Generative Adversarial Data Augmentation for Lung Nodule Detection

Zichen Cui
Minerva Schools at KGI
San Francisco, CA 94102
jeffcui@minerva.kgi.edu

March 24, 2019

Abstract

Medical datasets can suffer from significant class imbalance. In 3D patch-based lung nodule detection, for example, only 1% of all sampled volumes contain a nodule. To alleviate class imbalance, we propose using a 3D residual Wasserstein GAN with gradient penalty and an auxiliary classifier loss to synthesize fake samples that contain lung nodules to augment the imbalanced dataset, and then train a classifier with the generatively augmented dataset. We demonstrate that the GAN is able to generate novel samples distinct from the training set images, and that the generatively augmented classifier has a slightly lower sensitivity but significantly reduced false positive rate, improved F1 score, and higher precision compared to the baseline model.

1 Introduction

1.1 Motivation

This thesis is motivated by the class-imbalance problem in computer-assisted lung nodule detection in pulmonary computed tomography (CT) scans. Lung nodules are small, round growths in the lung. Most lung nodules are benign, but some are cancerous; effective detection of lung nodules can help with early detection of lung cancer. Recently, patch-based 3D convolutional architectures have proven to yield satisfactory results in assisting radiologists in lung nodule detection: for each case, the lung volume is extracted and cropped into 3D patches; then, a classifier neural network predicts whether there exists a nodule within each patch, alleviating the radiologist's workload and assisting with the discovery of lung nodules [1].

Like many medical datasets, however, the problem of class imbalance is especially significant in lung nodule detection. For example, in this thesis, we use the Lung Nodule Analysis 2016 (LUNA16) challenge dataset. After preprocessing, 1.4% of all patches are positive (with nodule), and 98.6% are negative (without nodule). Conventional data augmentation, such as random flips and rotations, is widely used to improve model performance. Recently, generative adversarial networks (GANs) have been shown to generate realistic images resembling real ones.

1.2 Thesis

Therefore, following the line of reasoning above, the thesis is that, for class-imbalance problems, conventional data augmentation only utilizes the minority-class information. Instead, we can generatively augment the dataset such that the generative model takes advantage of information across all classes.

In this paper, we demonstrate on the Lung Nodule Analysis 2016 (LUNA16) challenge dataset that an auxiliary classifier generative adversarial network (ACGAN) with a 3D residual structure, Wasserstein loss and a gradient penalty (reviewed in the section below) can synthesize novel samples to complement

the minority class and improve the classifier performance for the lung nodule detection problem, and point out possible further improvements based on the thesis experiments.

1.3 Literature Review

1.3.1 Neural Network Basics

Universal Approximation Theorem A neural network can be thought of as a sophisticated parametrized function that can approximate any mapping from a set of inputs to a set of outputs. Let us begin with a simple single-layer neural network. The universal approximation theorem [2] states that, a single-layer fully-connected neural network with a nonlinear activation function φ, represented as

$$F(x) = \sum_{i=1}^{N} v_i \varphi(w_i^T x + b_i); v_i, w_i, b_i \text{ parameters}$$

can approximate any function $f : \mathbb{R}^n \to \mathbb{R}$.

Supervised Training of Neural Networks Although a good theoretical foundation, [2] does not tell us how to obtain a working neural network. In practice, the training of a neural network is formulated as an optimization problem, typically to minimize a measure of error, or "wrongness", called the loss function. In machine learning, a **supervised** problem is one in which we have both the input information $\boldsymbol{X}$ (**features**) and the desired correct output $\boldsymbol{y}$ (**ground truth**). (If we do not have the ground truths, it is **unsupervised**.) Then, we formulate a loss function L that measures how wrong the model output is, given the ground truth $\boldsymbol{y}$ and the model prediction $\hat{\boldsymbol{y}} = F_{\boldsymbol{\theta}}(\boldsymbol{X})$, such that $L(\boldsymbol{y}, \hat{\boldsymbol{y}}))$ is minimized when $\boldsymbol{y} = \hat{\boldsymbol{y}}$. Given a single-layer neural network denoted $F_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$, finding the neural network parameters is solving the optimization

$$\boldsymbol{\theta} := \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, L(\boldsymbol{y}, \hat{\boldsymbol{y}})) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, L(\boldsymbol{y}, F_{\boldsymbol{\theta}}(\boldsymbol{X}))$$

given the input data. Suppose we have the single-layer network as formulated above; we can use gradient descent to optimize on v_i, w_i, b_i and minimize the loss. Each step, we numerically compute $\partial L/\partial \boldsymbol{v}, \partial L/\partial \boldsymbol{w}, \partial L/\partial \boldsymbol{b}$ and update $\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{b}$ accordingly.

Multilayer Perceptron Practical neural networks consist of a sequence of multiple layers. Therefore, to efficiently optimize on the parameters of multiple layers, there is one more development from the single-layer network. In a deep neural network consisting of many layers, each layer can be thought of as the single-layer function above. With this representation, let F_k be the k-th layer of

a N-layer neural network. Given an input $\boldsymbol{X}_0$, our model output becomes

$$\hat{\boldsymbol{y}} := F_N(F_{N-1}(F_{N-2}(F_{...}(F_2(F_1(\boldsymbol{X_0}))))))$$

where each F_k is formulated as

$$F(\boldsymbol{X}) = \varphi(\boldsymbol{w}^T\boldsymbol{X} + \boldsymbol{b})$$

Each layer's input is the previous layer's output, and the first layer's input is the raw features $\boldsymbol{X}_0$. This is called the *multilayer perceptron* [2]. For convenience below, here we denote the output of each layer F_k as $\boldsymbol{X}_k$.

