Generating Synthetic Chest X-Rays with generative modeling

Abhishek Kumar
Department of Computer Science
Stanford University
a6kme@stanford.edu

Juan Pablo Triana Martinez
Department of Computer Science
Stanford University
juan3112@stanford.edu

Abstract

In the field of medical imaging, accurate classification of Chest X-rays (CXRs) is vital for timely and effective diagnosis. The advances of Deep Learning in Computer Vision has enabled the possibility to build CAD (Computer Aided Diagnosis) systems, which can assist the radiologists with CXR diagnosis. However, medical imaging datasets like CXR images are limited in size and availability due to privacy concerns. Due to the diversity of health conditions, anatomical structures, and different angles of the X-rays (posterior-anterior, anterior-posterior, lateral) [1] simple image augmentation methods, such as rotation, flipping, varying lighting etc. would not capture the biological variance, resulting in generation of unrealistic images [6]. Recently, using generative models to augment datasets using synthetic X-rays, have shown to improve classification in multiple studies. [1] [12].

In this study, we explore three generative algorithms, Variational Autoencoders (VAE) [5], Gaussian Mixture Variational Autoencoders (GMVAE) [2], and Generative Adversarial Networks (GAN) [3], for generating synthetic chest X-ray images. Section 1 contains introduction and motivation of this study. Next, Section 2 contains related work which motivates the study. Section 3 and 4 contains description of the algorithms, their formulations, and the dataset used to train the models, CXR-14 [13]. Finally, Section 5 and 6 contains the details about our experiments and results, conclusion, and future work.

1. Introduction

Chest X-ray (CXR) imaging is one of the most relevant, non-invasive diagnostic tools in medical imaging. They are primary diagnostic tool for various conditions like pneumonia, lung cancer, tuberculosis and heart failure. They are often the first imaging technique when patient presents with symptoms like chest pain, persistent cough or shortness of breath. With the development of deep learning in Computer Vision, the possibility to build CAD (Computer Aided Diagnosis) systems which can assist in diagnosis of CXRs has become a reality. However, due to patient privacy laws and high cost of annotations, like other medical imaging datasets, CXR imaging datasets are limited. The need to have reliable generative augmentation methods to create synthetic CXR images for creating and improving CAD tools has become imperative, and also been shown empirically. For instance, during the 2019 COVID pandemic, there has been trials using GAN networks to generate synthetic COVID-19 X-rays for supervised

classification [9] [11]. The inputs to our different generative architectures (VAE, GMVAE, and GAN); are public CXR images - CXR14 dataset and our outputs are generated synthethic CXR images.

2. Related Work

We motivate our study by listing some studies, which have used generative models to generate synthetic images as an augmentation mechanism. Madani *et al*. [6] compared traditional augmentation technique with generated images from GAN and observed over 1% improvement over traditional augmentation and around 2.25% improvement over no augmentation in detecting cardiovascular abnormality. In another study [12], the authors observed around 3% improvement in detection of pneumonia when they augmented a VGG16 classifier with images generated from a GAN. During Covid-19 pandemic, Menon *et al*. used (Mean Teacher + Transfer GAN) MTT-GAN [9] to augment data for a classifier built for binary detection of Covid-19 using very limited Covid-19 CXR images. In another study, Shams *et al*. [11] presented a case study of using GAN networks for augmenting Covid-19 CXR images. A noticeable theme among researches is the common use of GAN networks to illustrate their capabilities to augment underrepresented classes, and capture coarse anatomical structures (lung, heart, and clavicles); however, their methodologies of assessment are purely based on qualitative analysis of the generated images.

Around 2020, latent difussion models came to the picture [10] to generate synthetic chest X-rays images. These showcased an improvement of 3.5% when used as a data augmentation technique for classification tasks. This study does have quantitative classification metrics, but the fact they didn't compare the difference of performance between a GAN network and the latent diffusion model; and focused solely on image generation, makes it a bit difficult to assess if its the best method.

Its only until 2022, where Chambon and et. al. developed RoentGen [1] using a latent diffusion model; composed of three main components: a VAE that compresses images to a pixel space, a conditional U-net for random Gaussian noise generation, and a conditional text encoder. Using this multi modal model, which leverages the radiologists natural language and the VAE for CXR images, is able to create visually convincing CXR images; based on a specific text prompt. They measure a 5% improvement of a classifier trained jointly on real and synthetic image, and a 3% improvement when trained on a larger but purely synthetic training images. This is the most complete actual work when comparing it to previous attempts; since they compare three main tasks performances: classification with a pretrained classifier, radiology and report generation, and image-image text image retrieval.

Lastly, in 2023, chest X-ray classification using Dirichlet VAEs [4] have proven to be superior than using GMVAEs, since its able to disentangle latent factors into class-specific visual features for multi classification settings. Because of this, research using GANs, VAEs, and latent diffusion models have proven to be promising in the development of CXR datasets.

3. Methods

In this section, we describe the algorithms that we used with their mathematical formulations.

