AI VIETNAM
All-in-One Course
(TA Session)

Scene Text Recognition

Project

Dinh-Thang Duong – TA

Year 2023

Outline

- Introduction
- Prepare datasets
- Text Detection
- Text Recognition
- End-to-End Pipeline
- Question

Introduction

❖ Getting Started

Iphone Live Text: Text Extraction on Images Feature

https://support.apple.com/en-vn/guide/iphone/iphcf0b71b0e/ios

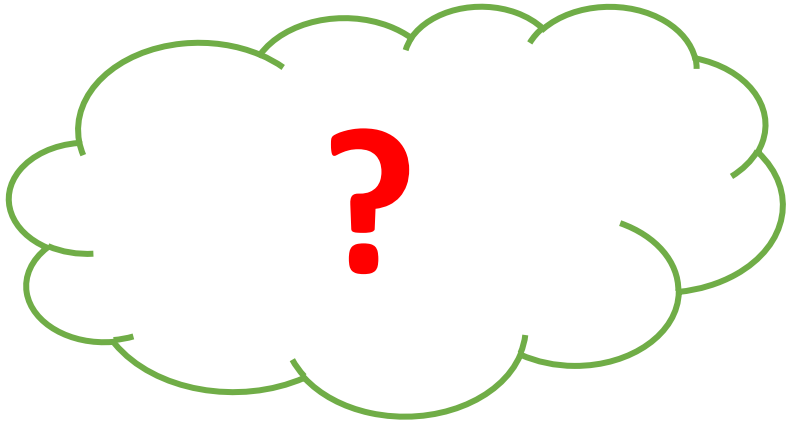
Introduction

❖ Getting Started

Given an image with textual information inside, how can we **use ML algorithms to extract** them?

Introduction

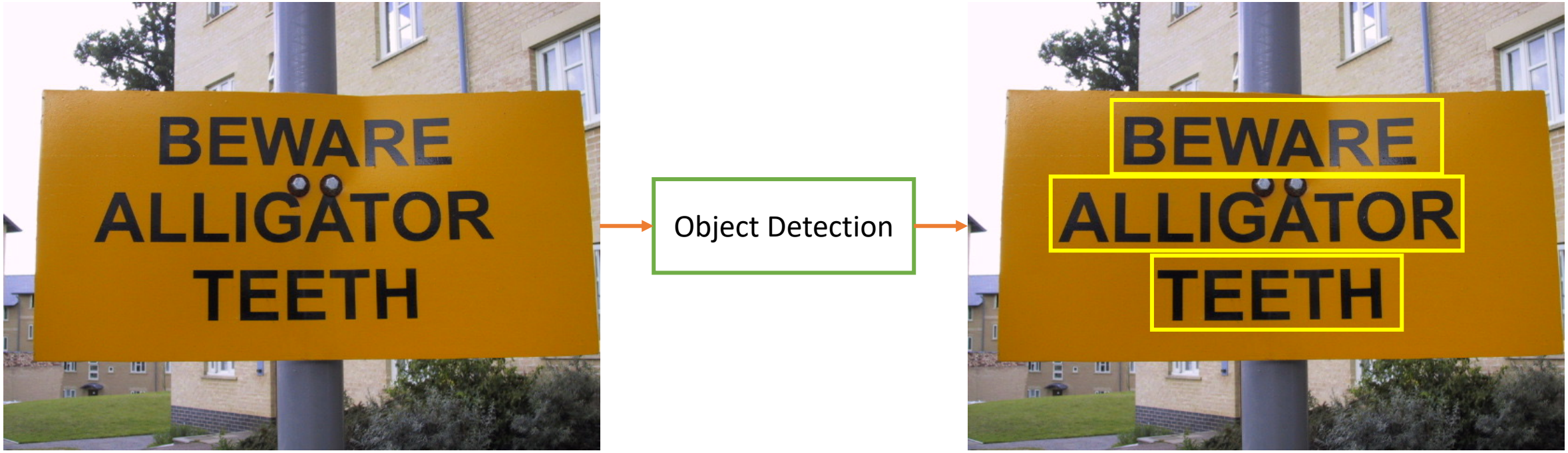
❖ Getting Started



How to build a program to automatically detect and recognize text inside an image?

