```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np

transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])


train_dataset = torchvision.datasets.CIFAR10(root="./data",
train=True, transform=transform, download=True)
test_dataset = torchvision.datasets.CIFAR10(root="./data",
train=False, transform=transform, download=True)


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

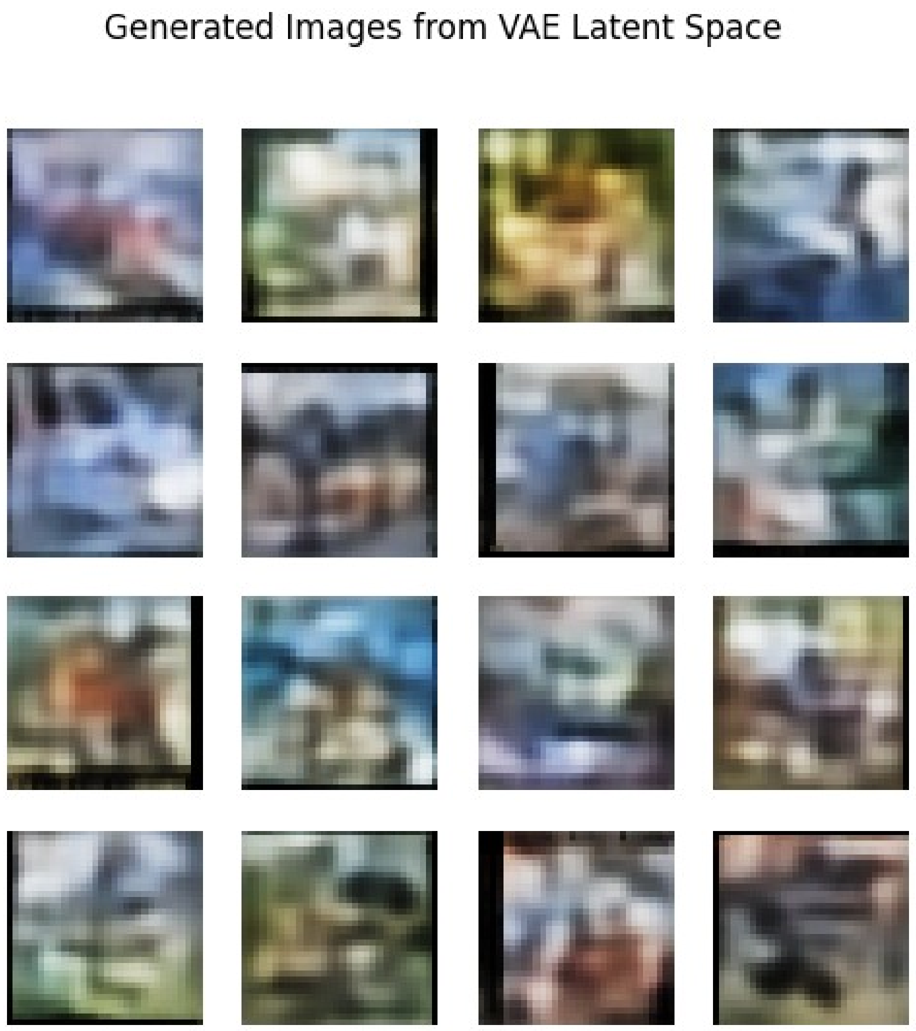
```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
./data\cifar-10-python.tar.gz

100%|██████████| 170M/170M [05:19<00:00, 534kB/s]

