```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import os
import time
import torch
import torch.nn as nn
import torchvision
from torchvision import datasets
from PIL import Image
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader

import numpy as np
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split


def aspect_ratio_preserving_resize(image, target_size):
    width, height = image.size
    target_width, target_height = target_size

    # Calculate the aspect ratio
    aspect_ratio = width / height

    if width > height:
        new_width = target_width
        new_height = int(new_width / aspect_ratio)
    else:
        new_height = target_height
        new_width = int(new_height * aspect_ratio)

    # Perform the resize
    image = transforms.functional.resize(image, (new_height, new_width))

    # Create a new image with the target size and paste the resized image in the center
    new_image = Image.new("L", target_size)
    new_image.paste(image, ((target_width - new_width) // 2, (target_height - new_height) // 2))

    return new_image


class MyDataset(Dataset):
    def __init__(self, data_dir, transform=None):
        self.data_dir = data_dir
        self.transform = transform
        self.image_paths = [os.path.join(data_dir, file) for file in os.listdir(data_dir)]

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        image_path = self.image_paths[idx]
```

```
        image = Image.open(image_path)

        # Apply aspect ratio-preserving resize
        resized_image = aspect_ratio_preserving_resize(image, (100, 100))

        if self.transform:
            transformed_image = self.transform(resized_image)
        else:
            transformed_image = resized_image

        return transformed_image


# Define your data transformation
train_transform = transforms.Compose([
    transforms.Resize((100, 100)),
    transforms.RandomHorizontalFlip(p=0.2),
    transforms.RandomVerticalFlip(p=0.2),
    transforms.RandomRotation(degrees=(5, 15)),
    transforms.ToTensor(),
])

test_transform = transforms.Compose([
    transforms.Resize((100, 100)),
    transforms.ToTensor(),
])



batch_size = 16

# Load data
train_dataset = MyDataset(data_dir='/content/drive/MyDrive/AE_xray/train', transform=test_transform)
test_dataset = MyDataset(data_dir='/content/drive/MyDrive/AE_xray/test', transform=test_transform)
train_data, valid_data = train_test_split(train_dataset, test_size=0.15, random_state=42)

# Dataloader
train_dl = DataLoader(train_data, batch_size=batch_size, shuffle=True)
valid_dl = DataLoader(valid_data, batch_size=batch_size, shuffle=True)
test_dl = DataLoader(test_dataset, batch_size=batch_size, shuffle=True)


class VEncoder(nn.Module):
    def __init__(self, input_size=10000, hidden_size1=5000, hidden_size2=2000, hidden_size3=1000, hidden_size4=
        super().__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1)
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.fc3 = nn.Linear(hidden_size2, hidden_size3)
        self.fc4 = nn.Linear(hidden_size3, hidden_size4)
        self.fc_mean = nn.Linear(hidden_size4, z_dim)
        self.fc_logvar = nn.Linear(hidden_size4, z_dim)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.relu(self.fc3(x))
        x = self.relu(self.fc4(x))
        mean = self.fc_mean(x)
```

```
        logvar = self.fc_logvar(x)
        return mean, logvar

class VDecoder(nn.Module):
    def __init__(self, output_size=10000, hidden_size1=5000, hidden_size2=2000, hidden_size3=1000, hidden_size4
        super().__init__()
        self.fc1 = nn.Linear(z_dim, hidden_size4)
        self.fc2 = nn.Linear(hidden_size4, hidden_size3)
        self.fc3 = nn.Linear(hidden_size3, hidden_size2)
        self.fc4 = nn.Linear(hidden_size2, hidden_size1)
        self.fc5 = nn.Linear(hidden_size1, output_size)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.relu(self.fc3(x))
        x = self.relu(self.fc4(x))
        x = torch.sigmoid(self.fc5(x))
        return x

class VAE(nn.Module):
    def __init__(self, z_dim=100):
        super().__init__()
        self.encoder = VEncoder(z_dim=z_dim)
        self.decoder = VDecoder(z_dim=z_dim)

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

    def forward(self, x):
        mean, logvar = self.encoder(x)
        z = self.reparameterize(mean, logvar)
        x_recon = self.decoder(z)
        return x_recon, mean, logvar
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

```
device(type='cuda')
```

```
vae = VAE(z_dim=100).to(device)
optimizer = torch.optim.Adam(vae.parameters(), lr=0.0000001,weight_decay=1e-5)
loss_function = nn.MSELoss()
```

```
num_epochs = 300
train_losses = []
valid_losses = []
path = "/content/drive/MyDrive/model/Autoencoder/another_xray_checkpoint_30z_5h_200e.pth"

