

AI VIET NAM – COURSE 2024

Scene Text Recognition - Project

Ngày 28 tháng 12 năm 2024

Phần I: Giới thiệu

Nhận dạng Văn bản trong Ảnh (Scene Text Recognition) là một bài toán ứng dụng các kỹ thuật xử lý hình ảnh và nhận dạng chữ viết để xác định các văn bản xuất hiện trong các bức ảnh chụp từ môi trường thực tế. Bài toán này có nhiều ứng dụng thực tế, chẳng hạn như:

- Xử lý văn bản trong ảnh: Nhận diện chữ trong các loại tài liệu, báo chí, biển hiệu,...
- Tìm kiếm thông tin: Nhận diện chữ trong ảnh trên internet để thu thập dữ liệu quan trọng.
- Tự động hóa quy trình: Nhận diện chữ trong ảnh để tự động hóa các công việc, ví dụ như xử lý đơn hàng, thanh toán,...

Quy trình Nhận dạng Văn bản trong Ảnh điển hình gồm hai giai đoạn chính:

- Phát hiện chữ viết (Detector): Xác định vị trí các khối văn bản trong ảnh.
- Nhận diện chữ viết (Recognizer): Giải mã văn bản tại các vị trí đã được xác định.



Hình 1: Minh họa về bài toán Nhận dạng Văn bản trong Ảnh.

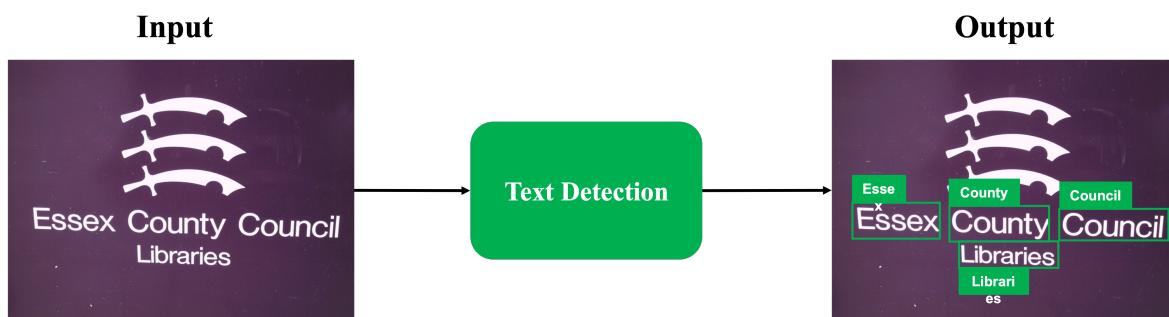
Trong dự án này, chúng ta sẽ phát triển một chương trình Nhận dạng Văn bản trong Ảnh sử dụng YOLOv11 (cho việc phát hiện văn bản) và CRNN (cho việc nhận dạng chữ). Đầu vào và đầu ra của chương trình như sau:

- **Đầu vào:** Một bức ảnh chứa văn bản.
- **Đầu ra:** Tọa độ vị trí và nội dung văn bản trong ảnh.

Phần II: Thiết lập chương trình

Chúng ta sẽ triển khai dự án này theo bốn giai đoạn, mỗi giai đoạn tập trung vào việc xây dựng các module đã được đề cập ở phần giới thiệu.

- Tải dữ liệu:** Các module trong chương trình Nhận dạng Văn bản trongẢnh của dự án này, bao gồm phần Text Detection và Nhận dạng Văn bản, sẽ được huấn luyện trên tập dữ liệu ICDAR2003. Bạn có thể tải tập dữ liệu này tại [đây](#).
- Thiết lập module Text Detection:** Trong module này, chúng ta sẽ xây dựng một chương trình nhận ảnh đầu vào và trả về các tọa độ bao quanh toàn bộ văn bản có trong ảnh. Cụ thể, chúng ta sẽ sử dụng YOLOv11 để thực hiện công việc này. Bạn có thể xem chi tiết hơn về input/output của bài toán này qua hình sau:



Hình 2: Input/Output của bài toán Text Detection.

- Cài đặt các thư viện cần thiết:** Mã nguồn của YOLOv11 được phát triển dựa trên một số thư viện Python khác nhau. Vì vậy, để chạy YOLOv11, chúng ta cần cài đặt các gói thư viện mà YOLOv11 yêu cầu. YOLOv11 đã bao gồm một thư viện có tên ultralytics, và chúng ta có thể cài đặt thư viện này thông qua lệnh pip như sau:

```

1 import ultralytics
2
3 ultralytics.checks()

```

```

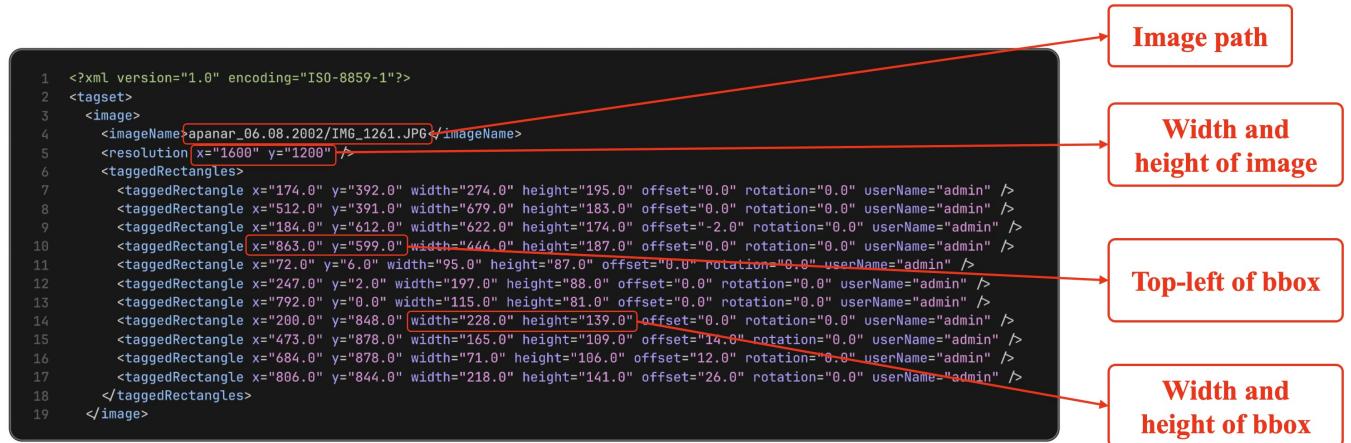
Ultralytics 8.3.51 🚀 Python-3.10.16 torch-2.3.0+cu118 CUDA:0 (NVIDIA GeForce RTX 3060, 11931MiB)
Setup complete ✅ (20 CPUs, 31.1 GB RAM, 604.6/915.3 GB disk)

```

Hình 3: Cài đặt thư viện ultralytics.

- Chuẩn bị bộ dữ liệu:** Cấu trúc thư mục và metadata của ICDAR2003 khác với yêu cầu của mô hình YOLOv11. Do đó, để có thể huấn luyện mô hình YOLOv11, bước đầu tiên là trích xuất thông tin và chuyển đổi chúng sang định dạng tương thích với YOLOv11.

- Cấu trúc file XML:** Dataset ICDAR2003 được lưu trữ trong file XML (`words.xml`). Dưới đây là một ví dụ về một hình ảnh trong file `words.xml`. Mỗi hình ảnh sẽ bao gồm các thông tin cơ bản như tên file, kích thước ảnh, và tọa độ của các bounding box cho từng từ xuất hiện trong ảnh.



