

# MSC 科研组 2025 招新题

Selen Su

2025 年 9 月 14 日

## 1 引言

许多学者认为，深度学习是一种压缩技术，主要解决的问题是将复杂的信息压缩为可以用少量参数表示的形式。Selen 老师非常喜欢自己的头像：图 1，因此也想把它通过神经网络压缩一下。这意味着，无论什么时候，Selen 都可以在找不到原图的时候通过这个网络来重建图像。



图 1: 这将是大家用于完成题目的图像，也就是代码中的 ye.png

具体来说，Selen 老师决定训练一个 MLP（多层感知机）来学习一个映射关系  $f : (x, y) \rightarrow (R, G, B)$ 。即，输入一个像素的坐标，网络输出该点的颜色值；输入一连串图像的每个像素点的坐标，网络就还原一个完整的图像。

Selen 老师很快写出来了代码 Listing 1。然而，这效果实在是太差了!! 经过 8000 轮的训练，模型输出的人物还是模糊的，如图 2。这个时候 Selen 想到，在深度学习的表示方式中，许多研究在处理图像的时候会采用位置编码的方式，即通过 sin-cos 编码图像不同位置的坐标。位置编码通过一系列不同频率的正弦和余弦函数，将低维的输入坐标  $\mathbf{p}$  映射到一个更高维的特征向量  $\gamma(\mathbf{p})$ 。其公式如下：

$$\gamma(\mathbf{p}) = \left( \dots, \sin(2^k \pi \mathbf{p}), \cos(2^k \pi \mathbf{p}), \dots \right)_{k=0}^{L-1}$$

Selen 觉得如果将原始的  $x, y$  坐标输入转换成这样编码以后的输入，模型就可以更好地了解不同坐标之间的位置关系，这样对于全局的理解是更好的。于是 Selen 为了控制变量，没有改变模型，