3.1. Variational Autoencoders

VAE is a type of autoencoder that uses principles of probabilistic inference and Bayesian statistics to learn a latent representation of data and then reconstruct the data. Intuition behind latent variables is to learn high level features in the lower dimensional space that can make it easy to determine $p(\text{data}|\text{latent variables})$ - Example: image of a human face can be approximated better, if the high level features are given like Gender, Ethnicity (Eye Color, Hair Color), Expression (Smiling/ Angry) etc. Unlike traditional autoencoders, which merely aim to compress data into a latent space and then reconstruct it, VAEs model the data distribution of the source data through a latent space and explicitly calculate the probability density of the data by integrating over this latent space.

$$VAE = P(x) = \sum_{z} P(x, z) = \sum_{z} P(z)P(x|z)$$

The major components of the VAE are

1. Encoder: It helps extract the features from higher dimension image to a lower dimension latent space.

2. Latent Space: The mean and variance parameters of the latent distribution of the data is learnt using a neural network. We then sample from this latent distribution and feed the sample into decoder to reconstruct the image. In case of GM-VAE, we have K mixture of means and variances while in case of VAE, we just have one.

3. Decoder: It takes those sampled point and reconstructs the input data from them.

An example is when $z \sim N(0, 1)$ and $P(x|z) = N(\mu_\theta(z), \Sigma_\theta(z))$. where the prior distribution is a Gaussian N and the class likelihood is defined by a Gaussian N or multivariate Gaussian N_k, defined by neural networks μ_θ and Σ_θ

The primary objective of the VAE is MLE (Maximum Likelihood Estimation) of the observed data. This is done via importance sampling; Monte Carlo, as well as application of a higher and lower bound using Jensen inequality:

$$log(\Pi_{x \in D} P_\theta(x)) = \sum_{x \in D} Log P_\theta(x) = \sum_{x \in D} log \sum_{z} P_\theta(x, z)$$

The reason comes from above, $\sum_z P_\theta(x, z)$ would become intractable; meaning we need to sample our z in an intelligent way using $q(z)$.

$$P_\theta(x) = \sum_{z} P_\theta(x, z) = \sum_{z \in Z} \frac{q(z)}{q(z)} P_\theta(x, z) = \mathbb{E}_{z \sim q(z)}[\frac{P_\theta(x, z)}{q(z)}]$$

With this expectation, we can use Monte Carlo sampling across a set (the batch size); and find the log likelihood. There is a little caveat; lets assume the case where k = 1.

$$\mathbb{E}_{z \sim q(z)}[\frac{P_\theta(x, z)}{q(z)}] \approx \frac{1}{K} \sum_{j=1}^{k} \frac{P_\theta(x, z^{(j)})}{q(z^{(j)})}$$

$$logP_\theta(x) \approx log(\frac{P_\theta(x, z^{(1)})}{q(z^{(1)})})$$

We know that by Jensen Inequality, for a concave function; $f\mathbb{E}[x] \geq \mathbb{E}[f(x)]$. Meaning that our upper bound is $logP_\theta(x)$ and lower bound is $\sum_z q(z)log\frac{P_\theta(x,z)}{q(z)}$

$$logP_\theta(x) \geq \sum_z q(z)logP_\theta(x, z) + \sum_z q(z)log(z)$$

$$logP_\theta(x) \geq \sum_z q(z)logP_\theta(x, z) + H(q)$$

This formulation for variation inference, the lower bound would be equal to the upper bound when $q(z) = P_\theta(z|x)$. Interestingly, this formulation does also come when we calculate $\mathrm{KL}(q(z)||p_\theta(z|x))$.

Then, let there be a $q_\phi(z) \approx P_\theta(z|x)$. The reason we do this is because $P_\theta(z|x)$ can also be intractable in practice. So we optimize a known distribution with parameters ϕ

$$logP_\theta(x) \geq \sum_z q_\phi(z)logP_\theta(x, z) + H(q_\phi(z)) \approx \mathcal{L}(\theta, \phi; x)$$

After applying reparametrization trick for continuous probability distributions, and amortization technique to obtain the sets of $\phi_1, \phi_2, ...$ coming from a neural network $f_\lambda(x)$. We get the following:

$$\mathcal{L}(\theta, \phi; x) = \sum_z q_\phi(z|x)logP_\theta(z, x) + H(q_\phi(z|x)) = \mathbb{E}_{q_\phi(z|x)}[\log P_\theta(z, x) - \log q_\phi(z|x)]$$

$$\mathbb{E}_{q_\phi(z|x)}[\log P_\theta(z, x) - \log(P(z)) + log(P(z)) - \log q_\phi(z|x)]$$

$$\mathbb{E}_{q_\phi(z|x)}[\log \frac{P_\theta(z, x)}{\log(P(z))} - \log \frac{P(z)}{q_\phi(z|x)}]$$

Giving us the final loss, which accounting for a β gradient; it is:

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta * \mathrm{KL}(q_\phi(z|x) \parallel p(z))$$

The loss function involves two key components.

1. Reconstruction Loss: This component encourages the decoder to reconstruct the data as accurately as possible from the sampled latent variables.