Extracting ./data\cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

```python
class VAE(nn.Module):
    def __init__(self, latent_dim=256):
        super(VAE, self).__init__()
        self.latent_dim = latent_dim

        # Encoder
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.Dropout(0.3),  # Regularization

            nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.Dropout(0.3),

            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.Dropout(0.3),
        )

        self.fc_mu = nn.Linear(128 * 4 * 4, latent_dim)
        self.fc_logvar = nn.Linear(128 * 4 * 4, latent_dim)


        self.decoder_input = nn.Linear(latent_dim, 128 * 4 * 4)
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2,
padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),

            nn.ConvTranspose2d(64, 32, kernel_size=4, stride=2,
padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),

            nn.ConvTranspose2d(32, 3, kernel_size=4, stride=2,
padding=1),
            nn.Tanh()
        )

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

    def forward(self, x):
        x = self.encoder(x)
        x = x.view(x.size(0), -1)
        mu, logvar = self.fc_mu(x), self.fc_logvar(x)
        z = self.reparameterization(mu, logvar)
        x = self.decoder_input(z).view(x.size(0), 128, 4, 4)
        x = self.decoder(x)
        return x, mu, logvar

def vae_loss_function(recon_x, x, mu, logvar, beta=0.1):
    reconstruction_loss = F.mse_loss(recon_x, x, reduction="sum")
    kl_divergence = -0.5 * torch.sum(1 + logvar - mu.pow(2) -
logvar.exp())
    return reconstruction_loss + beta * kl_divergence
```

```
vae = VAE(latent_dim=256).to(device)
optimizer = optim.Adam(vae.parameters(), lr=0.001, weight_decay=1e-4)

scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10,
gamma=0.5)

num_epochs = 300

for epoch in range(num_epochs):
    total_loss = 0
    vae.train()

    for images, _ in train_loader:
        images = images.to(device)
        optimizer.zero_grad()
        outputs, mu, logvar = vae(images)
        loss = vae_loss_function(outputs, images, mu, logvar)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    scheduler.step()
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {total_loss /
len(train_loader):.4f}")
```

```
Epoch [1/300], Loss: 41056.5003
Epoch [2/300], Loss: 23134.0243
Epoch [3/300], Loss: 20244.6839
Epoch [4/300], Loss: 18828.0475
Epoch [5/300], Loss: 17890.2783
Epoch [6/300], Loss: 17168.6471
Epoch [7/300], Loss: 16745.5004
Epoch [8/300], Loss: 16362.5803
Epoch [9/300], Loss: 16143.9096
Epoch [10/300], Loss: 15819.3558
Epoch [11/300], Loss: 15259.4053
Epoch [12/300], Loss: 15099.6694
Epoch [13/300], Loss: 14974.3914
Epoch [14/300], Loss: 14907.9837
Epoch [15/300], Loss: 14793.5950
Epoch [16/300], Loss: 14758.9677
Epoch [17/300], Loss: 14685.9862
Epoch [18/300], Loss: 14569.0417
Epoch [19/300], Loss: 14529.2722
Epoch [20/300], Loss: 14477.0306
Epoch [21/300], Loss: 14203.9459
Epoch [22/300], Loss: 14155.6405
Epoch [23/300], Loss: 14109.9339
Epoch [24/300], Loss: 14117.1192
Epoch [25/300], Loss: 14064.4828
Epoch [26/300], Loss: 14033.7029
Epoch [27/300], Loss: 13989.5016
Epoch [28/300], Loss: 13971.8817
Epoch [29/300], Loss: 13972.6813
Epoch [30/300], Loss: 13934.4682
Epoch [31/300], Loss: 13836.7414
Epoch [32/300], Loss: 13774.6444
Epoch [33/300], Loss: 13742.4931
Epoch [34/300], Loss: 13754.4894
Epoch [35/300], Loss: 13735.5845
Epoch [36/300], Loss: 13721.5866
Epoch [37/300], Loss: 13721.6212
Epoch [38/300], Loss: 13707.7300
Epoch [39/300], Loss: 13697.7476
Epoch [40/300], Loss: 13696.6488
Epoch [41/300], Loss: 13627.1819
Epoch [42/300], Loss: 13623.5583
Epoch [43/300], Loss: 13603.5233
Epoch [44/300], Loss: 13596.7970
Epoch [45/300], Loss: 13580.0073
Epoch [46/300], Loss: 13564.0093
Epoch [47/300], Loss: 13562.3508
Epoch [48/300], Loss: 13552.9885
Epoch [49/300], Loss: 13564.5701
Epoch [50/300], Loss: 13564.4669
Epoch [51/300], Loss: 13541.8415
Epoch [52/300], Loss: 13529.0318
Epoch [53/300], Loss: 13527.9808
Epoch [54/300], Loss: 13512.7375
Epoch [55/300], Loss: 13498.9709
Epoch [56/300], Loss: 13502.0036
Epoch [57/300], Loss: 13497.6613
Epoch [58/300], Loss: 13500.6924
Epoch [59/300], Loss: 13491.3651
Epoch [60/300], Loss: 13482.2075
Epoch [61/300], Loss: 13473.1164
Epoch [62/300], Loss: 13486.2408
Epoch [63/300], Loss: 13471.8649
Epoch [64/300], Loss: 13467.5726
Epoch [65/300], Loss: 13469.2186
