

人工智能12.25 上机课限时作业

by 基础医学院-顾桂榕

学号：2510305235

一.基本任务

code(已补全)

```
import os
import time
import math
import warnings

import numpy as np
import cv2
from PIL import Image

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torch.cuda.amp import autocast, GradScaler

from functools import partial

IMAGENET_MEAN = np.array([0.485, 0.456, 0.406], dtype=np.float32)
IMAGENET_STD = np.array([0.229, 0.224, 0.225], dtype=np.float32)

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

# 路径定义
DATASET_PATH = "/bohr/ai-biomedicine-course-5e8v/v1/MoNuSeg"
DINOVI_T_PATH = "/bohr/ai-biomedicine-course-5e8v/v1/dino_deit_small8_pretrain.pth"
CKPT_SAVE_PATH = "/personal/seg_checkpoints/"
RES_SAVE_PATH = "/personal/seg_results/"
```

超参数定义

TRAIN_IMG_SIZE = 256

PATCH_SIZE = 8

EMBEDED_DIM = 384

BATCH_SIZE = 4

EPOCH_NUM = 200

LEARNING_RATE = 1e-4

WEIGHT_DECAY = 1e-4

BCE_WEIGHT = 1.0

DICE_WEIGHT = 1.0

VALID_INTERVAL = 5

数据集类定义

class NucleiTilesDataset(Dataset):

def __init__(self, root_dir, split="train", train_img_size=256):

self.root_dir = root_dir

self.split = split # train / valid / test

self.train_img_size = train_img_size

self.img_dir = os.path.join(root_dir, split, "images")

if split != 'test':

self.ann_dir = os.path.join(root_dir, split, "masks")

exts = (".png", ".jpg")

self.img_paths = [os.path.join(self.img_dir, f) for f in
sorted(os.listdir(self.img_dir))

if f.lower().endswith(exts)]

if split != 'test':

self.ann_paths = [os.path.join(self.ann_dir, f) for f in
sorted(os.listdir(self.ann_dir))

if f.lower().endswith(exts)]

if len(self.img_paths) == 0:

raise RuntimeError(f"No images found in {self.img_dir}")

def __len__(self):

return len(self.img_paths)

def __getitem__(self, idx):

img_path = self.img_paths[idx]

```

        # 读取图像
        img = np.array(Image.open(img_path).convert("RGB")) #
(H,W,3)
        img = img.astype(np.float32) / 255.0 # 归一化到[0,1]
        img = (img - IMAGENET_MEAN) / IMAGENET_STD # 标准化
        img = torch.from_numpy(img).permute(2, 0, 1).float() #
(C,H,W)

        if self.split != 'test':
            ann_path = self.ann_paths[idx]
            ann = np.array(Image.open(ann_path).convert("L")) #
single-channel 0/255
            mask = (ann > 127).astype(np.uint8) # 0/1
            mask = torch.from_numpy(mask.astype(np.float32))[None,
...] # (1,H,W)

            return {
                "image": img,
                "mask": mask,
                "img_name": os.path.basename(img_path)
            }
        else:
            return {
                "image": img,
                "img_name": os.path.basename(img_path)
            }

# ===== ViT模型及相关组件定义 =====
# 这部分保持不变
def _no_grad_trunc_normal_(tensor, mean, std, a, b):
    def norm_cdf(x):
        return (1. + math.erf(x / math.sqrt(2.))) / 2.

    if (mean < a - 2 * std) or (mean > b + 2 * std):
        warnings.warn("mean is more than 2 std from [a, b] in
nn.init.trunc_normal_. "
                    "The distribution of values may be
incorrect.",
                    stacklevel=2)