Optimization by Backpropagation Optimizing the parameters of multiple layers requires the concept of backpropagation, first proposed by [3]. It answers this question: given the desired change (**gradient**) for the entire model's output, what is the gradient for each of the layer parameters, for all of the layers? In its essence, the idea of backpropagation is that to eventually change the output of a layer $\boldsymbol{X}_k = F_k(\boldsymbol{X}_{k-1}) = \varphi(\boldsymbol{w}_k^T\boldsymbol{X}_{k-1}+\boldsymbol{b}_k)$, there are three things we can change: $\boldsymbol{X}_{k-1}, \boldsymbol{w}_k$, and $\boldsymbol{b}_k$. Of these three, $\boldsymbol{X}_{k-1}$ is the input of this layer, which is also the output of the previous layer F_{k-1}; $\boldsymbol{w}_k, \boldsymbol{b}_k$ are the parameters of the layer. To compute the gradient for each of them is a straightforward application of chain rule:

$$\frac{\partial F_k(\boldsymbol{X}_{k-1})}{\partial \boldsymbol{w}_k} = \varphi'(\boldsymbol{w}_k^T\boldsymbol{X}_{k-1} + \boldsymbol{b}_k)\boldsymbol{X}_{k-1}^T$$

$$\frac{\partial F_k(\boldsymbol{X}_{k-1})}{\partial \boldsymbol{b}_k} = \varphi'(\boldsymbol{w}_k^T\boldsymbol{X}_{k-1} + \boldsymbol{b}_k)\mathbf{1} = \varphi'(\boldsymbol{w}_k^T\boldsymbol{X}_{k-1} + \boldsymbol{b}_k)$$

$$\frac{\partial F_k(\boldsymbol{X}_{k-1})}{\partial \boldsymbol{X}_{k-1}} = \varphi'(\boldsymbol{w}_k^T\boldsymbol{X}_{k-1} + \boldsymbol{b}_k)\boldsymbol{w}_k^T$$

And from there, since $\partial F_k(\boldsymbol{X}_{k-1})$ is given, we can extract $\partial \boldsymbol{w}, \partial \boldsymbol{b}, \partial \boldsymbol{X}_{k-1}$ to get the gradients.

The brilliance of backpropagation is in realizing that 1) we can compute the gradients for the layer parameters $\partial \boldsymbol{w}_k, \partial \boldsymbol{b}_k$ and the layer input $\partial \boldsymbol{X}_{k-1}$ as long as we know the gradient for the layer output $\partial F_k(\boldsymbol{X}_{k-1})$; and 2) this layer's input X_{k-1} is the previous layer's output, and therefore we now have the gradient for the previous layer's output, enabling us to apply backpropagation recursively, layer by layer.

Vanishing Gradients and Exploding Gradients Continuing the backpropagation explorations above, a common problem with backpropagation is vanishing gradients. In backpropagation, each previous layer's gradients depend on this layer's gradients. We can see that $\partial \boldsymbol{X}_{k-1}$ for $k = N, N-1, N-2, \ldots, 2$ is divided many times by $\varphi'(\boldsymbol{w}_k^T\boldsymbol{X_{k-1}} + \boldsymbol{b}_k)\boldsymbol{w}_k^T$. If this denominator term is consistently large in absolute value, we get vanishing gradients as we go deeper. If it is consistently small in absolute value, we get exploding gradients as we go deeper. Batch normalization can alleviate this issue and improve training stability. It is widely used and will not be discussed at length in this thesis, but the curious reader can refer to [4].

Generalization It is apparent that the line of reasoning above generalizes to any differentiable operation, not only the linear transformation $\boldsymbol{wX} + \boldsymbol{b}$ and the differentiable activation function φ in the example above. As long as we are going through a sequence of differentiable operations, we can recover gradients for all parameters involved. For example, convolutions are widely used in neural networks to process images.

1.3.2 Residual Network (ResNet)

Residual networks are neural networks where each layer is modeled as $F_k(\boldsymbol{X}_{k-1}) + \boldsymbol{X}_{k-1}$ instead of only $F_k(\boldsymbol{X}_{k-1})$. Instead of mapping an input to an entirely new output, each layer only adds a residual adjustment to the original input.

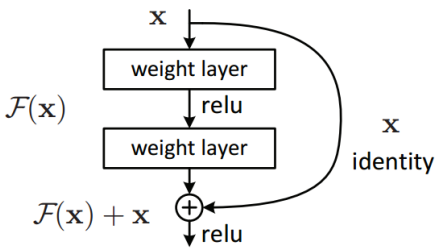

Figure 1: The residual block [5].

This "$+X_{k-1}$" operation is called a "skip connection" in the original paper [5]. We can see that given this residual form, the partial derivative $\partial(F_k(\boldsymbol{X}_{k-1}) + \boldsymbol{X}_{k-1})/\partial \boldsymbol{X}_{k-1} = F_k'(\boldsymbol{X}_{k-1}) + \mathbf{1}$. Indeed, this helps with the vanishing gradient problem discussed in 1.3.1: even if $F_k'(\boldsymbol{X}_{k-1})$ vanishes, we still have the identity passing along the gradient to deeper layers. This helps the flow of gradients, accelerates training convergence, and achieves significantly better performance [5].

1.3.3 Generative Adversarial Network (GAN)

The concept of Generative Adversarial Network (GAN) is first proposed in Goodfellow's seminal paper [6]. In a conventional neural network, the formulated optimization objective is to minimize some loss function measuring a distance between the prediction and the ground truth. This works well for supervised problems. However, with a conventional neural network, it is not immediately apparent how to approximate an arbitrary training data distribution and generate new samples that resemble the training data samples in a completely unsupervised manner. [6] solves this by introducing two networks trying to maximize the loss of one another: a generator and a discriminator.

The generator G maps a random latent noise $z \sim \mathcal{N}(0, 1)$ to a sample $G(z)$. The discriminator classifies whether a sample is real or fake. During training, the discriminator sees both real and fake samples.

Discriminator Loss The discriminator outputs `0` for fake, and `1` for real samples. Formally, the loss function for the discriminator is in effect a negated binary cross-entropy loss:

$$\max_{\theta_d} \left[\mathbb{E}_{x\sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z\sim p(z)} \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)\right]$$

Generator Loss The loss function for the generator is to maximize the likelihood of the discriminator classifying generated samples as real ones:

$$\max_{\theta_g} \mathbb{E}_{z\sim p(z)} \log\left(D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)$$

[6] further proves that this is equivalent to minimizing the Jenson-Shannon divergence (JS divergence) between the real and fake data distributions. However, training is often unstable with higher-dimensional data, as we will discuss below in 1.3.5.

1.3.4 Auxiliary Classifier GAN (ACGAN)

The original GAN discriminator only outputs a binary classification (real or fake). To generate conditional samples based on labeled classes (e.g. only dogs, only cats, etc.), [7] makes two additions to the original GAN.

The generator now takes in both the latent noise z, and one-hot encoded class labels.

The discriminator now outputs both a real-or-fake binary classification, and an auxiliary classification predicting which class the sample belongs to. Accordingly, we add a cross-entropy loss to the original discriminator loss function, in which the ground truths are either the labels fed to the generator for the fake data, or the real labels sampled from the training dataset.

We also add a cross-entropy loss to the original generator loss, in which the ground truths are the discriminator's class predictions.

