Xây dựng mô hình UNet cho bài toán phân đoạn ảnh y khoa

Lê Phạm Hoàng Trung

02/01/2025

1 Mô tả Đồ án

Bài toán phân đoạn hình ảnh (Image Segmentation) là một nhiệm vụ quan trọng trong Xử lý Hình Ảnh (Computer Vision). Mục tiêu của đồ án là xây dựng và huấn luyện mô hình UNet nhằm phân đoạn đối tượng trên hình ảnh.

2 Giới thiệu bộ dữ liệu

2.1 Tổng quan

Bộ dữ liệu Kvasir-SEG là một tập hợp hình ảnh y khoa được thu thập từ nội soi đường tiêu hóa (GI). Đây là một trong những bộ dữ liệu quan trọng giúp hỗ trợ nghiên cứu trong lĩnh vực phát hiện bệnh lý bằng trí tuệ nhân tạo. Bộ dữ liệu bao gồm các hình ảnh đã được phân loại và chú thích bởi các bác sĩ chuyên khoa tiêu hóa.

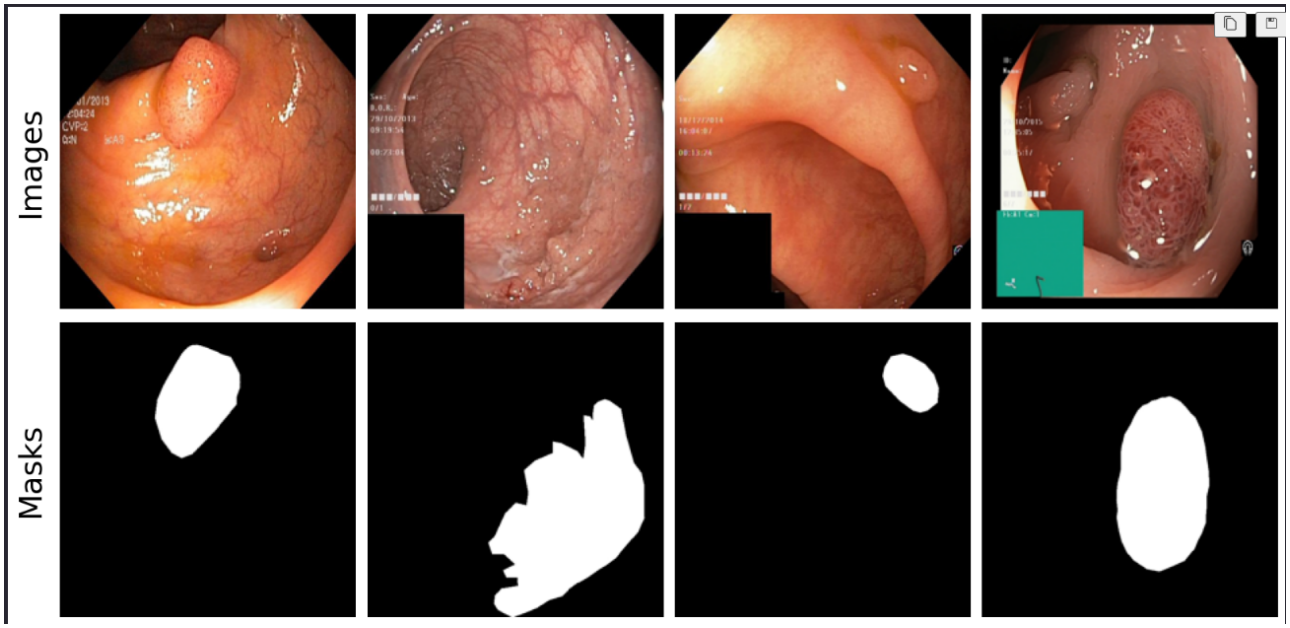
2.2 Quá trình thu thập dữ liệu

Bộ dữ liệu được thu thập bằng thiết bị nội soi tại Vestre Viken Health Trust (VV) ở Na Uy, một hệ thống bệnh viện phục vụ hơn 470.000 người. Các chuyên gia y tế từ VV và Cơ quan Đăng ký Ung thư Na Uy (CRN) đã chú thích và kiểm tra chất lượng của bộ dữ liệu để đảm bảo tính chính xác và nhất quán.