Introduction

❖ Getting Started



Introduction

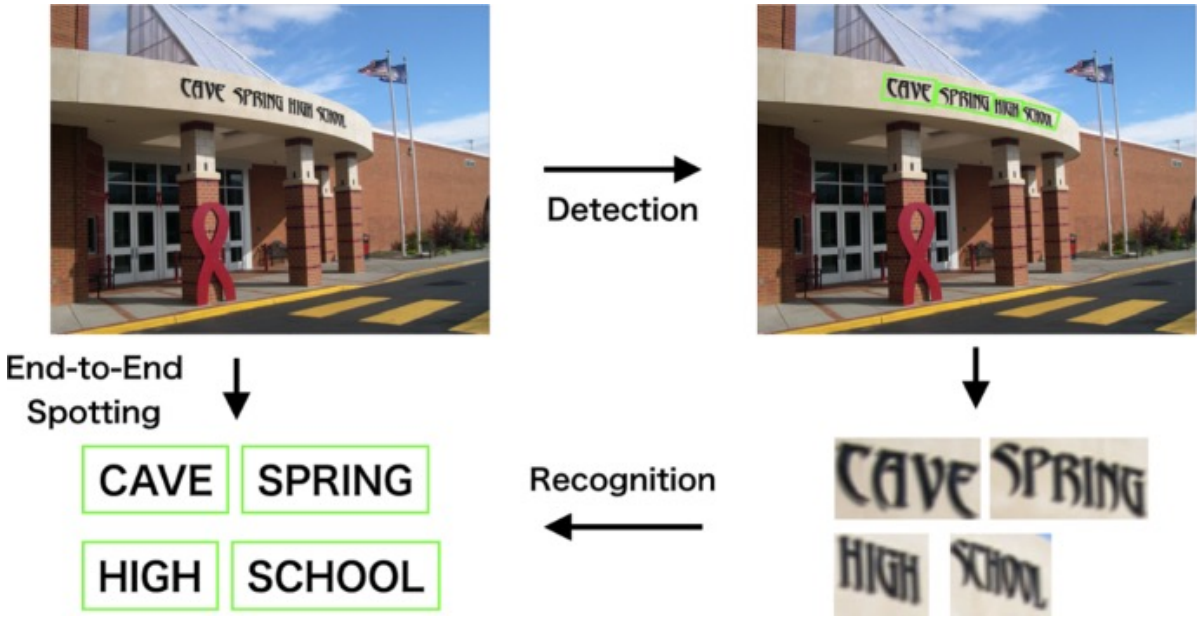
❖ Getting Started

Image Text Transcribe

- BEWARE
- ALLIGATOR
- TEETH

Introduction

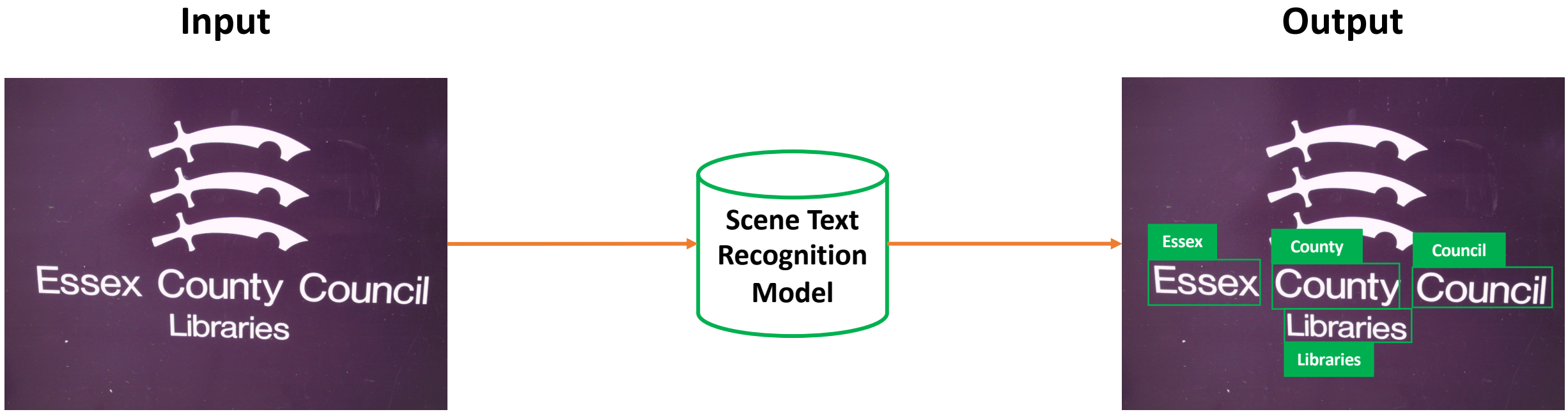
❖ Scene Text Recognition



Scene Text Recognition (STR): A task in Computer Vision that aims to automatically locate