```

```

with torch.no_grad():
    l = norm_cdf((a - mean) / std)
    u = norm_cdf((b - mean) / std)
    tensor.uniform_(2 * l - 1, 2 * u - 1)
    tensor.erfinv_()
    tensor.mul_(std * math.sqrt(2.))
    tensor.add_(mean)
    tensor.clamp_(min=a, max=b)
    return tensor

def trunc_normal_(tensor, mean=0., std=1., a=-2., b=2.):
    return _no_grad_trunc_normal_(tensor, mean, std, a, b)

def drop_path(x, drop_prob: float = 0., training: bool = False):
    if drop_prob == 0. or not training:
        return x
    keep_prob = 1 - drop_prob
    shape = (x.shape[0],) + (1,) * (x.ndim - 1)
    random_tensor = keep_prob + torch.rand(shape, dtype=x.dtype,
device=x.device)
    random_tensor.floor_()
    output = x.div(keep_prob) * random_tensor
    return output

class DropPath(nn.Module):
    def __init__(self, drop_prob=None):
        super(DropPath, self).__init__()
        self.drop_prob = drop_prob

    def forward(self, x):
        return drop_path(x, self.drop_prob, self.training)

class Mlp(nn.Module):
    def __init__(self, in_features, hidden_features=None,
out_features=None, act_layer=nn.GELU, drop=0.):
        super().__init__()
        out_features = out_features or in_features
        hidden_features = hidden_features or in_features
        self.fc1 = nn.Linear(in_features, hidden_features)
        self.act = act_layer()
        self.fc2 = nn.Linear(hidden_features, out_features)

```

```

        self.drop = nn.Dropout(drop)

    def forward(self, x):
        x = self.fc1(x)
        x = self.act(x)
        x = self.drop(x)
        x = self.fc2(x)
        x = self.drop(x)
        return x

class Attention(nn.Module):
    def __init__(self, dim, num_heads=8, qkv_bias=False,
qk_scale=None, attn_drop=0., proj_drop=0.):
        super().__init__()
        self.num_heads = num_heads
        head_dim = dim // num_heads
        self.scale = qk_scale or head_dim ** -0.5
        self.qkv = nn.Linear(dim, dim * 3, bias=qkv_bias)
        self.attn_drop = nn.Dropout(attn_drop)
        self.proj = nn.Linear(dim, dim)
        self.proj_drop = nn.Dropout(proj_drop)

    def forward(self, x):
        B, N, C = x.shape
        qkv = self.qkv(x).reshape(B, N, 3, self.num_heads, C //
self.num_heads).permute(2, 0, 3, 1, 4)
        q, k, v = qkv[0], qkv[1], qkv[2]
        attn = (q @ k.transpose(-2, -1)) * self.scale
        attn = attn.softmax(dim=-1)
        attn = self.attn_drop(attn)
        x = (attn @ v).transpose(1, 2).reshape(B, N, C)
        x = self.proj(x)
        x = self.proj_drop(x)
        return x, attn

class Block(nn.Module):
    def __init__(self, dim, num_heads, mlp_ratio=4., qkv_bias=False,
qk_scale=None, drop=0., attn_drop=0.,
drop_path=0., act_layer=nn.GELU,
norm_layer=nn.LayerNorm):
        super().__init__()

```

```

        self.norm1 = norm_layer(dim)
        self.attn = Attention(dim, num_heads=num_heads,
                               qkv_bias=qkv_bias, qk_scale=qk_scale,
                               attn_drop=attn_drop, proj_drop=drop)
        self.drop_path = DropPath(drop_path) if drop_path > 0. else
nn.Identity()
        self.norm2 = norm_layer(dim)
        mlp_hidden_dim = int(dim * mlp_ratio)
        self.mlp = Mlp(in_features=dim,
                        hidden_features=mlp_hidden_dim,
                        act_layer=act_layer, drop=drop)

```

```

def forward(self, x, return_attention=False):
    y, attn = self.attn(self.norm1(x))
    if return_attention:
        return attn
    x = x + self.drop_path(y)
    x = x + self.drop_path(self.mlp(self.norm2(x)))
    return x

```

```

class PatchEmbed(nn.Module):
    def __init__(self, img_size=224, patch_size=16, in_chans=3,
embed_dim=768):
        super().__init__()
        num_patches = (img_size // patch_size) * (img_size //
patch_size)
        self.img_size = img_size
        self.patch_size = patch_size
        self.num_patches = num_patches
        self.proj = nn.Conv2d(in_chans, embed_dim,
kernel_size=patch_size, stride=patch_size)

```

```

def forward(self, x):
    B, C, H, W = x.shape
    x = self.proj(x).flatten(2).transpose(1, 2)
    return x

```

```

class VisionTransformer(nn.Module):
    def __init__(self, img_size=[224], patch_size=16, in_chans=3,
num_classes=0, embed_dim=768, depth=12,
                num_heads=12, mlp_ratio=4., qkv_bias=False,