This allows us to use supervised class labels to train an otherwise unsupervised generative model, and generate conditional samples [7].

1.3.5 Wasserstein GAN with Gradient Penalty (WGAN-GP)

Problems with JS divergence The original GAN formulation minimized the JS divergence between the target distribution P_{real} and the generated distribution P_g, proven in section 4.1 in [6]. The JS divergence, however, does not provide a gradient when P_{real} and the fake distribution P_g has no overlapping support. Recall that the JS divergence is a symmetric arrangement of Kullback–Leibler divergence (KL divergence), here denoted KL:

$$\text{JS}(P_{\text{real}} \mid\mid P_g) = \frac{1}{2}\text{KL}(P_{\text{real}} \mid\mid M) + \frac{1}{2}\text{KL}(P_g \mid\mid M)$$

Where M is the average mixture between the two distributions in question:

$$M = \frac{1}{2}(P_{\text{real}} + P_g)$$

And the KL divergence is defined as

$$\mathrm{KL}(P \mid\mid Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

Note that when there is no support overlap between the two distributions (i.e. $\forall x \in \mathcal{X}, P_{\text{real}}(x) P_g(x) = 0$), the KL divergence will be constant:

$$\mathrm{KL}(P_{\text{real}} \mid\mid M) = \sum_{x \in \mathcal{X}} P_{\text{real}}(x) \log \left(\frac{P_{\text{real}}(x)}{M(x)} \right)$$

$$= \sum_{x \in \mathcal{X}} P_{\text{real}}(x) \log \left(\frac{P_{\text{real}}(x)}{\frac{1}{2} P_{\text{real}}(x) + \frac{1}{2} P_g(x)} \right) = \log 2$$

Same reasoning gives us $\mathrm{KL}(P_g \mid\mid M) = \log 2$. Therefore, in the case of no support overlap, the JS divergence is constant:

$$\mathrm{JS}(P_{\text{real}} \mid\mid P_g) = \log 2$$

Of course, this constant term gives us a 0 gradient. One can see this quickly becomes a problem in higher-dimensional generative problems such as synthesizing high-resolution human faces: while there are countless **possible** pixel values, there are only a few types of **probable** human faces, giving us an extremely sparse distribution over the space. In this case, the support overlap vanishes and training does not converge because we have a zero gradient.

Wasserstein distance [8] proposes that, instead of JS-divergence, we should use the Wasserstein distance defined on the 1^{st} moment as the loss function, denoted W below. The 1^{st} moment of a distribution is simply its mean.

Intuitively, the Wasserstein distance is also called the earth-mover distance: if we have two probability distributions μ, ν (piles of dirt), for all point masses (dirt particles) in μ, the minimal total distance required to move the a certain-shaped μ into a new shape ν is $W(\mu, \nu)$.

Formally, define a transport plan γ, which is a joint distribution over μ, ν. For each pair of coordinates $x, y \in \gamma$, x denotes the source, y denotes the destination, and $\gamma(x, y)$ denotes the planned amount of mass movement from x to y. Define all valid transport plans as $\Pi(\mu, \nu)$. l_2 norm is used as the distance metric. Then, the (1^{st}-moment) Wasserstein distance is defined as

$$W(\mu, \nu) = \inf_{\gamma \in \Pi(\mu, \nu)} \mathbb{E}_{(x,y) \sim \gamma} (\|x - y\|_2)$$

Which has a gradient everywhere even when there is no support overlap. This leads to stable training even on high-dimensional data.

W is hard to calculate, but [8] notes that the dual representation of W can be approximated:

$$W(\mu, \nu) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_\mu}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\nu}[f(x)]$$

Where $\|f\|_L \leq 1$ denotes 1-Lipschitzness, meaning that the $\|\nabla f(x)\| \leq 1$ everywhere.

Intuitively, the original GAN discriminator minimized binary cross-entropy, which was a classification problem requiring a sigmoid nonlinearity squishing $\mathbb{R} \to (0, 1)$ and diminishing its gradient by doing so. Now the discriminator (called **critic** in [8]) scores each sample and maximizes the score difference between real and fake samples, and the generator tries to maximize the score of its generated samples, yielding a much smoother gradient even when there are no support overlaps.

To enforce 1-Lipschitzness, [9] adds a regularization term to the critic loss such that the critic gives a well-behaved gradient with this soft constraint:

$$\texttt{gradient penalty} = \lambda(\|\nabla_{\hat{x}} f(\hat{x})\|_2 - 1)^2$$

Where λ is a scaling hyperparameter, $\hat{x}$ is a random linear combination of real and fake samples ($\hat{x} = \epsilon x_{\text{real}} + (1 - \epsilon)x_{\text{fake}}, \epsilon \sim U[0, 1]$). The regularization term penalizes any gradient away from norm 1.

1.3.6 Deep Convolutional GAN (DCGAN) Data Augmentation for Chest X-ray

[10] deals with a heavily class-imbalanced chest X-ray dataset. In their dataset, there were more than 12,000 cardiomegaly, effusion and normal cases, but only 3,018 edema cases and 2,013 pneumothorax cases. DCGAN data augmentation was applied to the original dataset to synthesize additional images to complement the original dataset and achieve class balance. They demonstrated an increase in classification accuracy from 70.9% to 92.1%.

1.3.7 GAN Data Augmentation for Liver Lesion Segmentation

[11] applied DCGAN to synthesize 3 classes of liver lesion samples: cysts, metastases, and hemangiomas. Their dataset consists of only 182 cases. In conclusion, they reported a conventionally augmented classifier yielding 78.6% sensitivity and 88.6% specificity, and a GAN augmented classifier yielding 85.7% sensitivity and 92.4% specificity.

2 Dataset

In this thesis, we work with the Lung Nodule Analysis 2016 (LUNA16) challenge dataset [12], which is a processed subset of the Lung Image Database Consortium and Image Database Resource Initiative (LIDC/IDRI) dataset [13]. The base dataset LIDC/IDRI contains 7371 lesions that are marked by at least 1 of the 4 reviewing radiologists; each annotation includes the position and the estimated nodule size. In short, the LUNA16 dataset is a subset of the LIDC/IDRI dataset including only CT scans with a slice thickness less than 2.5 mm, and only nodules that are at least 3 mm in size and marked by at least 3 of the 4 radiologists.