Hình 4: Các thông tin quan trọng trong file XML.

- **Trích xuất thông tin từ file XML:** Chúng ta sẽ sử dụng mảng (array) để trích xuất các thông tin quan trọng từ file XML. Đối với các bounding box và label, chúng ta sẽ sử dụng mảng hai chiều để lưu trữ vì mỗi hình ảnh có thể có nhiều hơn một bounding box và label. Chúng ta chỉ chọn những từ có label là các ký tự từ 0-9 hoặc a-z để huấn luyện mô hình (dòng 17).

```

1 def extract_data_from_xml(root_dir):
2     xml_path = os.path.join(root_dir, 'words.xml')
3     tree = ET.parse(xml_path)
4     root = tree.getroot()
5
6     img_paths = []
7     img_sizes = []
8     img_labels = []
9     bboxes = []
10
11    for img in root:
12        bbs_of_img = []
13        labels_of_img = []
14
15        for bbs in img.findall('taggedRectangles'):
16            for bb in bbs:
17                # check non-alphabet and non-number
18                if not bb[0].text.isalnum():
19                    continue
20
21                if ' ' in bb[0].text.lower() or ' ' in bb[0].text.
lower():
22                    continue
23
24                bbs_of_img.append(
25                  [
26                    float(bb.attrib['x']),
27                    float(bb.attrib['y']),
28                    float(bb.attrib['width']),
29                    float(bb.attrib['height'])]

```

```

30             ]
31         )
32         labels_of_img.append(bb[0].text.lower())
33
34         img_path = os.path.join(root_dir, img[0].text)
35         img_paths.append(img_path)
36         img_sizes.append((int(img[1].attrib['x']), int(img[1].attrib
37             ['y'])))
38         bboxes.append(bbs_of_img)
39         img_labels.append(labels_of_img)
40
41     return img_paths, img_sizes, img_labels, bboxes
42 dataset_dir = 'SceneTrialTrain'
43 img_paths, img_sizes, img_labels, bboxes = extract_data_from_xml(
44     dataset_dir)
45

```

- **Chuyển đổi sang định dạng YOLOv11:** Sau khi đã trích xuất được các thông tin, bước tiếp theo là chuyển chúng sang định dạng YOLOv11.



Hình 5: Cấu trúc thư mục của YOLOv11.

Hình 5 minh họa cấu trúc thư mục của YOLOv11 mà chúng ta cần chuyển đổi sang. Cấu trúc này bao gồm 3 thư mục chính: train, test, và val. Trong mỗi thư mục, sẽ có 2 thư mục con: images và labels. Thư mục images sẽ chỉ chứa các hình ảnh, và mỗi file hình ảnh trong đó sẽ có một file tương ứng cùng tên (nhưng khác đuôi, ví dụ *.png và *.txt) trong thư mục labels. Các file label này sẽ chứa thông tin về label và tọa độ của từng bounding box.

```
1 0 0.8162692847124825 0.8125854993160054 0.13183730715287517 0.09028727770177838
2 0 0.5007012622720898 0.9432284541723667 0.8583450210378681 0.10259917920656635
```

Hình 6: Ví dụ về file label theo định dạng YOLOv11.

Hình 6 là ví dụ về một file label theo cấu trúc YOLOv11. Mỗi file này chứa 2 dòng, tương ứng với 2 bounding box xuất hiện trong ảnh. Mỗi dòng bao gồm 5 giá trị: `class_id`, `x`, `y`, `width`, `height`, trong đó các tọa độ của bounding box đã được chuẩn hóa trong khoảng [0, 1]. (`x`, `y`) là tọa độ của điểm trung tâm (center) của bounding box.

Sau khi hiểu rõ về định dạng YOLOv11, chúng ta sẽ tiến hành chuyển đổi thông tin sang định dạng này. Các dòng 13-16 trong mã nguồn sẽ chuyển đổi từ tọa độ top left (theo ICDAR2003) sang tọa độ center của YOLOv11, đồng thời chuẩn hóa các giá trị vào khoảng [0, 1]. Mục tiêu của mô hình YOLOv11 trong dự án này là nhận diện vị trí của văn bản, vì vậy chúng ta chỉ cần một class duy nhất là "text".

```
1 def convert_to_yolo_format(image_paths, image_sizes, bounding_boxes):
2     :
3     yolo_data = []
4
5     for image_path, image_size, bboxes in zip(image_paths,
6         image_sizes, bounding_boxes):
6         image_width, image_height = image_size
7
8         yolo_labels = []
9
10        for bbox in bboxes:
11            x, y, w, h = bbox
12
13            # Calculate normalized bounding box coordinates
14            center_x = (x + w / 2) / image_width
15            center_y = (y + h / 2) / image_height
16            normalized_width = w / image_width
17            normalized_height = h / image_height
18
19            # Because we only have one class, we set class_id to 0
20            class_id = 0
21
22            # Convert to YOLO format
23            yolo_label = f"{class_id} {center_x} {center_y} {
24                normalized_width} {normalized_height}"
25            yolo_labels.append(yolo_label)
26
27        yolo_data.append((image_path, yolo_labels))
28
29    return yolo_data
30
31
32 # Define class labels
33 class_labels = ["text"]
34
35
36 # Convert data into YOLO format
37 yolo_data = convert_to_yolo_format(image_paths, image_sizes,
38         bounding_boxes)
```

- **Lưu data vào folder mới:** Sau khi đã chuyển đổi sang format của YOLOv11, bây giờ

chúng ta sẽ lưu data này thành 1 folder mới để có thể tiết kiệm thời gian cho những training sau.

```

1 def save_data(data, src_img_dir, save_dir):
2     # Create folder if not exists
3     os.makedirs(save_dir, exist_ok=True)
4
5     # Make images and labels folder
6     os.makedirs(os.path.join(save_dir, "images"), exist_ok=True)
7     os.makedirs(os.path.join(save_dir, "labels"), exist_ok=True)
8
9     for image_path, yolo_labels in data:
10        # Copy image to images folder
11        shutil.copy(
12            os.path.join(src_img_dir, image_path), os.path.join(
13                save_dir, "images")
14        )
15
16        # Save labels to labels folder
17        image_name = os.path.basename(image_path)
18        image_name = os.path.splitext(image_name)[0]
19
20        with open(os.path.join(save_dir, "labels", f"{image_name}.txt"), "w") as f:
21            for label in yolo_labels:
22                f.write(f"{label}\n")
23
24    seed = 0
25    val_size = 0.2
26    test_size = 0.125
27    is_shuffle = True
28    train_data, test_data = train_test_split(
29        yolo_data,
30        test_size=val_size,
31        random_state=seed,
32        shuffle=is_shuffle,
33    )
34    test_data, val_data = train_test_split(
35        test_data,
36        test_size=test_size,
37        random_state=seed,
38        shuffle=is_shuffle,
39    )
40    save_yolo_data_dir = "datasets/yolo_data"
41    os.makedirs(save_yolo_data_dir, exist_ok=True)
42    save_train_dir = os.path.join(save_yolo_data_dir, "train")
43    save_val_dir = os.path.join(save_yolo_data_dir, "val")
44    save_test_dir = os.path.join(save_yolo_data_dir, "test")
45
46    save_data(train_data, dataset_dir, save_train_dir)
47    save_data(test_data, dataset_dir, save_val_dir)
48    save_data(val_data, dataset_dir, save_test_dir)
49

```

- **Tạo file data.yaml để quản lý chung:** Để thuận tiện trong việc truy cập các đường dẫn của các thư mục train, test và val, cũng như số lượng và tên các class, chúng ta cần

tạo một file `data.yaml` để lưu trữ những thông tin này.