```

```

qk_scale=None, drop_rate=0., attn_drop_rate=0.,
                    drop_path_rate=0., norm_layer=nn.LayerNorm,
**kwargs):
    super().__init__()
    self.num_features = self.embed_dim = embed_dim
    self.patch_embed = PatchEmbed(img_size=img_size[0],
                                patch_size=patch_size,
                                in_chans=in_chans,
                                embed_dim=embed_dim)
    num_patches = self.patch_embed.num_patches
    self.cls_token = nn.Parameter(torch.zeros(1, 1, embed_dim))
    self.pos_embed = nn.Parameter(torch.zeros(1, num_patches +
1, embed_dim))
    self.pos_drop = nn.Dropout(p=drop_rate)
    dpr = [x.item() for x in torch.linspace(0, drop_path_rate,
depth)]
    self.blocks = nn.ModuleList([
        Block(dim=embed_dim, num_heads=num_heads,
mlp_ratio=mlp_ratio,
            qkv_bias=qkv_bias, qk_scale=qk_scale,
drop=drop_rate,
            attn_drop=attn_drop_rate, drop_path=dpr[i],
norm_layer=norm_layer)
        for i in range(depth)])
    self.norm = norm_layer(embed_dim)
    self.head = nn.Linear(embed_dim, num_classes) if num_classes
> 0 else nn.Identity()
    trunc_normal_(self.pos_embed, std=.02)
    trunc_normal_(self.cls_token, std=.02)
    self.apply(self._init_weights)

def _init_weights(self, m):
    if isinstance(m, nn.Linear):
        trunc_normal_(m.weight, std=.02)
        if isinstance(m, nn.Linear) and m.bias is not None:
            nn.init.constant_(m.bias, 0)
    elif isinstance(m, nn.LayerNorm):
        nn.init.constant_(m.bias, 0)
        nn.init.constant_(m.weight, 1.0)

def interpolate_pos_encoding(self, x, w, h):

```

```

    npatch = x.shape[1] - 1
    N = self.pos_embed.shape[1] - 1
    if npatch == N and w == h:
        return self.pos_embed
    class_pos_embed = self.pos_embed[:, 0]
    patch_pos_embed = self.pos_embed[:, 1:]
    dim = x.shape[-1]
    w0 = w // self.patch_embed.patch_size
    h0 = h // self.patch_embed.patch_size
    w0, h0 = w0 + 0.1, h0 + 0.1
    patch_pos_embed = nn.functional.interpolate(
        patch_pos_embed.reshape(1, int(math.sqrt(N)),
int(math.sqrt(N)), dim).permute(0, 3, 1, 2),
        scale_factor=(w0 / math.sqrt(N), h0 / math.sqrt(N)),
mode='bicubic')
    assert int(w0) == patch_pos_embed.shape[-2] and int(h0) ==
patch_pos_embed.shape[-1]
    patch_pos_embed = patch_pos_embed.permute(0, 2, 3,
1).view(1, -1, dim)
    return torch.cat((class_pos_embed.unsqueeze(0),
patch_pos_embed), dim=1)

def prepare_tokens(self, x):
    B, nc, w, h = x.shape
    x = self.patch_embed(x)
    cls_tokens = self.cls_token.expand(B, -1, -1)
    x = torch.cat((cls_tokens, x), dim=1)
    x = x + self.interpolate_pos_encoding(x, w, h)
    return self.pos_drop(x)

def forward(self, x):
    x = self.prepare_tokens(x)
    for blk in self.blocks:
        x = blk(x)
    x = self.norm(x)
    return x[:, 0]

def get_last_selfattention(self, x):
    x = self.prepare_tokens(x)
    for i, blk in enumerate(self.blocks):
        if i < len(self.blocks) - 1:

