**项目文档**

**项目简介**

**本项目基于 PyTorch 实现了一个端到端的图像文本识别（OCR）系统，采用了编码器-解码器（Encoder-Decoder）结构，结合了自定义的图像特征提取（如 Swin Transformer）、位置编码、束搜索（Beam Search）等技术。项目支持训练、推理和评估，能够对输入图片进行文本识别，并输出预测文本及相关性能指标（如 CER、准确率）。**

**主要代码结构与功能说明**

**1. 环境与依赖导入**

**import sys**

**sys.path.append("Encorder")**

**sys.path.append("Decorder")**

**sys.path.append("Datasets")**

**sys.path.append("数据集")**

**from torch import nn**

**import torch**

**from Encorder.Encoder import encoder**

**from Decorder.Decoder import decoder**

**from img\_drop\_embding import get\_peach**

**from torch.utils.data import Dataset, DataLoader**

**from PIL import Image**

**from torchvision import datasets, transforms**

**device='cuda' if torch.cuda.is\_available() else 'cpu'**

**from output import Generator**

**import numpy as np**

**from all\_dataset import custom\_dataset,test\_imgs,test\_labels**

**from pos\_embding import get\_positional\_encoding**

**from torch.optim.lr\_scheduler import StepLR**

**import random**

**from swin import swin**

**swin\_model=swin**

**作用说明：**

* **设置模块搜索路径，导入项目自定义模块和第三方库。**
* **配置设备（GPU/CPU）。**
* **加载编码器、解码器、特征提取、数据集、生成器等核心组件。**

**2. 束搜索（Beam Search）解码**

**def beam\_search(encoder\_decoder, test\_imgs, start\_token, end\_token, beam\_size=3, max\_length=20):**

**# ... 省略 ...**

**return best\_sequence, best\_word\_scores**

**作用说明：  
实现束搜索算法，用于推理阶段生成最优的输出序列（文本），提升解码质量。**

**3. 重复惩罚机制**

**def apply\_repetition\_penalty(logits, generated\_seq, penalty=1.2):**

**for token in set(generated\_seq):**

**logits[:, token] /= penalty**

**return logits**

**作用说明：  
对已生成的 token 进行概率惩罚，减少重复生成，提升生成文本的多样性和准确性。**

**4. 随机种子设置**

**seed=3407**

**torch.manual\_seed(seed)**

**torch.cuda.manual\_seed\_all(seed)**

**np.random.seed(seed)**

**random.seed(seed)**

**作用说明：  
保证实验的可复现性。**

**5. 字符错误率（CER）计算**

**def edit\_distance(str1, str2):**

**# ... 省略 ...**

**return dp[m][n]**

**作用说明：  
实现编辑距离算法，用于评估模型输出与真实标签的差异，计算 CER。**

**6. 编码器-解码器模型定义**

**class EncoderDecoder(nn.Module):**

**def \_\_init\_\_(self, encoder,decoder,source\_emb,target\_emb,generator):**

**# ... 省略 ...**

**def forward(self, src, tgt, type='train'):**

**# ... 省略 ...**

**def encode(self, src,type='train'):**

**# ... 省略 ...**

**def decode(self,src,tgt,src\_mask):**

**作用说明：  
定义整体模型结构，集成编码器、解码器、嵌入层、生成器等模块，支持训练和推理。**

**7. 模型初始化与优化器设置**

**vocab\_size=498**

**d\_model=768**

**Embd=nn.Embedding(vocab\_size,d\_model)**

**generator=Generator(d\_model=d\_model,vocab\_size=vocab\_size).to(device)**

**encoder\_decoder=EncoderDecoder(encoder,decoder,get\_peach,Embd,generator).to(device)**

**loss=nn.CrossEntropyLoss(ignore\_index=129)**

**optimizer=torch.optim.AdamW(encoder\_decoder.parameters(),lr=3e-5, betas=(0.9, 0.999),eps=1e-9,weight\_decay=1e-5)**

**scheduler = StepLR(optimizer, step\_size=30, gamma=0.8)**

**for p in encoder\_decoder.parameters():**

**if p.dim() > 1:**

**nn.init.xavier\_uniform\_(p)**

**作用说明：  
设置模型参数、损失函数、优化器和学习率调度器，并对参数进行初始化。**

**8. 数据增强**

**train\_transform = transforms.Compose([**

**transforms.ToTensor(),**

**transforms.Normalize(mean=[0.5, 0.5, 0.5,0.5], std=[0.5, 0.5, 0.5,0.5])**

**])**

**transform = transforms.Compose([**

**transforms.ToTensor(),**

**transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])**

**])**

**作用说明：  
对训练集和测试集图片进行归一化等预处理，提升模型泛化能力。**

**9. 训练与评估主循环**

**data\_loader = DataLoader(custom\_dataset, batch\_size=16, shuffle=True)**

**for epoch in range(600):**

**# 训练过程**

**for x,y,y1 in data\_loader:**

**# ... 省略 ...**

**# 每5个epoch评估一次**

**if epoch % 5 == 0:**

**with torch.no\_grad():**

**for x, y in zip(test\_img, test\_label):**

**# ... 省略 ...**

**# 更新最佳性能指标**

**# 打印和保存结果**

**作用说明：**

* **训练模型，计算损失并反向传播。**
* **定期在测试集上评估模型，输出 CER 和准确率，保存最佳结果。**