2. KL Divergence: This regularisation term ensures that latent space does not overfit to the training data and can generalise well. It does so by keeping the learned mean and standard deviations close to a prior distribution, typically a standard normal distribution.

If we consider the prior $p(z)$ to be the standard normal distribution from a multivariate gaussian , KL divergence has a closed form solution as given below. [1] [2]

$$\mathrm{KL}(q_\phi(z_i|x_i) \parallel p(z_i)) = -\frac{1}{2}\sum_{j=1}^{D}\left(1 + \log((\sigma_\phi^{(j)}(x_i))^2) - (\mu_\phi^{(j)}(x_i))^2 - (\sigma_\phi^{(j)}(x_i))^2\right)$$

Here $\mu_\phi^{(j)}(x_i)$ and $\sigma_\phi^{(j)}(x_i)$ are the mean and standard deviation respectively of the j_{th} dimension of i_{th} training sample. The reconstruction loss is taken to be MSE (Mean Squared Loss) between tensors of the original and generated image.

3.2. Generative Adversarial Network

GANs are generative model which learn to generate data by sampling from a learned distribution of the dataset without explicitly defining the distribution's density function. These models do not do an explicit maximum likelihood computation of the data distribution. GANs are composed of two main components, which can be considered as two agents trying to compete in a game.

1. Generator: The generator's role is to create data that is indistinguishable from real data. In a way, this agent tries to fool the discriminator into labeling the generated images as real images.

2. Discriminator: The discriminator's role is to be able to classify between real data and generated data. Its objective is to be able to classify real images from generated images.

We are using the Least Squares GAN [8] which adopt the least squares loss function for the model. This is different to the sigmoid cross entropy loss function in the original GAN paper. LSGans were empirically shown to be more stable during gradient backpropagation. The loss formulation for the generator and discriminator is given below.

$$\min_D V_{\text{LSGAN}}(D) = \frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}(x)}\left[(D(x)-1)^2\right] + \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}\left[(D(G(z)))^2\right]$$

$$\min_G V_{\text{LSGAN}}(G) = \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}\left[(D(G(z))-1)^2\right]$$

Intuitively, when $D(G(z))$, i.e the prediction from the discriminator of the generated image is 1, the generator's loss is 0 (the generator was successfully able to fool the discriminator into believing the generated image is real). Similariy, if $D(x)$ is 1, i.e discriminator labels real image as 1 and $D(G(z))$ is 0, i.e the discriminator labels generated images as 0, the Discrimator loss is 0.

4. Dataset

The sample images are given in Figure in the appendix. 10. The data comes from the CXR14 Dataset link. The following table illustrates the distribution of images found per chest x-ray condition.

[1] Taken from Variational Autoencoder post by Matthew N. Bernstein https://mbernste.github.io/posts/vae/

[2] Taken from CS 236, Homework 2 for VAEs, GMVAEs, and FSSVAEs Fall 2023; instructors: Stefano Ermon; ermon@cs.stanford.edu

Condition	Count
No Finding	59406
Infiltration	19894
Effusion	13317
Atelectasis	11559
Nodule	6331
Mass	5782
Pneumothorax	5302
Consolidation	4667
Pleural Thickening	3385
Cardiomegaly	2776
Emphysema	2516
Edema	2303
Subcutaneous Emphysema	1991
Fibrosis	1686
Pneumonia	1431
Tortuous Aorta	742
Calcification of the Aorta	455
Pneumoperitoneum	316
Pneumomediastinum	253
Hernia	227

Table 1. Frequency of Different Conditions in CXR-14 dataset

We used the publicly available CXR14 dataset released by Wang *et al*. [13] in 2017, from NIHCC (National Institutes of Health Clinical Center). The dataset includes chest X-rays in posterior-anterior (PA), anterior-posterior (AP), and lateral (LAT) projections. Due to time constraints, we did not perform augmentation for underrepresented classes or differentiate between CXR types. Future research can address these aspects to enhance model performance.

To use VGG16 and DenseNet121 as encoders, we transformed the images to the appropriate inputs. For VAEs, GANs, and GMVAEs V7 and V8, we resized images to 224, centered to crop, converted to grey channels (1 output for GAN and VAE, 3 channels for GMVAE), and applied histogram equalization. For GMVAEs V3 to V6, we also added a normalization block using mean values $[0.485, 0.456, 0.406]$ and standard deviations $[0.229, 0.224, 0.225]$.

Based on using generative models for classifiers on the CXR14 dataset [7], we created a transitional layer with a 1x1 convolution layer (feature size * 300 channels), BatchNormalization (300 channels), ReLU activation, Max Pool 2D (kernel size 7), and a linear layer (input 300, output desired z dimension). We developed various combinations of GMVAEs and FS-GMVAEs, with the latter concatenating the label $y = [1, 0, 1, ...]$ as a positional token in the Z space.

5. Experiments and Results

We used a batch size of 64 for all the algorithms. We found that batch size of 64 gave us optimum running time per epoch after experimenting with batch size of 32 and 128. We used Adam optimiser with default values of βs for VAE and with $\beta_1 = 0.5$ for GAN[3]. We used learning rate of 1e-4 for VAEs and 2e-4 for GAN (both discriminator and generator). To help the generator converge faster, we let the generator run 3 times for every run of discriminator. If we didn't do that, the discriminator loss came down rapidly and generator was not able to learn effectively. We provide further experimentations and results for our various algorithms below.