```

```

        x = blk(x)
    else:
        return blk(x, return_attention=True)

def get_intermediate_layers(self, x, n=1):
    x = self.prepare_tokens(x)
    output = []
    for i, blk in enumerate(self.blocks):
        x = blk(x)
        if len(self.blocks) - i <= n:
            output.append(self.norm(x))
    return output

def vit_small(patch_size=16, **kwargs):
    model = VisionTransformer(
        patch_size=patch_size, embed_dim=384, depth=12, num_heads=6,
        mlp_ratio=4,
        qkv_bias=True, norm_layer=partial(nn.LayerNorm, eps=1e-6),
        **kwargs)
    return model

# ===== 特征提取 =====
def extract_tokens(ckpt_path, model_name, patch_size, device, x):
    model = model_name(patch_size=patch_size).to(device)
    ckpt = torch.load(ckpt_path, map_location=device)
    model.load_state_dict(ckpt)
    model.eval()

    feats = {}
    def hook_fn(module, inp, out):
        feats["tokens"] = out

    h = model.norm.register_forward_hook(hook_fn)
    with torch.no_grad():
        out = model(x)
    h.remove()

    tokens = feats["tokens"][:, 1:, :] # 去掉 class token
    del model
    torch.cuda.empty_cache()

```

```
return tokens
```

```
# ===== 分割解码器 =====
```

```
class SegDecoder(nn.Module):
```

```
    def __init__(self, embed_dim=384, patch_size=8, H=256, W=256):
```

```
        super().__init__()
```

```
        self.embed_dim = embed_dim
```

```
        self.patch_size = patch_size
```

```
        self.H = H
```

```
        self.W = W
```

```
# 解码器: 从32x32上采样到256x256
```

```
self.decoder = nn.Sequential(
```

```
    nn.Conv2d(embed_dim, 256, 3, padding=1),
```

```
    nn.ReLU(inplace=True),
```

```
    nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2),
```

```
# 32->64
```

```
    nn.ReLU(inplace=True),
```

```
    nn.Conv2d(128, 128, 3, padding=1),
```

```
    nn.ReLU(inplace=True),
```

```
    nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2), #
```

```
64->128
```

```
    nn.ReLU(inplace=True),
```

```
    nn.Conv2d(64, 64, 3, padding=1),
```

```
    nn.ReLU(inplace=True),
```

```
    nn.ConvTranspose2d(64, 32, kernel_size=2, stride=2), #
```

```
128->256
```

```
    nn.ReLU(inplace=True),
```

```
    nn.Conv2d(32, 32, 3, padding=1),
```

```
    nn.ReLU(inplace=True),
```

```
)
```

```
self.head = nn.Conv2d(32, 1, kernel_size=1)
```

```
def forward(self, tokens):
```

```
    B, N, C = tokens.shape
```

```
    grid_size = int(N ** 0.5) # 应该是32
```

```
# 重塑为特征图
```

```

        feat = tokens.transpose(1, 2).contiguous().view(B,
self.embed_dim, grid_size, grid_size)

        # 通过解码器
        y = self.decoder(feat)
        y = self.head(y)

        # 确保输出尺寸正确
        if y.shape[-2:] != (self.H, self.W):
            y = F.interpolate(y, size=(self.H, self.W),
mode="bilinear", align_corners=False)

        return y

# ===== 损失函数和评估指标 =====
def dice_loss(logits, targets, eps=1e-6):
    probs = torch.sigmoid(logits)
    num = 2.0 * (probs * targets).sum(dim=(2,3))
    den = (probs + targets).sum(dim=(2,3)) + eps
    return 1.0 - (num / den).mean()

@torch.no_grad()
def compute_iou_dice(logits, targets, thr=0.5, eps=1e-6):
    probs = torch.sigmoid(logits)
    pred = (probs > thr).float()

    # 计算TP, FP, FN
    tp = (pred * targets).sum(dim=(1, 2, 3))
    fp = (pred * (1 - targets)).sum(dim=(1, 2, 3))
    fn = ((1 - pred) * targets).sum(dim=(1, 2, 3))

    # 计算IoU和Dice
    iou = tp / (tp + fp + fn + eps)
    dice = (2 * tp) / (2 * tp + fp + fn + eps)

    # 对整个batch求平均
    iou = iou.mean()
    dice = dice.mean()

    return iou, dice

