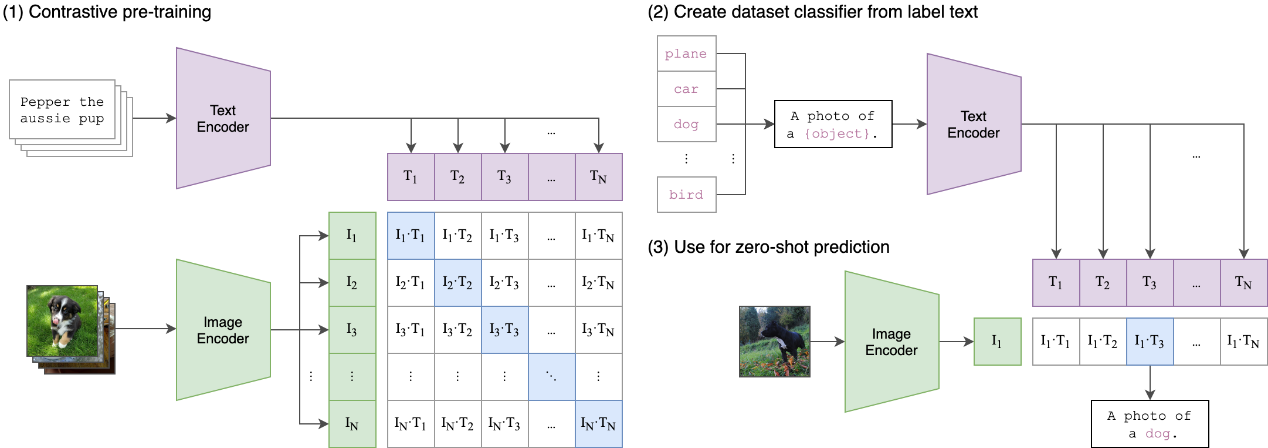
# 基础知识

1. 
   1. 左图为训练过程
      1. Clip使用图像和文本Encoder，将一组图片编码，Ii代表一个图片的编码向量；将一组文本编码，Tj代表一个文本的编码向量
      2. 随后使用对比学习的方法进行训练，目标函数是交叉熵函数
      3. 即使与图片编码匹配的文本编码的距离最小，而不匹配的距离最大
   2. 右图为零样本分类过程
      1. 使用Encoder将一张图片编码；将一组类别单词添加相同的prompt构成一组文本，并编码
      2. 计算图片编码与每一个文本编码的距离，距离最小的就是最有可能的类别
2. CLIP的精妙之处在于
   1. 用图片匹配一个句子而不是一个单词，这也使得训练数据量提升
   2. 使用了对比学习的方法
3. CoOp
   1. CoOp提供了可训练的prompt，相当于在原有的class embedding之前或之后添加了可训练的prompt embedding，实际上prompt embedding的训练结果往往无法解释

# 代码组织

此部分介绍CLIP仓库 (https://github.com/openai/CLIP)中代码的组织，代码主体部分为clip文件夹下的clip.py和model.py

## ./clip/model.py

1. 文本和图像的Encoder均使用了transformer结构，transformer模型定义位于./clip/model.py中，即Transformer类，在对图像和文本进行编码时会多次使用此模型
2. CLIP模型也在./clip/model.py中，即CLIP类，类中的encode\_image和encode\_text为图像文本编码函数。CLIP模型用于对文本和图像进行编码，并计算二者的相似度，从而实现分类

## ./clip/clip.py

此代码为clip实现零样本分类的主函数

1. \_transform函数为图像预处理函数
2. tokenize函数为文本预处理函数：将句子padding到77长度
3. 调用model中的encode\_image和encode\_text将图片和文本分别编码为(1, 512), (3, 512)
4. 调用model计算图片和文本的相似度

## zero-shot图片分类代码逻辑

python

>>> import torch

>>> from PIL import Image

>>> device = "cuda"

>>> model, preprocess = clip.load(“RN50", device=device)

>>> image = preprocess(Image.open("CLIP.png")).unsqueeze(0).to(device)

>>> text = clip.tokenize(["a diagram", "a dog", "a cat"]).to(device)

>>> with torch.no\_grad():

... image\_features = model.encode\_image(image)