Epoch [66/300], Loss: 13466.7630
Epoch [67/300], Loss: 13457.7100
Epoch [68/300], Loss: 13475.9750
Epoch [69/300], Loss: 13449.7811
Epoch [70/300], Loss: 13450.9202
Epoch [71/300], Loss: 13459.1907
Epoch [72/300], Loss: 13467.4232
Epoch [73/300], Loss: 13452.2366
Epoch [74/300], Loss: 13441.4497
Epoch [75/300], Loss: 13449.7135
Epoch [76/300], Loss: 13448.3436
Epoch [77/300], Loss: 13453.7525
Epoch [78/300], Loss: 13440.1929
Epoch [79/300], Loss: 13450.4237
Epoch [80/300], Loss: 13454.8413
Epoch [81/300], Loss: 13443.3803
Epoch [82/300], Loss: 13444.6198
Epoch [83/300], Loss: 13431.6759
Epoch [84/300], Loss: 13451.4774
Epoch [85/300], Loss: 13442.6393
Epoch [86/300], Loss: 13448.5618
Epoch [87/300], Loss: 13437.1840
Epoch [88/300], Loss: 13456.6973
Epoch [89/300], Loss: 13448.6833
Epoch [90/300], Loss: 13433.1088
Epoch [91/300], Loss: 13442.2916
Epoch [92/300], Loss: 13445.9358
Epoch [93/300], Loss: 13436.9515
Epoch [94/300], Loss: 13442.1185
Epoch [95/300], Loss: 13451.1528
Epoch [96/300], Loss: 13444.9185
Epoch [97/300], Loss: 13432.8716
Epoch [98/300], Loss: 13439.3113
Epoch [99/300], Loss: 13429.1630
Epoch [100/300], Loss: 13427.2026
Epoch [101/300], Loss: 13424.4160
Epoch [102/300], Loss: 13440.2190
Epoch [103/300], Loss: 13442.2894
Epoch [104/300], Loss: 13428.9470
Epoch [105/300], Loss: 13426.0131
Epoch [106/300], Loss: 13431.6653
Epoch [107/300], Loss: 13430.9275
Epoch [108/300], Loss: 13417.4533
Epoch [109/300], Loss: 13459.3067
Epoch [110/300], Loss: 13439.8516
Epoch [111/300], Loss: 13432.6645
Epoch [112/300], Loss: 13424.5698
Epoch [113/300], Loss: 13433.5537
Epoch [114/300], Loss: 13441.6157
Epoch [115/300], Loss: 13430.1758
Epoch [116/300], Loss: 13407.2238
Epoch [117/300], Loss: 13455.9120
Epoch [118/300], Loss: 13428.2918
Epoch [119/300], Loss: 13431.6896
Epoch [120/300], Loss: 13430.5893
Epoch [121/300], Loss: 13415.6491
Epoch [122/300], Loss: 13448.3393
Epoch [123/300], Loss: 13416.6836
Epoch [124/300], Loss: 13439.9022
Epoch [125/300], Loss: 13421.9258
Epoch [126/300], Loss: 13425.2611
Epoch [127/300], Loss: 13427.7082
Epoch [128/300], Loss: 13441.3644
Epoch [129/300], Loss: 13437.0451
Epoch [130/300], Loss: 13420.0017
Epoch [131/300], Loss: 13423.0496
Epoch [132/300], Loss: 13442.0044
Epoch [133/300], Loss: 13429.7460
Epoch [134/300], Loss: 13429.3776
Epoch [135/300], Loss: 13447.4645
Epoch [136/300], Loss: 13427.1638
Epoch [137/300], Loss: 13427.6618
Epoch [138/300], Loss: 13435.4448
Epoch [139/300], Loss: 13433.8923
Epoch [140/300], Loss: 13433.6267
Epoch [141/300], Loss: 13443.9895
Epoch [142/300], Loss: 13422.6779
Epoch [143/300], Loss: 13451.9071
Epoch [144/300], Loss: 13422.6070
Epoch [145/300], Loss: 13448.3795
Epoch [146/300], Loss: 13441.5626
Epoch [147/300], Loss: 13432.3090
Epoch [148/300], Loss: 13432.4518
Epoch [149/300], Loss: 13443.0548
Epoch [150/300], Loss: 13430.4295
Epoch [151/300], Loss: 13440.0925
Epoch [152/300], Loss: 13411.0621
Epoch [153/300], Loss: 13429.1664
Epoch [154/300], Loss: 13431.5538
Epoch [155/300], Loss: 13427.9330
Epoch [156/300], Loss: 13410.5123
Epoch [157/300], Loss: 13435.2071
Epoch [158/300], Loss: 13436.3000
Epoch [159/300], Loss: 13435.3474
Epoch [160/300], Loss: 13421.1356
Epoch [161/300], Loss: 13427.2609
Epoch [162/300], Loss: 13433.4918
Epoch [163/300], Loss: 13427.6639
Epoch [164/300], Loss: 13448.2454
Epoch [165/300], Loss: 13430.0669
Epoch [166/300], Loss: 13424.4483
Epoch [167/300], Loss: 13433.2522
Epoch [168/300], Loss: 13419.6874
Epoch [169/300], Loss: 13428.5343
Epoch [170/300], Loss: 13424.1021
Epoch [171/300], Loss: 13425.2310
Epoch [172/300], Loss: 13429.9082
Epoch [173/300], Loss: 13424.3354
Epoch [174/300], Loss: 13429.1704
Epoch [175/300], Loss: 13431.3104
Epoch [176/300], Loss: 13431.1536
Epoch [177/300], Loss: 13432.7325
Epoch [178/300], Loss: 13438.2352
Epoch [179/300], Loss: 13427.3942
Epoch [180/300], Loss: 13433.2611
Epoch [181/300], Loss: 13435.8407
Epoch [182/300], Loss: 13439.9747
Epoch [183/300], Loss: 13425.6768
Epoch [184/300], Loss: 13423.7574
Epoch [185/300], Loss: 13429.1247
Epoch [186/300], Loss: 13442.3590
Epoch [187/300], Loss: 13416.8780
Epoch [188/300], Loss: 13437.8119
Epoch [189/300], Loss: 13442.4203
Epoch [190/300], Loss: 13441.1032
Epoch [191/300], Loss: 13422.0585
Epoch [192/300], Loss: 13430.3835
Epoch [193/300], Loss: 13431.8166
Epoch [194/300], Loss: 13427.2200
Epoch [195/300], Loss: 13439.1555
Epoch [196/300], Loss: 13440.7277