5.1. GM-VAE

For GM-VAE, we used VGG-16 with ImageNet weights to extract the features from which we learnt the latent space distribution. We experimented with various dimensions of the latent space (Z space) and various possibilities of Z as Categorical with parameter K. For more information about which decoders and types used, follow the appendinx link 7.

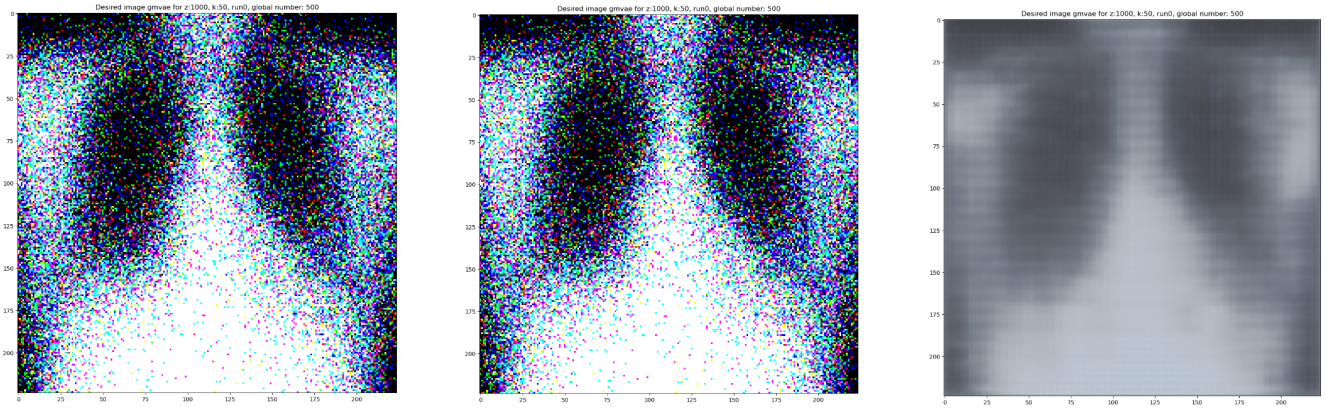

Figure 1. Samples from FS-GMVAE V1 to V3

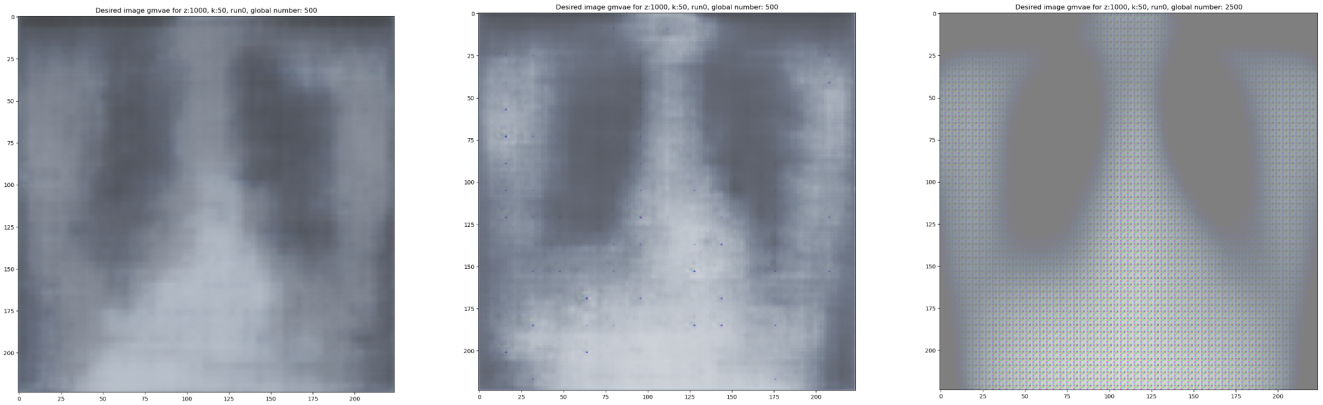

Figure 2. Samples from GMVAE V3 to V5

[3] $\beta_1 = 0.5$ was the value that was used in Assignment3 GAN of CS231. Also, further research revealed that we need more component of recent gradients during optimisation in case of GANs than historical momentum. Hence, the value of β_1 was chosen as 0.5