# Check if a checkpoint exists to resume training
if os.path.exists(path):
  checkpoint = torch.load(path)
  vae.load_state_dict(checkpoint["model_state_dict"])
```

```
      optimizer.load_state_dict(checkpoint["optimizer_state_dict"])
      train_losses = checkpoint["train_loss"]
      valid_losses = checkpoint["valid_loss"]
      start_epoch = checkpoint["epoch"] + 1  # Start from the next epoch after the loaded checkpoint
      print("Resume training from epoch", start_epoch)
    else:
      start_epoch = 1


    for epoch in range(start_epoch,num_epochs+1):
        vae.train()
        train_loss = 0.0
        reconstruction_loss = 0.0
        kl_loss = 0.0

        for batch in train_dl:
            batch = batch.to(device)
            optimizer.zero_grad()
            batch_size, num_channels, height, width = batch.shape
            batch = batch.view(batch_size, -1)

            # Forward pass through the model
            recon_batch, mean, logvar = vae(batch)

            # Compute the loss (including reconstruction loss and KL divergence)
            recon_loss = loss_function(recon_batch, batch)
            kl_divergence = -0.5 * torch.sum(1 + logvar - mean.pow(2) - logvar.exp())
            reconstruction_loss += recon_loss.item()
            kl_loss += kl_divergence.item()
            loss = recon_loss + kl_divergence

            loss.backward()
            train_loss += loss.item()
            optimizer.step()

        # Calculate average training loss for the epoch
        train_loss /= len(train_dl.dataset)
        train_losses.append(train_loss)

        # Validation loop
        vae.eval()
        valid_loss = 0.0

        with torch.no_grad():
            for batch in valid_dl:
                batch = batch.to(device)
                batch_size, num_channels, height, width = batch.shape
                batch = batch.view(batch_size, -1)
                recon_batch, mean, logvar = vae(batch)
                recon_loss = loss_function(recon_batch, batch)
                kl_divergence = -0.5 * torch.sum(1 + logvar - mean.pow(2) - logvar.exp())
                loss = recon_loss + kl_divergence
                valid_loss += loss.item()

        # Calculate average validation loss for the epoch
        valid_loss /= len(valid_dl.dataset)
        valid_losses.append(valid_loss)
```

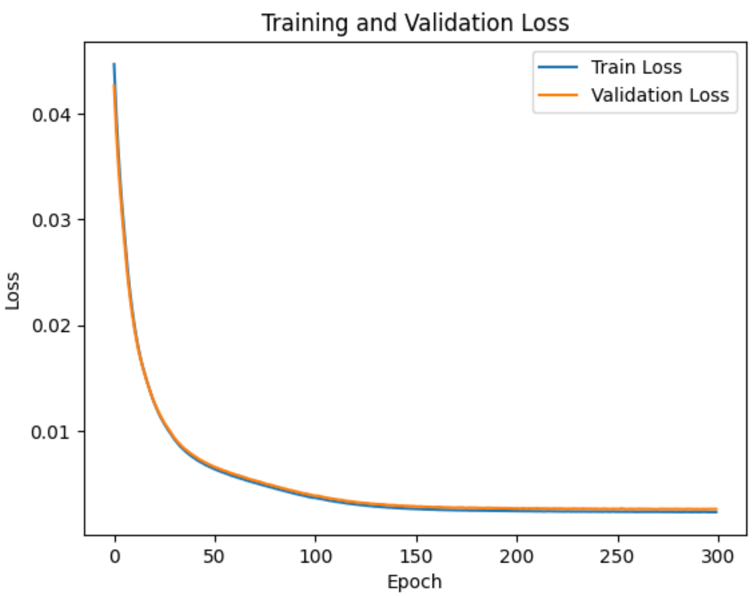
```
    # Print progress for each epoch
    print(f"Epoch [{epoch}/{num_epochs}] Train Loss: {train_loss:.4f}, Reconstruction Loss: {reconstruction_loss: .4f}, Kl Loss: {kl_loss: .4f} Valid Loss: {valid_loss:.4f

    # Save model checkpoint
    checkpoint = {
        'epoch': epoch,
        'model_state_dict': vae.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'train_loss': train_losses,
        'valid_loss': valid_losses
    }
    torch.save(checkpoint, path)
```