Epoch [197/300], Loss: 13419.3640
Epoch [198/300], Loss: 13420.5329
Epoch [199/300], Loss: 13438.2096
Epoch [200/300], Loss: 13433.0918
Epoch [201/300], Loss: 13422.1304
Epoch [202/300], Loss: 13420.3058
Epoch [203/300], Loss: 13429.1016
Epoch [204/300], Loss: 13435.9252
Epoch [205/300], Loss: 13415.7626
Epoch [206/300], Loss: 13427.7868
Epoch [207/300], Loss: 13444.3407
Epoch [208/300], Loss: 13438.4611
Epoch [209/300], Loss: 13419.8881
Epoch [210/300], Loss: 13430.8660
Epoch [211/300], Loss: 13431.9440
Epoch [212/300], Loss: 13445.4458
Epoch [213/300], Loss: 13422.7919
Epoch [214/300], Loss: 13429.2168
Epoch [215/300], Loss: 13434.2288
Epoch [216/300], Loss: 13435.8165
Epoch [217/300], Loss: 13435.3237
Epoch [218/300], Loss: 13425.0681
Epoch [219/300], Loss: 13418.5755
Epoch [220/300], Loss: 13435.3673
Epoch [221/300], Loss: 13432.2901
Epoch [222/300], Loss: 13441.4150
Epoch [223/300], Loss: 13436.5097
Epoch [224/300], Loss: 13425.3478
Epoch [225/300], Loss: 13426.7839
Epoch [226/300], Loss: 13423.2082
Epoch [227/300], Loss: 13412.7795
Epoch [228/300], Loss: 13445.8429
Epoch [229/300], Loss: 13425.1438
Epoch [230/300], Loss: 13428.2516
Epoch [231/300], Loss: 13432.2017
Epoch [232/300], Loss: 13424.4190
Epoch [233/300], Loss: 13417.7446
Epoch [234/300], Loss: 13436.5382
Epoch [235/300], Loss: 13420.5502
Epoch [236/300], Loss: 13436.9259
Epoch [237/300], Loss: 13433.3686
Epoch [238/300], Loss: 13421.6506
Epoch [239/300], Loss: 13426.9347
Epoch [240/300], Loss: 13422.9348
Epoch [241/300], Loss: 13447.0646
Epoch [242/300], Loss: 13430.8900
Epoch [243/300], Loss: 13421.2481
Epoch [244/300], Loss: 13433.7341
Epoch [245/300], Loss: 13437.2425
Epoch [246/300], Loss: 13429.0129
Epoch [247/300], Loss: 13432.9606
Epoch [248/300], Loss: 13431.0489
Epoch [249/300], Loss: 13433.6152
Epoch [250/300], Loss: 13433.9793
Epoch [251/300], Loss: 13440.1297
Epoch [252/300], Loss: 13423.9673
Epoch [253/300], Loss: 13432.5319
Epoch [254/300], Loss: 13446.2698
Epoch [255/300], Loss: 13432.3005
Epoch [256/300], Loss: 13431.5728
Epoch [257/300], Loss: 13423.6044
Epoch [258/300], Loss: 13434.7306
Epoch [259/300], Loss: 13433.9615
Epoch [260/300], Loss: 13420.8535
Epoch [261/300], Loss: 13437.3568
Epoch [262/300], Loss: 13443.7473
Epoch [263/300], Loss: 13444.0618
Epoch [264/300], Loss: 13435.0363
Epoch [265/300], Loss: 13420.5763
Epoch [266/300], Loss: 13437.0987
Epoch [267/300], Loss: 13421.0884
Epoch [268/300], Loss: 13437.6795
Epoch [269/300], Loss: 13407.4915
Epoch [270/300], Loss: 13427.7866
Epoch [271/300], Loss: 13422.5327
Epoch [272/300], Loss: 13435.1028
Epoch [273/300], Loss: 13427.7584
Epoch [274/300], Loss: 13430.7717
Epoch [275/300], Loss: 13436.3249
Epoch [276/300], Loss: 13447.8008
Epoch [277/300], Loss: 13421.0883
Epoch [278/300], Loss: 13433.8649
Epoch [279/300], Loss: 13414.6008
Epoch [280/300], Loss: 13413.4330
Epoch [281/300], Loss: 13427.2269
Epoch [282/300], Loss: 13420.0693
Epoch [283/300], Loss: 13430.1336
Epoch [284/300], Loss: 13420.6514
Epoch [285/300], Loss: 13444.1281
Epoch [286/300], Loss: 13443.0717
Epoch [287/300], Loss: 13432.2902
Epoch [288/300], Loss: 13437.8965
Epoch [289/300], Loss: 13424.2540
Epoch [290/300], Loss: 13440.2591
Epoch [291/300], Loss: 13437.7056
Epoch [292/300], Loss: 13439.9602
Epoch [293/300], Loss: 13428.3685
Epoch [294/300], Loss: 13458.4746
Epoch [295/300], Loss: 13431.9114
Epoch [296/300], Loss: 13430.4897
Epoch [297/300], Loss: 13427.7705
Epoch [298/300], Loss: 13441.2670
Epoch [299/300], Loss: 13436.5345
Epoch [300/300], Loss: 13412.4288
```

```
vae.eval()
with torch.no_grad():
    z = torch.randn(16, 256).to(device)
    generated_images = vae.decoder(vae.decoder_input(z).view(16, 128,
4, 4)).cpu()

# Plot generated images
fig, axes = plt.subplots(4, 4, figsize=(6, 6))
for i, ax in enumerate(axes.flat):
    ax.imshow(generated_images[i].permute(1, 2, 0).numpy() * 0.5 +
0.5)
    ax.axis("off")

plt.suptitle("Generated Images from VAE Latent Space")
plt.show()
```



```
torch.save(vae.state_dict(), "vae_cifar10.pth")
print("□ VAE model saved!")
```

```
□ VAE model saved!
```

```
test_images, _ = next(iter(test_loader))
test_images = test_images.to(device)


vae.eval()
with torch.no_grad():
    reconstructed_images, _, _ = vae(test_images)

loss = F.mse_loss(reconstructed_images, test_images).item()
print(f"□ VAE Reconstruction Loss: {loss:.4f}")
```

```
□ VAE Reconstruction Loss: 0.0207
```