2.3 Chi tiết bộ dữ liệu

- Bộ dữ liệu Kvasir-SEG chứa 1000 hình ảnh polyp và mặt nạ phân đoạn tương ứng.
- Độ phân giải của hình ảnh dao động từ 332x487 đến 1920x1072 pixel.
- Hình ảnh và mặt nạ được lưu trong các thư mục riêng biệt nhưng có cùng tên tệp để dễ dàng xử lý.
- Các hình ảnh có thể chứa thông tin vị trí và cấu hình của nội soi, điều này có thể hỗ trợ việc phân tích.

2.4 Ví dụ một số cặp dữ liệu: hình ảnh - mặt nạ trong bộ dữ liệu



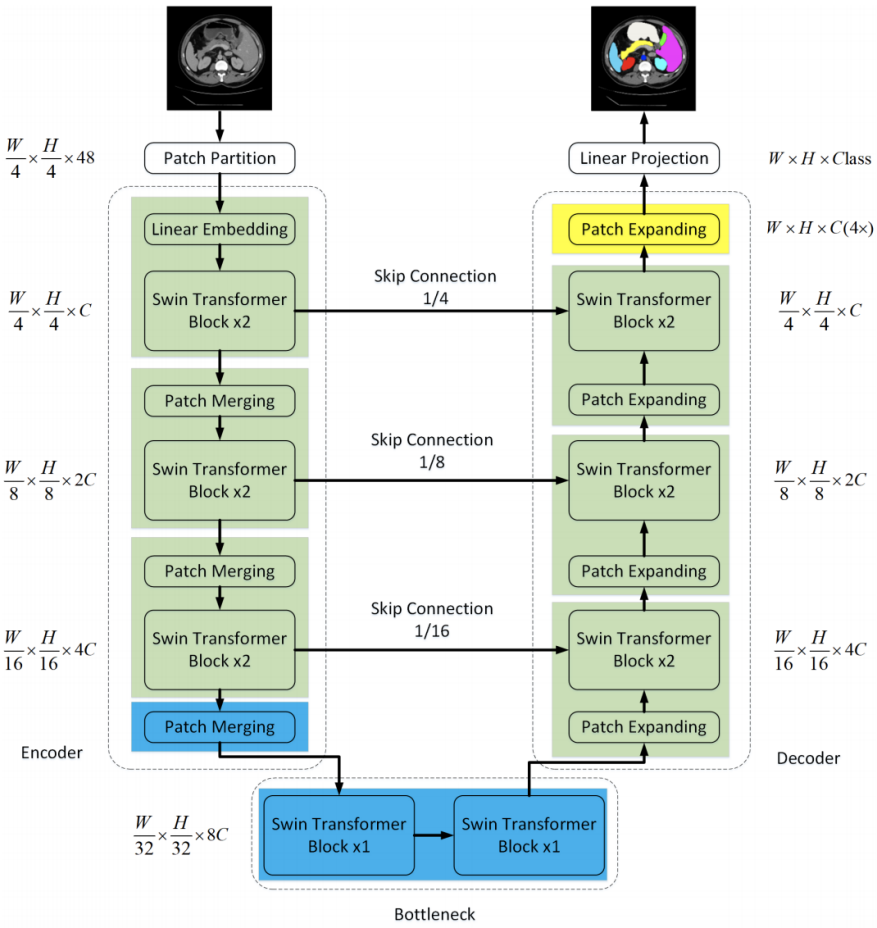
Hình 1: Một số cặp dữ liệu của bộ Kvasir-SEG

2.5 Cách tải bộ dữ liệu

Các bạn có thể tải bằng code (mình có cung cấp trong file notebook), tuy nhiên cần phải có tài khoản Kaggle `https://www.kaggle.com/` và đi vào phần Setting, chọn Create New Token. Sau đó dùng phần Token này cho đoạn code. Hoặc các bạn tải thẳng dataset này trên Kaggle về luôn cho lẹ `https://www.kaggle.com/datasets/abdallahwagih/kvasir-dataset-for-classification-a`

3 Kiến trúc tổng quan

UNet bao gồm ba phần chính: Encoder (bộ mã hóa), Center/Bottleneck (trung tâm), và Decoder (bộ giải mã). Mô hình sử dụng kiến trúc dạng U với các kết nối tắt (skip connections) giữa các tầng tương ứng của encoder và decoder để giúp phục hồi chi tiết hình ảnh tốt hơn. Thông thường, bộ encoder sẽ là các mô hình VGG, nếu sử dụng VGG11 thì sẽ gọi là UNet11, nếu sử dụng VGG16 thì là UNet16.