```

```

# ===== 训练函数 =====
def train_one_epoch(decoder, loader, optimizer, dino_ckpt_path,
device, patch_size=8,
                    bce_weight=1.0, dice_weight=0.5):
    decoder.train()
    bce = nn.BCEWithLogitsLoss()
    scaler = GradScaler() # 梯度缩放器

    loss_sum = 0.0
    iou_sum = 0.0
    dice_sum = 0.0
    n = 0

    for item in loader:
        # 获取数据
        img = item["image"].to(device)
        mask = item["mask"].to(device)

        optimizer.zero_grad(set_to_none=True)

        # 混合精度训练
        with autocast():
            # 提取tokens
            with torch.no_grad():
                tokens = extract_tokens(dino_ckpt_path, vit_small,
patch_size, device, img)

            # 计算logits
            logits = decoder(tokens)

            # 计算损失
            loss = bce_weight * bce(logits, mask) + dice_weight *
dice_loss(logits, mask)

        # 反向传播
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        # 计算IoU和Dice系数
        with torch.no_grad():

```

```

        iou, d = compute_iou_dice(logits, mask)

    bs = img.size(0)
    loss_sum += loss.item() * bs
    iou_sum += iou * bs
    dice_sum += d * bs
    n += bs

    # 清理缓存
    torch.cuda.empty_cache()

    return loss_sum / max(n,1), iou_sum / max(n,1), dice_sum /
max(n,1)

# ===== 评估函数 =====
@torch.no_grad()
def evaluate(decoder, loader, dino_ckpt_path, device, patch_size=8,
bce_weight=1.0, dice_weight=0.5):
    decoder.eval()
    bce = nn.BCEWithLogitsLoss()

    loss_sum = 0.0
    iou_sum = 0.0
    dice_sum = 0.0
    n = 0

    for item in loader:
        img = item["image"].to(device)
        mask = item["mask"].to(device)

        # 提取tokens
        tokens = extract_tokens(dino_ckpt_path, vit_small,
patch_size, device, img)

        # 计算logits
        logits = decoder(tokens)

        # 计算损失
        loss = bce_weight * bce(logits, mask) + dice_weight *
dice_loss(logits, mask)

```

```

# 计算IoU和Dice
iou, d = compute_iou_dice(logits, mask)

bs = img.size(0)
loss_sum += loss.item() * bs
iou_sum += iou * bs
dice_sum += d * bs
n += bs

return loss_sum / max(n,1), iou_sum / max(n,1), dice_sum /
max(n,1)

# ===== 测试函数 =====
@torch.no_grad()
def test(decoder, loader, dino_ckpt_path, device, patch_size,
save_dir):
    decoder.eval()
    os.makedirs(save_dir, exist_ok=True)

    for item in loader:
        img = item["image"].to(device)
        name = item["img_name"]

        # 提取tokens
        tokens = extract_tokens(dino_ckpt_path, vit_small,
patch_size, device, img)

        # 计算logits
        logits = decoder(tokens)

        # 转换为概率并二值化
        probs = torch.sigmoid(logits)
        pred_mask = (probs > 0.5).float()

        # 保存预测结果
        pred_np = pred_mask.squeeze().cpu().numpy().astype(np.uint8)
* 255
        for i in range(pred_np.shape[0]):
            # 保存为要求的格式: 输入图像文件名_mask.jpg
            base_name = os.path.splitext(name[i])[0]
            save_name = base_name + "_mask.jpg"

```

```

        cv2.imwrite(os.path.join(save_dir, save_name),
pred_np[i])

# ===== 主程序 =====
print("[INFO] device:", DEVICE)
print("[INFO] loading DINO backbone...")
print(f"[INFO] backbone patch_size={PATCH_SIZE}, embed_dim=
{EMBEDED_DIM}")

os.makedirs(CKPT_SAVE_PATH, exist_ok=True)

# 创建模型
decoder = SegDecoder(embed_dim=EMBEDED_DIM, patch_size=PATCH_SIZE,
H=TRAIN_IMG_SIZE, W=TRAIN_IMG_SIZE).to(DEVICE)

# 数据集和数据加载器
train_ds = NucleiTilesDataset(DATASET_PATH, split="train",
train_img_size=TRAIN_IMG_SIZE)
valid_ds = NucleiTilesDataset(DATASET_PATH, split="valid",
train_img_size=TRAIN_IMG_SIZE)

train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE,
shuffle=True, num_workers=2)
valid_loader = DataLoader(valid_ds, batch_size=BATCH_SIZE,
shuffle=False, num_workers=2)

# 优化器
params = [p for p in decoder.parameters() if p.requires_grad]
optimizer = torch.optim.Adamw(params, lr=LEARNING_RATE,
weight_decay=WEIGHT_DECAY)

best_dice = -1.0
best_path = os.path.join(CKPT_SAVE_PATH, "best.pt")
last_path = os.path.join(CKPT_SAVE_PATH, "last.pt")

print("[INFO] start training...")
for ep in range(1, EPOCH_NUM + 1):
    t0 = time.time()
    tr_loss, tr_iou, tr_dice = train_one_epoch(
        decoder, train_loader, optimizer, DINOVI_TCKPT_PATH, DEVICE,
        patch_size=PATCH_SIZE, bce_weight=BCE_WEIGHT,