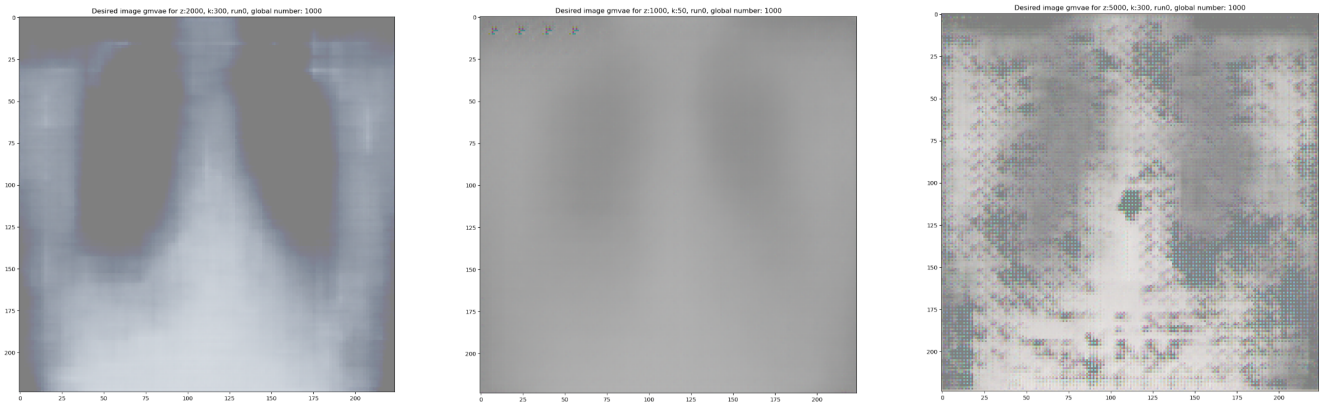

Figure 3. Samples from GMVAE V6 to V8

5.2. VAE

We then experimented with VAE. Since we did not have much success with a pre-trained encoder while training GM-VAE, we decided to create our own encoder and decoder architecture. We took inspiration from VGG-16 architecture in creating the encoder. The architecture of the encoder and decoders, as well as results are given in the following figures. As seen in the Figure 5, the training loss became flat after around 1 epoch and we stopped the training when we observed no further reduction in loss and no further improvement in quality of generated image.

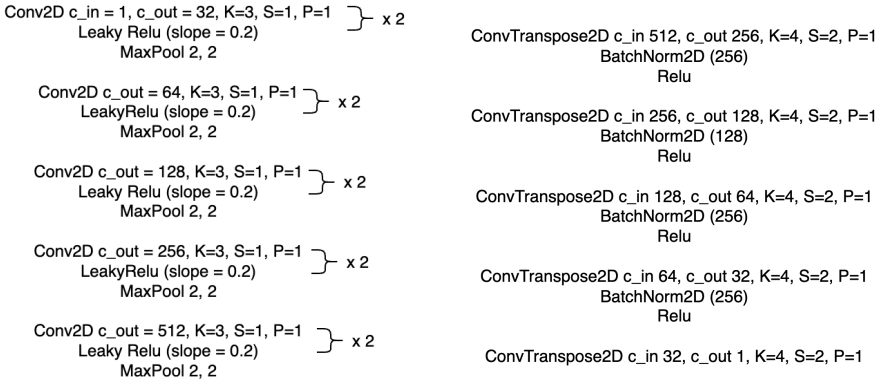

Figure 4. Architecture for VAE Encoder (left) and VAE Decoder (right)

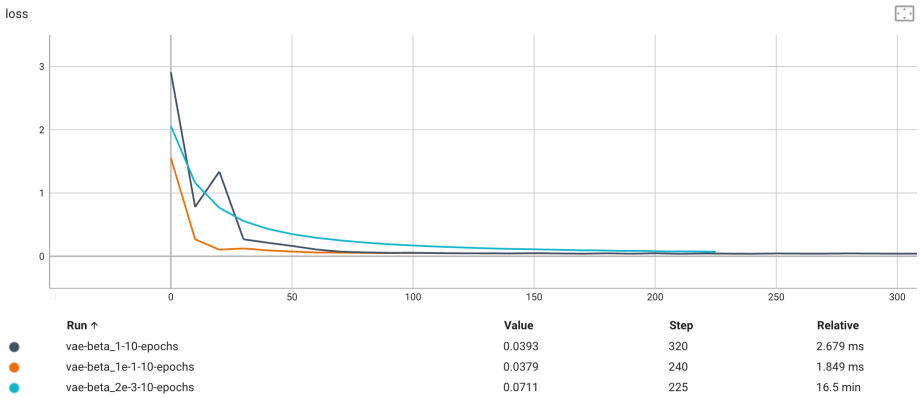

Figure 5. Training loss of VAE with $\beta = 1$, $\beta = 0.1$ and $\beta = 0.002$

