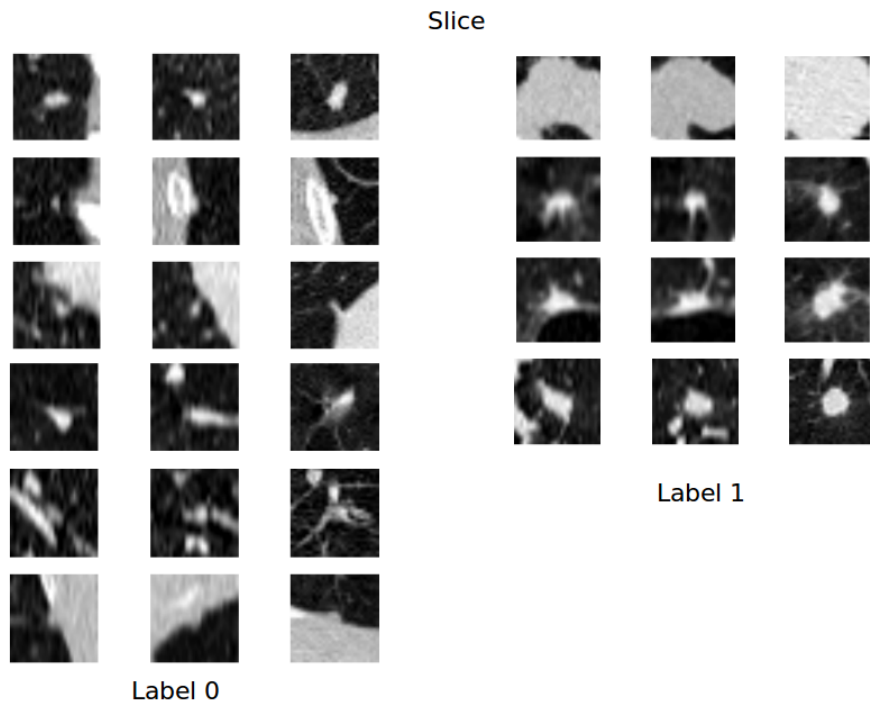


Figure 4.2: Malignant and benign lung nodules with axial, coronal and sagittal perspective.

4.2 Unsupervised manifold learning

In the context of GANs, the goal of the experiments was to find a model which had a good local minimum with respect to both generator and discriminator. This equilibrium is not always reflected by the loss score. A low loss score in the discriminator does not guarantee a good generator and vice versa in the vanilla GAN framework. For this reason, it is essential to run experiments and preview the generation of images during training. The assessment of the models were based on qualitative results. The experiment road map taken using the NLST dataset follows,

1. **DCGAN** We employed transposed convolutional GAN using the guidelines presented in section 2.1.2. This architecture converges relatively fast with big datasets such as MNIST, CelebA or LSUN bedrooms dataset. First, we attempt to use parallel slices of the lung nodules while using different slices as channels, $28 \times 28 \times 5$ or $32 \times 32 \times 3$. The lung nodules images presented visual repetition, i.e. for being to close a slice to another slice there was no significant or meaningful difference, when the slices are close to each other. After exhaustive hype-parameter tuning, the generator was not converging. We reduce the dimensionsof the problem by using only one slice of the lung nodules, $32 \times 32 \times 1$. The different trials did not converge.
2. **GAN hacks** Use training guidelines shown in [35], which can be applied to any GAN architecture. To mention a few of them: SGD for discriminator and ADAM for generator, train with batches of only real and only fake (do not combine). We also applied feature matching technique to DCGAN generator in order to have a more meaningful loss. From this point we change the direction to different GAN variations, such as: 3DGAN [41], ACGAN[26], SGAN[27], WGAN[28], VAEGAN[32] and WGAN-GP[29].
3. **3DGAN** The work from [41] inspired to use 3D convolutions with upsampling for the generator. Each trail took around four times more the amount of time compared to using only 5 slices. As a result, we opt to explore a different avenue. Most of the images from the generator are completely black of white.
4. **ACGAN and SGAN** Having labels significantly improves the generation of images as shown in [26, 27]. For this reason, we explore the architecture auxiliary conditional generative adversarial network and semi-supervised GAN. We first constructed an ACGAN using the MNIST dataset and extrapolate the architecture to the NLST dataset. At first the architecture did not converge using the MNIST dataset, a simple solution such as moving the learning rate of ADAM from 0.0002 to 0.0005 for both generator and discriminator generated the highest improvement. By improvement, we mean going from noisy images from the generator to actual seeing numbers. The images presented with ACGAN using MNIST dataset presented higher quality compared against the DCGAN architecture.

We attempt different experiments with ACGAN under the NLST dataset. The model presents almost complete mode collapse. However, this model was the first to present a lung nodule as shown in Figure 4.3. The other alternative which also uses labels is SGAN; however, our experiments did not converge to produce a good generator. By far, we consider ACGAN to be a strong candidate for generation of data.

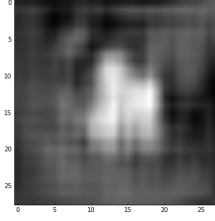


Figure 4.3: ACGAN lung nodule produced by the generator. Only 1 slice of lung nodules were used during training. The shape of the image is 28x28x1. We present only 1 image due to high mode collapse of the model.

5. **WGAN** One well known problem with GANs is having a meaningful loss function which can decrease as the quality of images from the generator increases. The work from [28] proved empirically this correlation. We consider this type of architecture for our experiments. However, there was no good generation of images. The WGAN model trains the discriminator n -batches for 1-batch of the generator. Contrary, to other GANs where the training is 1-batch for discriminator and 1-batch for generator. The computation cost is a factor of $n-1$ more batch training's. As a result, the convergence for each epoch is slower.
6. **VAEGAN** Based on the work from [32], we use this type of architecture only on the MNIST dataset. The generation of images was poor and the numbers had spiky strokes. Since not even with a simple dataset such as MNIST worked, we did not pursue experiment with this model under the NLST dataset.
7. **VAE** Until this point, most of the GAN architecture did not converge using the NLST dataset. We opt to explore the path of variational autoencoders as a second alternative for generative models. The model has showed generation of images with blurry effect. The model (decoder) produces high variability of images. This model is selected as a strong candidate. We use 8 multivariate Gaussian for the prior.
8. **WGAN-GP** The improved version of WGAN made a generator which was capable of creating lung nodules with low mode collapse. This model alongside VAE are the two main candidates from all our experiments.

The generator for most of our experiments have a 100 dimension vector as input with a uniform distribution either within $[-1, 1]$ or $[0, 1]$. After the input, our model has a stack of dense layers followed by a reshape. The reshape is used to produce a correct input to the transposed convolutions. The transposed convolution works to upsample the space to the desirable image size. Within each transposed convolution we can also stack convolutions layers.

The discriminator for most of our experiments is formed by blocks of: convolution layer, leaky relu and max pooling. After several blocks we flatten the last max pooling layer and use dense layers followed by cross-entropy.

When adding or removing layers of our model for the NLST dataset and tuning the hyperparameters, most of the different runs produced same noisy images, i.e. random black and white pixels. For this reason, it is hard to detect a course of action for improvement. We use

the benchmark dataset MNIST in our experiments as a testing dataset. Small modification, such as lowering the learning rate or adding convolution layers, produced qualitative better visual results for the MNIST. For such reason, parameters or layers that did not work under the MNIST dataset were not used for the NLST model.

4.3 Feature representation with GANs

We use the WGAN-GP model trained on both labels and using all the training data. After obtaining a representation of the manifold of lung nodules and using the feature representation as mentioned in our approach, we train multiple classifiers. The best classifier so far is support vector classifier. We use stratified sampling with cross validation (5 folds) using support vector classifier and weighted class using the validation data. Using grid search the best parameters used for SVC, from the library sklearn [42], are: {'C': 100, 'class_weight':{1:10}, 'coef0':0, 'decision_function_shape':None, 'degree':3, 'gamma': 0.001, 'kernel':rbf, 'max_iter':-1, 'probability':False, 'random state': None, 'shrinking': True, 'tol': 0.001}. The final model is trained with all the validation data and evaluated with the test set.

4.4 Anomaly detection with GANs

The GAN used for the approach of anomaly detection is WGAN-GP. The experiment assumes a pre-trained WGAN-GP model using the NLST dataset. As a recap from our approach presented in section 3, the experiments for this section involve mapping images to the latent space and performing anomaly detection.

For the MNIST dataset, we explore the use of residual loss (equation 3.1), discriminator loss (equation 3.2) and overall loss (equation 3.3). We use different thresholds for $\lambda \in \{0, 0.1, 0.5, 0.8, 1\}$. As mentioned in chapter 3, λ values closer to zero use fully the generator (pixel-wise). Our experiments show a higher AUC when using pixel-wise and feature-wise compared to only using parts of the discriminator.

For walking in the latent space, we use SGD using the overall loss. The parameters are: size of the steps α , momentum β and number of steps γ . The time taken doing SGD was around 1 minute for each target image, for this reason we explore the minimum number of steps and size of step when moving in the latent space. To decide how big the step size in the latent space and the number of steps required, we conduct an experiment where we left a static seed z_1 and attempt to approach a target image with different step sizes such as 0.1, 0.25, 0.5 while also varying the total amount of steps. Our experiments show an overall loss more or less the same with $\gamma = \{25, 100\}$ while maintaining $\alpha = 0.25$. The loss score oscillates within a range of values after γ gradient steps.

For the detection of regions of abnormality, we explore or compare the closest image in the manifold against the target image. The residual image, which is the difference between both images, showed regions which were not able to reconstruct. To highlight regions considered abnormal, we converted both images to 3 channels (RGB) and swap the colormap of the residual image. Afterwards, we merge both images to preview easier regions of abnormality. We explore the region of abnormality process with both datasets:

1. For the MNIST, we started with a random seed producing a number from 0-6 and then

attempt to reach in the manifold an abnormal number (7-9); the experiment, showed $G(z_\gamma)$ to be 1s when reaching 7s and 3s when reaching 8s.

2. For the NLST dataset, we realized the same experiment testing with benign and malignant lung nodules never seen during training while modifying the λ parameter. Such experiment showed that using $\lambda = 0$ creates a centered mass on the images, while a $\lambda = 0.8$ approximates more features but at the price of the image quality.

For the anomaly detection with NLST dataset, we use the overall loss with different thresholds of λ . For each of the thresholds, we calculated the AUC scores. We additionally, obtained the μ and σ AUC scores for malign and benign lung nodules using the validation dataset. The goal of the experiment was to disentangle or create some spacing between the confidence intervals of benign and malignant lung nodules. If we could reach this goal with the experiments, we could set a final threshold ϵ . Any element above the threshold would be considered malignant.