2.1 Preprocessing

2.1.1 Aspect Ratio

The LUNA16 dataset consists of 888 CT scans in DICOM format [14]. For each patient, the corresponding DICOM files are loaded to reconstruct the original CT scan. Because we intend to use a convolutional classifier to detect nodules, it is common practice to standardize the aspect ratios across three axes. Since DICOM files contain scanner metadata, we can do this by interpolating all scans according to the resolutions recorded in the metadata such that the interpolated axis resolutions are all $1 \times 1 \times 1$ (mm/pixel). This ensures that all nodules appear similarly shaped for the classifier.

2.1.2 Hounsfield Unit Normalization

CT scanners are calibrated to return voxel intensity values on the Hounsfield scale. The Hounsfield scale is a standard for radiodensity measurements: at standard pressure and temperature, the radiodensity of air is defined as -1000 Hounsfield Units (HU), and water defined as 0 HU. Cancellous bones typically measure 350 to 400 HU [15], and lung tissues typically measure -800 to -700 HU [16]. Most information falls within the range of $[-1000, 400]$. Therefore, raw volumes are clipped at $[-1000, 400]$ and linearly transformed to $[-1, 1]$ for the models.

2.1.3 Lung Segmentation

The lung is segmented with a traditional computer vision approach. First, the image is binarized such that all voxels with less than -500 HU is selected. This effectively selects all voxels that contain air. Then, we use OpenCV [17] to perform a morphological closing operation to remove the holes within the mask and to include the lung tissues. The largest volume is selected and considered as the lung [18].

2.1.4 Positive and Negative Cube Samples

Positive sample cubes are cropped from the LUNA16 dataset annotations. Negative cubes are generated by randomly sampling parts of lung volumes that do not contain the annotated nodules. All cubes are sized $48 \times 48 \times 48$ voxels.

2.1.5 Train / Validation / Test Split

In our experiments, 70% of the dataset was used for training, 10% for validation, and 20% is kept away as the test set. Both the GANs and the lung nodule detection classifiers are only trained on the training set to avoid information leakage.

In total, below are the resulting numbers of samples after all preprocessing steps.

	Positive	Negative
Training (70%)	5712	406791
Validation (10%)	816	58113
Test (20%)	1632	116226

Table 1: Dataset sample counts.

2.1.6 Conventional Data Augmentation

Random flips on the x, y, z axes are done to augment the positive samples. The negative samples are used as is.

2.1.7 Processed Samples

After preprocessing, typical positive and negative samples are $48 \times 48 \times 48$ voxels with a 1 mm / voxel resolution, normalized to $[-1, 1]$. Each sample is shown here slice by slice in reading order from superior (to the head) to inferior (to the feet).

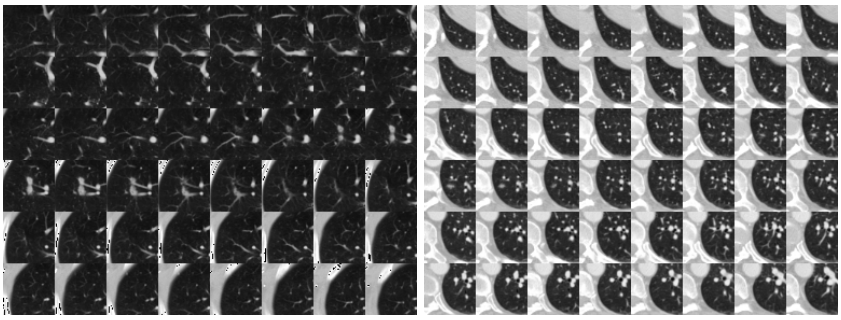

Figure 2: Positive samples from LUNA16

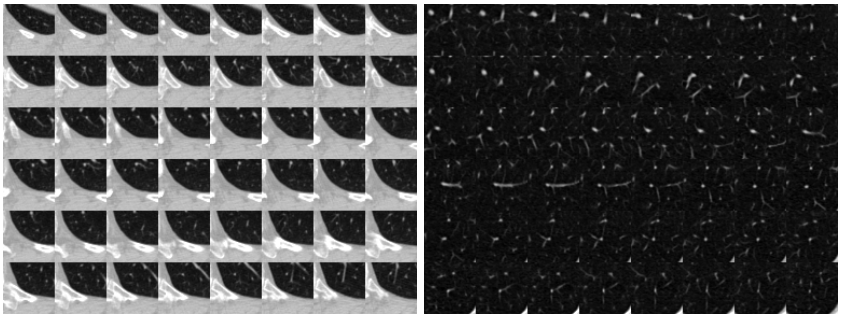

Figure 3: Negative samples from LUNA16

3 Model Architecture

3.1 Auxiliary Classifier Wasserstein GAN with Gradient Penalty

The GAN architecture combines all sections in the literature review. It is a Wasserstein GAN with gradient penalty (1.3.5) and an auxiliary classification loss (1.3.4). It has a 3D residual architecture (1.3.2).

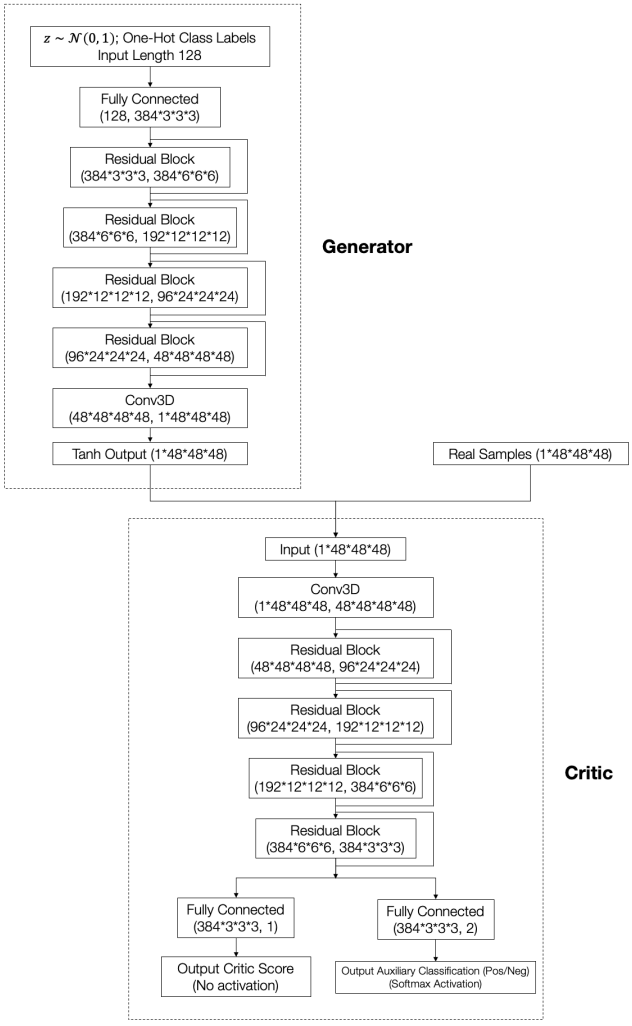

Figure 4: Auxiliary classifier Wasserstein GAN architecture.

