INT3404E 20 - Image Processing: Midterm

Sino-Nom character localization

Nhóm 10

Lưu Văn Đức Thiệu, Lê Hồng Đức
Nguyễn Thành Phát, Ngô Lê Hoàng

1 Giới thiệu bài toán

Bài toán định vị chữ Hán Nôm trong ảnh là một thách thức quan trọng trong lĩnh vực xử lý ảnh và nhận dạng ký tự quang học (OCR). Chữ Hán Nôm là hệ thống chữ viết cổ của người Việt, bao gồm chữ Hán (mượn từ Trung Quốc) và chữ Nôm (chữ sáng tạo bởi người Việt dựa trên chữ Hán). Mục tiêu của bài toán là xác định vị trí chính xác của các ký tự Hán Nôm trong một bức ảnh, từ đó hỗ trợ cho việc chuyển đổi văn bản viết tay hoặc in ấn thành dạng văn bản số hóa. Việc định vị chính xác các ký tự này không chỉ giúp bảo tồn và nghiên cứu di sản văn hóa mà còn mở ra nhiều ứng dụng trong các lĩnh vực giáo dục, lịch sử, và công nghệ thông tin.

2 Các hướng tiếp cận

2.1 YOLOv8

Model YoloV8 sử dụng là model $YoloV8x$ (phiên bản nhiều tham số nhất)
Loạt mô hình YOLO (You Only Look Once) đã trở nên nổi tiếng trong thế giới thị giác máy tính. Danh tiếng của YOLO được gán cho độ chính xác đáng kể của nó trong khi vẫn duy trì kích thước mô hình nhỏ gọn. Các mô hình YOLO có thể được huấn luyện trên một GPU duy nhất, điều này làm cho nó trở nên dễ tiếp cận với nhiều nhà phát triển.

YOLO đã được cộng đồng thị giác máy tính phát triển kể từ khi ra mắt lần đầu vào năm 2015 bởi Joseph Redmond. Trong những ngày đầu (các phiên bản từ 1-4), YOLO được duy trì bằng ngôn ngữ C sử dụng một deep learning framework do Redmond viết có tên là Darknet.

2.2 YOLOv9

Model YoloV9 được phát triển thêm dựa trên YoloV5. Nhóm sử dụng là model $YoloV9e$ (phiên bản nhiều tham số nhất)

2.3 Faster R-CNN

Faster R-CNN là một mô hình phát hiện đối tượng nổi bật, nối tiếp sự thành công của các phiên bản R-CNN trước đó. Mô hình này tích hợp mạng đề xuất vùng (RPN) và Fast R-CNN thành một kiến trúc duy nhất, cho phép việc phát hiện đối tượng nhanh chóng và chính xác hơn. Faster R-CNN đã đặt ra tiêu chuẩn mới trong lĩnh vực thị giác máy tính với khả năng xử lý hình ảnh đầu vào và dự đoán vị trí cũng như phân loại các đối tượng trong ảnh. Mặc dù có một số thách thức về mặt kỹ thuật và cài đặt, Faster R-CNN vẫn được cộng đồng nghiên cứu đánh giá cao vì khả năng tích hợp sâu các quy trình phát hiện đối tượng vào một mô hình thống nhất.

2.4 Detr-Resnet

Detr-ResNet-50 là kiến trúc mô hình deep learning kết hợp hai thành phần chính:

- Detr (DEtection Transformers): Đây là mô hình phát hiện đối tượng dựa trên Transfomers sử dụng cấu trúc encoder-decoder. Bộ mã hóa (encoder) xử lý hình ảnh đầu vào, trích xuất các đặc trưng, trong khi bộ giải mã (decoder) dự đoán các bounding-box và label cho các đối tượng được phát hiện.
- ResNet-50: Đây là kiến trúc mạng thần kinh tích chập (CNN) được biết đến với khả năng phân loại hình ảnh. Nó đóng vai trò là xương sống cho Detr trong trường hợp này, chịu trách nhiệm trích xuất các đặc điểm hình ảnh từ hình ảnh đầu vào.

3 Dữ liệu

3.1 Thu thập dữ liệu

Nhóm tìm hiểu các tập dữ liệu bao gồm:

1. Bộ dữ liệu mặc định do trợ giảng cung cấp bao gồm 70 ảnh cho huấn luyện, và 10 ảnh cho đánh giá
2. Tìm hiểu bộ dữ liệu NomnaOCR: bộ dữ liệu này bao gồm 2953 trang chữ Nôm viết tay và được phân tích và gán nhãn thành 38318 ảnh có bounding box và chữ Nôm ở dạng kỹ thuật số. Tuy nhiên trong bộ dữ liệu này, các bounding box được gán nhãn theo cụm chữ, và rất khó để có thể chuyển các label bounding box này về dạng giống với bộ dữ liệu mặc định. Do đó nhóm quyết định không dùng bộ dữ liệu này mà sẽ tập trung vào việc tăng cường dữ liệu có sẵn để cải thiện chất lượng các mô hình.