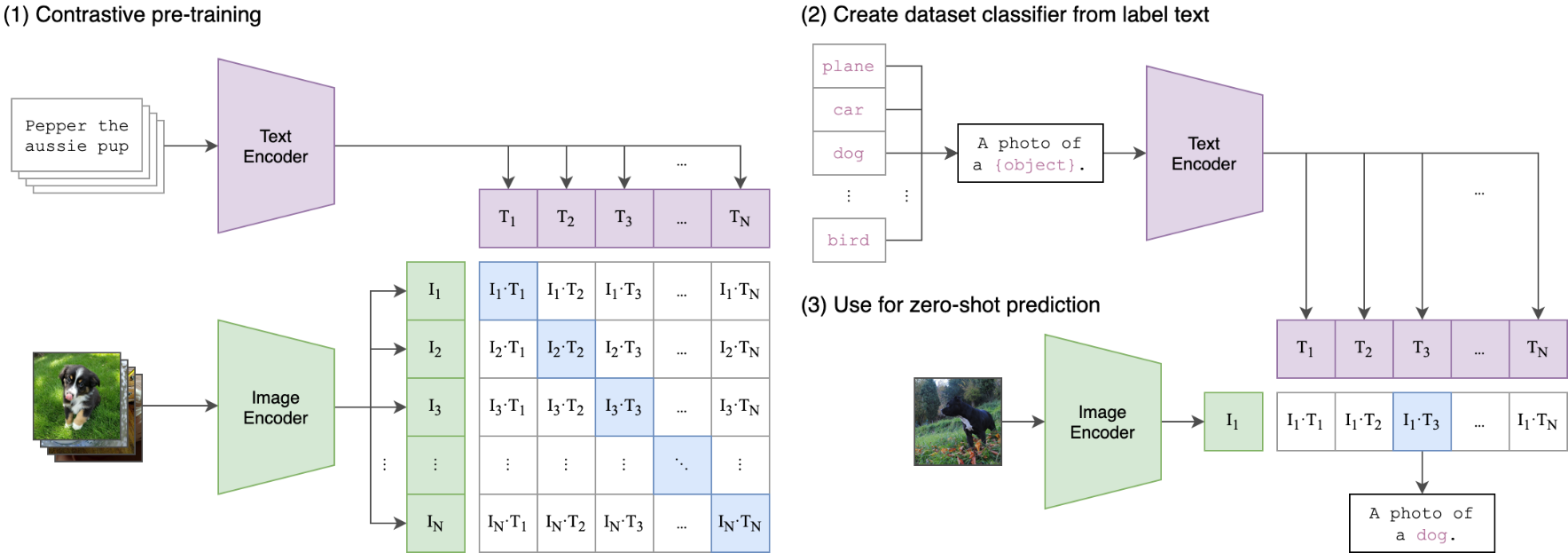
... text\_features = model.encode\_text(text)

... logits\_per\_image, logits\_per\_text = model(image, text)

... probs = logits\_per\_image.softmax(dim=-1).cpu().numpy()

...

>>> print("Label probs:", probs) # prints: [[0.9927937 0.00421068 0.00299572]]



1. 使用clip.load加载预训练的CLIP模型
2. 使用preprocess函数对图像进行预处理
3. 使用clip.tokenize函数对文本进行预处理，分词并将词汇转化为词汇表的id
4. 使用model（之前导入的预训练CLIP模型）对预处理后的图像和文本进行推理，前向传播的过程中会先编码图像和文本，再通过transformer编码位置信息，最后再计算图像和文本的对数概率
5. 使用softmax将logits转换为和为1的概率
6. 接着可以通过np.argmax找到最高概率的类别索引，进而实现了zero-shot分类