```
Resume training from epoch 201
Epoch [201/300] Train Loss: 0.0024, Reconstruction Loss:  2.4225, Kl Loss:  0.2911 Valid Loss: 0.0027
Epoch [202/300] Train Loss: 0.0024, Reconstruction Loss:  2.4244, Kl Loss:  0.2889 Valid Loss: 0.0026
Epoch [203/300] Train Loss: 0.0024, Reconstruction Loss:  2.4252, Kl Loss:  0.2877 Valid Loss: 0.0026
Epoch [204/300] Train Loss: 0.0024, Reconstruction Loss:  2.4172, Kl Loss:  0.2876 Valid Loss: 0.0027
Epoch [205/300] Train Loss: 0.0024, Reconstruction Loss:  2.4276, Kl Loss:  0.2864 Valid Loss: 0.0027
Epoch [206/300] Train Loss: 0.0024, Reconstruction Loss:  2.4252, Kl Loss:  0.2855 Valid Loss: 0.0026
Epoch [207/300] Train Loss: 0.0024, Reconstruction Loss:  2.4379, Kl Loss:  0.2847 Valid Loss: 0.0026
Epoch [208/300] Train Loss: 0.0024, Reconstruction Loss:  2.4186, Kl Loss:  0.2835 Valid Loss: 0.0026
Epoch [209/300] Train Loss: 0.0024, Reconstruction Loss:  2.4245, Kl Loss:  0.2824 Valid Loss: 0.0026
Epoch [210/300] Train Loss: 0.0024, Reconstruction Loss:  2.4138, Kl Loss:  0.2815 Valid Loss: 0.0026
Epoch [211/300] Train Loss: 0.0024, Reconstruction Loss:  2.4151, Kl Loss:  0.2808 Valid Loss: 0.0027
Epoch [212/300] Train Loss: 0.0024, Reconstruction Loss:  2.4284, Kl Loss:  0.2807 Valid Loss: 0.0026
Epoch [213/300] Train Loss: 0.0024, Reconstruction Loss:  2.4146, Kl Loss:  0.2792 Valid Loss: 0.0026
Epoch [214/300] Train Loss: 0.0024, Reconstruction Loss:  2.4096, Kl Loss:  0.2783 Valid Loss: 0.0026
Epoch [215/300] Train Loss: 0.0024, Reconstruction Loss:  2.4084, Kl Loss:  0.2770 Valid Loss: 0.0026
Epoch [216/300] Train Loss: 0.0024, Reconstruction Loss:  2.4283, Kl Loss:  0.2766 Valid Loss: 0.0026
Epoch [217/300] Train Loss: 0.0024, Reconstruction Loss:  2.4224, Kl Loss:  0.2756 Valid Loss: 0.0026
Epoch [218/300] Train Loss: 0.0024, Reconstruction Loss:  2.4203, Kl Loss:  0.2749 Valid Loss: 0.0026
Epoch [219/300] Train Loss: 0.0024, Reconstruction Loss:  2.4069, Kl Loss:  0.2745 Valid Loss: 0.0026
Epoch [220/300] Train Loss: 0.0024, Reconstruction Loss:  2.4084, Kl Loss:  0.2735 Valid Loss: 0.0026
Epoch [221/300] Train Loss: 0.0024, Reconstruction Loss:  2.4139, Kl Loss:  0.2725 Valid Loss: 0.0026
Epoch [222/300] Train Loss: 0.0024, Reconstruction Loss:  2.4087, Kl Loss:  0.2721 Valid Loss: 0.0026
Epoch [223/300] Train Loss: 0.0023, Reconstruction Loss:  2.4033, Kl Loss:  0.2704 Valid Loss: 0.0026
