ECE 175B - Variational Auto-Encoder

Anonymous Author(s)
Affiliation
Address
email

Abstract

This abstract summarizes a 4-layer Convolutional Variational Autoencoder (VAE) experiment trained on a dataset of chest X-rays. The VAE architecture consisted of four convolutional layers for both the encoder and decoder, enabling the model to learn representations of the chest X-ray images in a latent space. The experiment aimed to assess the VAE's ability to generate realistic and coherent chest X-ray images while preserving diagnostic information.

1 Introduction

In this report, a 4 layer convolutional variational autoencoder it built for the chest-xray dataaset with pneumonia. The architecture of the VAE consists of 4 convolutional layers accompanied with ReLU in the encoding and decoding end, and a linear layer with latent dimension of 256.

2 Training Configurations

Code for the neutral network is at the end of the document.
The following configurations were used for training.

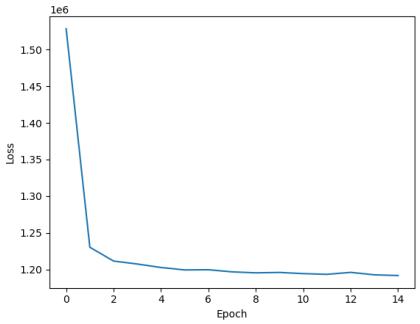
- Optimizer: Adam
- Input Size: 256 x 256
- Learning Rate: 0.001
- Batch Size: 32
- Epochs: 15
- KL-Divergence weight: 0.0001
- Loss Function: Binary Cross Entropy + KL-Divergence

For the second iteration, the linear layer is removed and the encoded layer output is directly fed into the decoding layer. This is to observe the effects of the probabilistic approximation via the linear layer.

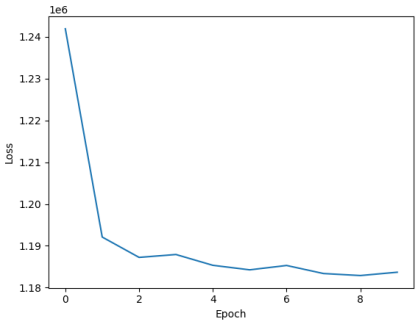
3 Training Loss Curves

For two different iterations, training loss tended to be large, probably due to the selection of the loss function.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.



(a) Loss Curve for First Iteration



(b) Loss Curve for Second Iteration

4 FID and IS Metrics

The FID and IS metrics are commonly used to evaluate the performance of Generative Adversarial Networks (GANs). The FID measures the similarity between real and generated data distributions, while the IS assesses the quality and diversity of generated samples. Both metrics provide valuable insights into GAN performance, but they have limitations and should be used alongside visual inspection and subjective evaluation for a comprehensive understanding of the model's capabilities. Scoring code was directly pulled from `https://pytorch-ignite.ai/blog/gan-evaluation-with-fid-and-is/`

The first iteration had a FID score of 0.447 and IS score of 2.479. This is significantly higher than the second iteration, where the FID score was 0.133 and IS score was 3.014. These metrics can soon be understood when the sample images are viewed. The second iteration, with a lower FID and higher IS metric, displayed significantly higher resolution images.

5 Reconstructed Images

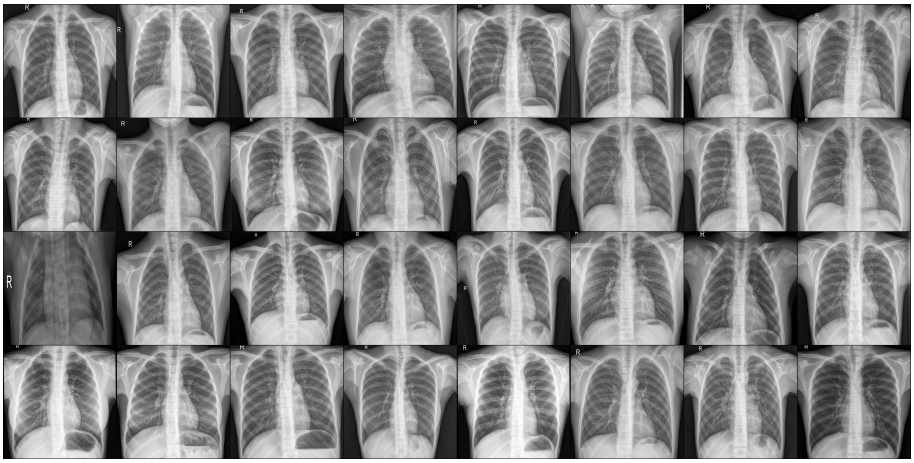


Figure 2: Original Image

The above batch of 32 images is the original batch fed into the model from the testing dataset.

The following images are the first batch for the first iteration. The larger inaccuracy may be due to a convergence to a local minima, but also small batch size and lack of depth in the network. More parameters with larger batches would improve generalization. Additionally, the model tries to fit 256 x 256 images into a 256-dimensional distribution, which already loses so much information.