# 实验环境

1. GPU：RTX3080
2. 系统：Linux

# 实验内容

## 数据集划分

1. 数据集：caltech-101数据集
2. 划分：
   1. 训练集：前30%的图片
   2. 测试集：后70%的图片

## CoOp: Context Optimization的实现

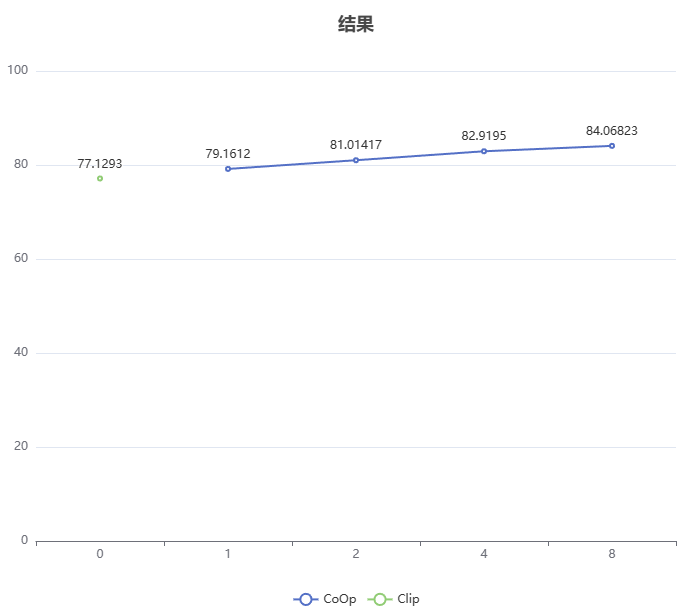
1. 初始化prompt embedding，并将其设为可训练的参数。其中M为embedding数量，end表示分类词在embedding后，prompt embedding初始化为均值为0方差为0.02的高斯分布
2. self.M = 16 # embedding数量
3. self.location = 'end' # 类别的位置
4. source\_prompt = torch.randn(1, self.M, 512) # 初始化prompt
5. for i in range(self.M):
6. source\_prompt[0, i, :].normal\_(mean=0, std=0.02)
7. self.prompt = nn.Parameter(source\_prompt) # 设置为可训练的参数
8. self.register\_parameter('prompt', self.prompt)
9. 取出分类词的start embedding，并将prompt embedding插入到start embedding和分类词embedding之间
10. if self.location == 'end':
11. start = x[:, :1, :]
12. text\_class = x[:, 1:77-self.M, :]
13. prompt = torch.cat([self.prompt] \* 101, dim=0).type(self.dtype)
14. text\_class = torch.cat((prompt, text\_class), dim=1)
15. x = torch.cat((start, text\_class), dim=1)
16. 训练使用SGD优化算法，初始学习率0.002，使用余弦退火以及warmup技巧（第一个epoch学习率为1e-5）
17. optimizer = optim.SGD(model.parameters(), lr=0.002) # 初始学习率
18. # optimizer = optim.Adam(model.parameters(), lr=0.02)
19. scheduler = CosineAnnealingLR(optimizer, epochs) # 余弦退火
20. warmup\_epochs = 1
21. warmup\_lr = 1e-5 # warmup技巧
22. warmup\_optimizer = optim.SGD(model.parameters(), lr=warmup\_lr)
23. for epoch in range(epochs):
24. if epoch < warmup\_epochs:
25. print("warmup")
26. logits\_per\_image, \_ = model(images, text)
27. probs = logits\_per\_image.softmax(dim=-1)
28. criterion = nn.CrossEntropyLoss()
29. loss = criterion(probs, targets)
30. warmup\_optimizer.zero\_grad()
32. loss.backward()
33. warmup\_optimizer.step()
34. else:
35. logits\_per\_image, \_ = model(images, text)
36. # probs = logits\_per\_image.softmax(dim=-1)
37. criterion = nn.CrossEntropyLoss()
38. loss = criterion(logits\_per\_image, targets)
39. optimizer.zero\_grad()
41. loss.backward()
42. optimizer.step()
43. scheduler.step() # 下一个学习率
45. if epoch%10 == 0:
46. print(f"epoch: {epoch}, loss: {loss}")

# 实验结果

## zero-shot图片分类

结果：success rate: 0.7712931175412784

## CoOp结果



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **1 shot** | **2 shot** | **4 shot** | **8 shot** |
| **seed 10** | 78.4513 | 81.4612 | 83.4161 | 84.1235 |
| **seed 20** | 79.8645 | 81.4165 | 83.2156 | 83.8256 |
| **seed 30** | 79.1678 | 80.1648 | 82.1268 | 84.2556 |

（embedding数目均为16，class token位于可训练embedding之后，训练epochs为200）

# 结果分析

CoOp主要的思想是自动设计提示文本，先保持预训练参数不变，然后利用少量数据去学习合适的 prompt，这样的 prompt 比人工设计的提示文本在测试时更有效，进而分类准确率高于zero-shot

最佳模型的分类成功率：0.8425564321254674