The generator transforms a conditional random Gaussian noise to a $48 \times 48 \times 48$

grayscale volume. To generate such a conditional random Gaussian noise, first we randomly sample a Gaussian noise vector $\boldsymbol{z} \sim \mathcal{N}(0, 1)$, and then overwrite the first N numbers with the one-hot encoding of desired class labels, where N is the number of classes. Specifically, in our case $N = 2$ and we encode `[0,1]` for positives and `[1,0]` for negatives. The resulting vector is fed through the generator to synthesize a fake sample cube.

The critic processes an input image (both real and fake samples) and gives two outputs: the critic score, and the auxiliary classification. The critic score is an estimate of the sample "realness". Formally, it is used in the dual form of the 1-Wasserstein distance between the real and synthesized data distributions, as described in section 1.3.5 above. The auxiliary classification is the critic prediction for the sample class (positive or negative), as described in section 1.3.4 above. This label is later used for the auxiliary classification loss to incentivize both the generator and the critic to not only synthesize and distinguish realistic samples, but also correctly approximate conditional distributions.

Layer normalization [19] and leaky ReLU activation [20] are used after each convolution layer except output layers.

3.1.1 Residual Block

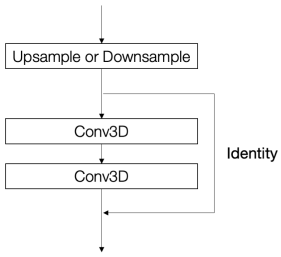

Figure 5: Upsamling / downsampling residual block.

As described in section 1.3.2 above, the residual block here is a 3D convolutional block containing a skip connection. In the generator, the residual blocks performs upsampling by nearest neighbor interpolation. In the discriminator, the residual blocks performs downsampling with 3D average pooling.

3.2 Lung Nodule Detection Classifier

The lung nodule detection classifier is a small model that quickly downsamples the input image with repeated max pooling and strided convolution operations. Both the max pooling and the strided convolution downsamples each axis in half. This model is designed to be quick to train and evaluate as the thesis is limited in time and compute.

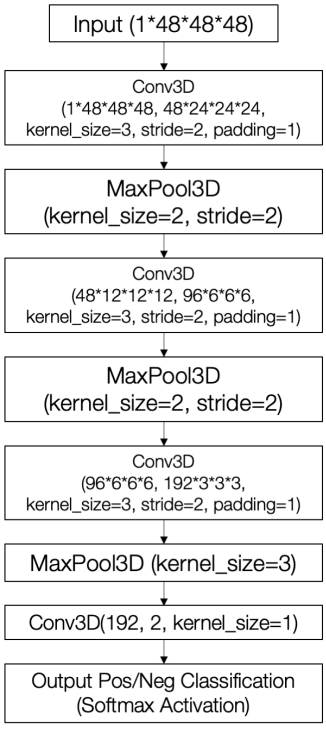

Figure 6: Lung nodule detection classifier architecture.

3.3 Loss Definitions

3.3.1 Critic Loss

The critic loss consists of the vanilla Wasserstein GAN loss, the gradient penalty, and the auxiliary classification cross-entropy loss.

Formally, suppose $(X_{\text{real}}, y_{\text{real}}) \sim \mathbb{P}_{\text{real}}$ are real images and labels sampled from the real distribution. Randomly sample y_{fake}, the desired class labels for the fake samples. Sample $\boldsymbol{z} \sim \mathcal{N}(0, 1)$, the corresponding conditional Gaussian noise. Let the critic score be $f(X)$, and the critic auxiliary classification be $\hat{y}$.

For each mini-batch, we sample both real and synthesized images and concatenate them into one batch. Let $X = X_{\text{real}} \frown X_{\text{fake}}, y = y_{\text{real}} \frown y_{\text{fake}}$, where $\frown$ denotes concatenation.

For the gradient penalty, as described by [9], let $\epsilon \sim \mathcal{U}[0, 1]$ be a uniform random scalar; let the generator be G, and generated samples be $X_{\text{fake}} = G(\boldsymbol{z})$.

Let $\hat{X} = \epsilon X_{\text{real}} + (1 - \epsilon) X_{\text{fake}}$, a linear interpolation between real and generated samples.

Now define two hyperparameters: let λ be the gradient penalty scale, and α be the auxiliary classification loss scale.

Then, the critic loss can be written as

$$L_f = f(X_{\text{fake}}) - f(X_{\text{real}}) + \lambda(\|\nabla_{\hat{X}} f(\hat{X})\|_2 - 1)^2 + \left(-\alpha \sum_{i=1}^{N} y_i \log(\hat{y}_i)\right)$$

Where

- $f(X_{\text{fake}}) - f(X_{\text{real}})$ is the dual form 1-Wasserstein distance;
- $\lambda(\|\nabla_{\hat{X}} f(\hat{X})\|_2 - 1)^2$ the gradient penalty;
- $-\alpha \sum_{i=1}^{N} y_i \log(\hat{y}_i)$ the cross-entropy loss for the critic auxiliary classification.

Intuitively, this incentivizes the critic to maximize the distance between the sample scores between the real and fake samples, keep the gradient norm close to 1, and correctly classify the samples. [9]

3.3.2 Generator Loss

To calculate the generator loss, we generate samples with the generator, and score and classify them with the critic. The generator loss function consists of two terms: the critic score for the fake samples, and the critic auxiliary classification cross-entropy loss, taking the desired label input as ground truth.

Formally, sample some random desired class labels y, sample a corresponding conditional Gaussian noise $z \sim \mathcal{N}(0, 1)$, and $X_{\text{fake}} = G(z)$ same as above. Predict critic score $f(X_{\text{fake}})$ and critic auxiliary classification $\hat{y}$.

Then, the generator loss can be written as

$$L_G = -f(X_{\text{fake}}) + \left(-\alpha \sum_{i=1}^{N} y_i \log(\hat{y}_i)\right)$$

Where

- $-f(X_{\text{fake}})$ is the critic score for the samples;
- $-\alpha \sum_{i=1}^{N} y_i \log(\hat{y}_i)$ is the auxiliary classification loss for the samples.