`data.yml` ×

```
1 names:
2 - text
3 nc: 1
4 path: yolo_data
5 test: test/images
6 train: train/images
7 val: val/images
```

Hình 7: Ví dụ về file `data.yaml`.

Hình 7 là ví dụ về file `data.yaml` trong bài toán của chúng ta. Trong file này, ta có các thành phần sau:

- **names:** tên của các class.
- **nc:** số lượng các class.
- **path:** đường dẫn đến thư mục gốc.
- **train, test, val:** đường dẫn đến các thư mục tương ứng.

Chúng ta cũng có thể tạo file `data.yaml` thông qua đoạn mã dưới đây:

```
1 # Create data.yaml file
2 data_yaml = {
3     "path": "./datasets/yolo_data",
4     "train": "train/images",
5     "test": "test/images",
6     "val": "val/images",
7     "nc": 1,
8     "names": class_labels,
9 }
10
11 yolo_yaml_path = os.path.join(save_yolo_data_dir, "data.yaml")
12 with open(yolo_yaml_path, "w") as f:
13     yaml.dump(data_yaml, f, default_flow_style=False)
14
```

- (c) **Training:** Sau khi đã chuẩn bị toàn bộ data, chúng ta đã có thể bắt đầu training model. Trong project này ta sẽ sử dụng YOLOs làm model chính, ngoài ra các bạn có thể sử dụng các phiên bản khác thông qua github của ultralytics tại [đây](#).

Ta sẽ train model thông qua đoạn code sau:

```
1 from ultralytics import YOLO
2
3 # Load a model
4 model = YOLO("yolo11m.pt")
5
6 # Train model
```

```

7 results = model.train(
8     data=yolo_yaml_path,
9     epochs=100,
10    imgsz=640,
11    cache=True,
12    patience=20,
13    plots=True,
14 )
15

```

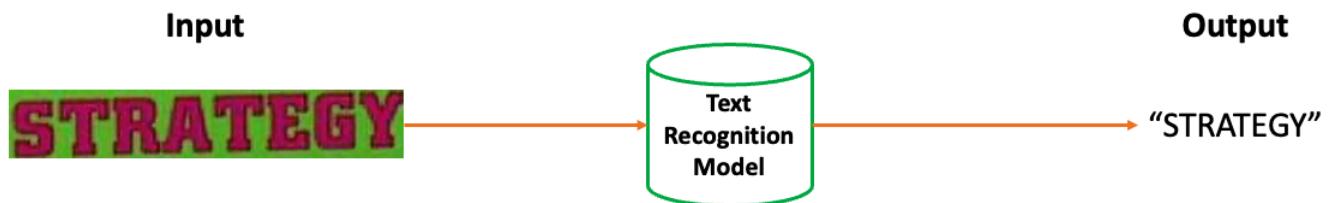
(d) **Evaluation:** Bây giờ chúng ta sẽ eval model đã train.

```

1 from ultralytics import YOLO
2
3 model_path = "runs/detect/train/weights/best.pt"
4 model = YOLO(model_path)
5
6 metrics = model.val()
7

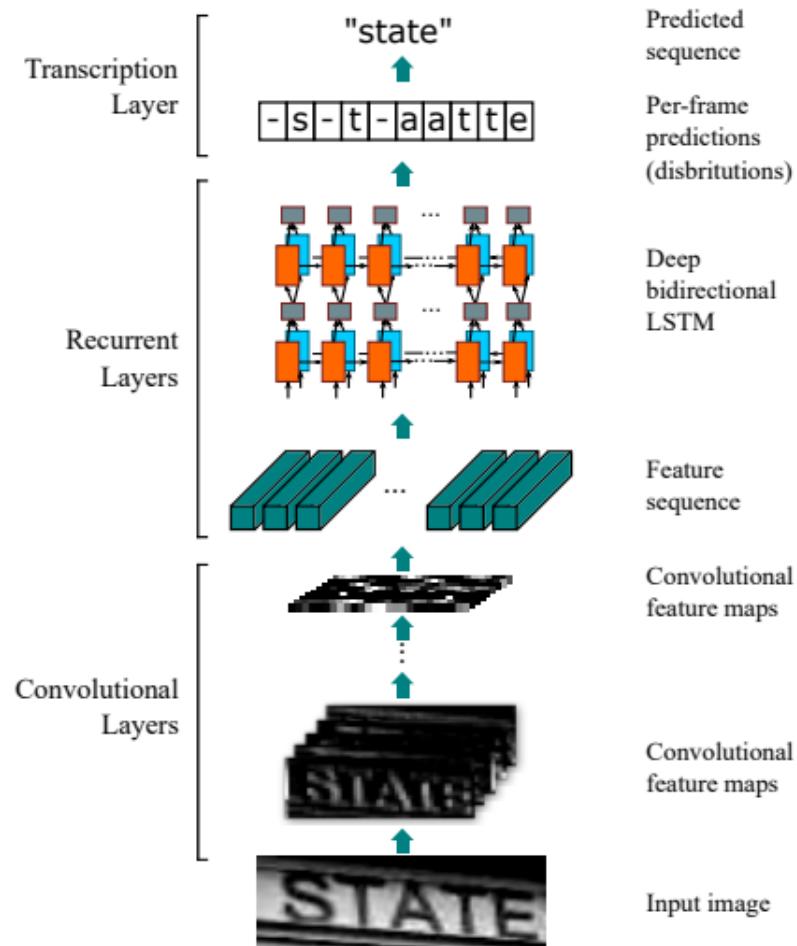
```

3. **Cài đặt module Text Recognition:** Trong module này, chúng ta sẽ xây dựng một chương trình nhận vào một ảnh chỉ chứa văn bản và trả về nội dung văn bản dưới dạng chuỗi (string). Các bạn có thể tham khảo rõ hơn về đầu vào và đầu ra (I/O) của bài toán này qua hình dưới đây:



Hình 8: Input/Output của bài toán Text Recognition

Để giải quyết bài toán này, chúng ta sẽ chọn mô hình Convolutional Recurrent Neural Networks (CRNN), một mô hình cơ bản nhưng hiệu quả. Đây là một trong những mô hình đầu tiên được phát triển nhằm giải quyết các bài toán dữ liệu vừa ở dạng hình ảnh vừa có tính chuỗi. Cấu trúc của mô hình CRNN như sau:



Hình 9: Mô phỏng kiến trúc của mạng CRNN

Mô hình này kết hợp giữa CNN (Convolutional Neural Networks) và RNN (Recurrent Neural Networks) để có thể trích xuất đặc trưng từ hình ảnh và chuỗi văn bản trong ảnh. Đồng thời, CRNN còn sử dụng một hàm mất mát đặc biệt, CTC Loss, nhằm cải thiện kết quả của mô hình. Từ các thông tin trên, chúng ta sẽ xây dựng module này như sau:

(a) Import các thư viện cần thiết:

```

1 import os
2 import random
3 import time
4 import xml.etree.ElementTree as ET
5
6 import cv2
7 import matplotlib.pyplot as plt
8 import numpy as np
9 import timm
10 import torch
11 import torch.nn as nn
12 import torchvision
13 from PIL import Image
14 from sklearn.model_selection import train_test_split