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision
from torch.utils.data import DataLoader


class Autoencoder(nn.Module):
    def __init__(self, latent_dim=100):
        super(Autoencoder, self).__init__()

        # Encoder
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 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.Flatten(),
            nn.Linear(128 * 4 * 4, latent_dim)
        )


        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 128 * 4 * 4),
            nn.ReLU(),
            nn.Unflatten(1, (128, 4, 4)),
            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, 3, kernel_size=4, stride=2,
padding=1),
            nn.Sigmoid()

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


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


transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_dataset = torchvision.datasets.CIFAR10(root="./data",
train=True, transform=transform, download=True)
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)


ae_model = Autoencoder(latent_dim=100).to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(ae_model.parameters(), lr=0.001)


num_epochs = 50
for epoch in range(num_epochs):
    total_loss = 0
    for images, _ in train_loader:
        images = images.to(device)
        optimizer.zero_grad()
        outputs = ae_model(images)
        loss = criterion(outputs, images)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

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


torch.save(ae_model.state_dict(), "autoencoder_cifar10.pth")
print("� Autoencoder training completed!")
```

```
Files already downloaded and verified
Epoch [1/50], AE Loss: 0.2609
Epoch [2/50], AE Loss: 0.2560
Epoch [3/50], AE Loss: 0.2560
Epoch [4/50], AE Loss: 0.2560
Epoch [5/50], AE Loss: 0.2560
Epoch [6/50], AE Loss: 0.2560
Epoch [7/50], AE Loss: 0.2560
Epoch [8/50], AE Loss: 0.2560
Epoch [9/50], AE Loss: 0.2560
Epoch [10/50], AE Loss: 0.2560
Epoch [11/50], AE Loss: 0.2560
Epoch [12/50], AE Loss: 0.2560
Epoch [13/50], AE Loss: 0.2560
Epoch [14/50], AE Loss: 0.2560
Epoch [15/50], AE Loss: 0.2560
Epoch [16/50], AE Loss: 0.2560
Epoch [17/50], AE Loss: 0.2560
Epoch [18/50], AE Loss: 0.2560
Epoch [19/50], AE Loss: 0.2560
Epoch [20/50], AE Loss: 0.2560
Epoch [21/50], AE Loss: 0.2560
Epoch [22/50], AE Loss: 0.2560
Epoch [23/50], AE Loss: 0.2560
Epoch [24/50], AE Loss: 0.2560
Epoch [25/50], AE Loss: 0.2560
Epoch [26/50], AE Loss: 0.2560
Epoch [27/50], AE Loss: 0.2560
Epoch [28/50], AE Loss: 0.2560
Epoch [29/50], AE Loss: 0.2560
Epoch [30/50], AE Loss: 0.2560
Epoch [31/50], AE Loss: 0.2559
Epoch [32/50], AE Loss: 0.2560
Epoch [33/50], AE Loss: 0.2560
Epoch [34/50], AE Loss: 0.2560
Epoch [35/50], AE Loss: 0.2560
Epoch [36/50], AE Loss: 0.2560
Epoch [37/50], AE Loss: 0.2560
Epoch [38/50], AE Loss: 0.2560
Epoch [39/50], AE Loss: 0.2560
Epoch [40/50], AE Loss: 0.2560
Epoch [41/50], AE Loss: 0.2560
Epoch [42/50], AE Loss: 0.2560
Epoch [43/50], AE Loss: 0.2560
Epoch [44/50], AE Loss: 0.2560
Epoch [45/50], AE Loss: 0.2560
Epoch [46/50], AE Loss: 0.2560
Epoch [47/50], AE Loss: 0.2560
Epoch [48/50], AE Loss: 0.2560
Epoch [49/50], AE Loss: 0.2560
Epoch [50/50], AE Loss: 0.2560
□ Autoencoder training completed!
```

```python
torch.save(ae_model.state_dict(), "autoencoder_cifar10.pth")
print("□ Autoencoder model weights saved successfully!")
```

```
□ Autoencoder model weights saved successfully!
```

```python
ae_model = Autoencoder(latent_dim=100).to(device)


ae_model.load_state_dict(torch.load("autoencoder_cifar10.pth"))


ae_model.eval()
print("□ Autoencoder model loaded successfully!")
```

```
□ Autoencoder model loaded successfully!
```

```
C:\Users\CSIR_prosthetic\AppData\Local\Temp\
ipykernel_16016\3136093701.py:5: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  ae_model.load_state_dict(torch.load("autoencoder_cifar10.pth"))
```

```
import torch


ae_model.load_state_dict(torch.load("autoencoder_cifar10.pth"))
ae_model.eval()


vae.load_state_dict(torch.load("vae_cifar10.pth"))
vae.eval()

print("□ Both AE and VAE models loaded successfully!")
```

```
□ Both AE and VAE models loaded successfully!
```

```
C:\Users\CSIR_prosthetic\AppData\Local\Temp\
ipykernel_16016\1444863018.py:4: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  ae_model.load_state_dict(torch.load("autoencoder_cifar10.pth"))
C:\Users\CSIR_prosthetic\AppData\Local\Temp\
ipykernel_16016\1444863018.py:8: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  vae.load_state_dict(torch.load("vae_cifar10.pth"))
```

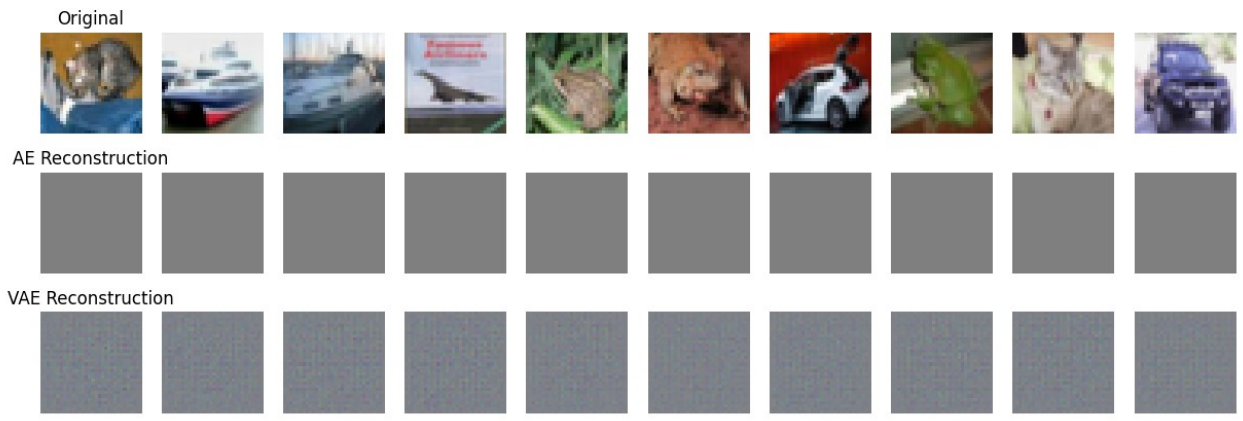
```
import torchvision.transforms as transforms
import torchvision
from torch.utils.data import DataLoader


transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])


test_dataset = torchvision.datasets.CIFAR10(root="./data",
train=False, transform=transform, download=True)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)

print("□ Test dataset loaded!")
```

```
Files already downloaded and verified
□ Test dataset loaded!
```

```
import torch.nn.functional as F
import matplotlib.pyplot as plt

# Get a batch of test images
test_images, _ = next(iter(test_loader))
test_images = test_images.to(device)

# Compute reconstructions
with torch.no_grad():
    ae_reconstructed = ae_model(test_images).cpu()
    vae_reconstructed, _, _ = vae(test_images)

ae_loss = F.mse_loss(ae_reconstructed.to(device), test_images).item()
vae_loss = F.mse_loss(vae_reconstructed, test_images).item()
```

```
print(f"□ Autoencoder Reconstruction Loss: {ae_loss:.4f}")
print(f"□ Variational Autoencoder Reconstruction Loss:
{vae_loss:.4f}")


fig, axes = plt.subplots(3, 10, figsize=(15, 5))

for i in range(10):

    axes[0, i].imshow(test_images[i].cpu().permute(1, 2, 0) * 0.5 +
0.5)
    axes[0, i].axis("off")


    axes[1, i].imshow(ae_reconstructed[i].cpu().permute(1, 2,
0).numpy() * 0.5 + 0.5)
    axes[1, i].axis("off")


    axes[2, i].imshow(vae_reconstructed[i].cpu().permute(1, 2,
0).numpy() * 0.5 + 0.5)
    axes[2, i].axis("off")

axes[0, 0].set_title("Original", fontsize=12)
axes[1, 0].set_title("AE Reconstruction", fontsize=12)
axes[2, 0].set_title("VAE Reconstruction", fontsize=12)
plt.show()
```

```
□ Autoencoder Reconstruction Loss: 0.2658
□ Variational Autoencoder Reconstruction Loss: 0.2752
```



VAE Without Regularization

```
transform = transforms.Compose([
    transforms.ToTensor(),
```

```python
    transforms.Normalize((0.5,), (0.5,))
])

train_dataset = torchvision.datasets.CIFAR10(root="./data",
train=True, transform=transform, download=True)
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
./data\cifar-10-python.tar.gz

100%|██████████| 170M/170M [00:18<00:00, 9.33MB/s]