Intuitively, this incentivizes the generator to improve (minimize) the sample realness score and correctly generate samples conditional on the class labels.

3.3.3 Lung Nodule Detection Classifier Loss

The lung nodule detection classifier predicts a softmax vector for our two classes (positive / negative) with a cross-entropy loss. [9]

4 Methodology

4.1 Wasserstein GAN

4.1.1 Training

The Wasserstein GAN is trained with mini-batch stochastic gradient descent with the Adam optimizer [21] with $\beta_1 = 0, \beta_2 = 0.9$ as suggested by [8]. Batch size 64, learning rate 10^{-4}. The gradient penalty scale is $\lambda = 10$ as suggested by [9]. The auxiliary classifier scale is $\alpha = 1$ at the beginning of the training; at the end of the training when both the generator and the discriminator have stabilized, we tune it to $\alpha = 10$ to penalize wrong classifications.

For each epoch, we train the critic for 5 iterations and the generator for 1 iteration as suggested by [8] to ensure an accurate estimation of the Wasserstein distance by the critic network.

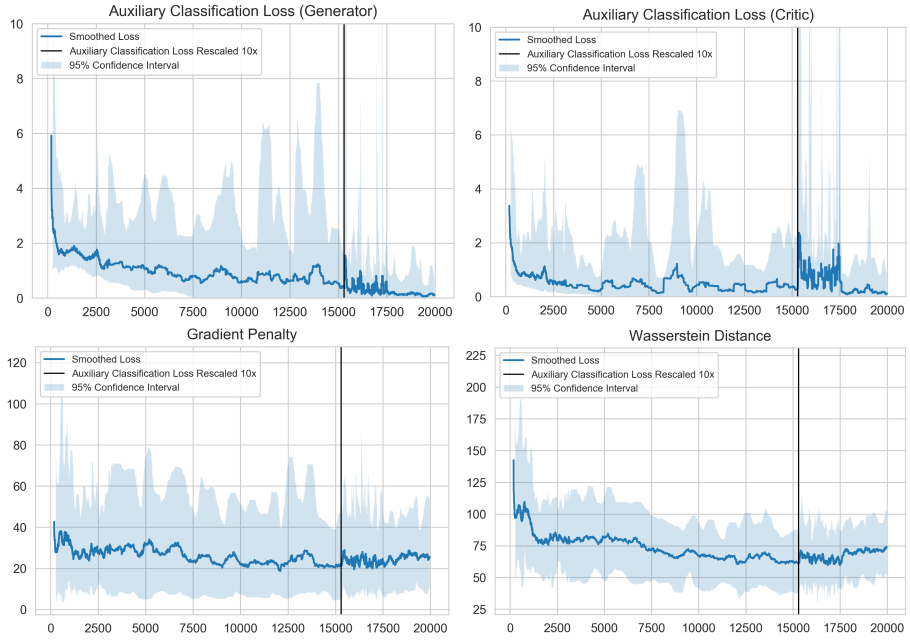

Figure 7: WGAN loss curves.

The Wasserstein GAN training is defined as an unsupervised or semi-supervised problem: for the generator, although it uses the class labels, it tries to fit the conditional distributions without supervised ground truth images that correspond to each Gaussian noise $\boldsymbol{z}$. For the critic, the training is supervised as it has access to ground truths (real/fake and positive/negative). Here we show that the training converges stably with relevant loss plots and generated samples seeded with a

$\boldsymbol{z}$ fixed throughout the training to visualize the progression of sample quality. For the loss plots, the smoothed loss and smoothed 95% confidence intervals for loss values are shown for the critic auxiliary classification cross-entropy, and the Wasserstein distance estimates. In total, the Wasserstein GAN was trained for 20,000 epochs. See loss curves above.

4.2 Conventional Augmentation

Conventional augmentation consists of only random flips along the x, y, z axes. Because the intensity of CT scan voxels is calibrated to a standard Hounsfield scale, we refrained from adding random noises to preserve information.

4.3 Generative Augmentation

We generate fake positive samples to augment the scarce real positive samples. First, we pass random Gaussian noises conditioned on the positive class into the generator and generate positive samples. Then, we use the critic to score and classify each sample, and select the best samples. For each sample, if its critic loss is below some threshold (i.e. it is considered real enough) and the auxiliary classification is confident above a threshold that the generated samples are indeed positives, we will include it in the generated dataset.

4.4 Lung Nodule Detection Classifier

4.4.1 Training

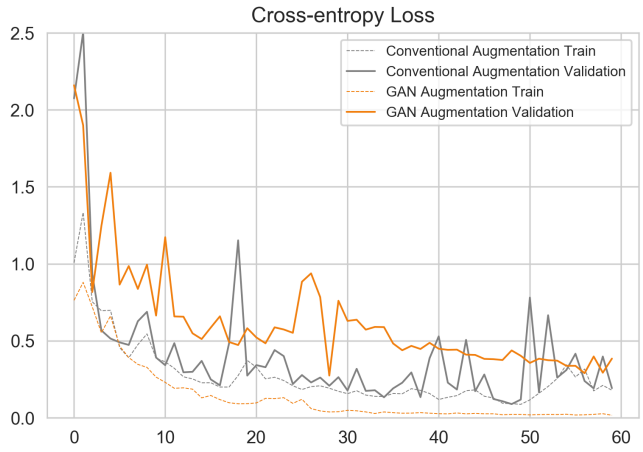

Figure 8: Classifier training and validation loss.

The classifier is trained on each of the two datasets: one run on only the conventionally augmented original dataset, and one run on the mixed dataset of

both real and synthesized data. Mini-batch stochastic gradient descent with the Adam optimizer ($\beta_1 = 0.9, \beta_2 = 0.999$). Batch size 64, initial learning rate 10^{-3}. At epoch 30, the learning rate is decayed to 10^{-4}. Each batch is class-balanced to have 32 positive and 32 negative samples. 60 epochs in total.

See training and validation cross-entropy loss curves above.

5 Results

5.1 GAN Generated Samples

The GAN was able to generate a variety of visually realistic samples that represent pulmonary nodules. To demonstrate that the GAN was in fact generating novel samples instead of simply memorizing the training data, here we show a few generated samples with its l_2-norm nearest neighbor side by side. We can observe the nodule at the center of the volume, and that the samples are indeed visually distinct from the training set. See Appendix A for more images.

