Segmentação do DeepLabV3+

Discente: Hector José Rodrigues Salgueiros

Instalação das dependências

```
!pip install roboflow pycocotools tqdm matplotlib torchvision
!pip install -q segmentation-models-pytorch
```

Importação do dataset

```
from roboflow import Roboflow

rf = Roboflow(api_key="API_KEY")
project = rf.workspace("questao2").project("spine_det-alvah")
version = project.version(1)
dataset = version.download("coco")
```

Importação das bibliotecas

```
import os
import torch
import torchvision
from torchvision.models.segmentation import deeplabv3_resnet50
from torchvision.transforms import functional as F
from torchvision.utils import draw_segmentation_masks
from torchvision.datasets import CocoDetection
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
from tqdm import tqdm
import json
from pycocotools.coco import COCO
from torchvision import transforms
from sklearn.metrics import f1_score
import torch.nn as nn
import torch.optim as optim
import segmentation_models_pytorch as smp
```

Dataset customizado com formato COCO

```python
class CocoSegmentationDataset(CocoDetection):
    def __init__(self, img_folder, ann_file, transforms=None):
        super().__init__(img_folder, ann_file)
        self._transforms = transforms

    def __getitem__(self, idx):
        img, target = super().__getitem__(idx)
        ann_ids = self.coco.getAnnIds(imgIds=self.ids[idx])
        anns = self.coco.loadAnns(ann_ids)

        mask = np.zeros((img.size[1], img.size[0]), dtype=np.uint8)
        for ann in anns:
            # Verifica se a anotação tem um polígono/RLE válido
            if not ann["segmentation"]:  # Filtra segmentações vazias
                continue

            try:
                cat_id = ann['category_id']
                rle = self.coco.annToMask(ann)  # Gera a máscara binária
                mask[rle > 0] = cat_id  # Atribui o category_id às regiões segmentadas
            except Exception as e:
                print(f"Ignorando anotação inválida (ID {ann['id']}): {e}")
                continue

        mask = Image.fromarray(mask)

        if self._transforms:
            img, mask = self._transforms(img, mask)

        img = transforms.functional.to_tensor(img)
        mask = torch.from_numpy(np.array(mask)).long()

        return img, mask
```

Transformação das imagens e carregamento em lotes

```python
class CustomTransform():
    def __call__(self, img, mask):
        img = transforms.functional.resize(img, (512, 512))
        mask = transforms.functional.resize(mask, (512, 512), interpolation=transforms.InterpolationMode.NEAREST)
        return img, mask

def get_loader(folder, batch_size=4):
    img_dir = os.path.join(folder)
    ann_path = os.path.join(folder, "_annotations.coco.json")
    dataset = CocoSegmentationDataset(img_dir, ann_path, transforms=CustomTransform())
    return DataLoader(dataset, batch_size=batch_size, shuffle=True, num_workers=2, pin_memory=True)

# Cria os DataLoaders
train_loader = get_loader(dataset.location + "/train")
val_loader = get_loader(dataset.location + "/valid")
test_loader = get_loader(dataset.location + "/test")
```

Treinamento e avaliação do modelo

```python
def train_one_epoch(model, loader, optimizer, criterion):
    model.train()
    total_loss = 0

    for imgs, masks in tqdm(loader, desc="Treinando"):
        imgs, masks = imgs.to(device), masks.to(device)
        optimizer.zero_grad()
        output = model(imgs)
        loss = criterion(output, masks.long())
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    return total_loss / len(loader)


def evaluate(model, loader):
    model.eval()
    y_true, y_pred = [], []
    with torch.no_grad():
        for imgs, masks in tqdm(loader):
            imgs, masks = imgs.to(device), masks.to(device)
            outputs = model(imgs)
            preds = torch.argmax(outputs, dim=1).cpu().numpy()
            y_pred.extend(preds.flatten())
            y_true.extend(masks.cpu().numpy().flatten())
    f1 = f1_score(y_true, y_pred, average='macro')
    return f1
```