```

```

15 from torch.nn import functional as F
16 from torch.utils.data import DataLoader, Dataset
17 from torchvision import transforms
18

```

- (b) **Chuẩn bị bộ dữ liệu:** Để huấn luyện mô hình Text Recognition, chúng ta cần có các cặp ảnh chứa chữ văn bản và label văn bản dạng string tương ứng. Song, dataset ICDAR2003 hiện tại chưa thỏa mãn điều này. Chính vì vậy, dựa vào file label của bộ ICDAR2003, ta sẽ cài đặt chương trình trích xuất thành một bộ dữ liệu dành cho bài toán Text Recognition. Tương tự như module trước, ta dùng hàm `extract_data_from_xml()` để lấy thông tin ảnh bounding box như sau:

```

1 def extract_data_from_xml(root_dir):
2     xml_path = os.path.join(root_dir, "words.xml")
3     tree = ET.parse(xml_path)
4     root = tree.getroot()
5
6     img_paths = []
7     img_sizes = []
8     img_labels = []
9     bboxes = []
10
11    for img in root:
12        bbs_of_img = []
13        labels_of_img = []
14
15        for bbs in img.findall("taggedRectangles"):
16            for bb in bbs:
17                # check non-alphabet and non-number
18                if not bb[0].text.isalnum():
19                    continue
20
21                if "e" in bb[0].text.lower() or "n" in bb[0].text.lower():
22                    continue
23
24                bbs_of_img.append(
25                    [
26                        float(bb.attrib["x"]),
27                        float(bb.attrib["y"]),
28                        float(bb.attrib["width"]),
29                        float(bb.attrib["height"]),
30                    ]
31                )
32                labels_of_img.append(bb[0].text.lower())
33
34    img_path = os.path.join(root_dir, img[0].text)
35    img_paths.append(img_path)
36    img_sizes.append((int(img[1].attrib["x"]), int(img[1].attrib["y"])))
37
38    bboxes.append(bbs_of_img)
39    img_labels.append(labels_of_img)
40
41    return img_paths, img_sizes, img_labels, bboxes

```

```

42 dataset_dir = "datasets/SceneTrialTrain"
43 img_paths, img_sizes, img_labels, bboxes = extract_data_from_xml(
    dataset_dir)
44

```

Với danh sách các ảnh và bounding box ứng với từng ảnh, ta sẽ xây dựng một hàm để cắt các bounding box thành các ảnh riêng biệt chỉ chứa văn bản. Nội dung văn bản từ các bounding box này sẽ được sử dụng làm label và lưu trữ vào một file .txt riêng biệt. Hàm thực hiện công việc này có nội dung như sau:

```

1 def split_bounding_boxes(img_paths, img_labels, bboxes, save_dir):
2     os.makedirs(save_dir, exist_ok=True)
3
4     count = 0
5     labels = [] # List to store labels
6
7     for img_path, img_label, bbs in zip(img_paths, img_labels, bboxes):
8         img = Image.open(img_path)
9
10        for label, bb in zip(img_label, bbs):
11            # Crop image
12            cropped_img = img.crop((bb[0], bb[1], bb[0] + bb[2], bb[1] +
13 bb[3]))
14
15            # filter out if 90% of the cropped image is black or white
16            if np.mean(cropped_img) < 35 or np.mean(cropped_img) > 220:
17                continue
18
19            if cropped_img.size[0] < 10 or cropped_img.size[1] < 10:
20                continue
21
22            # Save image
23            filename = f"{count:06d}.jpg"
24            cropped_img.save(os.path.join(save_dir, filename))
25
26            new_img_path = os.path.join(save_dir, filename)
27
28            label = new_img_path + "\t" + label
29
30            labels.append(label) # Append label to the list
31
32            count += 1
33
34        print(f"Created {count} images")
35
36        # Write labels to a text file
37        with open(os.path.join(save_dir, "labels.txt"), "w") as f:
38            for label in labels:
39                f.write(f"{label}\n")
40
41 save_dir = "datasets/ocr_dataset"
42 split_bounding_boxes(img_paths, img_labels, bboxes, save_dir)

```

Khi chạy đoạn code trên, ta sẽ có một thư mục dataset mới dành cho bài Text Recognition,

các ảnh này sẽ có dạng như sau:



Hình 10: Một số mẫu dữ liệu từ bộ dữ liệu OCR

- (c) **Đọc dữ liệu:** Với thư mục dữ liệu mới, ta viết chương trình đọc và lưu trữ vào hai list rỗng, một list dùng để chứa đường dẫn ảnh và một list để chứa string label tương ứng như sau:

```

1 root_dir = save_dir
2
3 img_paths = []
4 labels = []
5
6 # Read labels from text file
7 with open(os.path.join(root_dir, "labels.txt"), "r") as f:
8     for label in f:
9         labels.append(label.strip().split("\t")[1])
10        img_paths.append(label.strip().split("\t")[0])
11
12 print(f"Total images: {len(img_paths)}")
13

```

- (d) **Xây dựng bộ từ vựng (vocabulary):** Dối với loại bài toán này, để mô hình CRNN có thể xuất ra văn bản từ hình ảnh, chúng ta cần xác định một bộ từ vựng trước. Mô hình CRNN sẽ thực hiện dự đoán ở cấp độ ký tự, vì vậy bộ từ vựng sẽ bao gồm các ký tự có mặt trong dữ liệu. Do đó, chương trình của chúng ta sẽ như sau:

```

1 letters = [char.split(".")[0].lower() for char in labels]
2 letters = "".join(letters)
3 letters = sorted(list(set(list(letters))))
4
5 # create a string of all characters in the dataset
6 chars = "".join(letters)
7
8 # for "blank" character
9 blank_char = "-"
10 chars += blank_char
11 vocab_size = len(chars)
12
13 print(f"Vocab: {chars}")
14 print(f"Vocab size: {vocab_size}")
15

```

Trong đoạn mã trên, chúng ta duyệt qua tất cả các nhãn trong bộ dữ liệu và chọn ra các ký tự xuất hiện để tạo thành bộ từ vựng. Cần lưu ý rằng, khi sử dụng CTC Loss, ta phải định nghĩa một ký tự "blank". Dựa trên bộ từ vựng này, chúng ta sẽ xây dựng một từ điển để lưu trữ các ký tự và mã số thứ tự tương ứng như sau:

```

1 char_to_idx = {char: idx + 1 for idx, char in enumerate(sorted(chars))}
2 idx_to_char = {index: char for char, index in char_to_idx.items()}
3

```

Cuối cùng, ta xây dựng hàm dùng để chuyển đổi một string thành vector các kí tự được đại diện bằng mã số của chúng (token):

```

1 max_label_len = max([len(label) for label in labels])
2
3 def encode(label, char_to_idx, max_label_len):
4     encoded_labels = torch.tensor(
5         [char_to_idx[char] for char in label],
6         dtype=torch.int32
7     )
8     label_len = len(encoded_labels)
9     lengths = torch.tensor(
10        label_len,
11        dtype=torch.int32
12    )
13     padded_labels = F.pad(
14         encoded_labels,
15         (0, max_label_len - label_len),
16         value=0
17     )
18
19     return padded_labels, lengths
20