3.2 Tăng cường dữ liệu

Mỗi entry sẽ gồm một ảnh và một array gồm các bounding box được transform thành một image mới và một array gồm các bounding box tương ứng.

3.2.1 Lật ảnh (Flip)

Gồm 2 cách lật: lật ngang và lật dọc
Lật ngang ảnh sử dụng hàm **numpy.fliplr()**. Các bounding box được tính toán lại bằng cách lật x_{center}

```
    def get_flip(num):
        return 1 - num

    new_boxes = []
    for box in boxes:
        new_box = [get_flip(box[0]), box[1], box[2], box[3]]
        new_boxes.append(new_box)
```

Figure 1: Orignal vs Horizontal Flip Image

Lật dọc ảnh sử dụng hàm **numpy.flipud()**. Các bounding box được tính toán lại bằng cách lật y_{center}

```
    def get_flip(num):
        return 1 - num

    new_boxes = []
    for box in boxes:
        new_box = [box[0],get_flip(box[1]),box[2],box[3]]
        new_boxes.append(new_box)
```

Figure 2: Orignal vs Vertical Flip Image

3.2.2 Thêm nhiễu (Noise injection)

Sử dụng gaussian noise để thêm nhiễu vào trong ảnh.
Hàm **numpy.random.normal()** được dùng để tạo một noise matrix ứng với size của ảnh. Trong quá trình thêm nhiễu vào ảnh thì cần xử lý để tránh giá trị pixel vượt quá giới hạn, cụ thể là sử dụng hàm **numpy.clip()**

```
    noise = np.random.normal(self.mean, self.std, image.shape)
    # avoid out-of-range value and capture value from float to int
    added_noise_image = np.clip(image + noise, 0, 255).astype(np.uint8)
```

Figure 3: Orignal vs Noise Injection Image

3.2.3 Cắt ảnh (Crop)

Sử dụng class RandomCrop trong thư viện albumentations để cắt ảnh
Cách làm này có một nhược điểm là vài bounding box chỉ capture khoảng trống do crop ngẫu nhiên.

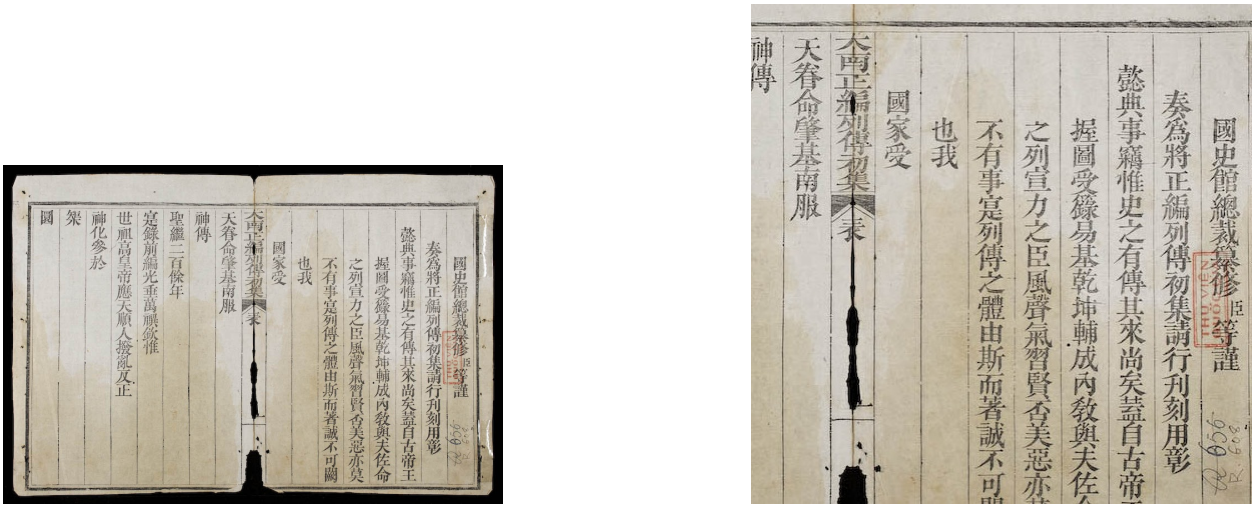

Figure 4: Orignal vs Crop Image

3.2.4 Làm mờ (Blur)

Sử dụng Gaussian kernel (có thể tùy chỉnh kernel size) để làm mờ ảnh. Cụ thể sử dụng hàm **cv2.getGaussianKernel()** để tạo một Gaussian Kernel, sau đó dùng hàm **cv2.filter2D()** để áp dụng bộ lọc Gaussian lên ảnh để làm mờ

```
    kernel = cv.getGaussianKernel(self.kernel_size, 0)
    gaussian_kernel = np.matmul(kernel, kernel.transpose())

    # Apply the Gaussian filter to the image to make it blur
    blur_image = cv.filter2D(image, -1, gaussian_kernel)
```