Epoch [224/300] Train Loss: 0.0024, Reconstruction Loss:  2.4111, Kl Loss:  0.2696 Valid Loss: 0.0026
Epoch [225/300] Train Loss: 0.0023, Reconstruction Loss:  2.4060, Kl Loss:  0.2691 Valid Loss: 0.0026
Epoch [226/300] Train Loss: 0.0024, Reconstruction Loss:  2.4137, Kl Loss:  0.2679 Valid Loss: 0.0026
Epoch [227/300] Train Loss: 0.0024, Reconstruction Loss:  2.4218, Kl Loss:  0.2670 Valid Loss: 0.0026
Epoch [228/300] Train Loss: 0.0023, Reconstruction Loss:  2.3951, Kl Loss:  0.2667 Valid Loss: 0.0026
Epoch [229/300] Train Loss: 0.0024, Reconstruction Loss:  2.4134, Kl Loss:  0.2667 Valid Loss: 0.0026
Epoch [230/300] Train Loss: 0.0023, Reconstruction Loss:  2.4017, Kl Loss:  0.2652 Valid Loss: 0.0026
Epoch [231/300] Train Loss: 0.0024, Reconstruction Loss:  2.4122, Kl Loss:  0.2646 Valid Loss: 0.0026
Epoch [232/300] Train Loss: 0.0023, Reconstruction Loss:  2.4110, Kl Loss:  0.2649 Valid Loss: 0.0026
Epoch [233/300] Train Loss: 0.0023, Reconstruction Loss:  2.3985, Kl Loss:  0.2631 Valid Loss: 0.0026
Epoch [234/300] Train Loss: 0.0023, Reconstruction Loss:  2.4128, Kl Loss:  0.2618 Valid Loss: 0.0026
Epoch [235/300] Train Loss: 0.0023, Reconstruction Loss:  2.4089, Kl Loss:  0.2611 Valid Loss: 0.0026
Epoch [236/300] Train Loss: 0.0023, Reconstruction Loss:  2.4000, Kl Loss:  0.2607 Valid Loss: 0.0026
Epoch [237/300] Train Loss: 0.0023, Reconstruction Loss:  2.4039, Kl Loss:  0.2599 Valid Loss: 0.0026
Epoch [238/300] Train Loss: 0.0023, Reconstruction Loss:  2.3975, Kl Loss:  0.2591 Valid Loss: 0.0026
Epoch [239/300] Train Loss: 0.0023, Reconstruction Loss:  2.4159, Kl Loss:  0.2585 Valid Loss: 0.0026
Epoch [240/300] Train Loss: 0.0023, Reconstruction Loss:  2.4048, Kl Loss:  0.2572 Valid Loss: 0.0026
Epoch [241/300] Train Loss: 0.0023, Reconstruction Loss:  2.4050, Kl Loss:  0.2570 Valid Loss: 0.0026
Epoch [242/300] Train Loss: 0.0023, Reconstruction Loss:  2.4093, Kl Loss:  0.2565 Valid Loss: 0.0026
Epoch [243/300] Train Loss: 0.0023, Reconstruction Loss:  2.4001, Kl Loss:  0.2557 Valid Loss: 0.0026
Epoch [244/300] Train Loss: 0.0023, Reconstruction Loss:  2.3993, Kl Loss:  0.2543 Valid Loss: 0.0026
Epoch [245/300] Train Loss: 0.0024, Reconstruction Loss:  2.4232, Kl Loss:  0.2536 Valid Loss: 0.0026
Epoch [246/300] Train Loss: 0.0023, Reconstruction Loss:  2.4109, Kl Loss:  0.2530 Valid Loss: 0.0026
Epoch [247/300] Train Loss: 0.0023, Reconstruction Loss:  2.3975, Kl Loss:  0.2523 Valid Loss: 0.0026
```