```

Khi chuyển đổi các nhãn thành tensor, cần đảm bảo rằng tất cả các tensor đều có kích thước đồng nhất để tránh lỗi khi sử dụng DataLoader. Tuy nhiên, các nhãn có độ dài khác nhau, vì vậy chúng ta cần xác định một chiều dài chung (max_label_len) và thực hiện padding nếu cần thiết để đảm bảo điều này (lưu ý rằng mã số của ký tự padding là 0).

```

1 def decode(encoded_sequences, idx_to_char, blank_char="-"):
2     decoded_sequences = []
3
4     for seq in encoded_sequences:
5         decoded_label = []
6         prev_char = None # To track the previous character
7
8         for token in seq:
9             if token != 0: # Ignore padding (token = 0)
10                 char = idx_to_char[token.item()]
11                 # Append the character if it's not a blank or the same
12                 # as the previous character
13                 if char != blank_char:
14                     if char != prev_char or prev_char == blank_char:

```

```

14             decoded_label.append(char)
15             prev_char = char # Update previous character
16
17         decoded_sequences.append("".join(decoded_label))
18
19     print(f"From {encoded_sequences} to {decoded_sequences}")
20
21     return decoded_sequences
22

```

Ngược lại với hàm `encode()`, ta cũng sẽ xây dựng một hàm để đổi ngược lại tensor các token thành dạng string của chúng như trên.

- (e) **Xây dựng hàm tiền xử lý dữ liệu:** Để kết quả huấn luyện được tốt hơn, ta sẽ xây dựng hàm tiền xử lý dữ liệu đầu vào như sau:

```

1 data_transforms = {
2     "train": transforms.Compose(
3         [
4             transforms.Resize((100, 420)),
5             transforms.ColorJitter(
6                 brightness=0.5,
7                 contrast=0.5,
8                 saturation=0.5,
9             ),
10            transforms.Grayscale(
11                num_output_channels=1,
12            ),
13            transforms.GaussianBlur(3),
14            transforms.RandomAffine(
15                degrees=1,
16                shear=1,
17            ),
18            transforms.RandomPerspective(
19                distortion_scale=0.3,
20                p=0.5,
21                interpolation=3,
22            ),
23            transforms.RandomRotation(degrees=2),
24            transforms.ToTensor(),
25            transforms.Normalize((0.5,), (0.5,)),
26        ]
27    ),
28    "val": transforms.Compose(
29        [
30            transforms.Resize((100, 420)),
31            transforms.Grayscale(num_output_channels=1),
32            transforms.ToTensor(),
33            transforms.Normalize((0.5,), (0.5,)),
34        ]
35    ),
36 }
37

```

Lưu ý rằng, trong quá trình huấn luyện, chúng ta cần áp dụng một số kỹ thuật tăng cường dữ liệu để cải thiện hiệu quả của mô hình. Do đó, sẽ có hai hàm chuyển đổi (`transforms`)

riêng biệt cho hai giai đoạn huấn luyện và đánh giá.

- (f) **Chia bộ dữ liệu train, val, test:** Với dữ liệu đã đọc, ta chia ba bộ dữ liệu train, val, test với tỉ lệ 7:2:1 như sau:

```

1 seed = 0
2 val_size = 0.1
3 test_size = 0.1
4 is_shuffle = True
5
6 X_train, X_val, y_train, y_val = train_test_split(
7     img_paths,
8     labels,
9     test_size=val_size,
10    random_state=seed,
11    shuffle=is_shuffle,
12 )
13
14 X_train, X_test, y_train, y_test = train_test_split(
15     X_train,
16     y_train,
17     test_size=test_size,
18     random_state=seed,
19     shuffle=is_shuffle,
20 )
21

```

- (g) **Xây dựng class pytorch datasets:** Ta xây dựng pytorch dataset cho bộ dữ liệu Text Recognition như sau:

```

1 class STRDataset(Dataset):
2     def __init__(
3         self,
4         X,
5         y,
6         char_to_idx,
7         max_label_len,
8         label_encoder=None,
9         transform=None,
10     ):
11         self.transform = transform
12         self.img_paths = X
13         self.labels = y
14         self.char_to_idx = char_to_idx
15         self.max_label_len = max_label_len
16         self.label_encoder = label_encoder
17
18     def __len__(self):
19         return len(self.img_paths)
20
21     def __getitem__(self, idx):
22         label = self.labels[idx]
23         img_path = self.img_paths[idx]
24         img = Image.open(img_path).convert("RGB")
25
26         if self.transform:

```

```
27         img = self.transform(img)
28
29     if self.label_encoder:
30         encoded_label, label_len = self.label_encoder(
31             label, self.char_to_idx, self.max_label_len
32         )
33     return img, encoded_label, label_len
34
```

- (h) **Xây dựng dataloader:** Với pytorch dataset đã định nghĩa, ta tạo dataloader cho ba bộ train, val, test như sau:

```
1 train_dataset = STRDataset(
2     X_train,
3     y_train,
4     char_to_idx=char_to_idx,
5     max_label_len=max_label_len,
6     label_encoder=encode,
7     transform=data_transforms["train"],
8 )
9 val_dataset = STRDataset(
10    X_val,
11    y_val,
12    char_to_idx=char_to_idx,
13    max_label_len=max_label_len,
14    label_encoder=encode,
15    transform=data_transforms["val"],
16 )
17 test_dataset = STRDataset(
18    X_test,
19    y_test,
20    char_to_idx=char_to_idx,
21    max_label_len=max_label_len,
22    label_encoder=encode,
23    transform=data_transforms["val"],
24 )
25
26 train_batch_size = 64
27 test_batch_size = 64 * 2
28
29 train_loader = DataLoader(
30     train_dataset,
31     batch_size=train_batch_size,
32     shuffle=True,
33 )
34 val_loader = DataLoader(
35     val_dataset,
36     batch_size=test_batch_size,
37     shuffle=False,
38 )
39 test_loader = DataLoader(
40     test_dataset,
41     batch_size=test_batch_size,
42     shuffle=False,
43 )
44
```

(i) **Xây dựng model:** Tại đây, chúng ta sẽ bắt đầu triển khai mô hình CRNN. Cụ thể, chúng ta sẽ kết hợp mô hình ResNet101 đã được huấn luyện trước với các lớp Bi-LSTM. Đặc trưng được trích xuất từ ResNet sẽ được biến đổi để đưa vào Bi-LSTM. Sau đó, chúng ta sử dụng trạng thái ẩn cuối cùng của Bi-LSTM để dự đoán chuỗi ký tự trong ảnh. Cách cài đặt sẽ như sau:

```

1  class CRNN(nn.Module):
2      def __init__(self, vocab_size, hidden_size, n_layers, dropout=0.2,
3                   unfreeze_layers=3):
4          super(CRNN, self).__init__()
5
6          backbone = timm.create_model("resnet152", in_chans=1, pretrained=True)
7          modules = list(backbone.children())[:-2]
8          modules.append(nn.AdaptiveAvgPool2d((1, None)))
9          self.backbone = nn.Sequential(*modules)
10
11         # Unfreeze the last few layers
12         for parameter in self.backbone[-unfreeze_layers: ].parameters():
13             parameter.requires_grad = True
14
15         self.mapSeq = nn.Sequential(
16             nn.Linear(2048, 512), nn.ReLU(), nn.Dropout(dropout))
17
18
19         self.gru = nn.GRU(
20             512,
21             hidden_size,
22             n_layers,
23             bidirectional=True,
24             batch_first=True,
25             dropout=dropout if n_layers > 1 else 0,
26         )
27         self.layer_norm = nn.LayerNorm(hidden_size * 2)
28
29         self.out = nn.Sequential(
30             nn.Linear(hidden_size * 2, vocab_size), nn.LogSoftmax(dim=2)
31         )
32
33
34     @torch.autocast(device_type="cuda")
35     def forward(self, x):
36         x = self.backbone(x)
37         x = x.permute(0, 3, 1, 2)
38         x = x.view(x.size(0), x.size(1), -1)  # Flatten the feature map
39         x = self.mapSeq(x)
40         x, _ = self.gru(x)
41         x = self.layer_norm(x)
42         x = self.out(x)
43         x = x.permute(1, 0, 2)  # Based on CTC
44
45     return x
46