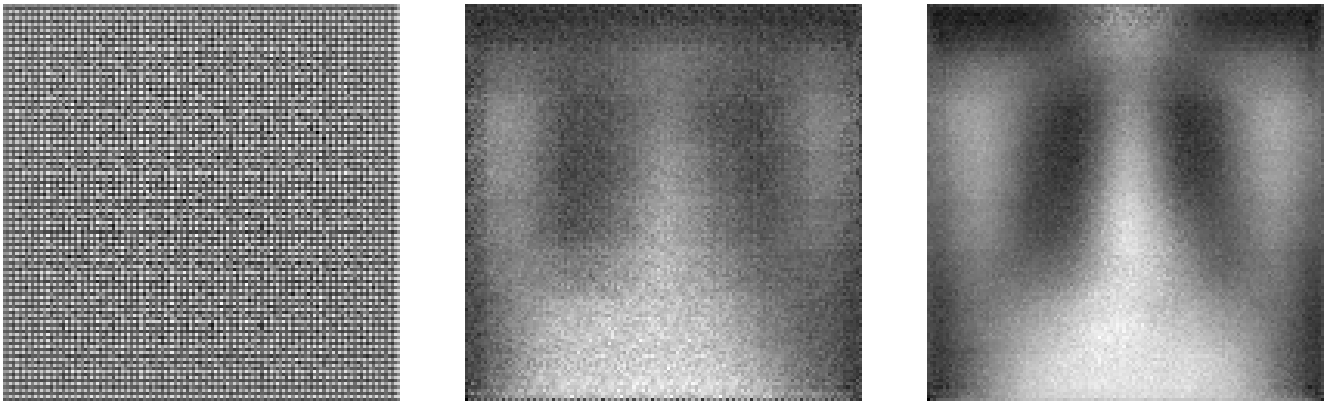

Figure 6. Samples from VAE with $\beta = 0.1$ at iteration 0, iteration 100 and iteration 250

5.3. GAN

We experimented with GAN after VAE. The architecture used is showcased in the next figure. We saw an equilibrium in the loss of generator and discriminator after around 3 epochs. We did not see any significant improvement in quality of image after 4 epochs and we also did not see the training loss coming down for generator and discriminator, so we stopped the training at around 5 epochs.

Conv2D c_in = 1, c_out = 32, K=4, S=2, P=1
LeakyRelu (slope = 0.2)

Conv2D c_in = 32, c_out = 64, K=4, S=2, P=1
BatchNorm2D (64)
LeakyRelu (slope = 0.2)

Conv2D c_in = 64, c_out = 128, K=4, S=2, P=1
BatchNorm2D (128)
LeakyRelu (slope = 0.2)

Conv2D c_in = 128, c_out = 256, K=4, S=2, P=1
BatchNorm2D (256)
LeakyRelu (slope = 0.2)

Conv2D c_in = 256, c_out = 512, K=4, S=2, P=1
BatchNorm2D (512)
LeakyRelu (slope = 0.2)

Conv2D c_in = 512, c_out = 1, K=4, S=2, P=1
(Classifier Layer)

ConvTranspose2D c_in = dim_z, c_out = 512, K=4, S=1, P=0
BatchNorm2D(512)
Relu

ConvTranspose2D c_in = 512, c_out = 256, K=4, S=2, P=1
BatchNorm2D(256)
Relu

ConvTranspose2D c_in = 256, c_out = 128, K=4, S=2, P=1
BatchNorm2D(128)
Relu

ConvTranspose2D c_in = 128, c_out = 64, K=4, S=2, P=1
BatchNorm2D(64)
Relu

ConvTranspose2D c_in = 64, c_out = 32, K=4, S=2, P=1
BatchNorm2D(32)
Relu

ConvTranspose2D c_in = 32, c_out = 1, K=4, S=2, P=1
Tanh

Figure 7. Architecture for GAN discriminator (left) and GAN generator (right)

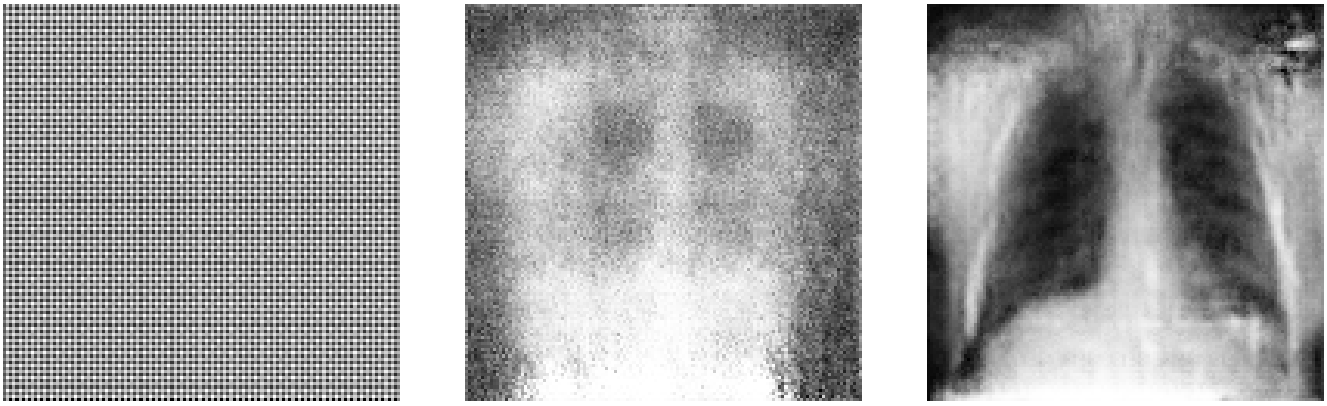

Figure 8. Samples from GAN at iteration 0, iteration 400 and final iteration after 5 epochs