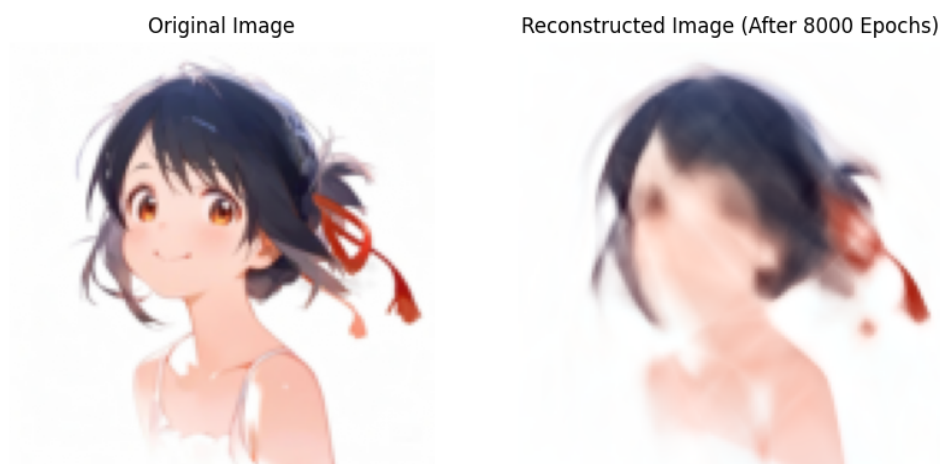


图 2: 模糊的重建图像

也没有改变训练的轮数等任何设置，仅仅将输入的  $x, y$  替换为了 sin-cos 编码以后的表示，并写好了代码 Listing 2。

经过同样的模型同样程度的训练，重建结果却清晰了好多！如图 3。

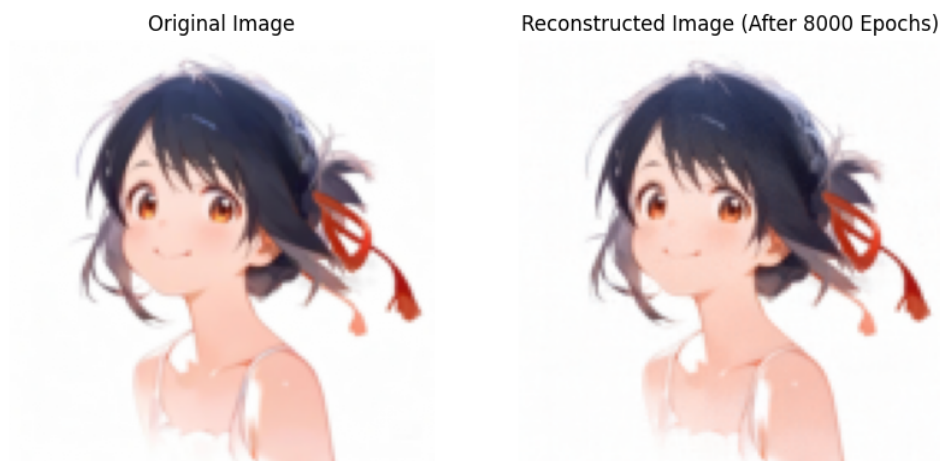


图 3: 清晰的重建图像

## 2 你需要解决的

- 为什么会出现这种差异呢？
- 有没有什么更好的重建方法呢？

- 你可以提交代码/文字报告到邮箱: s3702681@gmail.com, 或通过 qq 联系 Selen
- 相关资料会上传到群文件中, 请注意查收

### 3 代码

```
1 import torch
2 import torch.nn as nn
3 from torchvision.transforms import functional as TF
4 from PIL import Image
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import time
8 import os
9
10 # --- 1. 超参数设置 ---
11 IMAGE_PATH = 'ye.png'
12 IMG_SIZE = 128
13 HIDDEN_DIM = 256
14 N_HIDDEN_LAYERS = 4
15 EPOCHS = 8000
16 LEARNING_RATE = 1e-4
17 DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
18
19 print(f"Using device: {DEVICE}")
20
21 # 加载和预处理图像
22 img = Image.open(IMAGE_PATH).convert('RGB')
23 print(f"Successfully loaded image from: {IMAGE_PATH}")
24
25 img = TF.resize(img, (IMG_SIZE, IMG_SIZE))
26 img_tensor = TF.to_tensor(img).to(DEVICE) # shape: [3, H, W], 范围 [0, 1]
27
28 # 2. 对像素值进行标准化
29 # 计算每个通道的均值和标准差
30 # img_tensor shape is [C, H, W], so we calculate mean/std over H and W dimensions
31 mean = torch.mean(img_tensor, dim=[1, 2])
32 std = torch.std(img_tensor, dim=[1, 2])
33
34 # 为防止标准差为0 (例如纯色通道) 导致除零错误, 增加一个极小值
35 std = torch.max(std, torch.tensor(1e-6).to(DEVICE))
36
37 print(f"\nImage stats (per channel):")
38 print(f"Mean: {mean.cpu().numpy()}")
39 print(f"Std: {std.cpu().numpy()}")
40
41 # 应用标准化: (x - mean) / std
42 # 我们需要调整 mean 和 std 的形状以利用广播机制
43 img_tensor_standardized = (img_tensor - mean[:, None, None]) / std[:, None, None]
44
45 H, W = IMG_SIZE, IMG_SIZE
46 pixels = img_tensor_standardized.permute(1, 2, 0).view(-1, 3) # 使用标准化后的像素作为目标
47
48 # --- 3. 创建输入数据 (坐标网格) ---
49 grid_y, grid_x = torch.meshgrid(torch.linspace(0, H - 1, H), torch.linspace(0, W - 1, W), indexing='ij')
50 coords = torch.stack([
51     grid_x / (W - 1) * 2 - 1,
52     grid_y / (H - 1) * 2 - 1
53 ], dim=-1).to(DEVICE)
54 coords = coords.view(-1, 2)
55
56 print(f"\nImage Size: {H}x{W}")
```

```

57 print(f"Input coordinates shape: {coords.shape}")
58 print(f"Target pixels shape: {pixels.shape}")
59
60
61 # --- 4. 构建 MLP 模型 ---
62 class MLPImageReconstructorLinear(nn.Module):
63     def __init__(self, in_features, hidden_features, hidden_layers, out_features):
64         super().__init__()
65         layers = []
66         layers.append(nn.Linear(in_features, hidden_features))
67         layers.append(nn.ReLU())
68         for _ in range(hidden_layers):
69             layers.append(nn.Linear(hidden_features, hidden_features))
70             layers.append(nn.ReLU())
71
72         layers.append(nn.Linear(hidden_features, out_features))
73
74         self.net = nn.Sequential(*layers)
75
76     def forward(self, x):
77         return self.net(x)
78
79 model = MLPImageReconstructorLinear(
80     in_features=2,
81     hidden_features=HIDDEN_DIM,
82     hidden_layers=N_HIDDEN_LAYERS,
83     out_features=3
84 ).to(DEVICE)
85
86 print("\nModel Architecture:")
87 print(model)
88
89 # --- 5. 定义损失函数和优化器 ---
90 loss_fn = nn.MSELoss()
91 optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
92
93 # --- 6. 训练模型 ---
94 print("\nStarting training...")
95 start_time = time.time()
96 for epoch in range(EPOCHS):
97     predicted_pixels = model(coords)
98     loss = loss_fn(predicted_pixels, pixels)
99     optimizer.zero_grad()
100    loss.backward()
101    optimizer.step()
102    if (epoch + 1) % 100 == 0:
103        print(f"Epoch [{epoch+1}/{EPOCHS}], Loss: {loss.item():.6f}")
104
105 end_time = time.time()
106 print(f"\nTraining finished in {end_time - start_time:.2f} seconds.")
107
108 # --- 7. 重建与可视化 ---
109 model.eval()
110 with torch.no_grad():
111     reconstructed_pixels_standardized = model(coords)
112
113 # 将标准化的输出逆向转换回 [0, 1] 范围以便显示
114 # 逆向操作: x_norm = x_std * std + mean
115 # 调整 mean 和 std 的形状以匹配 [N, C] 的像素列表