4.5 Anomaly detection with VAE

The experiments were based on the reconstruction loss. The model was trained using the training dataset with only benign lung nodules. We assume the model would not be able to reconstruct malignant lung nodules, since the model was never trained with such data. Different architecture of VAE always gave blurry images. We used the decoder to generate several images and qualitatively assess the variance of the images. For the evaluation and parameter selection of the model, we use cross validation with 5 folds using training and validation data. The evaluation of the AnoVAE was performed using the test set.

We constructed a histogram and calculated the mean and variance of the reconstruction for each group: benign and malignant lung nodules. We use the histograms with parameters μ and σ to compare against the anomaly detection with GANs. We research how much difference or spacing existed between the confidence intervals. If we could have non-overlapping c.i., then we could linearly split malignant and benign lung nodules.

Chapter 5

Results

5.1 DCGAN

For the NLST dataset, the images from the generator at different epochs are presented in Figure 5.1. The loss of the model is shown at Figure 5.2. We use only one slice of lung nodules. After several trials, even when the loss has the same value, the quality of the images is relatively low. The images present some variability and mode collapse was not observed.

For the MNIST dataset, using numbers in the range 0-6, the loss is presented in figure 5.4 and a subset of numbers generated is illustrated in figure 5.3. The convergence, i.e. start seeing numbers, is realized after 40 epochs which is relatively low.

5.2 ACGAN

ACGAN model with MNIST dataset generates the best quality and variation of images as shown in figure 5.5, the loss of such model is shown in figures 5.6, 5.7. Since the results under the MNIST dataset were of high quality, we attempt to use the architecture with the NLST dataset. For the NLST dataset, using 5 parallel slices, the produced images are shown in figure 5.8, the loss of such model is shown in figures 5.9, 5.10.

5.3 AnoDCGAN

The section presents the results of using MNIST in the context of anomaly detection with the generative adversarial network of DCGAN. The results are split into three sections: unsupervised manifold learning, mapping images to the latent space and anomaly detection. Additionally, we include the effect of using only the discriminator as anomaly detection without any participation from the generator against using both generator and discriminator.

5.3.1 Unsupervised Representation

DCGAN model trained with numbers in the range 0-6. A few samples are shown in figure 5.3 and the loss of the model is presented in figure 5.4. In spite of ACGAN being a better model for the generation of images, we opt for the DCGAN because it only has a continuous

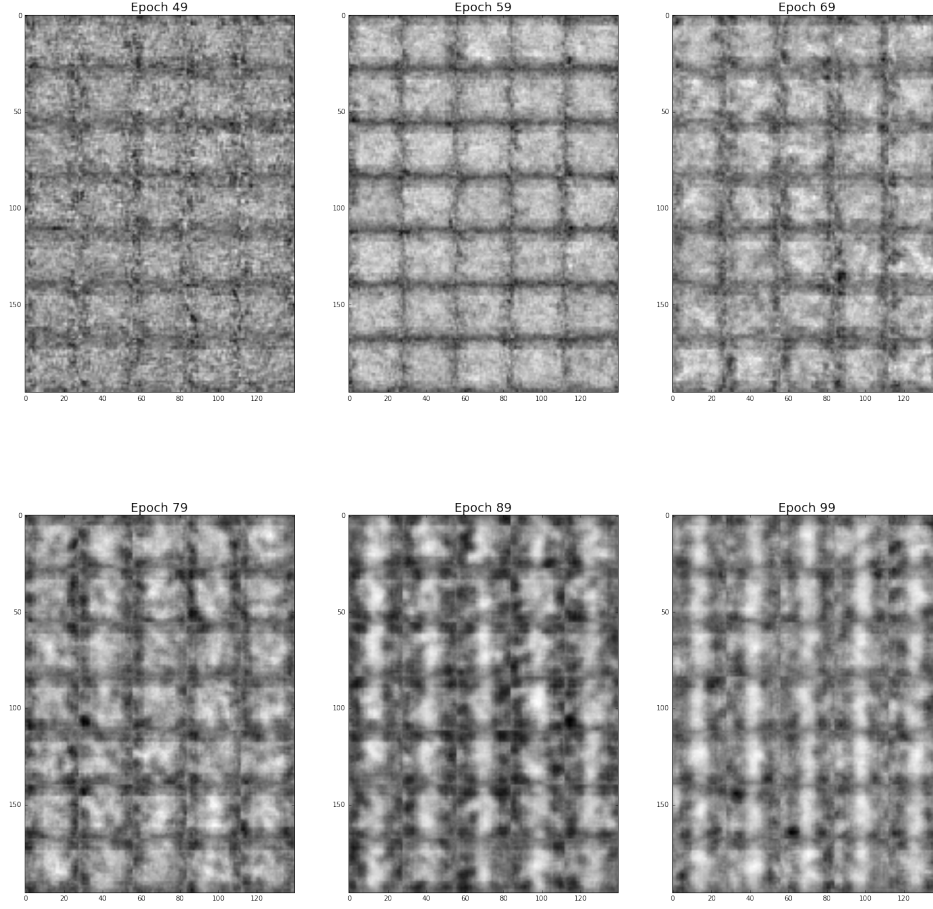


Figure 5.1: Different images from generator $G(z)$ with DCGAN architecture at different epochs. The first epochs are mostly noise until reaching epoch 89, where we start reaching some shape. After that point the loss increases, as shown in figure 5.2, and the images return with heavier noise.

vector for the input, which facilitate calculating the gradient of the loss w.r.t to the input when mapping images to the latent space.

5.3.2 Mapping images to the latent space

We present the outcome of searching in the latent space using SGD in figure 5.11, including the preview of abnormal regions shown in 5.12 which are based on 5.11.

5.3.3 Anomaly Scores

The overall loss, as mention in our approach in chapter 3, has two components: discriminator loss and residual loss. We contrast in this section the effect of using only the discriminator loss against using an overall loss (equation 3.3 with $\alpha = 0.5$. The ROC curve with only discriminator participation is shown in figure 5.13; with discriminator and residual component is shown in figure 5.14.

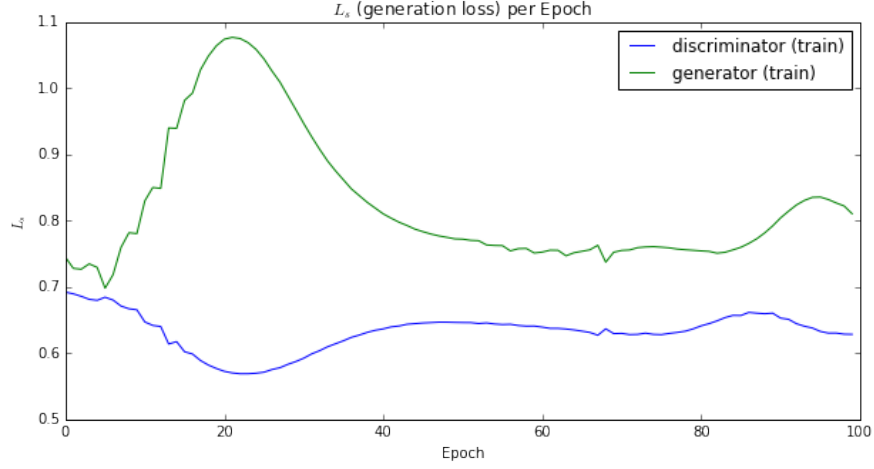


Figure 5.2: DCGAN loss with binary cross entropy using NLST dataset.



Figure 5.3: DCGAN with MNIST dataset using only number within the range 0-6.

5.4 AnoWGAN-GP

The section presents anomaly detection with wasserstein generative adversarial network using gradient penalty with NLST dataset. There are three sections: unsupervised representation, which focuses on GAN outcomes; images to the latent space, focuses on dealing with the inverse process of $G(z) \rightarrow z$; anomaly scores, previews the classification results when setting anomaly detection.

5.4.1 Unsupervised Representation

We present examples of the generator $G(z)$ at figures 5.15, 5.16 and 5.17. The loss of the model is presented in Figure 5.18, and the loss of the discriminator and generator is presented in Figure 5.19.

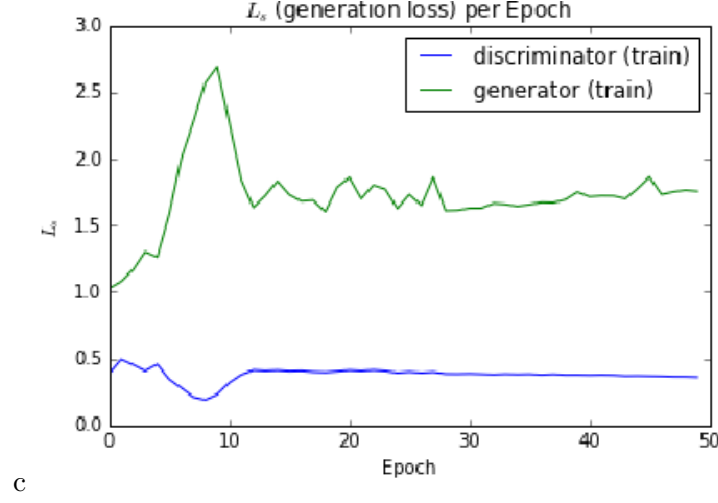


Figure 5.4: DCGAN loss using the MNIST dataset.



Figure 5.5: ACGAN generator images for different numbers.

5.4.2 Images to the latent space

We start at a random point in the learned manifold of healthy lung nodules z_1 , with target image I . We use SGD of the loss w.r.t to the input z . After 100 iterations with a step of 0.25 and $\lambda = 0.1$ we obtain z_γ . We preview the image $G(z_\gamma)$ and the original I at Figure 5.20.

The previous example, shown in figure 5.20, used a healthy nodule not seen during training. What if we use a malignant nodule? Figure 5.21. presents visual results with 100 gradient iterations with step size of 0.25 and $\lambda = 0.1$.

A target image (healthy lung nodule never seen during training) is approximated with different thresholds of λ in figure 5.22. With lower values of λ , the model tends to put a mass over the center, as the values of λ increases the image blurs more and match more the original image. Additionally, we include the effect of approximating a malignant lung nodule with different thresholds of λ in figure 5.23.