Extracting ./data\cifar-10-python.tar.gz to ./data
```

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


class VAE_NoReg(nn.Module):
    def __init__(self, latent_dim=256):
        super(VAE_NoReg, self).__init__()
        self.latent_dim = latent_dim


        self.encoder = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),

            nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),

            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
        )


        self.fc_mu = nn.Linear(128 * 4 * 4, latent_dim)
        self.fc_logvar = nn.Linear(128 * 4 * 4, latent_dim)


        self.decoder_input = nn.Linear(latent_dim, 128 * 4 * 4)
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2,
padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),

            nn.ConvTranspose2d(64, 32, kernel_size=4, stride=2,
padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),

            nn.ConvTranspose2d(32, 3, kernel_size=4, stride=2,
padding=1),
            nn.Tanh()
        )

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

    def forward(self, x):
        x = self.encoder(x)
        x = x.view(x.size(0), -1)
        mu, logvar = self.fc_mu(x), self.fc_logvar(x)
        z = self.reparameterization(mu, logvar)
        x = self.decoder_input(z).view(x.size(0), 128, 4, 4)
        x = self.decoder(x)
        return x, mu, logvar

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


transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_dataset = torchvision.datasets.CIFAR10(root="./data",
train=True, transform=transform, download=True)
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)


vae_noreg = VAE_NoReg(latent_dim=256).to(device)


optimizer_noreg = optim.Adam(vae_noreg.parameters(), lr=0.001)


scheduler_noreg = optim.lr_scheduler.StepLR(optimizer_noreg,
step_size=10, gamma=0.5)


def vae_loss_function(recon_x, x, mu, logvar, beta=0.1):
    reconstruction_loss = F.mse_loss(recon_x, x, reduction="sum")  #
MSE loss
    kl_divergence = -0.5 * torch.sum(1 + logvar - mu.pow(2) -
logvar.exp())
    return reconstruction_loss + beta * kl_divergence


num_epochs = 50

for epoch in range(num_epochs):
    total_loss = 0
    vae_noreg.train()

    for images, _ in train_loader:
        images = images.to(device)
        optimizer_noreg.zero_grad()
        outputs, mu, logvar = vae_noreg(images)
        loss = vae_loss_function(outputs, images, mu, logvar)
        loss.backward()
        optimizer_noreg.step()
        total_loss += loss.item()

    scheduler_noreg.step()
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {total_loss /
len(train_loader):.4f}")


torch.save(vae_noreg.state_dict(), "vae_noreg_cifar10.pth")
print("□ VAE Without Regularization training completed!")
```

```
Files already downloaded and verified
Epoch [1/50], Loss: 31211.1843
Epoch [2/50], Loss: 17457.9144
Epoch [3/50], Loss: 14738.9099
Epoch [4/50], Loss: 13450.2570
Epoch [5/50], Loss: 12701.9288
Epoch [6/50], Loss: 12173.6998
Epoch [7/50], Loss: 11763.4107
Epoch [8/50], Loss: 11497.8540
Epoch [9/50], Loss: 11283.7870
Epoch [10/50], Loss: 11027.9619
Epoch [11/50], Loss: 10573.3972
Epoch [12/50], Loss: 10472.9955
Epoch [13/50], Loss: 10445.9973
Epoch [14/50], Loss: 10357.2275
Epoch [15/50], Loss: 10244.6717
Epoch [16/50], Loss: 10198.1254
Epoch [17/50], Loss: 10146.5468
Epoch [18/50], Loss: 10088.9099
Epoch [19/50], Loss: 10048.2824
Epoch [20/50], Loss: 9970.0887
Epoch [21/50], Loss: 9818.2931
Epoch [22/50], Loss: 9763.9938
Epoch [23/50], Loss: 9753.7868
Epoch [24/50], Loss: 9736.8847
Epoch [25/50], Loss: 9719.2252
Epoch [26/50], Loss: 9690.5807
Epoch [27/50], Loss: 9657.5045
Epoch [28/50], Loss: 9666.0228
Epoch [29/50], Loss: 9629.4477
Epoch [30/50], Loss: 9623.6919
Epoch [31/50], Loss: 9519.7519
Epoch [32/50], Loss: 9496.3041
Epoch [33/50], Loss: 9488.5218
Epoch [34/50], Loss: 9475.4387
Epoch [35/50], Loss: 9472.3994
Epoch [36/50], Loss: 9457.6601
Epoch [37/50], Loss: 9453.8410
Epoch [38/50], Loss: 9445.1707
Epoch [39/50], Loss: 9430.7967
Epoch [40/50], Loss: 9439.0696
Epoch [41/50], Loss: 9380.0093
Epoch [42/50], Loss: 9357.0304
Epoch [43/50], Loss: 9369.4039
Epoch [44/50], Loss: 9360.2726
Epoch [45/50], Loss: 9352.8620
Epoch [46/50], Loss: 9341.1434
Epoch [47/50], Loss: 9364.0040
Epoch [48/50], Loss: 9349.8577
Epoch [49/50], Loss: 9343.4700
Epoch [50/50], Loss: 9340.7514
□ VAE Without Regularization training completed!
```

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

# Define VAE Model with Regularization
class VAE(nn.Module):
    def __init__(self, latent_dim=256):
        super(VAE, self).__init__()
        self.latent_dim = latent_dim

        # Encoder
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.Dropout(0.3),  # Regularization

            nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.Dropout(0.3),

            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.Dropout(0.3),
        )

        # Latent space
        self.fc_mu = nn.Linear(128 * 4 * 4, latent_dim)
        self.fc_logvar = nn.Linear(128 * 4 * 4, latent_dim)

        # Decoder
        self.decoder_input = nn.Linear(latent_dim, 128 * 4 * 4)
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2,
padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),

            nn.ConvTranspose2d(64, 32, kernel_size=4, stride=2,
padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),

            nn.ConvTranspose2d(32, 3, kernel_size=4, stride=2,
padding=1),
            nn.Tanh()  # Outputs [-1,1] to match input normalization
        )

    def reparameterization(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std  # Reparameterization trick

    def forward(self, x):
        x = self.encoder(x)
        x = x.view(x.size(0), -1)
        mu, logvar = self.fc_mu(x), self.fc_logvar(x)
        z = self.reparameterization(mu, logvar)
        x = self.decoder_input(z).view(x.size(0), 128, 4, 4)
        x = self.decoder(x)
        return x, mu, logvar

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

# Initialize VAE
vae = VAE(latent_dim=256).to(device)

# Define optimizer with weight decay (L2 regularization)
optimizer = optim.Adam(vae.parameters(), lr=0.001, weight_decay=1e-4)

scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10,
gamma=0.5)

# Define VAE Loss Function
def vae_loss_function(recon_x, x, mu, logvar, beta=0.1):
    reconstruction_loss = F.mse_loss(recon_x, x, reduction="sum")  #
MSE loss
    kl_divergence = -0.5 * torch.sum(1 + logvar - mu.pow(2) -
logvar.exp())
    return reconstruction_loss + beta * kl_divergence

num_epochs = 50

for epoch in range(num_epochs):
    total_loss = 0
    vae.train()

    for images, _ in train_loader:
        images = images.to(device)
        optimizer.zero_grad()
        outputs, mu, logvar = vae(images)
        loss = vae_loss_function(outputs, images, mu, logvar)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    scheduler.step()
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {total_loss /
len(train_loader):.4f}")

torch.save(vae.state_dict(), "vae_cifar10.pth")
print("� VAE with Regularization training completed & saved!")
```