```

```

116 reconstructed_pixels = reconstructed_pixels_standardized * std.view(1, -1) + mean.view(1, -1)
117
118 # 重要：逆向转换后，数值可能略微超出[0,1]范围，需要裁剪
119 reconstructed_pixels.clamp_(0.0, 1.0)
120
121 reconstructed_img_tensor = reconstructed_pixels.view(H, W, 3)
122 reconstructed_img_np = reconstructed_img_tensor.cpu().numpy()
123 original_img_np = img_tensor.permute(1, 2, 0).cpu().numpy() # 原始图像依然使用[0,1]范围的张量
124
125 # 显示
126 plt.figure(figsize=(10, 5))
127 plt.subplot(1, 2, 1)
128 plt.title("Original Image")
129 plt.imshow(original_img_np)
130 plt.axis('off')
131 plt.subplot(1, 2, 2)
132 plt.title(f"Reconstructed Image (After {EPOCHS} Epochs)")
133 plt.imshow(reconstructed_img_np)
134 plt.axis('off')
135 plt.show()

```

Listing 1: MLP 重建图像

```

1 import torch
2 import torch.nn as nn
3 from torchvision.transforms import functional as TF
4 from PIL import Image
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import time
8 import os
9
10 # --- 1. 参数设置 ---
11 IMAGE_PATH = 'ye.png'
12 IMG_SIZE = 128
13 HIDDEN_DIM = 256
14 N_HIDDEN_LAYERS = 4
15 EPOCHS = 8000
16 LEARNING_RATE = 1e-4
17 DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
18
19 # 新增：位置编码参数
20 # L: 决定了编码后维度的参数。维度 = 2 * 输入维度 * L
21 # 越高的 L 能表示越高的频率
22 N_ENCODING_FUNCTIONS = 10
23
24 print(f"Using device: {DEVICE}")
25
26
27 # 新增：位置编码器模块
28 class PositionalEncoder(nn.Module):
29     def __init__(self, input_dims, num_functions):
30         super().__init__()
31         self.input_dims = input_dims
32         self.num_functions = num_functions
33
34         # 创建频率列表 [1, 2, 4, 8, ..., 2^(L-1)]
35         self.freq_bands = 2.0 ** torch.arange(num_functions)
36
37         # 计算编码后的输出维度
38         # 对于每个输入维度(x, y)，每个频率都产生sin和cos两个值
39         self.output_dims = input_dims * num_functions * 2
40
41     def forward(self, x):
42         # x shape: [N, input_dims]
43         # unsqueeze(dim=-1) -> [N, input_dims, 1]
44         # self.freq_bands -> [num_functions]
45         # x * self.freq_bands -> [N, input_dims, num_functions]
46         scaled_inputs = x.unsqueeze(-1) * self.freq_bands.to(x.device)
47
48         # 将 sin 和 cos 的结果拼接起来
49         # torch.sin(scaled_inputs) -> [N, input_dims, num_functions]
50         # torch.cos(scaled_inputs) -> [N, input_dims, num_functions]
51         # shape -> [N, input_dims, num_functions * 2]
52         encoded = torch.cat([torch.sin(scaled_inputs), torch.cos(scaled_inputs)], dim=-1)
53
54         # 展平为 [N, output_dims]
55         return torch.flatten(encoded, start_dim=1)
56
57
58 # --- 2. 加载和预处理图像 ---
59 img = Image.open(IMAGE_PATH).convert('RGB')