The residual image, as mentioned in our Approach, is the difference between the closest image in the manifold and the original image. In Figure 5.25, we present the residual image

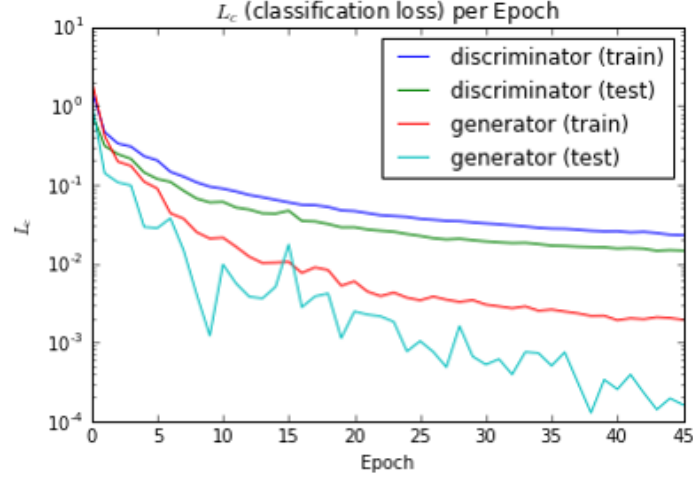


Figure 5.6: ACGAN classification loss.

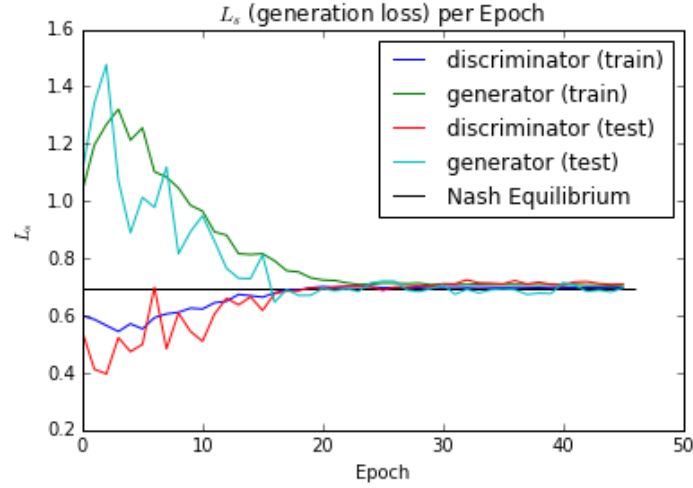


Figure 5.7: ACGAN loss associated with distinguishing real from fake.

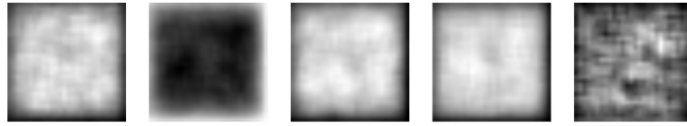


Figure 5.8: ACGAN generator images for different lung nodules slices.

which was constructed using the images from 5.24. We include more examples of unseen healthy data points shown in figures 5.26, 5.27.

5.4.3 Anomaly Scores

The anomaly score $A(x)$, as mentioned in our approach, is an analogy of the loss used for mapping images to the latent space. With parameters $\lambda = 0.1$, 100 backpropagations steps

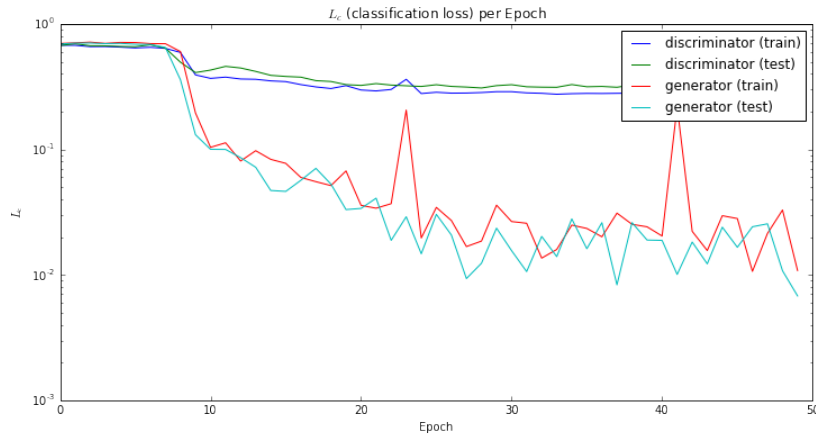


Figure 5.9: ACGAN classification loss using NLST dataset.

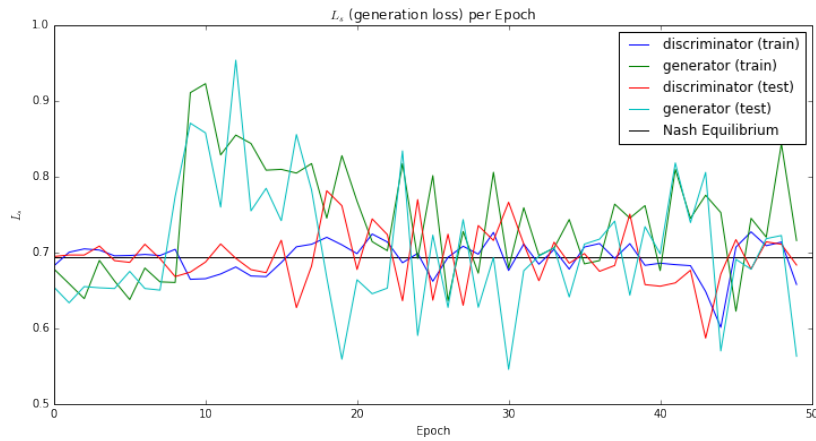


Figure 5.10: ACGAN loss associated with distinguishing real from fake using NLST dataset.

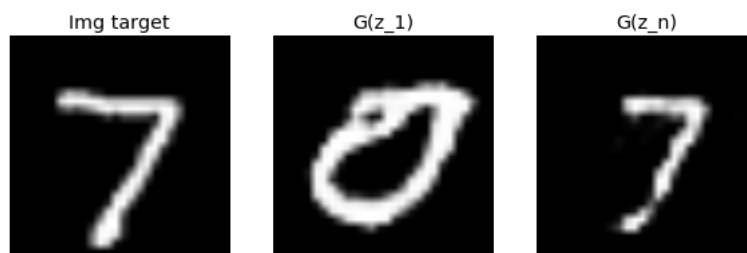


Figure 5.11: We initialize $z_1 \sim P(z)$ and create the image using the generator $G(z_1)$. We select a target image, belonging to the subset of abnormal images, and approximate the target image in the latent space. The closest image in the latent space is $G(z_\gamma)$. Recall, the model was never trained using number 7.

and with a step size of 0.1, we show in Figure 5.28 the ROC curve. Since the results did not show promising outcomes, we explored why was this effect by creating histograms of the

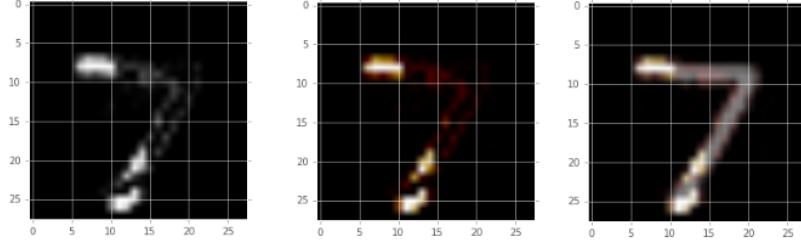


Figure 5.12: We show the residual image, which is based on the results from figure 5.11. Counting from left to right: the first image, is the difference between the target image and $G(z_\gamma)$; the middle image, is $G(z_\gamma)$ with different colormap; the last image, is an overlap between residual image and target image. The last image shows regions considered abnormal for such target image.

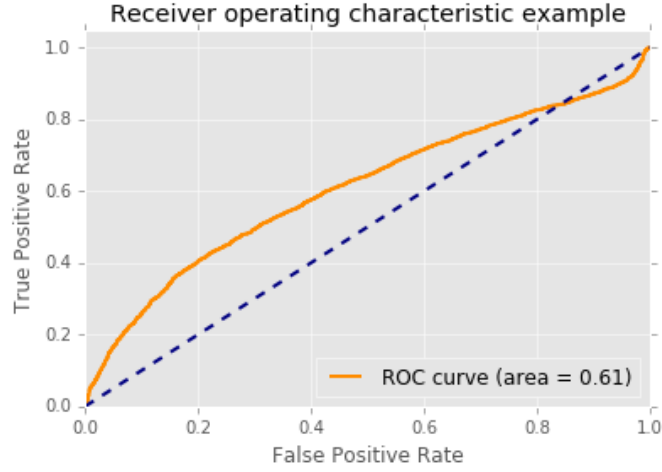


Figure 5.13: Anomaly scores using only the features from the discriminator without the participation of the residual loss (loss associated with the generator).

outcome values of $A(x)$ as shown in Figure 5.29. The results are derived using the validation data, which was not used during GAN training.

5.5 WGAN-GP as feature extraction

The results are split into two sections. The first section, unsupervised learning, is concern with the outputs of WGAN-GP trained with both labels in an unsupervised manner. The second section, feature extraction, presents results after training a classifier on top of the features extracted from the discriminator.

5.5.1 Unsupervised Learning

We present image from generator $G(z)$ in Figure 5.30. The loss of the model is shown in Figure 5.31.

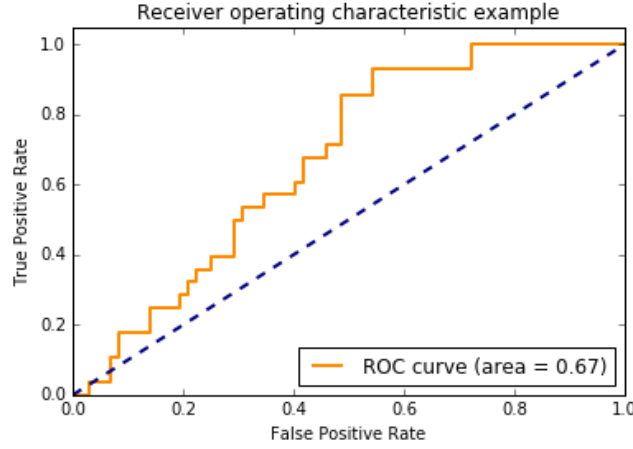


Figure 5.14: Anomaly ROC scores using the overall loss. The overall loss contains the discriminator loss (loss associated with features of the discriminator) and the residual loss (loss associated with the generator).