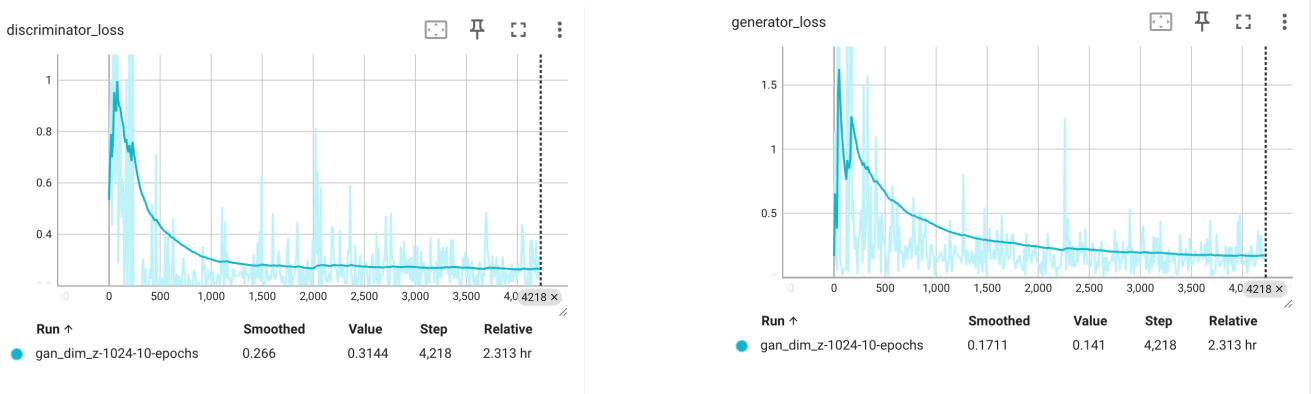

Figure 9. Discriminator and Generator loss for the GAN Network

We compare the parameter counts of our architectures and average training time per epoch on table 2.

6. Conclusions and Future Work

We found that GANs produced qualitatively better results than VAE and GMVAE. We think that our VAEs did not perform well because of the Gaussian Prior assumption on the distribution of latent space. We experimented with various encoder architectures, with and without pre-trained ImageNet weights, but our results did not improve significantly. GANs used fewer parameters, compute and were able to produce qualitatively better results. As future work, we can explore other architectures that try to do explicit modeling of data distribution, like Diffusion Models and try to dig deeper on why our VAEs were not able to model the distribution well. We can also try with other priors like Dirichlet prior or Cauchy prior and see how it impacts the modeling of the latent space. We can also use these generated images as augmentation in classification models and report the effect that they have on such models.

7. Appendices

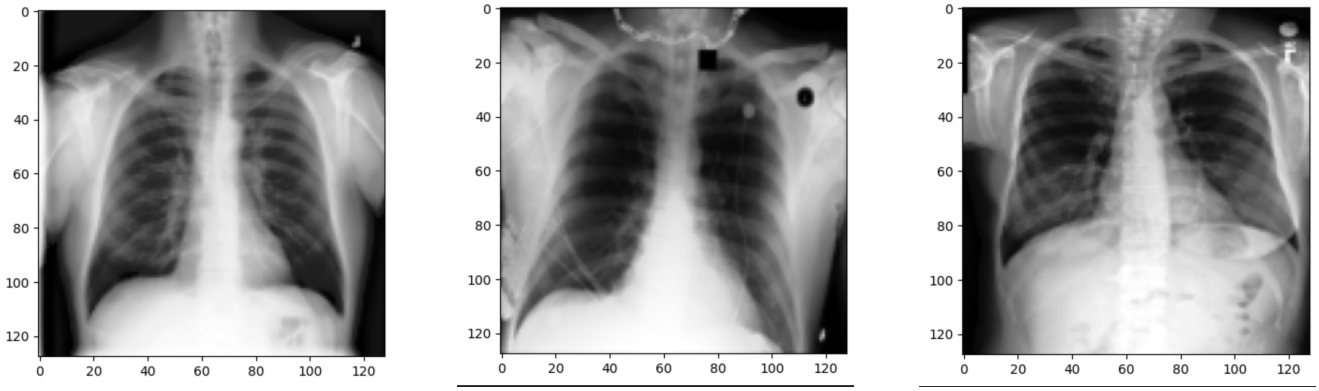

Figure 10. Sample images from the dataset after applying the transformations

Formula for calculating convolution transpose 2D: $O_h = s_h(I_h - 1) + k_h - 2p$ and $O_w = s_w(I_w - 1) + K_w - 2p$

$$N * C_{in} * H * W -> C_{out} * C_{in} * K_h * K_w -> N * C_{out} * O_w * O_h$$

For instance, assuming a padding = 1, and stride = 2.

$$1 * 14 * 14 * 14 -> 6 * 14 * 4 * 4 -> 1 * 6 * 28 * 28$$

Layer	Activation Volumes	Weights	Biases
Input	1*14*14*14	N/A	N/A
CONVT2D (14,6,4,2,1)	1*6*28*28	4*4*14*6	6
CONVT2D (6,3,4,2,1)	1*3*56*56	4*4*6*3	3
CONVT2D (3,3,4,2,1)	1*3*112*112	4*4*3*3	3
CONVT2D (3,3,4,2,1)	1*3*224*224	4*4*3*3	3