Hình 2: Overview of UNet

3.1 Encoder - Bộ mã hoá

Bộ mã hóa có nhiệm vụ giảm kích thước ảnh đầu vào và trích xuất các đặc trưng quan trọng. Nó hoạt động tương tự như một mạng nơ-ron tích chập (CNN) truyền thống, với các lớp tích chập và hàm kích hoạt (ReLU) xen kẽ với các lớp max-pooling để giảm dần kích thước không gian của ảnh.

- Dùng để downsampling
- Dùng VGG11 hoặc VGG16 (có thể dùng pretrained trên ImageNet để mã hoá)
- Bao gồm các lớp Convolutional + ReLU xen kẽ với MaxPooling để trích xuất đặc trưng.

Bảng chi tiết cho phần encoder, gồm các layer như sau:

Layer	Input Shape	Output Shape	Description
Conv1	(B, 3, H, W)	(B, 64, H, W)	1 Conv2D + ReLU
Pool1	(B, 64, H, W)	(B, 64, H/2, W/2)	MaxPooling (2x2)
Conv2	(B, 64, H/2, W/2)	(B, 128, H/2, W/2)	1 Conv2D + ReLU
Pool2	(B, 128, H/2, W/2)	(B, 128, H/4, W/4)	MaxPooling (2x2)
Conv3	(B, 128, H/4, W/4)	(B, 256, H/4, W/4)	2 Conv2D + ReLU
Pool3	(B, 256, H/4, W/4)	(B, 256, H/8, W/8)	MaxPooling (2x2)
Conv4	(B, 256, H/8, W/8)	(B, 512, H/8, W/8)	2 Conv2D + ReLU
Pool4	(B, 512, H/8, W/8)	(B, 512, H/16, W/16)	MaxPooling (2x2)
Conv5	(B, 512, H/16, W/16)	(B, 512, H/16, W/16)	2 Conv2D + ReLU

Bảng 1: EncoderLayer-wise architecture of UNet

Ví dụ về một block **Conv** trong bảng 1

```
if pretrained:
    self.encoder = models.vgg11(weights=models.vgg.VGG11_Weights.DEFAULT).
    features
else:
    self.encoder = models.vgg11().features

self.conv1 = nn.Sequential(self.encoder[0], self.relu)
```

trong đó, một **self.encoder** là các features của mô hình VGG11, bao gồm:

Index (i)	Lớp (self.encoder[i])
0	Conv2d(3, 64, 3×3)
1	ReLU(inplace=True)
2	MaxPool2d(kernel_size=2, stride=2)
3	Conv2d(64, 128, 3×3)
4	ReLU(inplace=True)
5	MaxPool2d(kernel_size=2, stride=2)
6	Conv2d(128, 256, 3×3)
7	ReLU(inplace=True)
8	Conv2d(256, 256, 3×3)
9	ReLU(inplace=True)
10	MaxPool2d(kernel_size=2, stride=2)
11	Conv2d(256, 512, 3×3)
12	ReLU(inplace=True)
13	Conv2d(512, 512, 3×3)
14	ReLU(inplace=True)
15	MaxPool2d(kernel_size=2, stride=2)
16	Conv2d(512, 512, 3×3)
17	ReLU(inplace=True)
18	Conv2d(512, 512, 3×3)
19	ReLU(inplace=True)
20	MaxPool2d(kernel_size=2, stride=2)

Bảng 2: Cấu trúc lớp của VGG11

Tức nếu sử dụng:

```
self.conv1 = nn.Sequential(self.encoder[0], self.relu)
```

ta sẽ lấy self.encoder[0] là Conv2d$(3, 64, 3 \times 3)$. Còn nếu sử dụng:

```
self.conv2 = nn.Sequential(self.encoder[3], self.relu)
```

thì self.encoder[3] là Conv2d$(64, 128, 3 \times 3)$

3.2 Center - Trung tâm và Decoder - Bộ giải mã

- Gồm 1 block Center và các Decoder Blocks, giúp phục hồi kích thước ảnh bằng Upsampling hoặc Transposed Convolution
- Mỗi tầng của Decoder sẽ nhận thông tin từ tầng tương ứng của encoder qua skip connections