Figure 5: Orignal vs Blur Image

3.2.5 Loại bỏ, thêm pixel biên (Erosion, Dilation)

Sử dụng class Morphological với 2 mode là erosion, dilation.
Dilation augment data làm giảm accuracy của model do chữ bị nhạt đi tương đối nhiều.

3.2.6 Xóa ngẫu nhiên (Random erasing)

Sử dụng class CoarseDropout trong thư viện albumentations để xóa ngẫu nhiên các vùng ảnh (hole), trong cách implement các vùng ảnh bị xóa đi sẽ có màu trắng và bounding box giữ nguyên. Số lượng vùng ảnh bị xóa đi giới hạn ở mức 100 để giảm khả năng một vùng ảnh trong một bounding box bị xóa đi hoàn toàn.

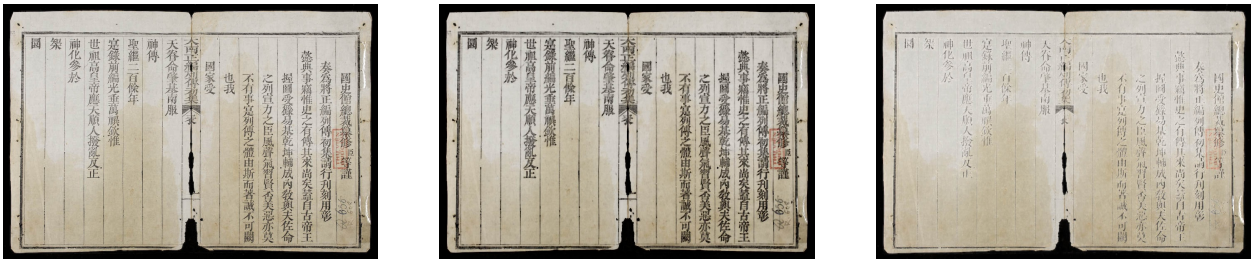

Figure 6: Orignal vs Erosion vs Dilation Image

Figure 7: Orignal vs Random Erase Image

4 Kết quả và phân tích

4.1 Môi trường thực nghiệm

Nhóm tiến hành các thực nghiệm trên môi trường Google Colab và Kaggle:

- Google Colab: GPU T4
- Kaggle: GPU P100, 2 x GPU T4.

4.2 Cách thực hiện

Đối với các model YOLO nói chung: Khi data không có nhãn, YOLO sẽ xếp nó vào loại background image (ảnh nền), đây là một tính năng của YOLO để giúp giảm thiểu False Positive [1]. Tuy nhiên sau khi thử, nhóm nhận thấy model học rất tệ khi áp dụng, do đó nhóm quyết định sử dụng toàn bộ data có nhãn.

Đối với YOLOv8:

- **100 epoch đầu tiên**: Sử dụng dataset mặc đinh (được cung cấp), đây sẽ là baseline của nhóm, các lần train tiếp theo sẽ lần từ *best_weight* của lần train này.
- **100 epoch sử dụng một phần data augmentation**: Đạt được kết quả tốt nhất hiện tại, 0.881 trên tập *val*.
- **100 epoch sử dụng data augmentation (Các loại augmentation được thêm sau: Erosion, Dilation))**: Model có dấu hiệu decay, không lấy kết quả này.
- **100 epoch sử dụng dataset gốc và data augmentation**: Nhóm thực hiện train lại từ weight ban đầu, sử dụng dataset được gộp từ data gốc và các data đã được biến đổi, kết quả thu được là 0.8779 trên tập *val*.

[1] https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/

Đối với YOLOv9:
Do không có pretrained weight nên nhóm đã train từ đầu, dưới đây là các bước và nhận xét:

- **100 epoch đầu tiên**: Sử dụng dataset mặc định, đây sẽ là baseline của nhóm, các lần train tiếp theo sẽ lần từ *best_weight* của lần train này. Sau khi train xong, kết quả thu được không tốt hơn YoloV8 nên nhóm đã quyết định dừng.