```

Sau khi đã định nghĩa class cho CRNN, ta khai báo mô hình như sau:

```

1 hidden_size = 256
2 n_layers = 3
3 dropout_prob = 0.2
4 unfreeze_layers = 3
5 device = "cuda" if torch.cuda.is_available() else "cpu"
6
7 model = CRNN(
8     vocab_size=vocab_size,
9     hidden_size=hidden_size,
10    n_layers=n_layers,
11    dropout=dropout_prob,
12    unfreeze_layers=unfreeze_layers,
13 ).to(device)
14

```

- (j) **Xây dựng hàm train và evaluate:** Ta định nghĩa hàm dùng để huấn luyện mô hình và đánh giá mô hình như sau:

```

1 def evaluate(model, dataloader, criterion, device):
2     model.eval()
3     losses = []
4     with torch.no_grad():
5         for inputs, labels, labels_len in dataloader:
6             inputs = inputs.to(device)
7             labels = labels.to(device)
8             labels_len = labels_len.to(device)
9
10            outputs = model(inputs)
11            logits_lens = torch.full(
12                size=(outputs.size(1),), fill_value=outputs.size(0),
13                dtype=torch.long
14            ).to(device)
15
16            loss = criterion(outputs, labels, logits_lens, labels_len)
17            losses.append(loss.item())
18
19            loss = sum(losses) / len(losses)
20
21        return loss
22
23 def fit(
24     model, train_loader, val_loader, criterion, optimizer, scheduler,
25     device, epochs
26 ) :
27     train_losses = []
28     val_losses = []
29
30     for epoch in range(epochs):
31         start = time.time()
32
33         batch_train_losses = []
34
35         model.train()
36         for idx, (inputs, labels, labels_len) in enumerate(train_loader)
37         :
38             inputs = inputs.to(device)
39
40             outputs = model(inputs)
41             loss = criterion(outputs, labels, logits_lens, labels_len)
42
43             batch_train_losses.append(loss.item())
44
45             loss = sum(batch_train_losses) / len(batch_train_losses)
46
47             optimizer.zero_grad()
48             loss.backward()
49             optimizer.step()
50
51             end = time.time()
52             print(f'Epoch {epoch+1}/{epochs}, Step {idx+1}/{len(train_loader)}, Loss: {loss:.4f}, Time: {end - start:.2f} s')
53
54         scheduler.step()
55
56         val_loss = evaluate(model, val_loader, criterion, device)
57
58         train_losses.append(loss)
59         val_losses.append(val_loss)
60
61     return train_losses, val_losses

```

```

36         labels = labels.to(device)
37         labels_len = labels_len.to(device)
38
39         optimizer.zero_grad()
40
41         outputs = model(inputs)
42
43         logits_lens = torch.full(
44             size=(outputs.size(1),),
45             fill_value=outputs.size(0),
46             dtype=torch.long,
47         ).to(device)
48
49         loss = criterion(outputs, labels, logits_lens, labels_len)
50
51         loss.backward()
52         torch.nn.utils.clip_grad_norm_(model.parameters(), 5)
53         optimizer.step()
54
55         batch_train_losses.append(loss.item())
56
57         train_loss = sum(batch_train_losses) / len(batch_train_losses)
58         train_losses.append(train_loss)
59
60         val_loss = evaluate(model, val_loader, criterion, device)
61         val_losses.append(val_loss)
62
63         print(
64             f"EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tVal loss: {val_loss:.4f}\tTime: {time.time() - start:.2f} seconds"
65         )
66
67         scheduler.step()
68
69     return train_losses, val_losses
70

```

- (k) **Khai báo hàm loss, optimizer và scheduler:** Sau khi xây dựng mô hình, chúng ta cần khai báo hàm loss, optimizer và learning rate scheduler. Như đã đề cập ở phần đầu, chúng ta sẽ sử dụng hàm CTC Loss cho bài toán này. Learning rate scheduler sẽ giúp giảm learning rate trong các epoch sau để cải thiện hiệu quả huấn luyện.

```

1 epochs = 100
2 lr = 5e-4
3 weight_decay = 1e-5
4 scheduler_step_size = epochs * 0.5
5
6 criterion = nn.CTCLoss(
7     blank=char_to_idx[blank_char],
8     zero_infinity=True,
9     reduction="mean",
10 )
11 optimizer = torch.optim.Adam(
12     model.parameters(),
13     lr=lr,
14     weight_decay=weight_decay,

```

```

15 )
16 scheduler = torch.optim.lr_scheduler.StepLR(
17     optimizer, step_size=scheduler_step_size, gamma=0.1
18 )
19

```

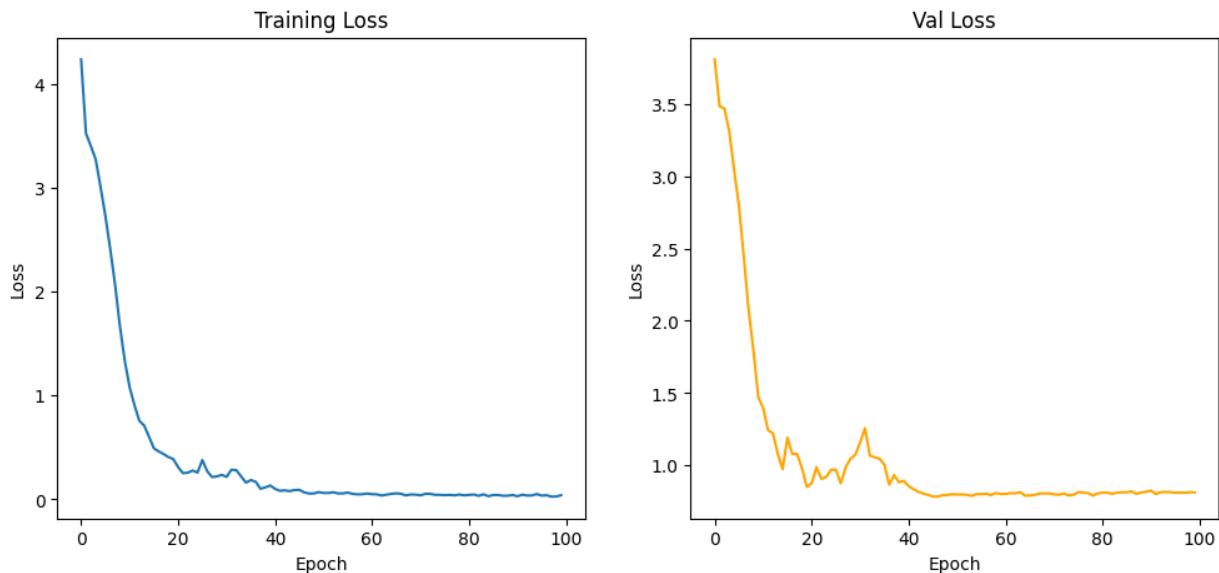
- (l) **Training:** Ta gọi hàm `fit()` kèm các tham số phù hợp để bắt đầu quá trình huấn luyện:

```

1 train_losses, val_losses = fit(
2     model, train_loader, val_loader, criterion, optimizer, scheduler,
3     device, epochs
4 )