```

```

60 print(f"Successfully loaded image from: {IMAGE_PATH}")
61
62 img = TF.resize(img, (IMG_SIZE, IMG_SIZE))
63 img_tensor = TF.to_tensor(img).to(DEVICE)
64
65 # 标准化流程保持不变
66 mean = torch.mean(img_tensor, dim=[1, 2])
67 std = torch.std(img_tensor, dim=[1, 2])
68 std = torch.max(std, torch.tensor(1e-6).to(DEVICE))
69 img_tensor_standardized = (img_tensor - mean[:, None, None]) / std[:, None, None]
70 H, W = IMG_SIZE, IMG_SIZE
71 pixels = img_tensor_standardized.permute(1, 2, 0).view(-1, 3)
72
73 # --- 3. 创建输入数据 (坐标网格) ---
74 grid_y, grid_x = torch.meshgrid(torch.linspace(0, H - 1, H), torch.linspace(0, W - 1, W), indexing='
    ij')
75 coords = torch.stack([
76     grid_x / (W - 1) * 2 - 1,
77     grid_y / (H - 1) * 2 - 1
78 ], dim=-1).to(DEVICE)
79 coords = coords.view(-1, 2)
80
81 # 修改点 1: 对坐标进行位置编码
82 encoder = PositionalEncoder(input_dims=2, num_functions=N_ENCODING_FUNCTIONS)
83 coords_encoded = encoder(coords) # 新的、高维的坐标输入
84
85 print(f"\nOriginal coords shape: {coords.shape}")
86 print(f"Encoded coords shape: {coords_encoded.shape}") # 维度显著增加
87 print(f"Target pixels shape: {pixels.shape}")
88
89
90 # --- 4. 构建 MLP 模型 ---
91 class MLPImageReconstructorLinear(nn.Module):
92     def __init__(self, in_features, hidden_features, hidden_layers, out_features):
93         super().__init__()
94         layers = []
95         layers.append(nn.Linear(in_features, hidden_features))
96         layers.append(nn.ReLU())
97         for _ in range(hidden_layers):
98             layers.append(nn.Linear(hidden_features, hidden_features))
99             layers.append(nn.ReLU())
100         layers.append(nn.Linear(hidden_features, out_features))
101         self.net = nn.Sequential(*layers)
102
103     def forward(self, x):
104         return self.net(x)
105
106 # 修改点 2: 使用编码后的维度作为模型输入
107 model = MLPImageReconstructorLinear(
108     in_features=encoder.output_dims, # <-- 使用编码后的维度
109     hidden_features=HIDDEN_DIM,
110     hidden_layers=N_HIDDEN_LAYERS,
111     out_features=3
112 ).to(DEVICE)
113
114 print("\nModel Architecture:")
115 print(model)
116
117 # --- 5. 定义损失函数和优化器 ---

```



```

118 loss_fn = nn.MSELoss()
119 optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
120
121 # --- 6. 训练模型 ---
122 print("\nStarting training...")
123 start_time = time.time()
124 for epoch in range(EPOCHS):
125     # 修改点 3: 使用编码后的坐标进行训练
126     predicted_pixels = model(coords_encoded) # <-- 使用编码后的坐标
127
128     loss = loss_fn(predicted_pixels, pixels)
129     optimizer.zero_grad()
130     loss.backward()
131     optimizer.step()
132     if (epoch + 1) % 100 == 0:
133         print(f"Epoch [{epoch+1}/{EPOCHS}], Loss: {loss.item():.6f}")
134
135 end_time = time.time()
136 print(f"\nTraining finished in {end_time - start_time:.2f} seconds.")
137
138 # --- 7. 重建与可视化 ---
139 model.eval()
140 with torch.no_grad():
141     # 重建时同样需要对坐标进行编码
142     reconstructed_pixels_standardized = model(coords_encoded)
143
144 # 逆向标准化流程保持不变
145 reconstructed_pixels = reconstructed_pixels_standardized * std.view(1, -1) + mean.view(1, -1)
146 reconstructed_pixels.clamp_(0.0, 1.0)
147 reconstructed_img_tensor = reconstructed_pixels.view(H, W, 3)
148 reconstructed_img_np = reconstructed_img_tensor.cpu().numpy()
149 original_img_np = img_tensor.permute(1, 2, 0).cpu().numpy()
150
151 # 显示
152 plt.figure(figsize=(10, 5))
153 plt.subplot(1, 2, 1)
154 plt.title("Original Image")
155 plt.imshow(original_img_np)
156 plt.axis('off')
157 plt.subplot(1, 2, 2)
158 plt.title(f"Reconstructed Image (After {EPOCHS} Epochs)")
159 plt.imshow(reconstructed_img_np)
160 plt.axis('off')
161 plt.show()

```

Listing 2: MLP 位置编码后重建图像