```

```

dice_weight=DICE_WEIGHT,
)

if ep % VALID_INTERVAL == 0:
    te_loss, te_iou, te_dice = evaluate(
        decoder, valid_loader, DINOVIK_CKPT_PATH, DEVICE,
        patch_size=PATCH_SIZE, bce_weight=BCE_WEIGHT,
dice_weight=DICE_WEIGHT
    )
    dt = time.time() - t0

    print(f"[Epoch {ep:03d}/{EPOCH_NUM}] "
          f"train: loss={tr_loss:.4f} iou={tr_iou:.4f} dice=
{tr_dice:.4f} | "
          f"valid: loss={te_loss:.4f} iou={te_iou:.4f} dice=
{te_dice:.4f} | "
          f"time={dt:.1f}s")

# 保存最后一个检查点
torch.save({
    "epoch": ep,
    "model": decoder.state_dict(),
    "optimizer": optimizer.state_dict(),
    "patch_size": PATCH_SIZE,
    "embed_dim": EMBEDDED_DIM,
    "train_img_size": TRAIN_IMG_SIZE,
}, last_path)

# 保存最佳检查点
if te_dice > best_dice:
    best_dice = te_dice
    torch.save({
        "epoch": ep,
        "model": decoder.state_dict(),
        "optimizer": optimizer.state_dict(),
        "patch_size": PATCH_SIZE,
        "embed_dim": EMBEDDED_DIM,
        "train_img_size": TRAIN_IMG_SIZE,
        "best_dice": best_dice,
    }, best_path)
    print(f"[INFO] saved best -> {best_path} (dice=

```

```

{best_dice:.4f}"))

print("[DONE] best_dice:", best_dice)
print("  best ckpt:", best_path)
print("  last ckpt:", last_path)

# ===== 测试 =====
print("[INFO] Testing...")
test_ds = NucleiTilesDataset(DATASET_PATH, split="test",
train_img_size=TRAIN_IMG_SIZE)
test_loader = DataLoader(test_ds, batch_size=8, shuffle=False)

# 加载最佳模型
decoder_best = SegDecoder(embed_dim=EMBEDED_DIM,
patch_size=PATCH_SIZE,
                        H=TRAIN_IMG_SIZE,
W=TRAIN_IMG_SIZE).to(DEVICE)
decoder_ckpt = torch.load(best_path, map_location=DEVICE)
decoder_best.load_state_dict(decoder_ckpt["model"])

# 运行测试
test(decoder_best, test_loader, DINOVI_T_CKPT_PATH, DEVICE,
PATCH_SIZE, RES_SAVE_PATH)
print(f"[INFO] Test results saved to {RES_SAVE_PATH}")

```

train.pyVersion control

Project

train.py8522073858948442822
train.py
External Libraries
Scratches and Consoles

train.py

```
22 IMAGENET_STD = np.array([0.229, 0.224, 0.225], dtype=np.float32)
23
24 DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
25
26 # 路径定义
27 DATASET_PATH = r"/root/Desktop/我的网盘/MoNuSeg/"
28 DINOVI_T_CKPT_PATH = r"/root/Desktop/我的网盘/dino_deit-small8_pretrain.pth"
29 CKPT_SAVE_PATH = r"/root/Desktop/我的网盘/save1/"
30 RES_SAVE_PATH = r"/root/Desktop/我的网盘/save2/"
31
32 # 超参数定义
33 TRAIN_IMG_SIZE = 256
34 PATCH_SIZE = 8
35 EMBED_DIM = 384
36 BATCH_SIZE = 4
37 EPOCH_NUM = 200
```