Bảng chi tiết cho phần này, các bạn tham khảo để code theo:

Layer	Input Shape	Output Shape	Description
Center Block	(B, 512, H/16, W/16)	(B, 256, H/16, W/16)	Decoder Block
Dec5	(B, 768, H/16, W/16)	(B, 256, H/16, W/16)	Decoder Block
Dec4	(B, 768, H/8, W/8)	(B, 128, H/8, W/8)	Decoder Block
Dec3	(B, 384, H/4, W/4)	(B, 64, H/4, W/4)	Decoder Block
Dec2	(B, 192, H/2, W/2)	(B, 32, H/2, W/2)	Decoder Block
Dec1	(B, 96, H, W)	(B, 32, H, W)	Conv + ReLU
Final Conv	(B, 32, H, W)	(B, num_classes, H, W)	1×1 Conv

Bảng 3: DecoderLayer-wise architecture of UNet

4 Yêu cầu bài tập

- **Bài tập 1:** Hãy viết một đoạn mã Python để hiển thị **n** cặp ảnh và mask từ dataset. Mỗi cặp bao gồm:
  - Một ảnh màu (*RGB*).
  - Một mask nhị phân (*grayscale*).

  Gợi ý: Bạn có thể sử dụng thư viện `matplotlib` để hiển thị ảnh và mask theo dạng lưới.

  1. Sử dụng `plt.subplots(2, n, figsize=(n*5, 10))` để tạo một lưới gồm 2 hàng và **n** cột.
  2. Duyệt qua **n** ảnh và mask:
     - Ở hàng đầu tiên, hiển thị ảnh màu.
     - Ở hàng thứ hai, hiển thị mask với màu *grayscale*.
  3. Ẩn trục tọa độ để dễ quan sát kết quả.
  4. Thêm tiêu đề cho hai hàng ảnh và mask.
  5. Điều chỉnh khoảng cách giữa các ảnh để có bố cục đẹp hơn.

  Mã mẫu:

```
import matplotlib.pyplot as plt
from PIL import Image

n = 4

fig, axs = plt.subplots(2, n, figsize=(n*5, 10))

for i in range(n):
    image = Image.open(image_paths[i]).convert('RGB')
    image = image.resize((224, 224))
    axs[0, i].imshow(image)
    axs[0, i].axis('off')

    mask = Image.open(mask_paths[i]).convert('L')
    mask = mask.resize((224, 224))
    axs[1, i].imshow(mask, cmap='gray')
    axs[1, i].axis('off')

plt.text(-0.15, 0.5, 'Images', va='center', rotation='vertical',
transform=axs[0, 0].transAxes, fontsize=28)
plt.text(-0.15, 0.5, 'Masks', va='center', rotation='vertical',
transform=axs[1, 0].transAxes, fontsize=28)

plt.subplots_adjust(wspace=0.04, hspace=0.04)
plt.show()

```

Yêu cầu thêm: Hãy thử thay đổi số lượng ảnh **n** và kiểm tra xem kết quả hiển thị có đúng không.

- **Bài tập 2:** Trong bài tập này, bạn sẽ thực hiện các phép biến đổi (*transforms*) trên ảnh và mask để chuẩn bị dữ liệu đầu vào cho mô hình học sâu. Hãy hoàn thiện các phần còn thiếu trong đoạn mã (phần todo). **Gợi ý:** Sử dụng các phương thức có sẵn trong thư viện `torchvision.transforms` để thực hiện:
  - Chuẩn hóa ảnh sử dụng giá trị `mean` và `std` đã được tính toán trước.
  - Chuyển đổi mask sang dạng nhị phân, chỉ bao gồm hai giá trị 0 và 1.
- **Bài tập 3:** Xây dựng mô hình UNet, các bạn dựa vào các bảng layer chi tiết ở Encoder và Decoder đã cung cấp ở trên, hoàn thành các **todo** trong đoạn mã gốc.
- **Bài tập 4:** Hoàn thiện hàm tính toán IOU. Công thức:

$$IoU = \frac{\text{target } \cap \text{ prediction}}{\text{target } \cup prediction}$$

5 Yên cầu bài nộp

- Nộp file notebook định dạng **.ipnb** đã hoàn thành. Lưu ý bấm **Run all** để đảm bảo code chạy được hết.
- Báo cáo, định dạnng file pdf về cách thực hiện và kết quả thu được.