The next batch of images are significantly better reconstructions of the original images. However, with closer observation of the images, the fine lines and outlines of the bones are slightly more blurrier than the original. Probabilistic reconstruction and decoding of images can not be perfect, so this must be pretty optimal. Not much information is lost since there is not latent layer between the encoding and decoding layers.

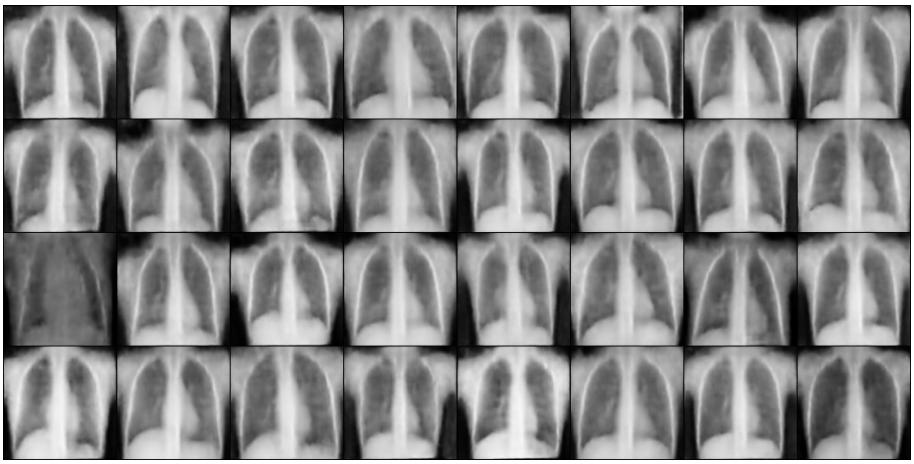


Figure 3: First Iteration

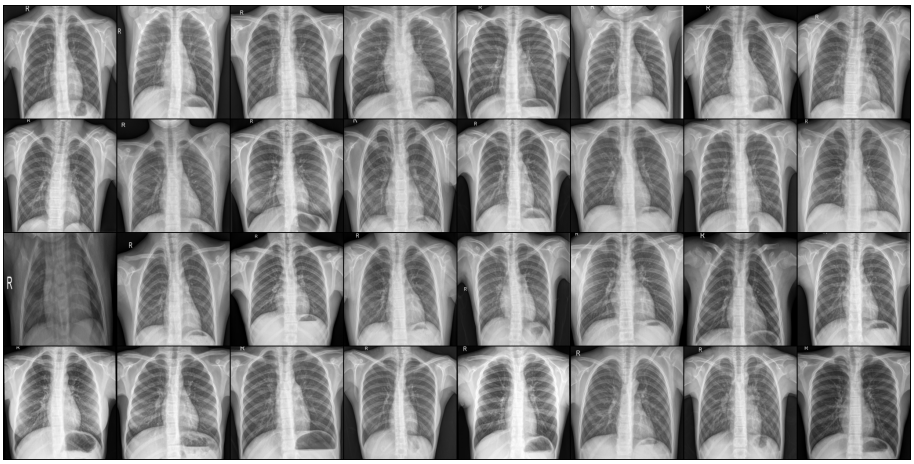


Figure 4: Second Iteration

6 Conclusion

In conclusion, the project on Variational Autoencoders (VAEs) and their application in encoding and decoding images has provided valuable insights into the power of latent space representation for image generation. By training VAE models, we were able to effectively capture the underlying structure and variations in the image data. The encoding process allowed us to compress images into a lower-dimensional latent space, while the decoding process successfully reconstructed the original images with minimal loss of information. This project highlights the potential of VAEs in various applications, such as image compression, generation, and data representation, paving the way for further advancements in the field of deep learning and computer vision.

7 Code

```
# Define the VAE model
class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()

        # Define the encoder
        self.encoder = nn.Sequential(
            nn.Conv2d(input_shape[0], 32, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
            nn.ReLU()
        )

        # Define the fully connected layers
        self.fc_mu = nn.Linear(64 * 32 * 32, latent_dim)
        self.fc_logvar = nn.Linear(64 * 32 * 32, latent_dim)

        # Define the decoder
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 64 * 32 * 32),
            nn.ReLU(),
            nn.Unflatten(1, (256, 16, 16)),
            nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 32, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(32, input_shape[0], kernel_size=4, stride=2, padding=1),
            nn.Sigmoid()
        )

    def encode(self, x):
        x = self.encoder(x)
        x = x.view(x.size(0), -1)
        mu = self.fc_mu(x)
        logvar = self.fc_logvar(x)
        return mu, logvar

    def decode(self, z):
        z = self.decoder(z)
        return z

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        z = mu + eps * std
        return z
```

```
    def forward(self, x):
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
        x_hat = self.decode(z)
        return x_hat, mu, logvar

# Training loop
num_epochs = 15
kld_rate = 0.0001
for epoch in range(num_epochs):
    running_loss = 0.0
    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.to(device)

        # Forward pass
        out, mu, logvar = model(data)

        kl_divergence = kld_rate * 0.5 * torch.sum(-1 - logvar + mu.pow(2) + logvar.exp())
        loss = F.binary_cross_entropy(out, data, size_average=False) + kl_divergence

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
```