Table 2. Decoder 1 using convolutional transpose 2d layers

Layer	Activation Volumes	Weights	Biases
Input	1*128*14*14	N/A	N/A
CONVT2D (128,64,4,2,1)	1*64*28*28	4*4*128*64	64
CONVT2D (64,32,4,2,1)	1*32*56*56	4*4*64*32	32
CONVT2D (32,16,4,2,1)	1*16*112*112	4*4*32*16	16
CONVT2D (16,3,4,2,1)	1*3*224*224	4*4*16*3	3

Table 3. Decoder 2 using convolutional transpose 2d layers

Version	Z	K	Transition Net	Normalize Blue	Encoder	Decoder
FS-GMVAE V1	1000	50	True	True	VGG16	Linear Layers
FS-GMVAE V2	1000	50	True	False	VGG16	Linear Layers
FS-GMVAE V3	1000	50	True	True	VGG16	Decoder 1
GMVAE V3	1000	50	False	True	VGG16	Decoder 1
GMVAE V4	1000	50	True	True	VGG16	Decoder 1
GMVAE V5	1000	50	False	True	VGG16	Decoder 2
GMVAE V6	2000	300	True	True	DenseNet121	Decoder 2
GMVAE V7	2000	300	False	False	VGG16	Decoder 2
GMVAE V8	2000	300	True	True	VGG16	Decoder 2

Table 4. Experiments with GM-VAE Hyperparameters Z: Dimension of Latent Space, K: Number of Gaussian Mixture

Architecture	Parameter Count	Average Train Time per Epoch
GAN	Discriminator: 2,795,904, Generator: 3,606,976 (Total: 6,402,880)	24 minutes
VAE	Encoder: 4,711,648, Decoder: 11,184,033, Mean and Logvar: 8,389,632 (Total: 32,674,945)	48 minutes
GM-VAE (Encoder in Eval)	Decoder1: 1920; Decoder2: 172800	30 min (Image generation)
GM-VAE (Encoder in Train)	VGG16 Features Encoder: 14,714,688; Decoder1: 1920; Decoder2: 172,800 (Total: 14,716,608 - 14,887,488)	65 min

Table 5. Parameter counts and train time for reported architectures. The training time is average training time per epoch on 1 Nvidia T4 GPU on Google Cloud Compute

8. Contributions

Abhishek worked on experiments with VAE and GAN and helped write the report. Juan worked on experiments with GM-VAE and helped write the report. We wrote the training code ourselves taking inspirations from the CS231 and CS236 assignments.

References

[1] P. Chambon, C. Bluethgen, J.-B. Delbrouck, R. V. der Sluijs, M. Połacin, J. M. Z. Chaves, T. M. Abraham, S. Purohit, C. P. Langlotz, and A. Chaudhari. Roentgen: Vision-language foundation model for chest x-ray generation, 2022.

[2] N. Dilokthanakul, P. A. Mediano, M. Garnelo, M. C. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders. *arXiv preprint arXiv:1611.02648*, 2016.

[3] 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 (NeurIPS)*, volume 27, 2014.

[4] R. Harkness, A. F. Frangi, K. Zucker, and N. Ravikumar. Learning disentangled representations for explainable chest x-ray classification using dirichlet vaes, 2023.

[5] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014.

[6] A. Madani, M. Moradi, A. Karargyris, and T. Syeda-Mahmood. Chest x-ray generation and data augmentation for cardiovascular abnormality classification. In *Medical Imaging 2018: Image Processing*, volume 10574, pages 415–420. SPIE, 2018.

[7] C. Mao, Y. Pan, Z. Zeng, L. Yao, and Y. Luo. Deep generative classifiers for thoracic disease diagnosis with chest x-ray images, 2018.

[8] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, and Z. Wang. Multi-class generative adversarial networks with the L2 loss function. *CoRR*, abs/1611.04076, 2016.

[9] S. Menon, J. Galita, D. Chapman, A. Gangopadhyay, J. Mangalagiri, P. Nguyen, Y. Yesha, Y. Yesha, B. Saboury, and M. Morris. Generating realistic covid19 x-rays with a mean teacher + transfer learning gan, 2020.

[10] K. Packhäuser, L. Folle, F. Thamm, and A. Maier. Generation of anonymous chest radiographs using latent diffusion models for training thoracic abnormality classification systems, 2022.

[11] M. Y. Shams, O. M. Elzeki, M. A. Elfattah, T. Medhat, and A. E. Hassanien. Why are generative adversarial networks vital for deep neural networks? a case study on covid-19 chest x-ray images. *Studies in Big Data*, pages 147–162, 2020.

[12] D. Srivastav, A. Bajpai, and P. Srivastava. Improved classification for pneumonia detection using transfer learning with gan based synthetic image augmentation. In *2021 11th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, pages 433–437. IEEE, 2021.

[13] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. National Institutes of Health Clinical Center, 2017.