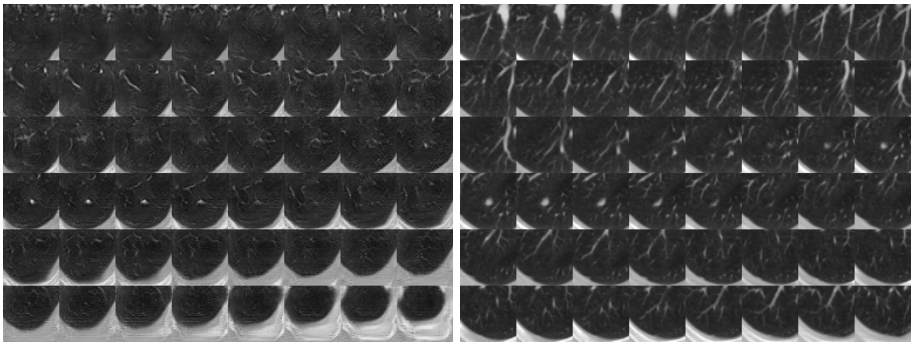

Figure 9: Generated (left) vs. nearest real training sample (right).

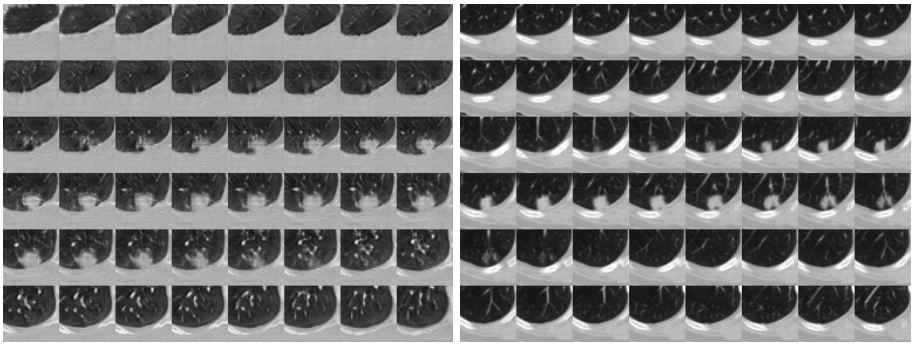

Figure 10: Generated (left) vs. nearest real training sample (right).

5.2 Classification Metrics

The classifier was trained on:

- Only conventionally augmented real data
- Mix of real and GAN synthesized data
- Only GAN synthesized data (no real data)

For each run, the classifier with the best validation performance was kept and evaluated on the test set.

Aside from that, we also compare these specifically trained classifier with the auxiliary classifier in the critic. The critic auxiliary classifiers did not go through additional training; it was directly evaluated on the test set for classification performance.

Below we report the F1 score, sensitivity, specificity, precision, and the confusion matrix on the test set.

5.2.1 Conventional Augmentation Baseline

		Ground Truth	
		Positive	Negative
Prediction	Positive	1583	2776
	Negative	50	113451

Sensitivity:	0.969
Specificity:	0.976
Precision:	0.363
F1:	0.528

Table 2: Classification metrics, conventional augmentation baseline.

5.2.2 Generative Augmentation Mixed with Real Data

		Ground Truth	
		Positive	Negative
Prediction	Positive	1377	750
	Negative	256	115477

Sensitivity:	0.843
Specificity:	0.994
Precision:	0.647
F1:	0.732

Table 3: Classification metrics, generative augmentation mixed with reals.

5.2.3 Completely Fake Data

		Ground Truth	
		Positive	Negative
Prediction	Positive	919	8539
	Negative	714	107688

Sensitivity:	0.563
Specificity:	0.927
Precision:	0.097
F1:	0.166

Table 4: Classification metrics, only generated fake samples.

5.2.4 Critic Auxiliary Classifier Performance

		Ground Truth	
		Positive	Negative
Prediction	Positive	1215	815
	Negative	418	115412

Sensitivity:	0.744
Specificity:	0.993
Precision:	0.598
F1:	0.663

Table 5: Classification metrics, critic auxiliary classifier (no fine-tuning).

5.3 Conclusions

Compared to the conventional augmentation baseline, generative adversarial data augmentation was able to significantly decrease false positives in the lung nodule classification task, resulting in a significant increase in F1 score, precision and specificity; it was, however, somewhat worse in sensitivity compared to the conventional augmentation baseline.

We also make three observations: 1) the classifier trained on completely fake data converged rapidly on the training loss but performed badly on the validation and the test sets; 2) the generatively augmented classifier behaves similarly to the critic auxiliary classifier, failing to detect more subtle nodules; and 3) the Wasserstein distance flattened out around 70 (see 4.1.1) and the Wasserstein GAN failed to drive the Wasserstein distance down to 0 during training.

These observations signify that the model capacity of both the generator and the critic in our Wasserstein GAN are not sufficiently large to generate samples that are indistinguishable from the real data. Given the limited time and computational resources, we believe that this thesis experiment is a promising

proof-of-concept that conditional generative adversarial augmentation can be used to combat class imbalance in lung nodule detection or in other heavily imbalanced datasets, but larger experiments need to be conducted to yield definitive conclusions.

6 Further Research

Based on the experiment results above, we suggest two further areas of research.

One is training a larger Wasserstein GAN architecture and a larger lung nodule detection classifier. As mentioned above in 5.3, we were able to conclude that our model capacity was insufficient for the task of sample generation. Experiments with a larger architecture could yield better results.

Two is adding finer features to condition the GAN. This thesis only conditions the GAN to distinguish between positive and negative classes. An extension to this would be to extract the nodule size (annotated nodule diameter), nodule transparency (solid, part-solid, non-solid), and malignancy score from the LUNA16 dataset and condition the GAN on these additional features [12] [13].

7 Acknowledgements

The author would like to thank LPixel Inc. for providing computational resources for this capstone thesis research, Antoine Choppin for scoping out the project and providing frequent feedback, and Dr. Dennis Romero for his valuable insights on various parts of my thesis.

The author would like to thank Prof. Tony Drummond for advising this capstone thesis project throughout the senior year.

References

[1] Wei Shen, Mu Zhou, Feng Yang, Caiyun Yang, and Jie Tian. Multi-scale convolutional neural networks for lung nodule classification. In *International Conference on Information Processing in Medical Imaging*, pages 588–599. Springer, 2015.

[2] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[3] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[7] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2642–2651. JMLR. org, 2017.

[8] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[9] Martin Arjovsky Vincent Dumoulin Aaron Courville Ishaan Gulrajani, Faruk Ahmed. Improved training of wasserstein gans, 2017.

[10] Hojjat Salehinejad, Shahrokh Valaee, Tim Dowdell, Errol Colak, and Joseph Barfett. Generalization of deep neural networks for chest pathology classification in x-rays using generative adversarial networks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 990–994. IEEE, 2018.