Đối với Detr-Resnet-50: sử dụng pretrained model "facebook/detr-resnet-50" và finetune trên tập dữ liệu training đã được tăng cường, batch-size 2

4.3 Phân tích

Nhóm đã có ý tưởng sử dụng một mô hình thứ ba để extract các đặc trưng riêng của chữ Nôm. Tuy nhiên, việc này rất khó để cài đặt tích hợp với các kiến trúc mô hình có sẵn như YOLO hay các mô hình khác. Trong thời gian thực hiện bài tập này, nhóm đã không cài đặt. Tuy nhiên, nhóm có một đề xuất để có thể làm được điều đó. Nhóm đề xuất sử dụng mô hình Autoencoder để chắt lọc features chữ Nôm, tạo một lớp embedding đầu vào cho mô hình YOLO hoặc ResNet. Điều này được kì vọng sẽ mang lại kết quả tốt.

Về các bước training, nhóm sẽ bắt đầu lý giải và phân tích dưới đây:
Với YOLOv8:
Nhóm đã thực hiện 2 nhánh chính:

- Fine-tune 100 epoch đầu với tập dữ liệu gốc, sau đó train thêm với augmented data.
- Fine-tune với bộ data được merge cả hai bộ dữ liệu.

Nguyên nhân dẫn đến sự khác biệt giữa kết quả 2 model trên là vì:

- 100 epochs đầu chính là để cho mô hình học được các đặc trưng của chữ Nôm, điều này giúp tạo được nền tảng vững chắc cho việc fine-tune thêm các data được augmented.
- Các epoch sau sử dụng augmented data mục đích chính là để tăng sự khỏe mạnh của model (robustness) và sự ổn định (consistency). Điều này có thể thấy rõ qua kết quả trên tập test, mặc dù chỉ số $mAP@[.5, .8]$ thấp hơn các nhóm khác ($\approx 97,8\%$) tuy nhiên chỉ số $mAP@[.85, .95]$ lại cao hơn so với các nhóm khác. Đây là minh chứng vững chắc cho luận điểm này.
- Việc train với loại augmentation còn lại dẫn đến kết quả tệ hơn các lần trước có thể do 2 nguyên nhân theo nhóm đề xuất:
  - Các augmentation này quá khó để model nhận dạng và khác quá nhiều với dữ liệu được train trước đó.
  - Model đã bị bão hòa.

  Từ các nhận xét trên nhóm đã có dự định sinh lại và train lại, tuy nhiên do thời gian có hạn nên bỏ qua hướng tiếp cận này.
- Từ việc trên, nếu chúng ta thực hiện việc train với dataset đã được merge lại, model trong quá trình học sẽ bị lẫn lộn và không học được các đặc trưng quan trọng của chữ Nôm do có quá nhiều nhiễu (**lượng data được augmented gấp 9 lần dataset ban đầu**). Do đó, hướng tiếp cận này là không khả thi.

Với YOLOv9:
Nhóm chỉ sử dụng kiến trúc của YOLOv9 mà không có pretrained weights. Điều này có thể dẫn chúng ta đến với các kết luận sau:

- Việc train trên các dataset khác là rất cần thiết trong quá trình làm bài toán này. Vì dataset sẵn có của chúng ta quá ít (70 samples), kể cả khi train qua 100 epochs cũng sẽ không thể can thiệp sâu đến các layer của mô hình.
- Nếu tiếp tục với việc thêm nhiễu vào dataset, kết quả thu được có thể sẽ tệ đi vì model không đảm bảo được cơ sở ban đầu.

Với Detr-ResNet-50:
Nhóm sử dụng pretrained weights, tuy nhiên đây là weight đã được train trên dataset được gán nhãn để định vị động vật trong ảnh, do vậy thời gian để model thích ứng với kiểu dữ liệu mới khá lâu. Do thời gian và tài nguyên có hạn cũng như kết quả quá tệ nên nhóm đã quyết định dừng lại.

4.4 Kết quả

Bảng 1 thể hiện các kết quả thực nghiệm.

Table 1: Tổng kết các hướng thử nghiệm

Hướng thử nghiệm	mAP@[.5,.95]	Đánh giá của nhóm
YOLOv8	0.86	thành công
YOLOv8 + data augmentation	0.881	thành công
YOLOv9	0.85	thất bại
Detr-ResNet-50	0.002	thất bại

5 Kết luận

Bài toán localization chữ Nôm là một bài toán mới với cả nhóm, việc tìm hướng giải quyết và tiếp cận bài toán ban đầu đã gây ra nhiều khó khăn. Tuy nhiên, với những phân tích và kết quả thực nghiệm cũng như kết quả chấm bài, những dẫn chứng đó đã chứng minh cho việc chọn mô hình cũng như chiến lược huấn luyện đã đạt được kết quả tốt.