Báo cáo về mô hình phân đoạn ảnh y khoa trong tập dữ liệu Kvasir sử dụng kiến trúc UNet11

Nguyễn Trường Giang

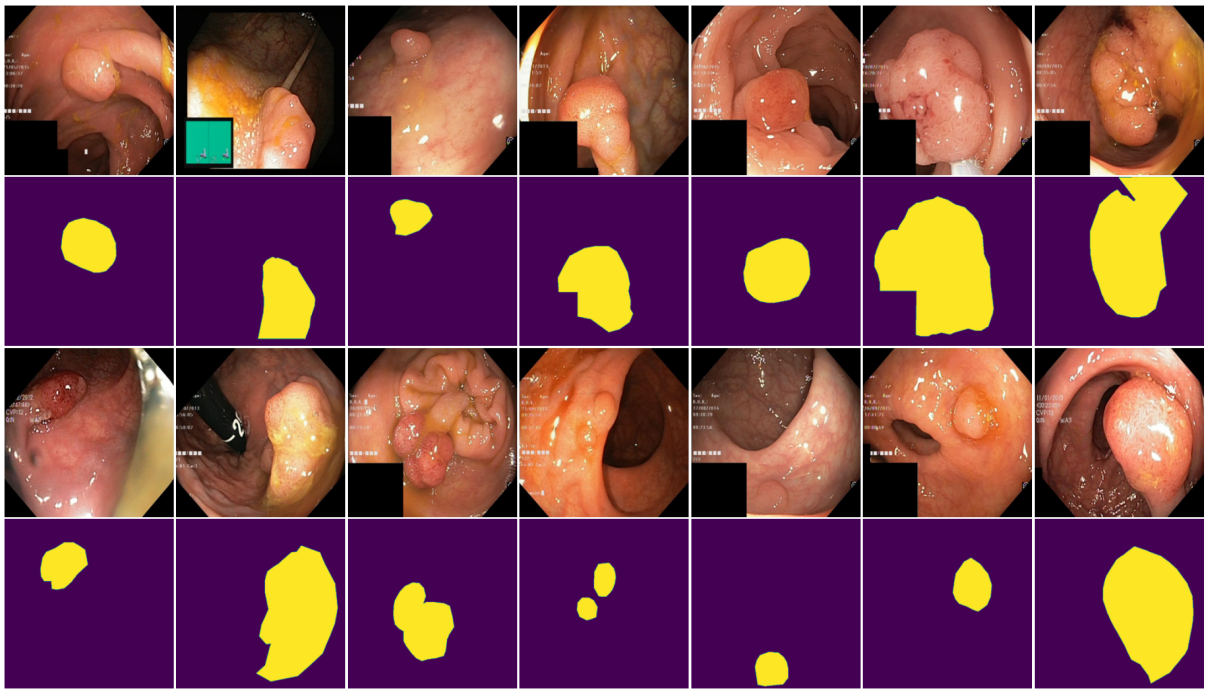
Ngày 10 tháng 3 năm 2025

1 Tập Dataset.

1.1 Tập dữ liệu đầu vào đầu vào.

- Tập dữ liệu đầu vào có 1000 image (ảnh) và 1000 mask (”mặt nạ”).

Dữ liệu thô: Trong tập dữ liệu Kvasir này các image (ảnh) và mask (”mặt nạ”) có nhiều kích thước khác nhau. Trong đó image gồm 3 kênh màu (RGB) và mask có 1 kênh màu với giá trị nhị phân 0 hoặc 255 tức là vùng không chứa đối tượng và vùng chứa đối tượng.



Hình 1: Trực quan về tập dữ liệu Kvasir

1.2 Tập dữ liệu sau khi được xử lý.

Sau bước xử lý dữ liệu: Các image và mask đều được đưa về kích thước (224×224). Cụ thể hơn:

- Ảnh train data: $(8 \times 3 \times 224 \times 224)$, các giá trị trong ảnh sẽ được đưa về phân phối chuẩn với trung bình cộng ≈ 0 và độ lệch chuẩn ≈ 1 (qua bước trung gian .ToTensor() sẽ đưa giá trị trong ảnh từ $[0 \dots 255]$ về giá trị trong khoảng $[0 \dots 1]$).
- Mặt nạ train data: $(8 \times 1 \times 224 \times 224)$, các giá trị trong ảnh là các giá trị 0 hoặc 1.
- Ảnh validate và test data: $(4 \times 3 \times 224 \times 224)$, các giá trị tương tự như ảnh train data.
- Mặt nạ train data: $(4 \times 1 \times 224 \times 224)$, các giá trị tương tự như ảnh test data.