Runtrain

```
/root/miniconda3/envs/myconda/bin/python /mnt/train.py
[INFO] device: cuda
[INFO] loading DINO backbone...
[INFO] backbone patch_size=8, embed_dim=384
/mnt/train.py:421: FutureWarning: `torch.cuda.amp.GradScaler(args...)` is deprecated. Please use `torch.amp.GradScaler('cuda', args...)` instead.
scaler = GradScaler() if torch.cuda.is_available() else None
[INFO] start training...
/mnt/train.py:437: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
with autocast():
/mnt/train.py:315: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickling module that is unsafe when loading untrusted data.
ckpt = torch.load(ckpt_path, map_location=device)
[Epoch 005/200] train: loss=0.5142 iou=0.6493 dice=0.7849 | valid: loss=0.5752 iou=0.6590 dice=0.7935 | time=74.6s
[INFO] saved best -> /root/Desktop/我的网盘/save1/best.pt (dice=0.7935)
```

34.15CRLFUTF-84 spacesmyconda

```
Project <
  train.py8522073858948442822
    train.py
  External Libraries
  Scratches and Consoles

train.py
22 IMAGENET_STD = np.array([0.229, 0.224, 0.225], dtype=np.float32)
23
24 DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
25
26 # 路径定义
27 DATASET_PATH = r"/root/Desktop/我的网盘/MoNuSeg/"
28 DINOVI_T_CKPT_PATH = r"/root/Desktop/我的网盘/dino_deit-small8_pretrain"
29 CKPT_SAVE_PATH = r"/root/Desktop/我的网盘/save1/"
30 RES_SAVE_PATH = r"/root/Desktop/我的网盘/save2/"
31
32 # 超参数定义
33 TRAIN_IMG_SIZE = 256
34 PATCH_SIZE = 8
35 EMBEDDED_DIM = 384
36 BATCH_SIZE = 4
37 EPOCH_NUM = 200

Run train

/root/miniconda3/envs/myconda/bin/python /mnt/train.py
[INFO] device: cuda
[INFO] loading DINO backbone...
[INFO] backbone patch_size=8, embed_dim=384
/mnt/train.py:421: FutureWarning: `torch.cuda.amp.GradScaler(args...)` is deprecated. Please use `torch.amp.GradScaler('cuda', scaler = GradScaler()) if torch.cuda.is_available() else None`
[INFO] start training...
/mnt/train.py:437: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', at with autocast()):
/mnt/train.py:315: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which use ckpt = torch.load(ckpt_path, map_location=device)
[Epoch 005/200] train: loss=0.5142 iou=0.6493 dice=0.7849 | valid: loss=0.5752 iou=0.6590 dice=0.7935 | time=74.6s
[INFO] saved best -> /root/Desktop/我的网盘/save1/best.pt (dice=0.7935)

warnings.warn(
[Epoch 005/200] train: loss=0.5047 iou=0.6536 dice=0.7881 | valid: loss=0.5719 iou=0.6555 dice=0.7910 | time=46.4s
[INFO] saved best -> /root/autodl-tmp/save1/best.pt (dice=0.7910)
[Epoch 010/200] train: loss=0.4351 iou=0.6963 dice=0.8194 | valid: loss=0.5546 iou=0.6665 dice=0.7989 | time=48.0s
[INFO] saved best -> /root/autodl-tmp/save1/best.pt (dice=0.7989)
```

```
train.py
22 IMAGENET_STD = np.array([0.229, 0.224, 0.225], dtype=np.float32)
23
24 DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
25
26 # 路径定义
27 DATASET_PATH = r"/root/Desktop/我的网盘/MoNuSeg/"
28 DINOVI_T_CKPT_PATH = r"/root/Desktop/我的网盘/dino_deit-small8_pretrain"
29 CKPT_SAVE_PATH = r"/root/Desktop/我的网盘/save1/"
30 RES_SAVE_PATH = r"/root/Desktop/我的网盘/save2/"
31
32 # 超参数定义
33 TRAIN_IMG_SIZE = 256
34 PATCH_SIZE = 8
35 EMBEDDED_DIM = 384
36 BATCH_SIZE = 4
37 EPOCH_NUM = 200
```