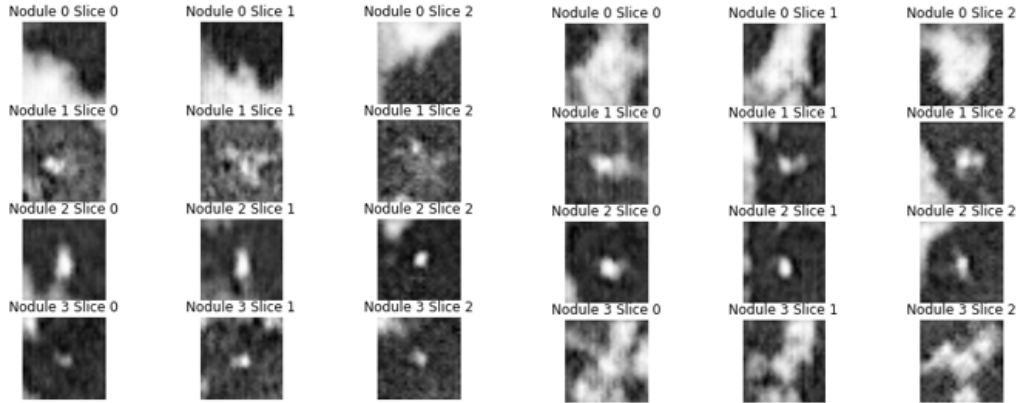


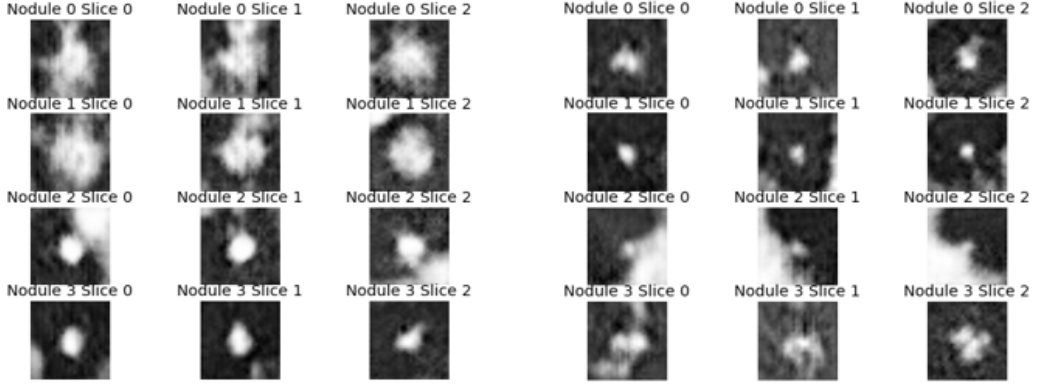
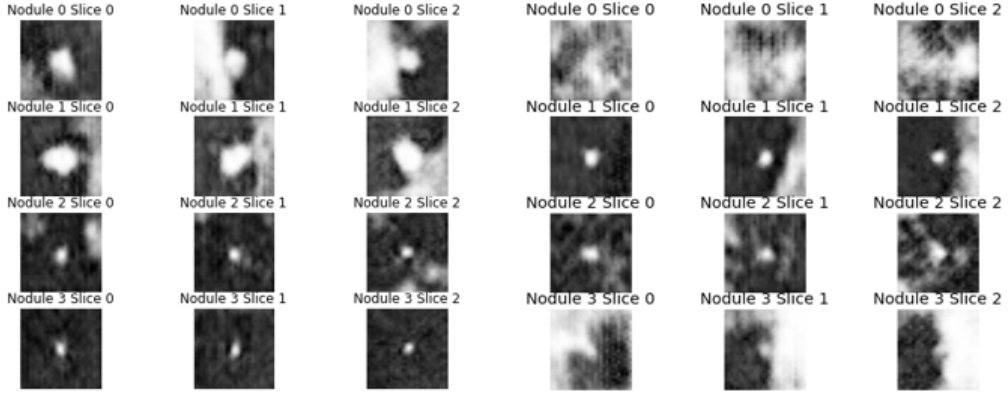
Figure 5.15: Random images from the generator $G(z)$.

5.5.2 Feature Extraction

Stratified sampling with cross validation using support vector classifier and weighted class gave AUC score of 0.722481 (+/- 0.082380) using validation data. The final model was constructed using all the validation data and evaluated on the testing data set. Classification report is presented in table 5.1.

Table 5.1: Classification report for feature extraction using WGAN-GP

	Precision	Recall	f1-score	Support
Class 0	0.86	0.81	0.83	556
Class 1	0.40	0.51	0.45	144
avg/total	0.77	0.74	0.75	700

Figure 5.16: Random images from the generator $G(z)$.Figure 5.17: Random images from the generator $G(z)$.

The confusion matrix is shown in Figure 5.32 and the final AUC score gave a value of 0.66 on the test set.

5.6 AnoVAE

Anomaly detection with variational autoencoders. The results are split into unsupervised manifold learning representation and anomaly detection.

5.6.1 Unsupervised Representation

The model was also trained with only one class. We show the reconstruction of lung nodules in Figure 5.33, 5.34.

We produce 10 spaced intervals for 8 Gaussians. We pass the linearly spaced values through the decoder $P(X|z)$, the results are shown in Figure 5.35.

Random generated images using the decoder are shown in Figures 5.36. Compared to the results with GANs, the images look blurry but have nice diversity.

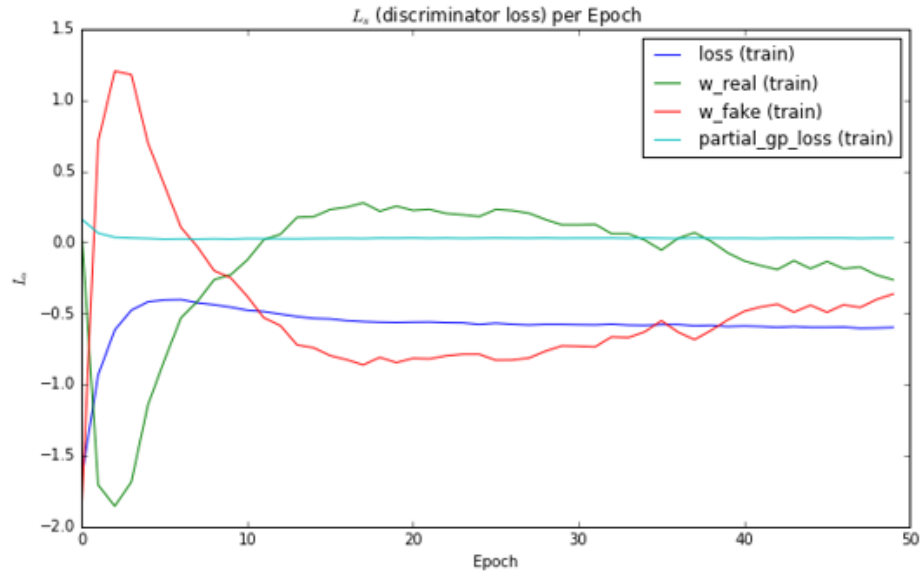


Figure 5.18: Loss of the model in strong blue, which is the sum of all the loss components: loss of real images, loss of fake images and the gradient penalty loss.

The loss of the model is presented in Figure 5.37. We use early stopping, i.e. after several epochs of score not changing within a range we stop training, with the validation loss.

5.6.2 Anomaly Detection

As mentioned in the approach, the reconstruction is used as the anomaly score. We expect to have high reconstruction loss with malignant lung nodules since we never train with any of them. The ROC curve is shown in Figure 5.38. We also explore the average reconstruction for healthy and unhealthy lung nodules: the results are shown in Figure 5.39.

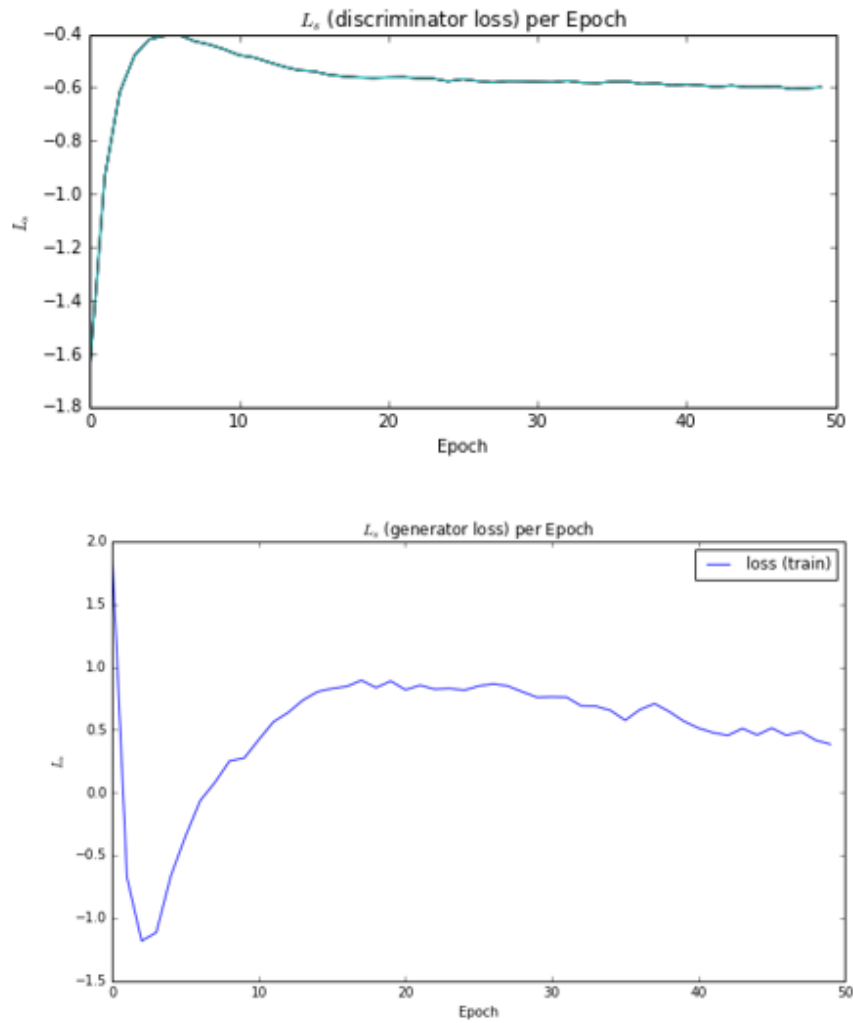


Figure 5.19: We present the loss of the generator and the discriminator.