2 Kiến trúc mô hình

- Như các mô hình UNet khác, mô hình được chia thành 2 phần: **Contracting path** và **Expansive path**, ngoài ra phần giữa chứa một khối **bottleneck**.

- Mô hình sử dụng liên tục các lớp **Convolution** với hàm kích hoạt là **ReLU** và lớp **Pooling** ở phần **Encode**. Còn về phần **Decode** mô hình sử dụng **Transposed Convolution** để tăng kích thước ảnh và giảm số kênh, và như các mô hình **UNet** khác, phần **Decode** cũng copy các kênh từ phần **Encode** ở **"cùng vị trí"** làm tăng số lượng kênh của phần **Decode** nhằm mục đích để lấy lại hình dạng ban đầu của đối tượng.

- Ngoài ra ở phần Encode, mình có thêm một lớp BatchNorm trong mỗi khối Encode, giúp chuẩn hóa dữ liệu tại mỗi block. Còn về phần Decode do mình quên và lúc sau cũng lười nên chưa thêm vào.

2.1 Contracting path:

- Phần Contracting path sử dụng phần trích xuất đặc trưng (Feature Extraction) của mô hình **VGG11**.
Ban đầu kích thước đầu vào mỗi batch là $(3 \times 224 \times 224)$.

1. Conv + ReLU **(1)**: $(64 \times 224 \times 224)$.
2. Pooling: $(64 \times 112 \times 112)$.
3. Conv + ReLU **(2)**: $(128 \times 112 \times 112)$.
4. Pooling: $(128 \times 56 \times 56)$.
5. Conv + ReLU + Conv + ReLU **(3)**: $(256 \times 56 \times 56)$.
6. Pooling: $(256 \times 28 \times 28)$.
7. Conv + ReLU + Conv + ReLU **(4)**: $(512 \times 28 \times 28)$.
8. Pooling: $(512 \times 14 \times 14)$.
9. Conv + ReLU + Conv + ReLU **(5)**: $(512 \times 14 \times 14)$.
10. Pooling: $(512 \times 7 \times 7)$.

- Giữa Contracting path và Expansive path là một khối bottleneck với các layer tương tự với phần decoder block.

2.2 Bottleneck:

- Bottleneck (Conv + ReLU + UpConv + ReLU): $(256 \times 14 \times 14)$.

2.3 Expansive path:

1. Decoder block **(5)**: $(256 \times 28 \times 28)$.
    - Cat: Thêm các channels từ lần Conv thứ 5: $256 + 512 = 768$ channels.
    - Conv + ReLU: $(512 \times 14 \times 14)$.
    - UpConv (Transposed Convolution) + ReLU: $(256 \times 28 \times 28)$.
2. Decoder block **(4)**: $(256 \times 56 \times 56)$.
    - Cat: Thêm các channels từ lần Conv thứ 4: $256 + 512 = 768$ channels.
    - Conv + ReLU: $(512 \times 28 \times 28)$.
    - UpConv (Transposed Convolution) + ReLU: $(256 \times 56 \times 56)$.
3. Decoder block **(3)**: $(128 \times 112 \times 112)$.
    - Cat: Thêm các channels từ lần Conv thứ 3: $256 + 256 = 512$ channels.
    - Conv + ReLU: $(256 \times 56 \times 56)$.
    - UpConv (Transposed Convolution) + ReLU: $(128 \times 112 \times 112)$.
4. Decoder block **(2)**: $(64 \times 224 \times 224)$.
    - Cat: Thêm các channels từ lần Conv thứ 2: $128 + 128 = 256$ channels.

- Conv + ReLU: $(128 \times 112 \times 112)$.
- UpConv (Transposed Convolution) + ReLU: $(64 \times 224 \times 224)$.