and recognize textual content within natural scenes. It serves as a foundational component for numerous applications, including navigation aids, information retrieval systems…

Introduction

❖ Scene Text Recognition I/O



Introduction

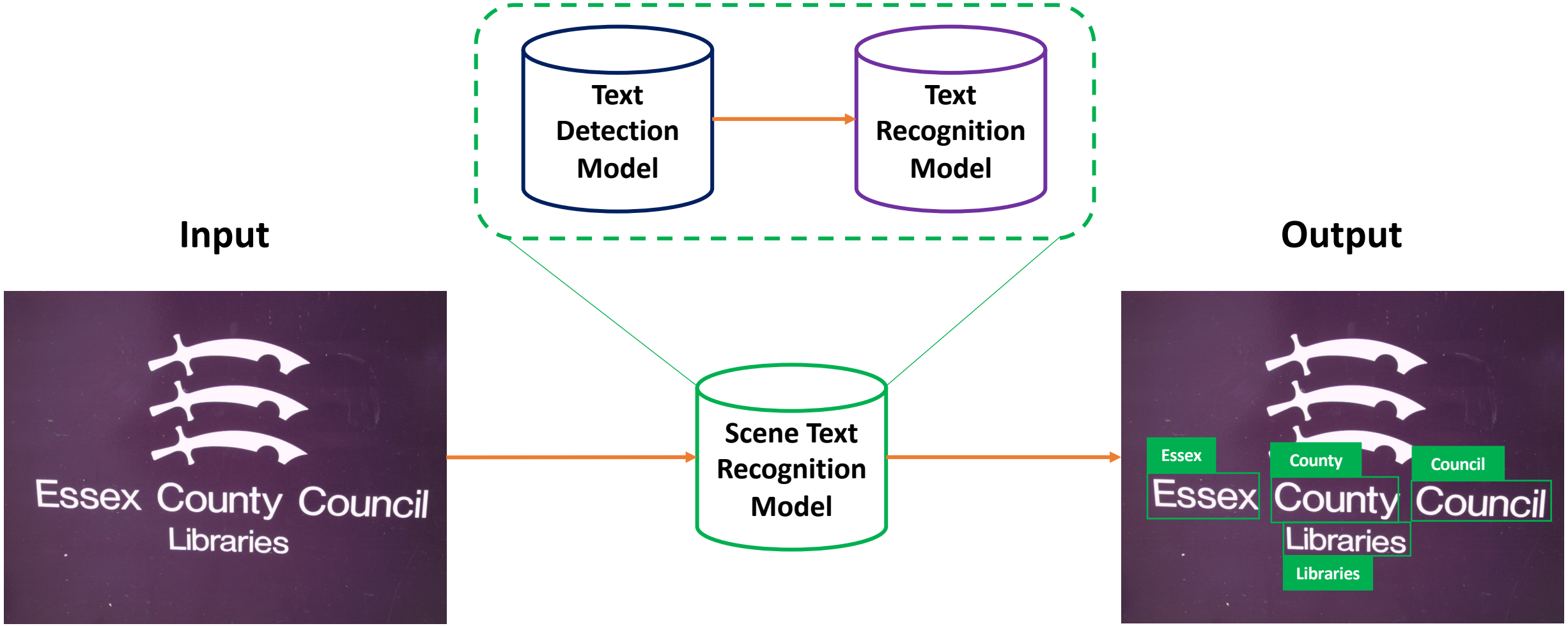
❖ Scene Text Recognition I/O

Example of a STR result:

- ([263.30096435546875, 193.81906127929688, 415.0815124511719, 286.1605529785156], 'text', 0.9180512428283691, ['not’]
- ([119.19277954101562, 287.2218322753906, 433.1628723144531, 389.69293212890625], 'text', 0.8851116895675659, ['ddisturb’]
- ([133.73980712890625, 189.6825408935547, 235.85116577148438, 281.9341735839844], 'text', 0.8539453148841858, ['ddo’])
- ([133.2864990234375, 89.10581970214844, 414.71649169921875, 183.2499542236328], 'text', 0.3512883484363556, ['pleaase'])

Introduction

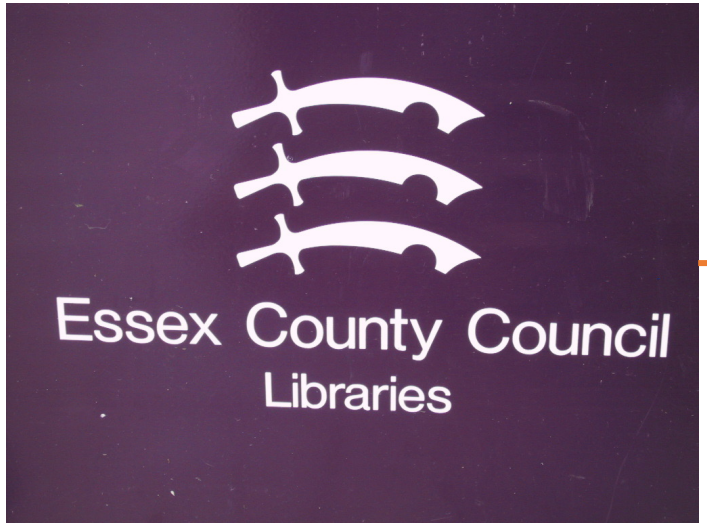
❖ Scene Text Recognition I/O

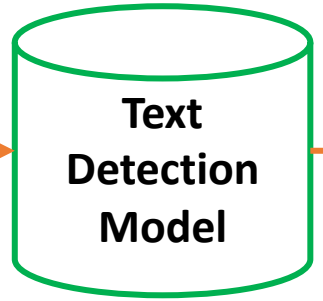


Introduction

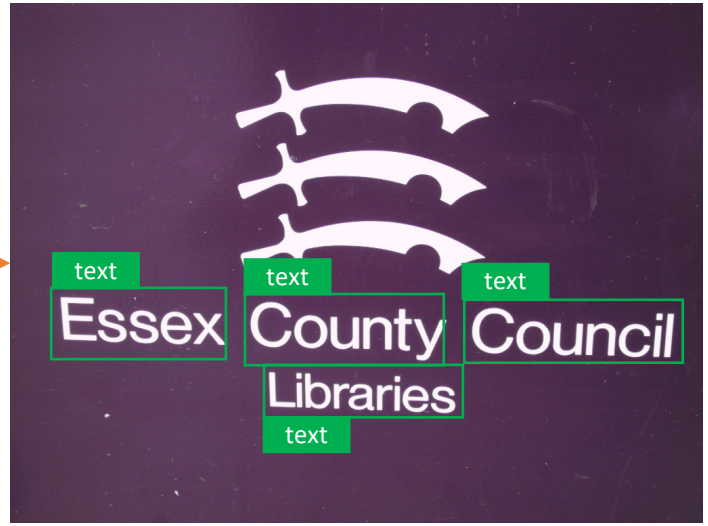
❖ Text Detection I/O

Input





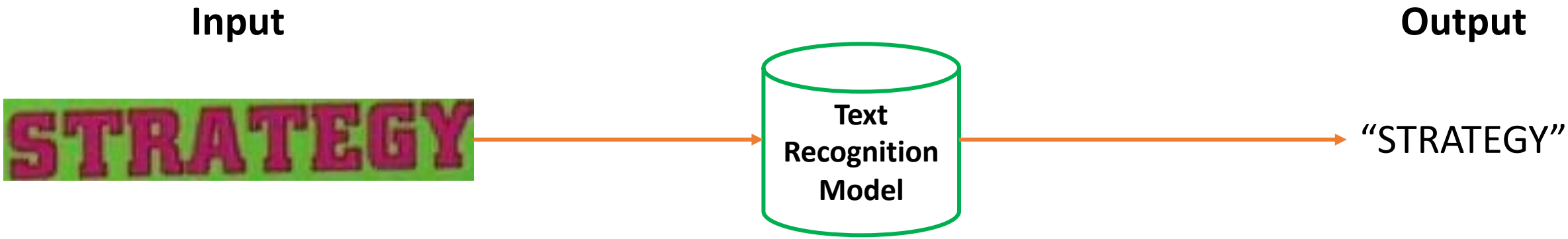
Output



Objective: Get the coodinate (x1, y1, x2, y2) of all text on input image.

Introduction