```

Khi quá trình huấn luyện kết thúc, ta có thể dùng kết quả loss ghi nhận được qua từng epoch trên hai tập train và val lên đồ thị như sau:



Hình 11: Kết quả huấn luyện của mô hình CRNN trên hai tập train và val

- (m) **Evaluation:** Sau khi quá trình huấn luyện hoàn tất, ta đánh giá mô hình trên hai tập val và test:

```

1 val_loss = evaluate(model, val_loader, criterion, device)
2 test_loss = evaluate(model, test_loader, criterion, device)
3
4 print("Evaluation on val/test dataset")
5 print("Val loss: ", val_loss)
6 print("Test loss: ", test_loss)
7

```

- (n) **Lưu model:** Cuối cùng, chúng ta lưu lại model đã train được vào máy.

```

1 save_model_path = "ocr_crnn.pt"
2 torch.save(model.state_dict(), save_model_path)
3

```

4. **Cài đặt toàn bộ pipeline:** Sau khi đã hoàn tất xây dựng hai mô hình Text Detection và Text Recognition, chúng ta sẽ xây dựng chương trình Scene Text Recognition hoàn chỉnh thông qua việc kết hợp hai mô hình trên lại. Các bước thực hiện như sau:

(a) **Import các thư viện cần thiết:**

```

1 %pip install ultralytics
2 import ultralytics
3 ultralytics.checks()
4
5 import os
6 import numpy as np
7 import timm
8 import matplotlib.pyplot as plt
9
10 import torch
11 import torch.nn as nn
12 from torchvision import models
13 from torchvision import transforms
14 from PIL import Image
15
16 from ultralytics import YOLO
17

```

(b) **Load các mô hình đã huấn luyện:** Khởi tạo hai model về Text Detection và Text Recognition đã huấn luyện:

i. **Text Detection:**

```

1 text_det_model_path = 'runs/detect/train/weights/best.pt'
2 yolo = YOLO(text_det_model_path)
3

```

ii. **Text Recognition:**

```

1 class CRNN(nn.Module):
2     def __init__(self, vocab_size, hidden_size, n_layers, dropout
3                  =0.2, unfreeze_layers=3):
4         super(CRNN, self).__init__()
5
6         backbone = timm.create_model(
7             'resnet101',
8             in_chans=1,
9             pretrained=True
10        )
11        modules = list(backbone.children())[:-2]
11        modules.append(nn.AdaptiveAvgPool2d((1, None)))

```

```
12         self.backbone = nn.Sequential(*modules)
13
14     # Unfreeze the last few layers
15     for parameter in self.backbone[-unfreeze_layers:].parameters
16     ():
17         parameter.requires_grad = True
18
19     self.mapSeq = nn.Sequential(
20         nn.Linear(2048, 512),
21         nn.ReLU(),
22         nn.Dropout(dropout)
23     )
24
25     self.lstm = nn.LSTM(
26         512, hidden_size,
27         n_layers, bidirectional=True, batch_first=True,
28         dropout=dropout if n_layers > 1 else 0
29     )
30     self.layer_norm = nn.LayerNorm(hidden_size * 2)
31
32     self.out = nn.Sequential(
33         nn.Linear(hidden_size * 2, vocab_size),
34         nn.LogSoftmax(dim=2)
35     )
36
37     def forward(self, x):
38         x = self.backbone(x)
39         x = x.permute(0, 3, 1, 2)
40         x = x.view(x.size(0), x.size(1), -1) # Flatten the feature
41         map
42         x = self.mapSeq(x)
43         x, _ = self.lstm(x)
44         x = self.layer_norm(x)
45         x = self.out(x)
46         x = x.permute(1, 0, 2) # Based on CTC
47
48     chars = '0123456789abcdefghijklmnopqrstuvwxyz-'
49     vocab_size = len(chars)
50     char_to_idx = {char: idx + 1 for idx, char in enumerate(sorted(chars
51     ))}
52     idx_to_char = {idx: char for char, idx in char_to_idx.items()}
53
54     hidden_size = 256
55     n_layers = 3
56     dropout_prob = 0.2
57     unfreeze_layers=3
58     device = 'cuda' if torch.cuda.is_available() else 'cpu'
59     model_path = 'ocr_crnn_base_best.pt'
60
61     crnn_model = CRNN(
62         vocab_size=vocab_size,
63         hidden_size=hidden_size,
64         n_layers=n_layers,
65         dropout=dropout_prob,
66         unfreeze_layers=unfreeze_layers
67     ).to(device)
68     crnn_model.load_state_dict(torch.load(model_path))
```

68

- (c) **Xây dựng hàm decode:** Khởi tạo hàm `decode()` để chuyển kết quả dự đoán của mô hình thành string:

```

1 def decode(encoded_sequences, idx_to_char, blank_char='-' ):
2     decoded_sequences = []
3
4     for seq in encoded_sequences:
5         decoded_label = []
6         for idx, token in enumerate(seq):
7             if token != 0:
8                 char = idx_to_char[token.item()]
9                 if char != blank_char:
10                     decoded_label.append(char)
11
12     decoded_sequences.append(''.join(decoded_label))
13
14 return decoded_sequences
15

```

- (d) **Xây dựng hàm Text Detection:** Ta dùng hàm này để phát hiện vị trí của tất cả văn bản xuất hiện trong một ảnh sử dụng mô hình YOLOv11 đã huấn luyện:

```

1 def text_detection(img_path, text_det_model):
2     text_det_results = text_det_model(img_path, verbose=False)[0]
3
4     bboxes = text_det_results.boxes.xyxy.tolist()
5     classes = text_det_results.boxes.cls.tolist()
6     names = text_det_results.names
7     confs = text_det_results.boxes.conf.tolist()
8
9     return bboxes, classes, names, confs
10

```

- (e) **Xây dựng hàm nhận diện văn bản:** Ta dùng hàm này để nhận diện văn bản bên trong một ảnh bắt kì với model CRNN đã huấn luyện:

```

1 def text_recognition(img, data_transforms, text_reg_model, idx_to_char,
2 device):
3     transformed_image = data_transforms(img)
4     transformed_image = transformed_image.unsqueeze(0).to(device)
5     text_reg_model.eval()
6     with torch.no_grad():
7         logits = text_reg_model(transformed_image).detach().cpu()
8         text = decode(logits.permute(1, 0, 2).argmax(2), idx_to_char)
9
10    return text

```

- (f) **Xây dựng hàm trực quan hóa kết quả dự đoán:**

```

1 def visualize_detections(img, detections):
2     plt.figure(figsize=(12, 8))
3     plt.imshow(img)
4     plt.axis('off')
5
6     for bbox, detected_class, confidence, transcribed_text in detections:
7         x1, y1, x2, y2 = bbox
8         plt.gca().add_patch(plt.Rectangle((x1, y1), x2 - x1, y2 - y1,
9                                         fill=False, edgecolor='red', linewidth=2))
10        plt.text(
11            x1, y1 - 10, f'{detected_class} ({confidence:.2f}): {transcribed_text}',
12            fontsize=9, bbox=dict(facecolor='red', alpha=0.5)
13        )
14
15    plt.show()