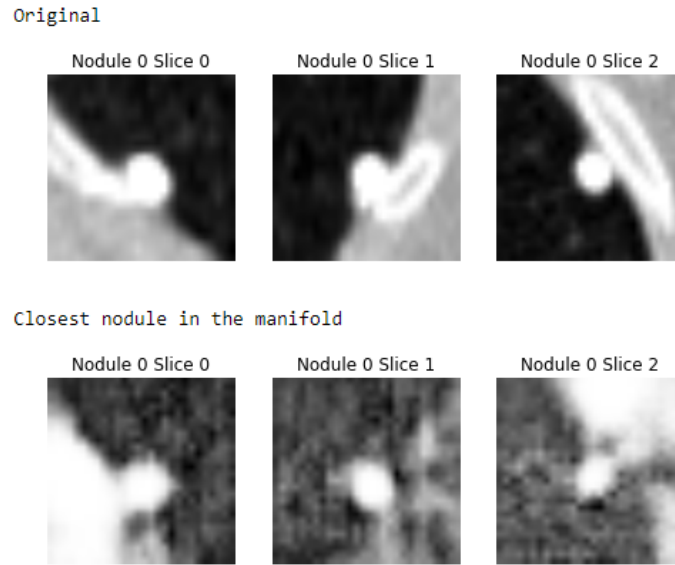


Figure 5.20: Original image (target) against the closest image found in the latent space using healthy unseen data point during training. Parameters $\lambda = 0.1$, 100 gradient steps with size of 0.1

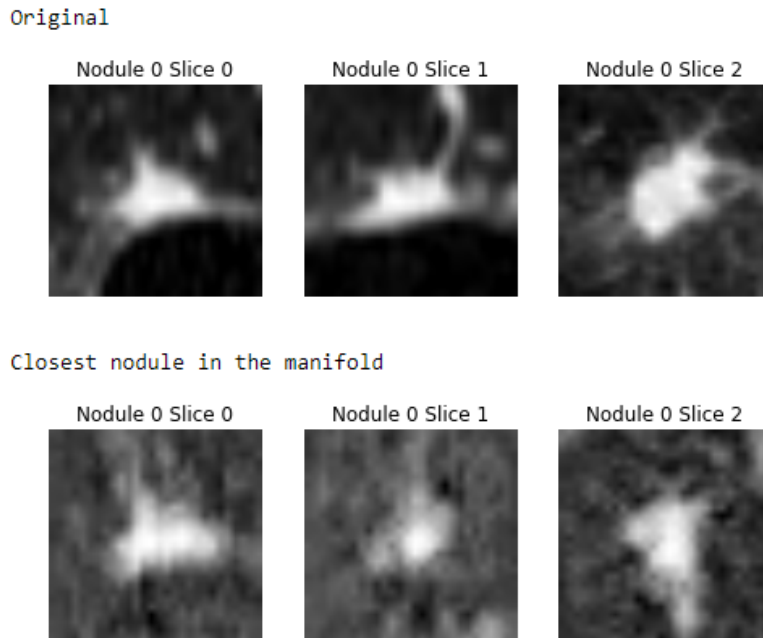


Figure 5.21: Original image (target) against the closest image found in the latent space using unhealthy data point. Parameters $\lambda = 0.1$, 100 gradient steps with size of 0.1

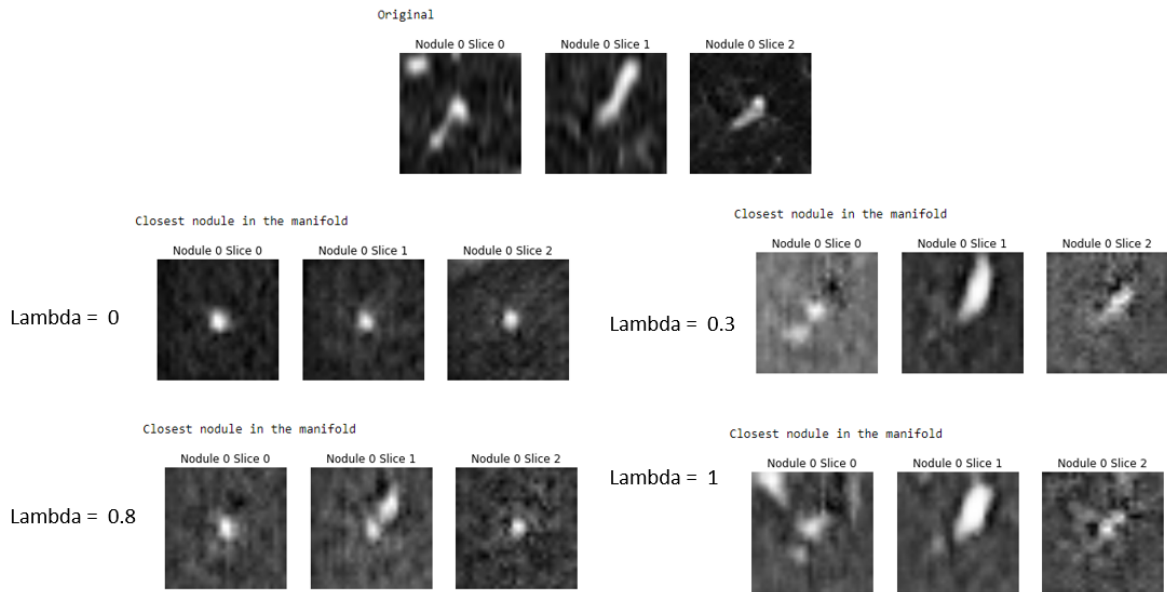


Figure 5.22: Modification of λ across different values using a data point with label 0 (healthy lung nodule). Modifying the values of λ change the resultant image.

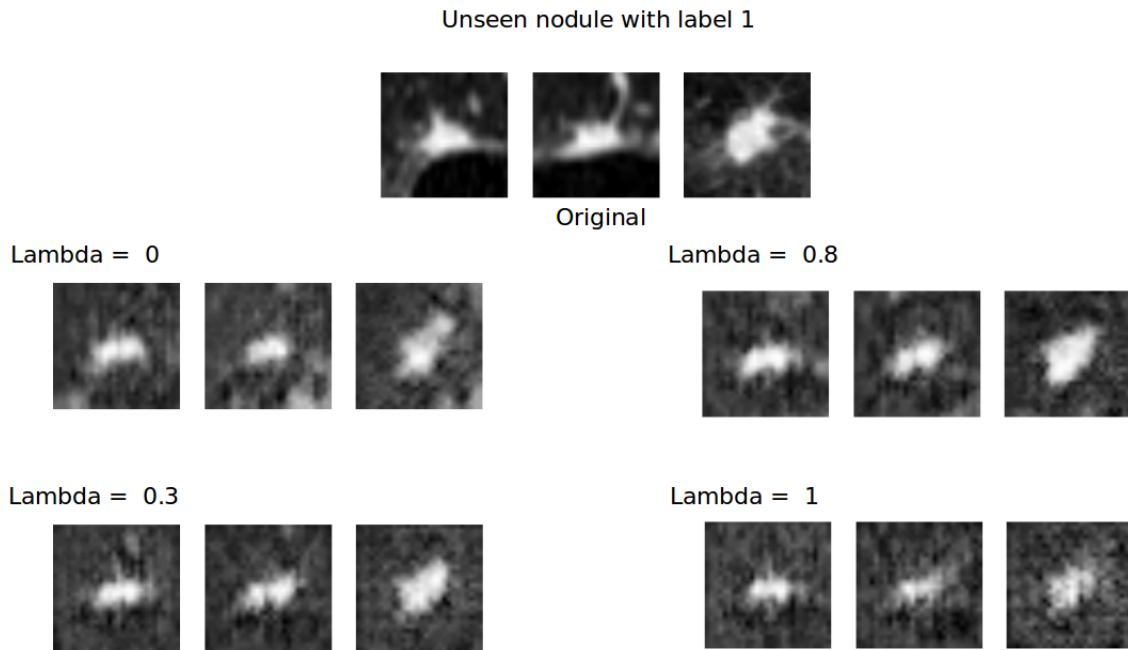


Figure 5.23: Modification of λ across different values using a data point with label 1. Most of the images generated with different thresholds did not significantly change the closest image in the manifold.

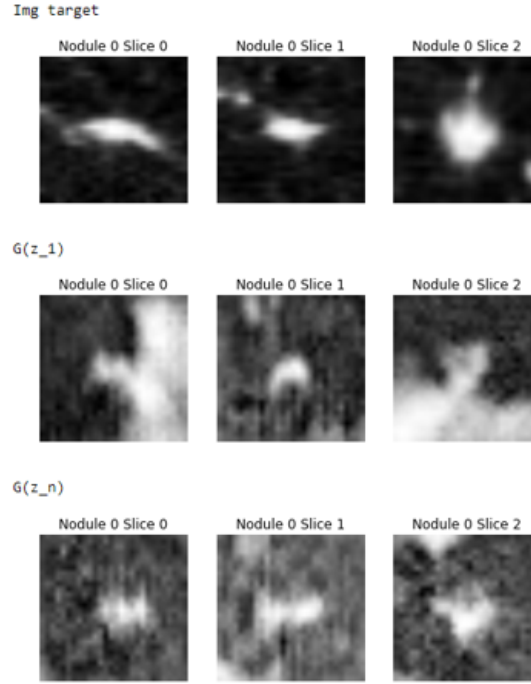


Figure 5.24: Original image, random starting point z_1 with image $G(z_1)$. After 200 back-propagations we arrive at $G(z_n)$.

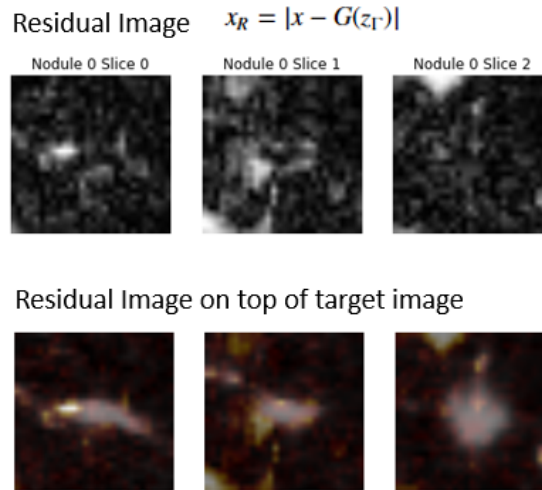


Figure 5.25: Residual image using a healthy lung nodule. On the bottom row we transform the residual image grayscale into a color-map. The color-map shows low and high intensity of regions considered different.