```
Epoch [248/300] Train Loss: 0.0023, Reconstruction Loss:  2.4072, Kl Loss:  0.2515 Valid Loss: 0.0026
Epoch [249/300] Train Loss: 0.0023, Reconstruction Loss:  2.4219, Kl Loss:  0.2508 Valid Loss: 0.0026
Epoch [250/300] Train Loss: 0.0023, Reconstruction Loss:  2.4006, Kl Loss:  0.2504 Valid Loss: 0.0026
Epoch [251/300] Train Loss: 0.0023, Reconstruction Loss:  2.3985, Kl Loss:  0.2499 Valid Loss: 0.0026
Epoch [252/300] Train Loss: 0.0023, Reconstruction Loss:  2.4034, Kl Loss:  0.2490 Valid Loss: 0.0026
Epoch [253/300] Train Loss: 0.0023, Reconstruction Loss:  2.4022, Kl Loss:  0.2481 Valid Loss: 0.0026
Epoch [254/300] Train Loss: 0.0023, Reconstruction Loss:  2.4063, Kl Loss:  0.2473 Valid Loss: 0.0026
Epoch [255/300] Train Loss: 0.0023, Reconstruction Loss:  2.4046, Kl Loss:  0.2463 Valid Loss: 0.0026
Epoch [256/300] Train Loss: 0.0023, Reconstruction Loss:  2.4083, Kl Loss:  0.2467 Valid Loss: 0.0026
Epoch [257/300] Train Loss: 0.0023, Reconstruction Loss:  2.4082, Kl Loss:  0.2452 Valid Loss: 0.0026
```

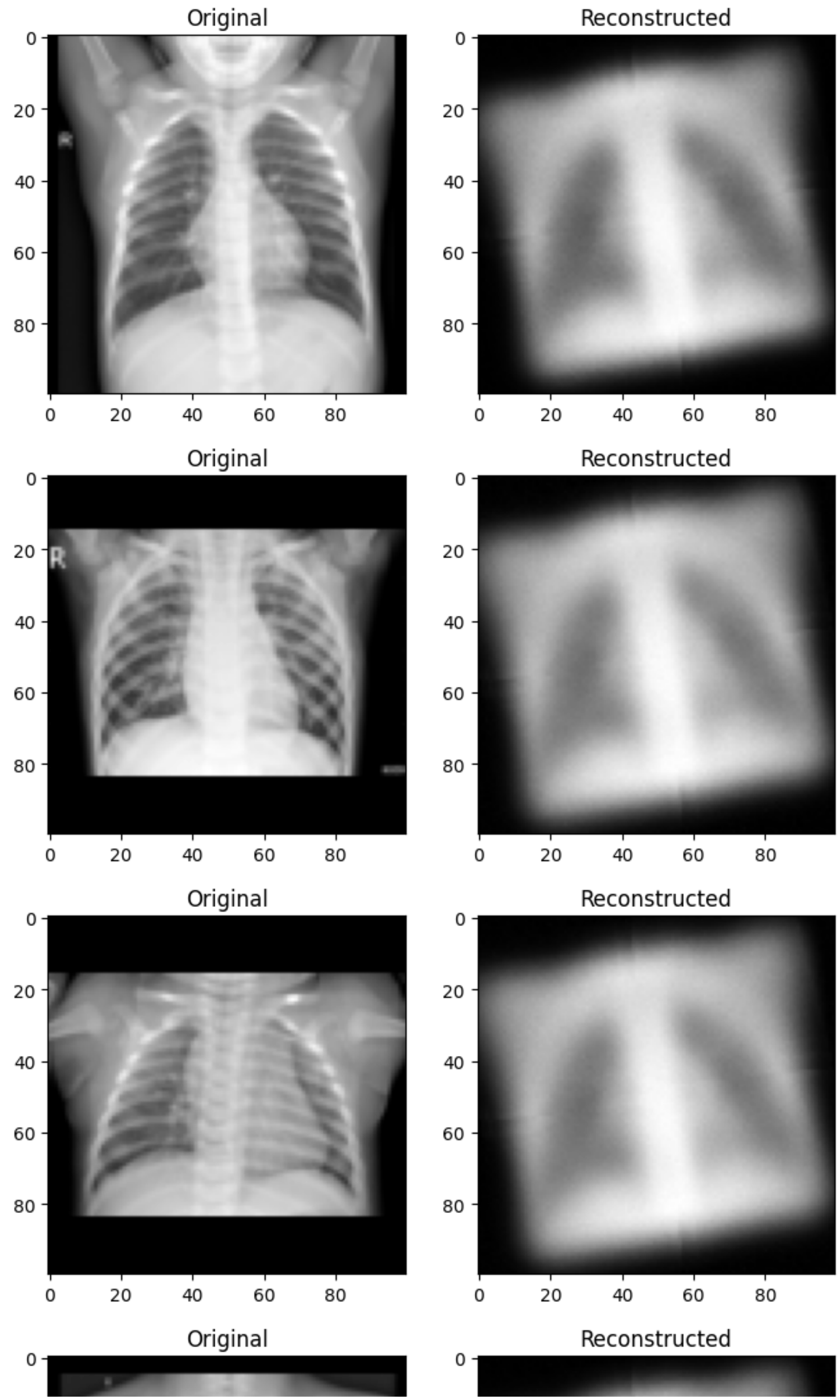
```
# Plot the loss graph
plt.figure()
plt.plot(train_losses, label='Train Loss')
plt.plot(valid_losses, label='Validation Loss')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.show()
```




```
# Plot some original and reconstructed images
n_samples = 5  # Number of samples to visualize
with torch.no_grad():
    for i, batch in enumerate(test_dl):
        if i >= n_samples:
            break
        batch = batch.to(device)
        batch = batch.flatten(1)
        reconstructed,_,_ = vae(batch)  # Pass the batch through your VAE
        plt.figure(figsize=(8, 4))
        plt.subplot(1, 2, 1)
        plt.title('Original')
        plt.imshow(batch[0].view(100, -1).cpu().numpy(), cmap='gray')  # Reshape to original size
```

```
    plt.subplot(1, 2, 2)
    plt.title('Reconstructed')
    plt.imshow(reconstructed[0].view(100, -1).cpu().numpy(), cmap='gray')  # Reshape to original size

    plt.show()
```


Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed


20


20