[11] Maayan Frid-Adar, Idit Diamant, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, 321:321–331, 2018.

[12] Arnaud Arindra Adiyoso Setio, Alberto Traverso, Thomas De Bel, Moira SN Berens, Cas van den Bogaard, Piergiorgio Cerello, Hao Chen, Qi Dou, Maria Evelina Fantacci, Bram Geurts, et al. Validation, comparison, and combination of algorithms for automatic detection of pulmonary nodules in computed tomography images: the luna16 challenge. *Medical image analysis*, 42:1–13, 2017.

[13] Samuel G Armato, Geoffrey McLennan, Luc Bidaut, Michael F McNitt-Gray, Charles R Meyer, Anthony P Reeves, Binsheng Zhao, Denise R Aberle, Claudia I Henschke, Eric A Hoffman, et al. The lung image database consortium (lidc) and image database resource initiative (idri): a completed reference database of lung nodules on ct scans. *Medical physics*, 38(2):915–931, 2011.

[14] Mario Mustra, Kresimir Delac, and Mislav Grgic. Overview of the dicom standard. In *2008 50th International Symposium ELMAR*, volume 1, pages 39–44. IEEE, 2008.

[15] Sanjana Patrick, N Praveen Birur, Keerthi Gurushanth, A Shubhasini Raghavan, Shubha Gurudath, et al. Comparison of gray values of cone-beam computed tomography with hounsfield units of multislice computed tomography: An in vitro study. *Indian Journal of Dental Research*, 28(1):66, 2017.

[16] John Kalef-Ezra, A Karantanas, and P Tsekeris. Ct measurement of lung density. *Acta Radiologica*, 40(3):333–337, 1999.

[17] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[18] Guido Zuidhof. Full preprocessing tutorial, national data science bowl 2017, 2017.

[19] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[20] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

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

A More Generated Samples and Nearest Reals

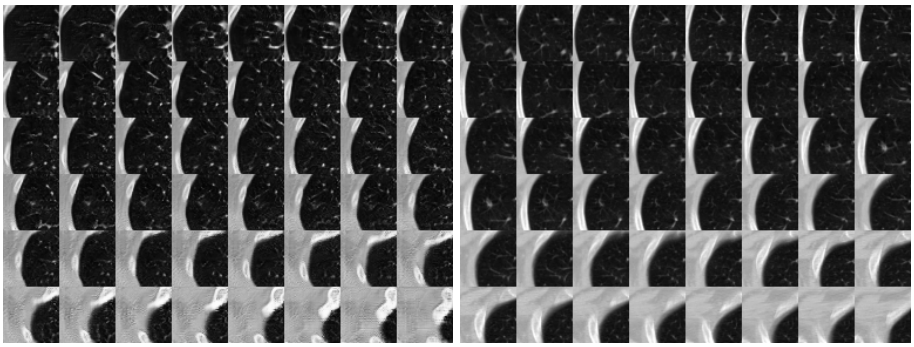

Figure 11: Generated (left) vs. closest real training sample (right).

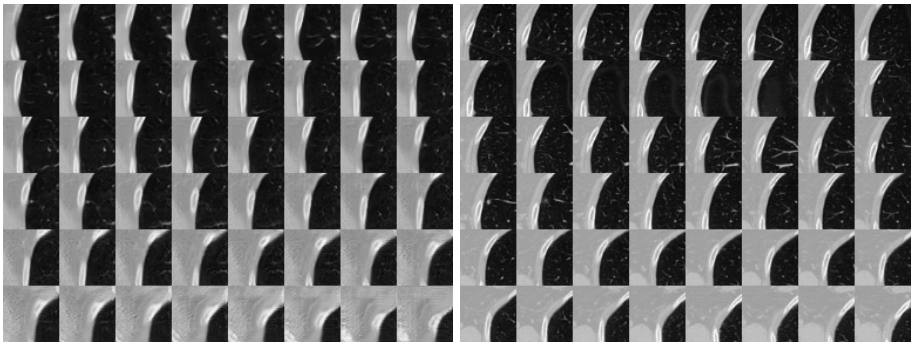

Figure 12: Generated (left) vs. closest real training sample (right).

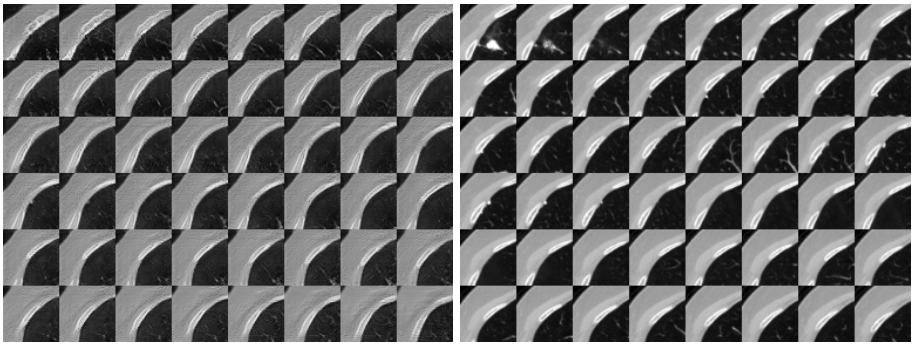

Figure 13: Generated (left) vs. closest real training sample (right).

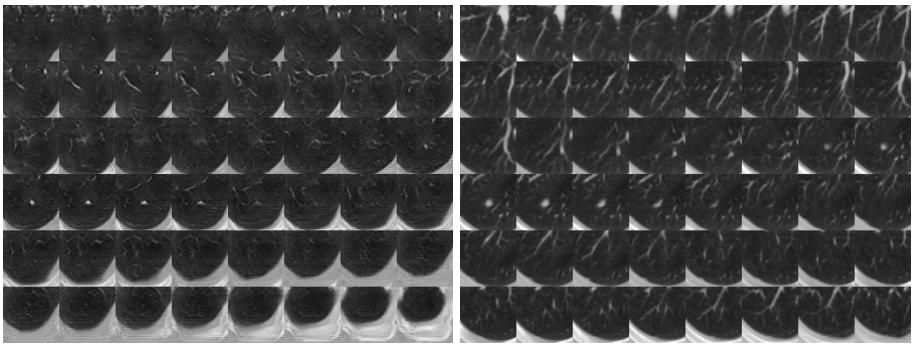

Figure 14: Generated (left) vs. closest real training sample (right).