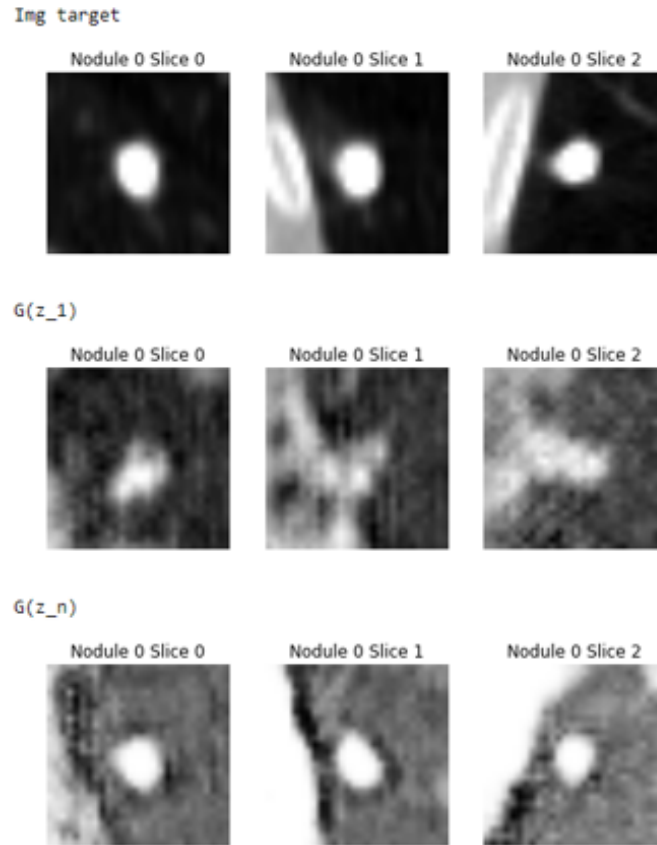


Figure 5.26: Unseen healthy lung nodule is search in the manifold by an iterative backpropagation process. The target image is the search data point in the manifold. The image $G(z_1)$ is the random initial state at some point in space. The image $G(z_\gamma)$ is the closest image found in the manifold after the iterative process with SGD.

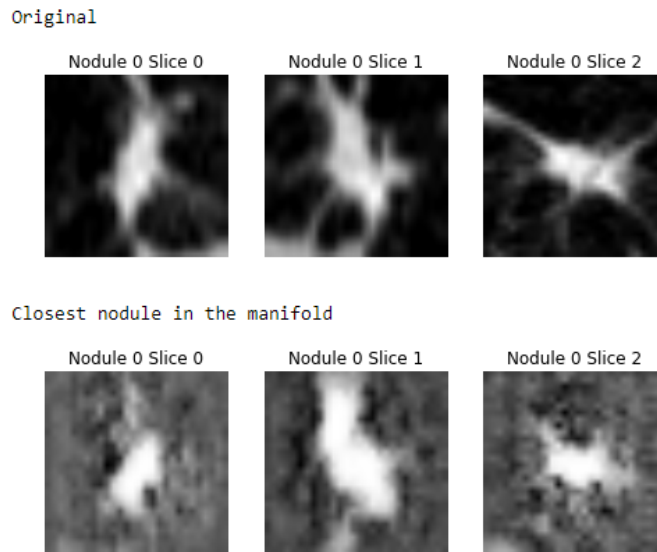


Figure 5.27: Unseen healthy lung nodule is search in the manifold by an iterative backpropagation process. The target image is the search data point in the manifold. The image $G(z_1)$ is the random initial state at some point in space. The image $G(z_\gamma)$ is the closest image found in the manifold after the iterative process with SGD.

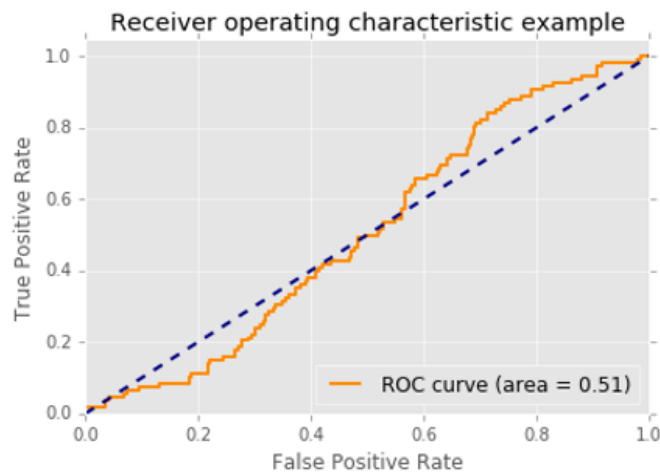
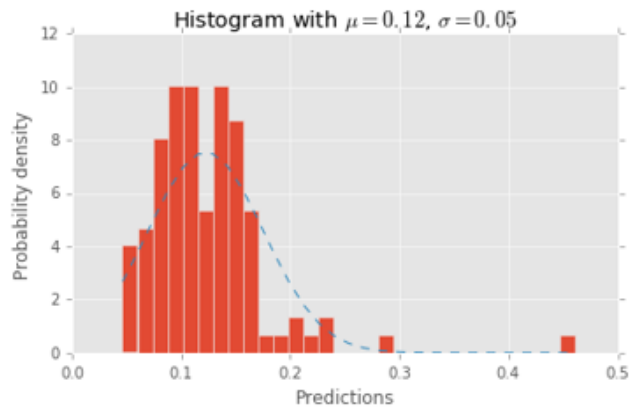


Figure 5.28: ROC curve using the anomaly score $A(x)$

Size of nodules with label 1: 108
 median 0.11309212446212769, mean 0.12249697530987086, std 0.05291958361524216
 conf int 95% (0.11251646766449433, 0.13247748295524739)



Size of nodules with label 0: 417
 median 0.11379603296518326, mean 0.12089512155031701, std 0.05179080812823496
 conf int 95% (0.11592424759297461, 0.12586599550765942)

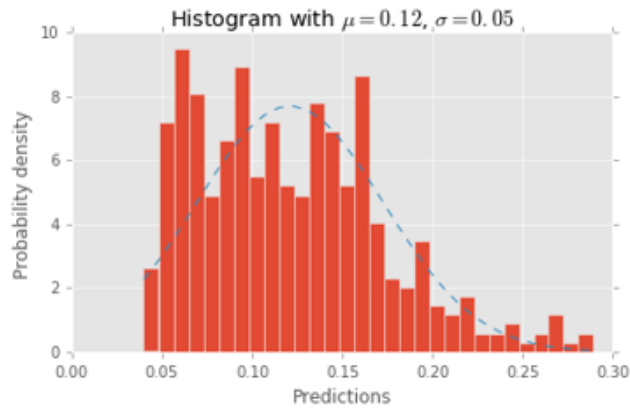


Figure 5.29: Histogram of the values from $A(x)$ based on the results from Figure 5.28.

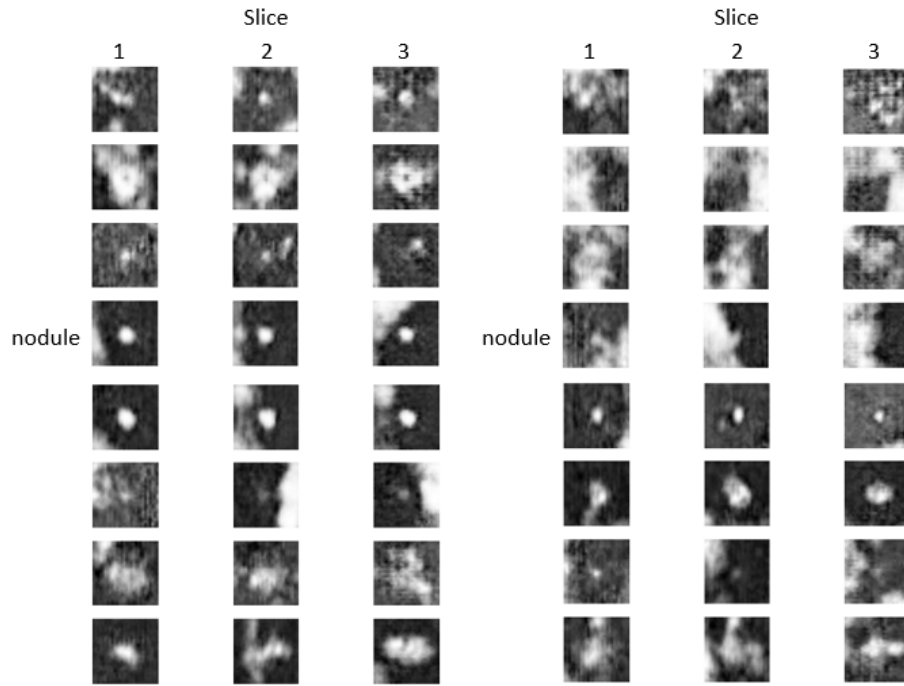


Figure 5.30: Images from the generator $G(z)$ trained with both labels.

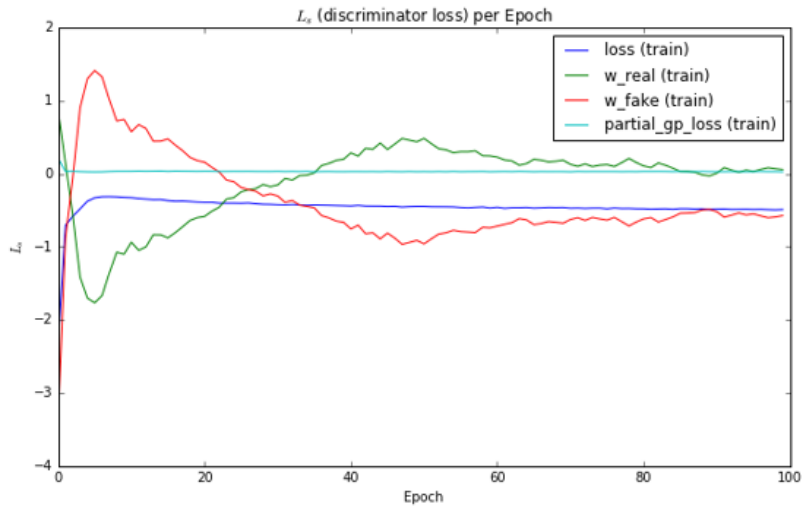


Figure 5.31: Loss of the model including its components

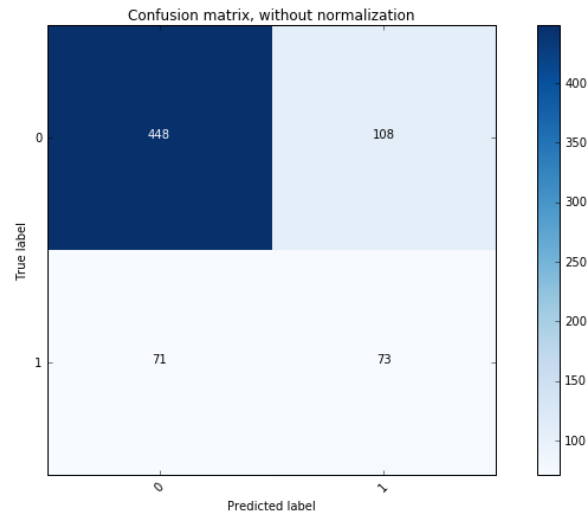


Figure 5.32: Confusion matrix using the test set.