DeepLabV3+ com ResNet50 e treinamento

```python
# Configuração do dispositivo
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Modelo DeepLabV3+ com encoder ResNet-50
model = smp.DeepLabV3Plus(
    encoder_name="resnet50",         # backbone
    encoder_weights="imagenet",      # pesos pré-treinados
    in_channels=3,                   # RGB
    classes=2                        # fundo e escoliose
).to(device)


# CrossEntropy para segmentação com 2 classes
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4)

n_epochs = 25
for epoch in range(n_epochs):
    print(f"\nÉpoca {epoch+1}/{n_epochs}")
    train_loss = train_one_epoch(model, train_loader, optimizer, criterion)
    val_f1 = evaluate(model, val_loader)
    print(f"Loss de treino: {train_loss:.4f} | F1 Val: {val_f1:.4f}")
```

Resultado da última época

```
Época 25/25
Treinando: 100%|██████████| 607/607 [03:43<00:00,  2.72it/s]
100%|██████████| 25/25 [00:06<00:00,  3.70it/s]
Loss de treino: 0.0106 | F1 Val: 0.9135
```

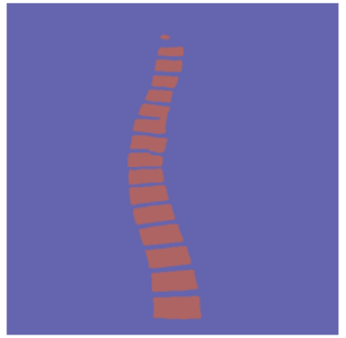
Plotagem

```python
def show_predictions(model, loader, n=3):
    model.eval()
    imgs, masks = next(iter(loader))
    imgs, masks = imgs.to(device), masks.to(device)
    with torch.no_grad():
        outputs = model(imgs)
        preds = torch.argmax(outputs, dim=1)

    for i in range(n):
        img = imgs[i].cpu()
        mask = preds[i].cpu().numpy()

        plt.figure(figsize=(10,3))
        plt.subplot(1,2,1)
        plt.title("Imagem")
        plt.imshow(img.permute(1,2,0))
        plt.axis('off')

        plt.subplot(1,2,2)
        plt.title("Máscara Predita")
        plt.imshow(mask, cmap='jet', alpha=0.6)
        plt.axis('off')
        plt.show()

print("\nVisualizando previsões...")
show_predictions(model, test_loader)
```

Resultado da Segmentação

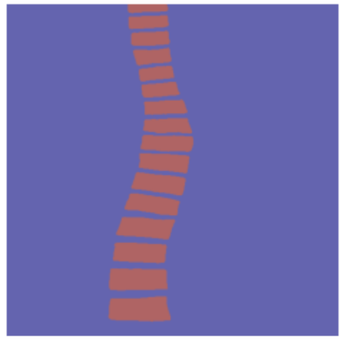
Imagem

Máscara Predita



Imagem

Máscara Predita

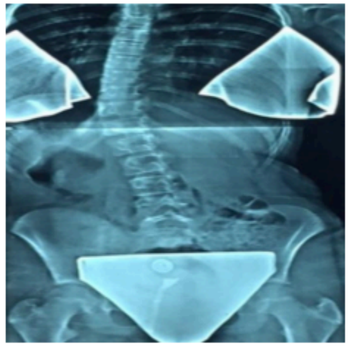


Imagem

Máscara Predita

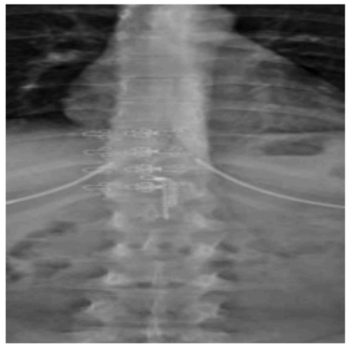
Resultado da Segmentação

Imagem



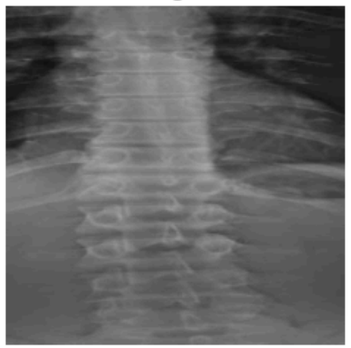
Máscara Predita

Imagem



Máscara Predita

Imagem



Máscara Predita