```
Epoch [1/50], Loss: 36622.1502
Epoch [2/50], Loss: 22355.5351
Epoch [3/50], Loss: 19684.5893
Epoch [4/50], Loss: 18240.7286
Epoch [5/50], Loss: 17484.1456
Epoch [6/50], Loss: 16873.3327
Epoch [7/50], Loss: 16453.1217
Epoch [8/50], Loss: 16107.1204
Epoch [9/50], Loss: 15873.8878
Epoch [10/50], Loss: 15616.4633
Epoch [11/50], Loss: 15129.2866
Epoch [12/50], Loss: 15004.5605
Epoch [13/50], Loss: 14946.0318
Epoch [14/50], Loss: 14873.9146
Epoch [15/50], Loss: 14793.3773
Epoch [16/50], Loss: 14714.7136
Epoch [17/50], Loss: 14673.6771
Epoch [18/50], Loss: 14634.6020
Epoch [19/50], Loss: 14562.6632
Epoch [20/50], Loss: 14548.8211
Epoch [21/50], Loss: 14280.1390
Epoch [22/50], Loss: 14253.6773
Epoch [23/50], Loss: 14219.1534
Epoch [24/50], Loss: 14209.1928
Epoch [25/50], Loss: 14160.7959
Epoch [26/50], Loss: 14159.2915
Epoch [27/50], Loss: 14147.2288
Epoch [28/50], Loss: 14106.4327
Epoch [29/50], Loss: 14080.5106
Epoch [30/50], Loss: 14090.6420
Epoch [31/50], Loss: 13965.3719
Epoch [32/50], Loss: 13947.2107
Epoch [33/50], Loss: 13910.9031
Epoch [34/50], Loss: 13928.2878
Epoch [35/50], Loss: 13933.1986
Epoch [36/50], Loss: 13898.3767
Epoch [37/50], Loss: 13884.4297
Epoch [38/50], Loss: 13886.3403
Epoch [39/50], Loss: 13878.5189
Epoch [40/50], Loss: 13876.2656
Epoch [41/50], Loss: 13805.0497
Epoch [42/50], Loss: 13822.1130
Epoch [43/50], Loss: 13770.3236
Epoch [44/50], Loss: 13773.6785
Epoch [45/50], Loss: 13777.5534
Epoch [46/50], Loss: 13771.4209
Epoch [47/50], Loss: 13766.8313
Epoch [48/50], Loss: 13759.1609
Epoch [49/50], Loss: 13763.7416
Epoch [50/50], Loss: 13754.3169
□ VAE with Regularization training completed & saved!
```

```python
import torch

# Load VAE with Regularization
vae_with_reg = VAE(latent_dim=256).to(device)
vae_with_reg.load_state_dict(torch.load("vae_cifar10.pth"))
vae_with_reg.eval()

# Load VAE without Regularization
vae_noreg = VAE_NoReg(latent_dim=256).to(device)
vae_noreg.load_state_dict(torch.load("vae_noreg_cifar10.pth"))
vae_noreg.eval()

print("▯ Both VAE models (with and without regularization) loaded
successfully!")
```

```
▯ Both VAE models (with and without regularization) loaded
successfully!

C:\Users\CSIR_prosthetic\AppData\Local\Temp\
ipykernel_21772\2812163446.py:5: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  vae_with_reg.load_state_dict(torch.load("vae_cifar10.pth"))
C:\Users\CSIR_prosthetic\AppData\Local\Temp\
ipykernel_21772\2812163446.py:10: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
```

```
related to this experimental feature.
  vae_noreg.load_state_dict(torch.load("vae_noreg_cifar10.pth"))
```

```
import torchvision.transforms as transforms
import torchvision
from torch.utils.data import DataLoader

# Define transform
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Load CIFAR-10 test dataset
test_dataset = torchvision.datasets.CIFAR10(root="./data",
train=False, transform=transform, download=True)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)

print("□ Test dataset loaded!")
```

```
Files already downloaded and verified
□ Test dataset loaded!
```

```
import torch.nn.functional as F
import matplotlib.pyplot as plt

# Get a batch of test images
test_images, _ = next(iter(test_loader))
test_images = test_images.to(device)

# Compute reconstructions
with torch.no_grad():
    vae_reg_reconstructed, _, _ = vae_with_reg(test_images)
    vae_noreg_reconstructed, _, _ = vae_noreg(test_images)

# Compute MSE Loss for both models
vae_reg_loss = F.mse_loss(vae_reg_reconstructed, test_images).item()
vae_noreg_loss = F.mse_loss(vae_noreg_reconstructed,
test_images).item()

print(f"□ VAE with Regularization Loss: {vae_reg_loss:.4f}")
print(f"□ VAE without Regularization Loss: {vae_noreg_loss:.4f}")
```

```
□ VAE with Regularization Loss: 0.0220
□ VAE without Regularization Loss: 0.0150
```

```
# Plot Original vs. Reconstructed Images
fig, axes = plt.subplots(3, 10, figsize=(15, 5))

for i in range(10):
    # Original images
    axes[0, i].imshow(test_images[i].cpu().permute(1, 2, 0) * 0.5 +
0.5)
    axes[0, i].axis("off")

    # VAE without Regularization
    axes[1, i].imshow(vae_noreg_reconstructed[i].cpu().permute(1, 2,
0) * 0.5 + 0.5)
    axes[1, i].axis("off")

    # VAE with Regularization
    axes[2, i].imshow(vae_reg_reconstructed[i].cpu().permute(1, 2, 0)
* 0.5 + 0.5)
    axes[2, i].axis("off")

axes[0, 0].set_title("Original")
axes[1, 0].set_title("VAE No Reg")
axes[2, 0].set_title("VAE With Reg")
plt.show()
```