5. Decoder block **(1)**: $(32 \times 448 \times 448)$.
    - Cat: Thêm các channels từ lần Conv thứ 2: $64 + 64 = 128$ channels.
    - Conv + ReLU: $(64 \times 224 \times 224)$.
    - UpConv (Transposed Convolution) + ReLU: $(32 \times 448 \times 448)$.

- Pooling: $(32 \times 224 \times 224)$ thêm một layer phụ này để đảm bảo đầu ra có cùng kích thước ban đầu là 224×224.
- Conv (filter: 1x1): $(1 \times 224 \times 224)$.

2.4 Nhận xét về kích thước mô hình:

Mô hình có kích thước khá lớn với hơn 31 triệu tham số (đều là tham số cần phải train). Với kích thước input khá ít (chỉ 0.57 MB), nguyên nhân là vì dataset của mô hình chỉ lưu đường dẫn của các ảnh, chỉ khi nào cần train thì mới lấy một batch (8 ảnh) từ url ra để train. Trong khi đó phần lớn tài nguyên bộ nhớ đến từ việc **lan truyền** và **lan truyền ngược** (chiếm gần 1 GB). Tóm gọn lại mô hình chiếm mất 1072 MB RAM. Vấn đề lớn hơn là trong thực nghiệm, bộ nhớ của GPU (VRAM) chiếm tới tận 3 GB, một tải nguyên khá lớn khi chạy trên máy tính cá nhân.

```
    DecoderBlock-114         [-1, 32, 448, 448]               0
       MaxPool2d-115         [-1, 32, 224, 224]               0
          Conv2d-116          [-1, 1, 224, 224]              33
================================================================
Total params: 31,834,177
Trainable params: 31,834,177
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 950.33
Params size (MB): 121.44
Estimated Total Size (MB): 1072.34
----------------------------------------------------------------
```

Hình 2: Tóm tắt về kích thước mô hình

2.5 Mô tả sơ bộ (nhận xét cá nhân):

- Các block ở cả 2 phần Encoder block và Decoder block càng ở giữa thì càng dùng nhiều Conv trong cùng một block và tốc độ tăng/giảm kích thước ảnh cũng càng chậm hơn, lý do là vì càng vào trong các chi tiết càng nhiều hơn so với phần ngoài của Encode và Decode, do đó cần phải xử lý chậm hơn.
- Lý do có phần Cat là để phần Decode có thể ”học lại” hình dạng ban đầu của ảnh, tránh trường hợp thu nhỏ ảnh quá để xử lý rồi tới lúc phóng ảnh to ra lại mất hình dạng ban đầu.
- UNet không sử dụng bất kỳ lớp Fully connected nào, vì khi thu nhỏ ảnh tới tận kích thước 7×7 là gần như là fully connected rồi (với kích thước ảnh là (7×7) với số lượng kênh khủng, tại đó mô hình sẽ dễ dàng xác định và phân vùng được đối tượng), khi đó Conv và UpConv của bottleneck layer đóng vai trò tương tự như fully connected vậy, dĩ nhiên không chỉ do Bottleneck không mà do toàn bộ các Decoder block đằng sau cũng đóng vai trò đó nữa.
- Nguyên nhân dẫn đến việc không sử dụng bất kỳ FC (fully connected) nào là vì khi sử dụng FC sẽ bị mất thông tin về tính không gian.

3 Kết quả thực nghiệm

3.1 Loss function được sử dụng:

Sử dụng loss function: **Binary cross-enthropy loss (BCE)**.

$$\mathcal{L}_{BCE}(y, \hat{y}) = -\sum_i \left[y_i \log \sigma(\hat{y}_i) + (1 - y_i) \log(1 - \sigma(\hat{y}_i)) \right].$$

Vì mask có các pixel chỉ gồm 2 giá trị 0 hoặc 1, ngoài ra do đây là phân đoạn ảnh nên không cần thiên về 0 hoặc 1 (để tránh trường hợp toàn bộ mask phần lớn là 0 hoặc 1 thì sẽ bị đánh giá sai). Do đó loss function **BCE** là hợp lý.

3.2 Độ đánh giá được sử dụng:

- Độ đánh giá được sử dụng cho model là IOU thay vì Accuracy. Nguyên nhân chính là IOU đánh giá tốt hơn trên các bộ dữ liệu bị mất cân bằng vì IOU tập trung vào đối tượng cần xác định.

