Pratham Nagar - 64 Experiment 05 ~ Design the architecture and implement the autoencoder model for Image denoising.

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

# Transform: Convert to tensor and normalize
transform = transforms.Compose([
    transforms.ToTensor(),
])

# Load Fashion-MNIST dataset
train_dataset = torchvision.datasets.FashionMNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = torchvision.datasets.FashionMNIST(root='./data', train=False, download=True, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)

# Function to add Gaussian noise
def add_noise(images, noise_factor=0.3):
    noisy = images + noise_factor * torch.randn_like(images)
    noisy = torch.clip(noisy, 0., 1.)
    return noisy

class DenoisingAutoencoder(nn.Module):
    def __init__(self):
        super(DenoisingAutoencoder, self).__init__()

        # Encoder
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, stride=2, padding=1),  # [32, 14, 14]
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, stride=2, padding=1), # [64, 7, 7]
            nn.ReLU(),
        )

        # Decoder
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(64, 32, kernel_size=3, stride=2, output_padding=1, padding=1), # [32, 14, 14]
            nn.ReLU(),
            nn.ConvTranspose2d(32, 1, kernel_size=3, stride=2, output_padding=1, padding=1),  # [1, 28, 28]
            nn.Sigmoid()  # to keep outputs in [0,1]
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = DenoisingAutoencoder().to(device)

criterion = nn.MSELoss()  # reconstruction loss
optimizer = optim.Adam(model.parameters(), lr=0.001)

epochs = 5
for epoch in range(epochs):
    model.train()
    running_loss = 0.0

    for images, _ in train_loader:
        images = images.to(device)
        noisy_images = add_noise(images).to(device)

        # Forward pass
        outputs = model(noisy_images)
        loss = criterion(outputs, images)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
```

```
            optimizer.step()

            running_loss += loss.item()

        print(f"Epoch [{epoch+1}/{epochs}], Loss: {running_loss/len(train_loader):.4f}")

    model.eval()
    with torch.no_grad():
        for images, _ in test_loader:
            images = images.to(device)
            noisy_images = add_noise(images).to(device)
            outputs = model(noisy_images)
            break  # take one batch for visualization

    # Plot original, noisy, and denoised images
    def show_images(orig, noisy, denoised, n=10):
        plt.figure(figsize=(15, 5))
        for i in range(n):
            # Original
            plt.subplot(3, n, i+1)
            plt.imshow(orig[i].cpu().squeeze(), cmap="gray")
            plt.axis("off")
            if i == 0: plt.ylabel("Original")

            # Noisy
            plt.subplot(3, n, i+1+n)
            plt.imshow(noisy[i].cpu().squeeze(), cmap="gray")
            plt.axis("off")
            if i == 0: plt.ylabel("Noisy")

            # Denoised
            plt.subplot(3, n, i+1+2*n)
            plt.imshow(denoised[i].cpu().squeeze(), cmap="gray")
            plt.axis("off")
            if i == 0: plt.ylabel("Denoised")
        plt.show()

    show_images(images, noisy_images, outputs)
```

```
100%|██████████| 26.4M/26.4M [00:02<00:00, 11.6MB/s]
100%|██████████| 29.5k/29.5k [00:00<00:00, 211kB/s]
100%|██████████| 4.42M/4.42M [00:01<00:00, 3.89MB/s]
100%|██████████| 5.15k/5.15k [00:00<00:00, 18.5MB/s]
Epoch [1/5], Loss: 0.0200
Epoch [2/5], Loss: 0.0113
Epoch [3/5], Loss: 0.0106
Epoch [4/5], Loss: 0.0102
Epoch [5/5], Loss: 0.0100
```