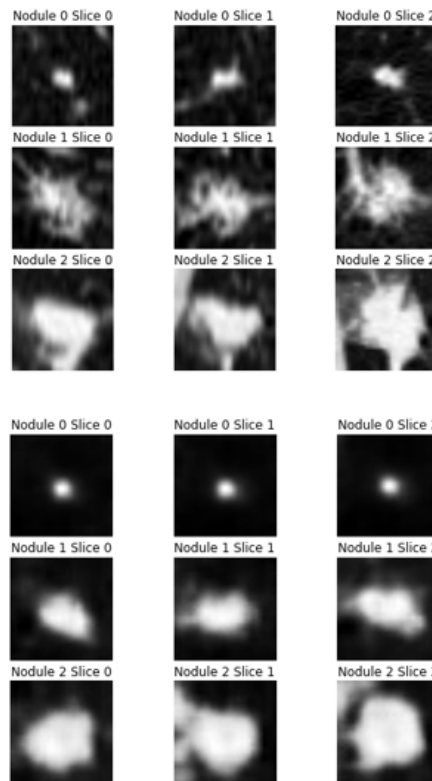


Figure 5.33: Reconstruction of images with label 0. The model was trained with only label 0. The top 3 rows are the original ones and the bottom rows are the reconstruction.

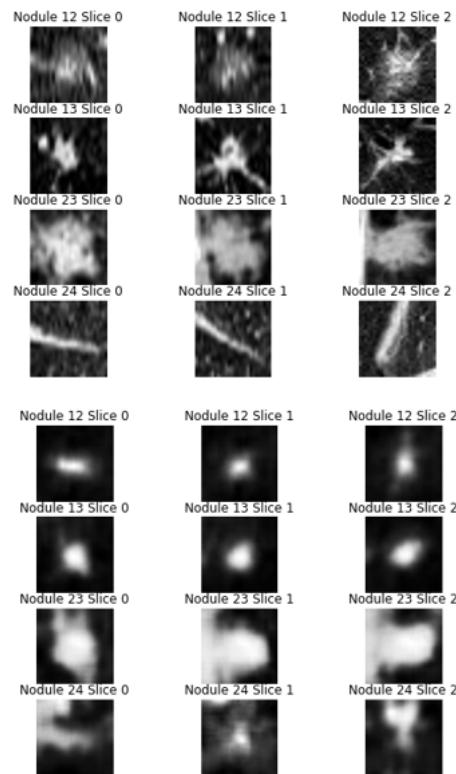


Figure 5.34: Reconstruction of images with label 1. The model was trained with only label 0. The top 3 rows are the original ones and the bottom rows are the reconstruction.

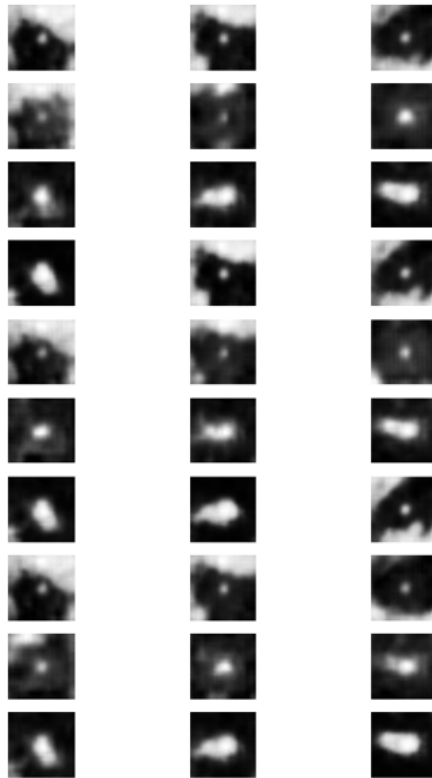


Figure 5.35: The images are generated from the decoder. Each row is a nodule and the columns are the slices. We use 10 linearly spaced values for the prior. The image shows some repetitions even when values of z are different for some Gaussians.

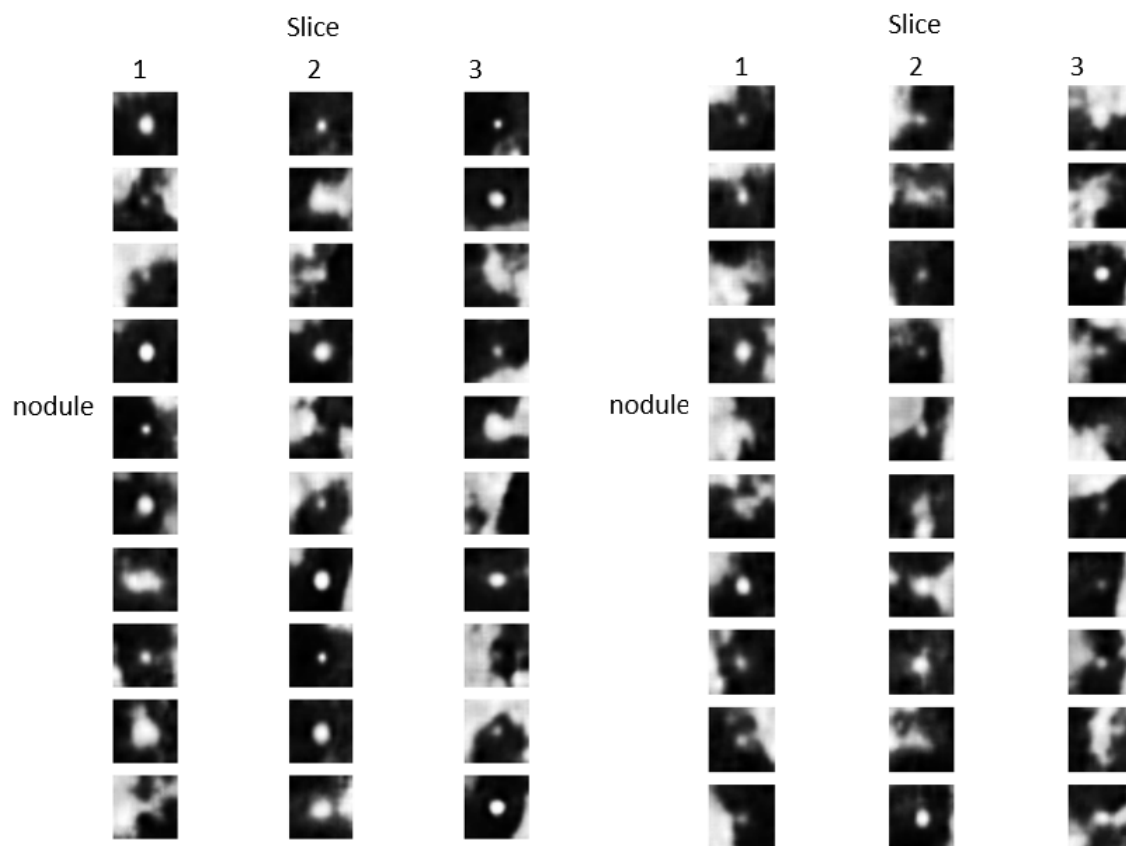


Figure 5.36: Random images from the decoder.

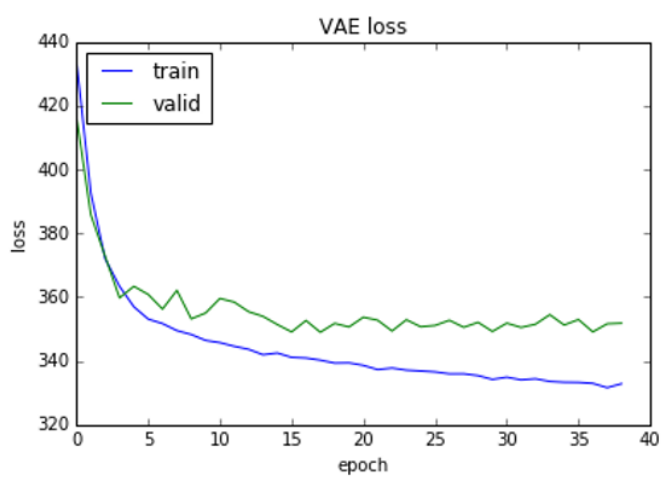


Figure 5.37: Loss of the variational autoencoder.

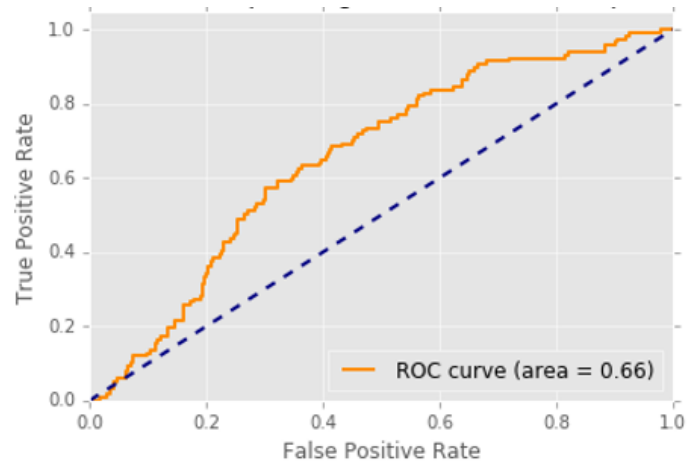
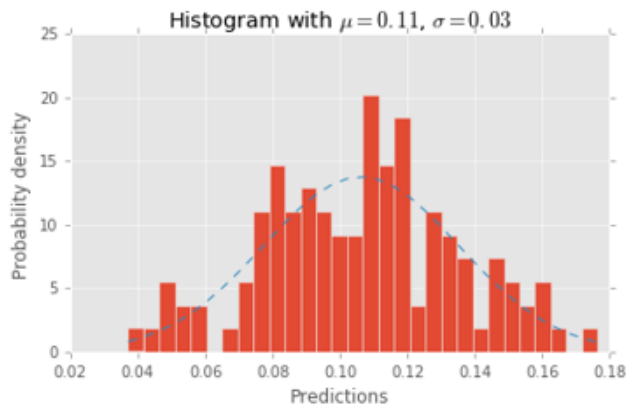


Figure 5.38: ROC using VAE as anomaly detection framework.