  Công thức IoU:

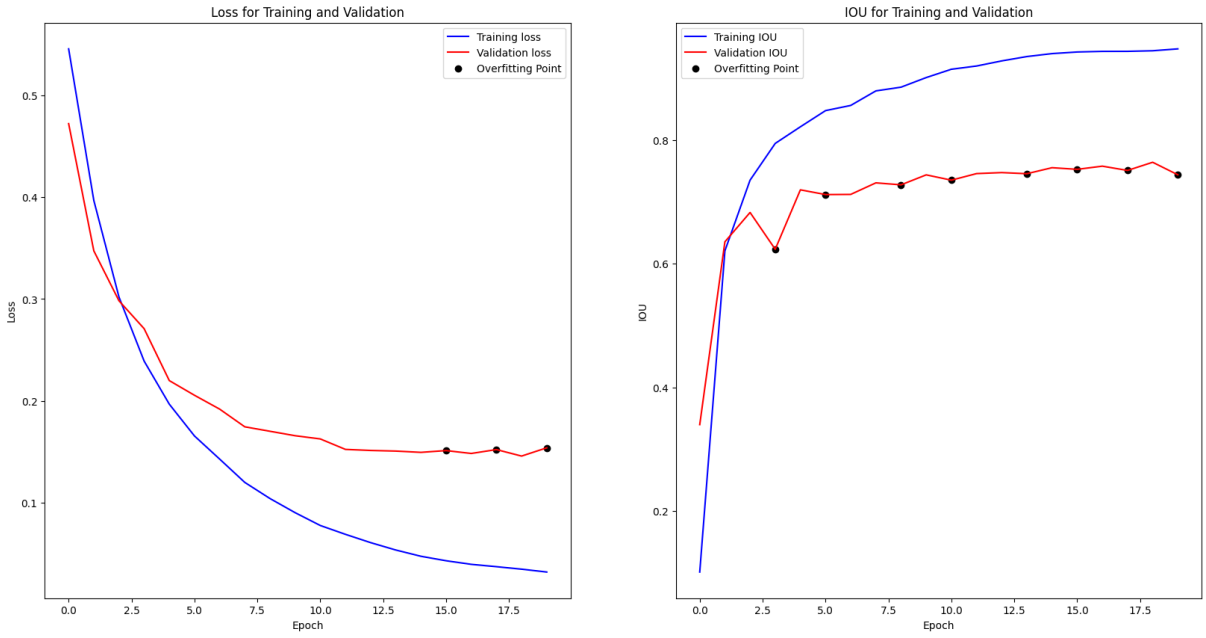
$$IoU = \frac{\text{intersection} + \text{SMOOTH}}{\text{union} + \text{SMOOTH}}$$

  Trong đó:

  - **Intersection** (phần giao nhau): tức là số pixel được dự đoán đúng (phần pixel của cả mask thực tế và mask được dự đoán đều có pixel bằng 1).
  - **Union** (phần hợp): là tổng số pixel bằng 1 thuộc mask của dự đoán hoặc mask thực tế (tức là số lượng pixel của đối tượng trong thực tế và số lượng pixel dự đoán là 1 nhưng lại bị sai so với thực tế).
  - **SMOOTH**: chỉ đơn giản là tránh trường hợp chia cho 0 thôi, vì số này thường để rất nhỏ.

- Ví dụ như khi phân đoạn một ảnh có vùng cần xác định nhỏ, accuracy sẽ đánh giá kết quả rất cao nhưng thực tế predict chỉ toàn số 0, trong khi đó IOU có công thức là **kích thước vật thể** trên tổng của **kích thước vật thể** và **phần sai khác**, công thức trên sẽ tập trung vào đối tượng hơn là Accuracy.

4 Đánh giá mô hình



Hình 3: Biểu đồ trực quan về loss và IOU đối với tập train và validation

Dựa theo biểu đồ về giá trị loss trên, ta nhận thấy rằng mô hình đã bị **overfitting** tại epoch thứ 15 trở đi, vì không có *Early Stopping* hay *Learning Schedule* nên mô hình vẫn tiếp tục train với learning rate cũ.