```

- (g) **Xây dựng hàm tiền xử lý liệu:** Ta sử dụng lại hàm transforms đã triển khai cho mô hình CRNN:

```

1 data_transforms = {
2     'train': transforms.Compose([
3         transforms.Resize((100, 420)),
4         transforms.ColorJitter(brightness=0.5, contrast=0.5, saturation=0.5),
5         transforms.Grayscale(num_output_channels=1),
6         transforms.GaussianBlur(3),
7         transforms.RandomAffine(degrees=1, shear=1),
8         transforms.RandomPerspective(distortion_scale=0.2, p=0.3,
9             interpolation=3),
10        transforms.RandomRotation(degrees=2),
11        transforms.ToTensor(),
12        transforms.Normalize((0.5,), (0.5,)),
13    ]),
14    'val': transforms.Compose([
15        transforms.Resize((100, 420)),
16        transforms.Grayscale(num_output_channels=1),
17        transforms.ToTensor(),
18        transforms.Normalize((0.5,), (0.5,)),
19    ]),
20 }

```

- (h) **Xây dựng hàm dự đoán:** Cuối cùng, ta xây dựng hàm predict() để sử dụng chương trình STR:

```

1 def predict(img_path, data_transforms, text_det_model, text_reg_model,
2            idx_to_char, device):
3     # Detection
4     bboxes, classes, names, confs = text_detection(img_path,
5                                                    text_det_model)

```

```
4
5      # Load the image
6      img = Image.open(img_path)
7
8      predictions = []
9
10     # Iterate through the results
11     for bbox, cls, conf in zip(bboxes, classes, confs):
12         x1, y1, x2, y2 = bbox
13         confidence = conf
14         detected_class = cls
15         name = names[int(cls)]
16
17         # Extract the detected object and crop it
18         cropped_image = img.crop((x1, y1, x2, y2))
19
20         transcribed_text = text_recognition(
21             cropped_image,
22             data_transforms,
23             text_reg_model,
24             idx_to_char,
25             device
26         )
27
28         predictions.append((bbox, name, confidence, transcribed_text))
29
30     visualize_detections(img, predictions)
31
32     return predictions
33
```



Hình 12: Một số kết quả của mô hình mà chúng ta đã xây dựng

Phần III: Câu hỏi trắc nghiệm

1. Bài toán Scene Text Recognition nhằm mục đích gì??

- (a) Nhận diện văn bản trong hình ảnh.
- (b) Phân loại hình ảnh.
- (c) Xác định đối tượng trong video.
- (d) Tính toán biểu đồ dữ liệu.

2. Ứng dụng phổ biến của Scene Text Recognition bao gồm:

- (a) Tự động chụp ảnh selfie.
- (b) Dịch ngôn ngữ tự động từ văn bản trong hình ảnh.
- (c) Nhận diện khuôn mặt.
- (d) Phân loại thẻ loại sách.

3. Mục tiêu của Detector trong bài Scene Text Recognition là gì?

- (a) Nhận diện ngôn ngữ tự nhiên.
- (b) Xác định vị trí của văn bản trong hình ảnh.
- (c) Phân loại thẻ loại sách.
- (d) Dự đoán từ ngữ chính trong câu.

4. Mục tiêu của Regconizer trong bài Scene Text Recognition là gì?

- (a) Phân loại hình ảnh.
- (b) Chuyển đổi văn bản hình ảnh thành dạng văn bản có thể đọc được.
- (c) Nhận diện ngôn ngữ tự nhiên.
- (d) Đoán từ ngữ chính trong câu.

5. Đặc điểm chung của hình ảnh gây thách thức trong bài toán Scene Text Recognition là gì?

- | | |
|-----------------------------------|----------------------------|
| (a) Ánh sáng đồng đều. | (c) Nền phức tạp và nhiễu. |
| (b) Kích thước văn bản đồng nhất. | (d) Màu sắc đa dạng. |

6. Deep learning models thường được ưa chuộng trong Scene Text Recognition vì:

- | | |
|--|--|
| (a) Chúng có khả năng nhận diện đối tượng. | (c) Chúng chỉ chấp nhận dữ liệu ảnh đen trắng. |
| (b) Chúng tự động học các đặc trưng. | (d) Chúng không ổn định và khó điều chỉnh. |

7. YOLOv11 là một model detection thuộc loại:

- | | |
|------------------|-----------------|
| (a) Single-stage | (c) Three-stage |
| (b) Two-stage | (d) Multi-stage |

8. YOLOv11 hoạt động như thế nào?

- (a) Chia hình ảnh thành các ô nhỏ và dự đoán các đối tượng trong mỗi ô.
- (b) Dự đoán các đối tượng trong hình ảnh bằng cách sử dụng các bounding box.

- (c) Dự đoán các đối tượng trong hình ảnh bằng cách sử dụng các đặc trưng.
(d) Tất cả các đáp án trên.
9. CRNN hoạt động như thế nào?
(a) Sử dụng một mạng CNN để trích xuất các đặc trưng từ hình ảnh văn bản.
(b) Sử dụng một mạng RNN để dự đoán các ký tự trong văn bản.
(c) Sử dụng một mạng CTC để chuyển đổi các dự đoán của mạng RNN thành văn bản.
(d) Tất cả các đáp án trên.
10. Trong CRNN, mô hình CNN (Convolutional Neural Network) thường được sử dụng để làm gì?
(a) Nhận diện vị trí văn bản.
(b) Phân loại hình ảnh.
(c) Trích xuất đặc trưng từ hình ảnh.
(d) Xác định kích thước hình ảnh.
11. Quy trình tích hợp giữa YOLOv11 và CRNN trong bài toán Scene Text Recognition thường như thế nào?
(a) CRNN chỉ hoạt động độc lập, không cần YOLOv11.
(b) YOLOv11 chỉ được sử dụng để định vị vị trí của văn bản, sau đó CRNN đọc văn bản.
(c) YOLOv11 và CRNN hoạt động độc lập, không tương tác.
(d) CRNN chỉ được sử dụng để cải thiện kết quả của YOLOv11.

1 Phụ lục

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 6 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Triển khai streamlit:** Các bạn có thể truy cập vào web streamlit và link GitHub tại [đây](#).
4. **Rubric:**

Scene Text Recognition Project - Rubric		
Câu	Kiến Thức	Đánh Giá
II.2.	<ul style="list-style-type: none"> - Kiến thức về YOLOv11. - Kiến thức về cách huấn luyện YOLOv11 cho bài toán Text Detection. 	<ul style="list-style-type: none"> - Biết cách ứng dụng YOLOv11 để xây dựng mô hình Text Detection.
II.3.	<ul style="list-style-type: none"> - Kiến thức về Convolution Recurrent Neural Network (CRNN). - Cách về việc xây dựng mô hình CRNN sử dụng pytorch. - Kiến thức về CTCLoss. - Khả năng ứng dụng mô hình CRNN trong bài toán Text Recognition có trong ảnh. 	<ul style="list-style-type: none"> - Hiểu được các khái niệm liên quan đến mô hình CRNN. - Cách lập trình và ứng dụng mô hình CRNN vào giải quyết bài toán Text Recognition.
II.3.	<ul style="list-style-type: none"> - Kiến thức về cách xây dựng chương trình Scene Text Recognition (STR). 	<ul style="list-style-type: none"> - Hiểu được cách vận dụng hai mô hình Text Detection và Text Recognition để xây dựng một chương trình về Scene Text Detection.

- *Hết* -