Size of nodules with label 1: 117
 median 0.10884270817041397, mean 0.10622835490438673, std 0.02899008588297753
 conf int 95% (0.10097538803022063, 0.11148132177855283)



Size of nodules with label 0: 408
 median 0.0856676958501339, mean 0.09071673558312743, std 0.0341688316896447
 conf int 95% (0.087401242402311397, 0.09403222876394346)

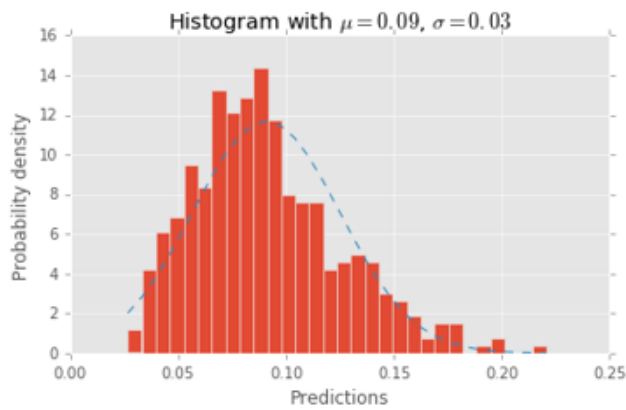


Figure 5.39: Histogram of the average reconstruction for different type of lung nodules. The overall loss reconstruction is higher for nodules with label 1 compared against healthy lung nodules.

Chapter 6

Conclusions

In this work, we explore how generative models could assist with classification of unbalanced and small dataset in the context of lung cancer. The generative models chosen were VAE and GANs. For GANs, we explored feature extraction and transformed the problem of classification into anomaly detection; additionally, we investigated the problem for mapping images to the latent space. For VAE, we based our experiments on the reconstruction loss by using a framework of anomaly detection. We compared both generative models using the generation of images and anomaly scores.

The advantage of using GANs over VAE is the quality of images and the ability to explore the latent space for the detection of regions of abnormality. On the other hand, VAE produces more blurry images and are easier to train. The VAE is based on Bayesian methods, while GANs is constructed with a min-max game between two players. The VAE model presents more rigorous mathematical derivation; as a result, we consider VAE a more complex model. Both models can sample new data points relatively fast, which is an important factor for generative models. The models which contain GANs have a three step process: unsupervised representation, mapping images to the latent space and anomaly detection; whereas, the model AnoVAE contains only two steps: unsupervised representation and anomaly detection.

A suitable model for anomaly detection based on the test set are either AnoVAE (auc = 0.65) or feature extraction with WGAN-GP (auc = 0.66). When performing cross validation using the training and validation data, the AUC score for AnoVAE is 0.63(+/- 0.16), while feature extraction with WGAN-GP it is 0.72 (+/- 0.08). Other models, which do not fold in the generative model context and train directly $P(y|x)$, have shown higher scores for classification. However, the value of using generative models is the ability to create images that generalize from a small subset of images, train with only one class and detect regions of abnormality.

The results under AnoWGAN-GP showed a technique for mapping images to the latent space, including regions of abnormality which is not the case for AnoVAE and feature extraction with WGAN-GP. The downside of the model is low performance of anomaly detection. However, we are able to reach or reconstruct benign lung nodules never seen during training as shown in our results. The experiments taken with different λ (threshold between perceptual and feature-wise loss) showed the necessity of a middle point between feature and pixel-wise. With pixel-wise, $\lambda = 0.0$, the images produced focus on a white mass in the center, while going fully feature-wise, $\lambda = 1.0$, the images obtained become noisier but with more strokes

and characteristics.

Future Work

Merging ACGAN and WGAN-GP for lung nodules could potentially increase the quality of images. The auxiliary classifier creates better quality and diversity of images than a DCGAN or SGAN model while using the MNIST dataset. Unfortunately, ACGAN for NLST did not produce desirable results. On the other hand, as shown in our results, WGAN-GP is able to produce visually appealing lung nodules images.

The comparison of different generative models is a hard problem. One of the approaches, as mentioned in chapter 2, was to use inception score. The inception score depends on a good classifier, which is usually at hand for benchmark datasets such as MNIST or CIFAR. For the NLST dataset, we could use the same idea and use pre-defined classifier as the evaluator of different generative models.

In the context of mapping to the latent space using backpropagation, we set an initial point z_1 at random. Each time we run the algorithm, we match the target image X with a different score since the noise z_i is stochastic. Instead, we could rely on multiple initial points z_1, \dots, z_n and calculate the average score.

The inverse mapping technique used, for mapping images to the latent space, was implicit type. We leave ALIGAN [33] and BIGAN [34] as future research. The model WGAN-GP could also be combined with this two models to produce a new type of architecture.

The construction of the histograms done during anomaly detection were used with the validation dataset. Would be interesting to see how much the mean between training set and validation set differ. The difference between training and validation score could give indications of overfitting.

$$\frac{1}{n} \sum |x - P_{\hat{x} \sim Q(z|x)}(\hat{x}|z)|$$

Bibliography

- [1] I. Goodfellow, “Nips 2016 tutorial: Generative adversarial networks,” *arXiv preprint arXiv:1701.00160*, 2016.
- [2] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” *arXiv preprint arXiv:1611.07004*, 2016.
- [3] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, “Photo-realistic single image super-resolution using a generative adversarial network,” *arXiv preprint arXiv:1609.04802*, 2016.
- [4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, pp. 2672–2680, 2014.
- [5] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.
- [6] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” in *Advances in Neural Information Processing Systems*, pp. 2234–2242, 2016.
- [7] hindupuravinash, “the-gan-zoo,” jul 2017. Obtained from <https://github.com/hindupuravinash/the-gan-zoo>.
- [8] GKalliatakis, “Delving-deep-into-gans,” jul 2017.
- [9] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, “Unsupervised anomaly detection with generative adversarial networks to guide marker discovery,” in *International Conference on Information Processing in Medical Imaging*, pp. 146–157, Springer, 2017.
- [10] K. Kamnitsas, C. Baumgartner, C. Ledig, V. Newcombe, J. Simpson, A. Kane, D. Menon, A. Nori, A. Criminisi, D. Rueckert, *et al.*, “Unsupervised domain adaptation in brain lesion segmentation with adversarial networks,” in *International Conference on Information Processing in Medical Imaging*, pp. 597–609, Springer, 2017.
- [11] S. Kohl, D. Bonekamp, H.-P. Schlemmer, K. Yaqubi, M. Hohenfellner, B. Hadaschik, J.-P. Radtke, and K. Maier-Hein, “Adversarial networks for the detection of aggressive prostate cancer,” *arXiv preprint arXiv:1702.08014*, 2017.

- [12] D. Nie, R. Trullo, J. Lian, C. Petitjean, S. Ruan, Q. Wang, and D. Shen, “Medical image synthesis with context-aware generative adversarial networks,” in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 417–425, Springer, 2017.
- [13] P. Costa, A. Galdran, M. I. Meyer, M. D. Abràmoff, M. Niemeijer, A. M. Mendonça, and A. Campilho, “Towards adversarial retinal image synthesis,” *arXiv preprint arXiv:1701.08974*, 2017.
- [14] Y. Xue, T. Xu, H. Zhang, R. Long, and X. Huang, “Segan: Adversarial network with multi-scale l_1 loss for medical image segmentation,” *arXiv preprint arXiv:1706.01805*, 2017.
- [15] A. Ng, “Machine learning.” Video, jun 2017. Week 9 Anomaly Detection.
- [16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, jun.
- [17] L. C. L. T. Marco A.F. Pimentel, David A. Clifton, “A review of novelty detection,” jan 2014.
- [18] P. Seeböck, S. Waldstein, S. Klimesch, B. S. Gerendas, R. Donner, T. Schlegl, U. Schmidt-Erfurth, and G. Langs, “Identifying and categorizing anomalies in retinal imaging data,” *arXiv preprint arXiv:1612.00686*, 2016.
- [19] S. Mohamed and B. Lakshminarayanan, “Learning in implicit generative models,” *arXiv preprint arXiv:1610.03483*, 2016.
- [20] S. Nowozin, B. Cseke, and R. Tomioka, “f-gan: Training generative neural samplers using variational divergence minimization,” in *Advances in Neural Information Processing Systems*, pp. 271–279, 2016.
- [21] F. Chollet *et al.*, “Keras.” <https://github.com/fchollet/keras>, 2015.
- [22] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” *arXiv preprint arXiv:1603.07285*, 2016.
- [23] vdumoulin, “Convolution arithmetic.” https://github.com/vdumoulin/conv_arithmetic, 2017.
- [24] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv preprint arXiv:1511.06434*, 2015.
- [25] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International Conference on Machine Learning*, pp. 448–456, 2015.
- [26] A. Odena, C. Olah, and J. Shlens, “Conditional image synthesis with auxiliary classifier gans,” *arXiv preprint arXiv:1610.09585*, 2016.
- [27] A. Odena, “Semi-supervised learning with generative adversarial networks,” *arXiv preprint arXiv:1606.01583*, 2016.

- [28] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein gan,” *arXiv preprint arXiv:1701.07875*, 2017.
- [29] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, “Improved training of wasserstein gans,” *arXiv preprint arXiv:1704.00028*, 2017.
- [30] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *arXiv preprint arXiv:1607.06450*, 2016.
- [31] R. Yeh, C. Chen, T. Y. Lim, M. Hasegawa-Johnson, and M. N. Do, “Semantic image inpainting with perceptual and contextual losses,” *arXiv preprint arXiv:1607.07539*, 2016.
- [32] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, “Autoencoding beyond pixels using a learned similarity metric,” *arXiv preprint arXiv:1512.09300*, 2015.
- [33] V. Dumoulin, I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. Courville, “Adversarially learned inference,” *arXiv preprint arXiv:1606.00704*, 2016.
- [34] J. Donahue, P. Krähenbühl, and T. Darrell, “Adversarial feature learning,” *arXiv preprint arXiv:1605.09782*, 2016.
- [35] M. A. M. M. Soumith Chintala, Emily Denton, “How to train a gan? tips and tricks to make gans work,” jul 2017.
- [36] A. Odena, V. Dumoulin, and C. Olah, “Deconvolution and checkerboard artifacts,” *Distill*, 2016.
- [37] L. Devroye, “Random variate generation in one line of code,” in *Simulation Conference, 1996. Proceedings. Winter*, pp. 265–272, IEEE, 1996.
- [38] Shakir, “Machine learning trick of the day (4): Reparameterisation tricks,” oct 2015.
- [39] L. Theis, A. v. d. Oord, and M. Bethge, “A note on the evaluation of generative models,” *arXiv preprint arXiv:1511.01844*, 2015.
- [40] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. Software available from tensorflow.org.
- [41] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, “Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling,” in *Advances in Neural Information Processing Systems*, pp. 82–90, 2016.
- [42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.