Experiment no 7

Aim

To design, implement, and train a convolutional autoencoder neural network for compressing and reconstructing images, demonstrating how learned latent representations can be used for image compression.

Apparatus

- **Hardware:**
    - Computer with GPU support (optional, but recommended for faster training)
- **Software:**
    - Python 3.x
    - PyTorch (deep learning framework)
    - torchvision (for image transforms)
    - PIL (Python Imaging Library) for image processing
    - matplotlib (for visualization)
- **Data:**
    - A sample RGB image resized to 128x128 pixels for compression and reconstruction

Theory

An **autoencoder** is a type of artificial neural network used for unsupervised learning of efficient codings. The network is trained to encode the input into a latent space representation and then decode that representation back to a reconstruction of the original input.

- **Encoder:** Compresses the input image into a smaller, dense latent representation (bottleneck).
- **Latent Space:** Contains the compressed encoded data, ideally retaining the most important features.
- **Decoder:** Reconstructs the image from the latent space representation.

This process can be used for **lossy image compression**, where the reconstructed image approximates the original with fewer data.

Procedure

1. **Data Preparation:**
    - Load a sample image.
    - Resize the image to 128x128 pixels.
    - Convert the image to a tensor and normalize pixel values to [0,1].
2. **Model Design:**
    - Construct an autoencoder with:
        - **Encoder:** Four convolutional layers with ReLU activations, reducing spatial dimensions.
        - **Decoder:** Four transpose convolutional layers with ReLU activations and a Sigmoid output layer for pixel values.
3. **Training:**
    - Define Mean Squared Error (MSE) loss between the original and reconstructed image.
    - Use Adam optimizer with a learning rate of 0.001.
    - Train the model for 500 epochs on the single image to learn to reconstruct it accurately.
    - Monitor and print loss every 50 epochs.
4. **Testing and Visualization:**
    - Use the trained autoencoder to reconstruct the image.
    - Display the original and reconstructed images side-by-side.
    - Compare visual similarity to assess compression quality.

Code:-

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

IMAGE_PATH = "/content/sunflower_yellow_flowers_215332.jpg"
IMAGE_SIZE = 128

transform = transforms.Compose([
    transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
```

```
    transforms.ToTensor(),
])

image = Image.open(IMAGE_PATH).convert('RGB')
input_image = transform(image).unsqueeze(0)

class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 16, 3, stride=2, padding=1),
            nn.ReLU(True),
            nn.Conv2d(16, 32, 3, stride=2, padding=1),
            nn.ReLU(True),
            nn.Conv2d(32, 64, 3, stride=2, padding=1),
            nn.ReLU(True),
            nn.Conv2d(64, 128, 3, stride=2, padding=1),
            nn.ReLU(True)
        )
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(128, 64, 3, stride=2, padding=1, output_padding=1),
            nn.ReLU(True),
            nn.ConvTranspose2d(64, 32, 3, stride=2, padding=1, output_padding=1),
            nn.ReLU(True),
            nn.ConvTranspose2d(32, 16, 3, stride=2, padding=1, output_padding=1),
            nn.ReLU(True),
            nn.ConvTranspose2d(16, 3, 3, stride=2, padding=1, output_padding=1),
            nn.Sigmoid()
        )

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

model = Autoencoder()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
epochs = 1000

for epoch in range(epochs):
    optimizer.zero_grad()
    output, _ = model(input_image)
    loss = criterion(output, input_image)
    loss.backward()
    optimizer.step()
    if (epoch+1) % 50 == 0:
        print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.6f}")
```

```
model.eval()
with torch.no_grad():
    reconstructed, compressed = model(input_image)

input_size = input_image.numel()
compressed_size = compressed.numel()

print(f"Input image size (number of values): {input_size}")
print(f"Compressed latent size (number of values): {compressed_size}")
print(f"Compression ratio: {input_size / compressed_size:.2f}x")

def show_image(tensor_img, title):
    img = tensor_img.squeeze(0).permute(1, 2, 0).numpy()
    plt.imshow(img)
    plt.title(title)
    plt.axis('off')
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
show_image(input_image, "Original Image")
plt.subplot(1, 2, 2)
show_image(reconstructed, "Reconstructed Image")
plt.tight_layout()
plt.show()
```

Results

```
Epoch [50/1000], Loss: 0.055425
Epoch [100/1000], Loss: 0.027625
Epoch [150/1000], Loss: 0.017442
Epoch [200/1000], Loss: 0.012070
Epoch [250/1000], Loss: 0.009431
Epoch [300/1000], Loss: 0.007958
Epoch [350/1000], Loss: 0.006203
Epoch [400/1000], Loss: 0.005360
Epoch [450/1000], Loss: 0.004516
Epoch [500/1000], Loss: 0.004004
Epoch [550/1000], Loss: 0.003414
Epoch [600/1000], Loss: 0.003116
Epoch [650/1000], Loss: 0.003107
Epoch [700/1000], Loss: 0.002473
Epoch [750/1000], Loss: 0.002315
Epoch [800/1000], Loss: 0.002246
Epoch [850/1000], Loss: 0.001982
Epoch [900/1000], Loss: 0.001881
Epoch [950/1000], Loss: 0.001802
Epoch [1000/1000], Loss: 0.001655
Input image size (number of values): 49152
Compressed latent size (number of values): 8192
Compression ratio: 6.00x
```

Conclusion

The convolutional autoencoder effectively compressed and reconstructed the image using a compact latent space. Training on a single image demonstrated the core concept of learned image compression. The model achieved reasonable reconstruction quality after sufficient training. With larger datasets, such autoencoders can generalize for practical lossy image compression tasks.