❖ Text Recognition I/O



Objective: Transcribe the text within input image into string.

Introduction

❖ STR Approach

1. Two-stage Approach

Text Detection

Text Recognition

- RESERVED
- FOR
- CLUB
- SECRETARY

2. End-to-End Approach (Text Spotting)

End-to-End Model

Introduction

❖ STR Approach: Two-stage

Text Detection

- YOLO
- SSD
- Mask R-CNN
- DBNet
- PANet
- …

Text Recognition

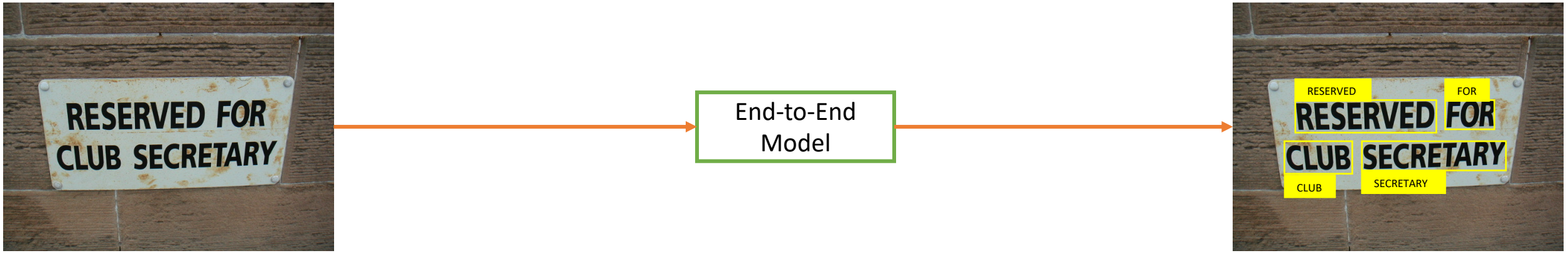
- CRNN
- Transformers

- RESERVED
- FOR
- CLUB
- SECRETARY

Two-stage Approach: Handle Text Detection and Text Recognition separately.

Introduction

❖ STR Approach: End-to-End (Text Spotting)



End-to-End Approach: Handle both Text Detection and Text Recognition in a single, unfied process, from the intial input.

Introduction

❖ STR Approach: End-to-End (Text Spotting) Example

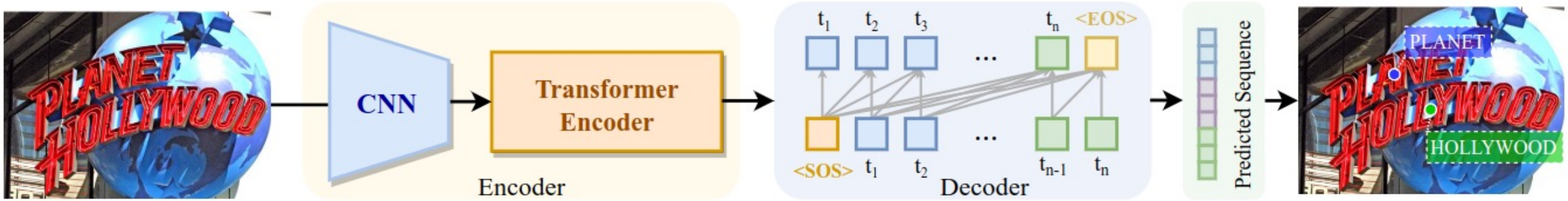
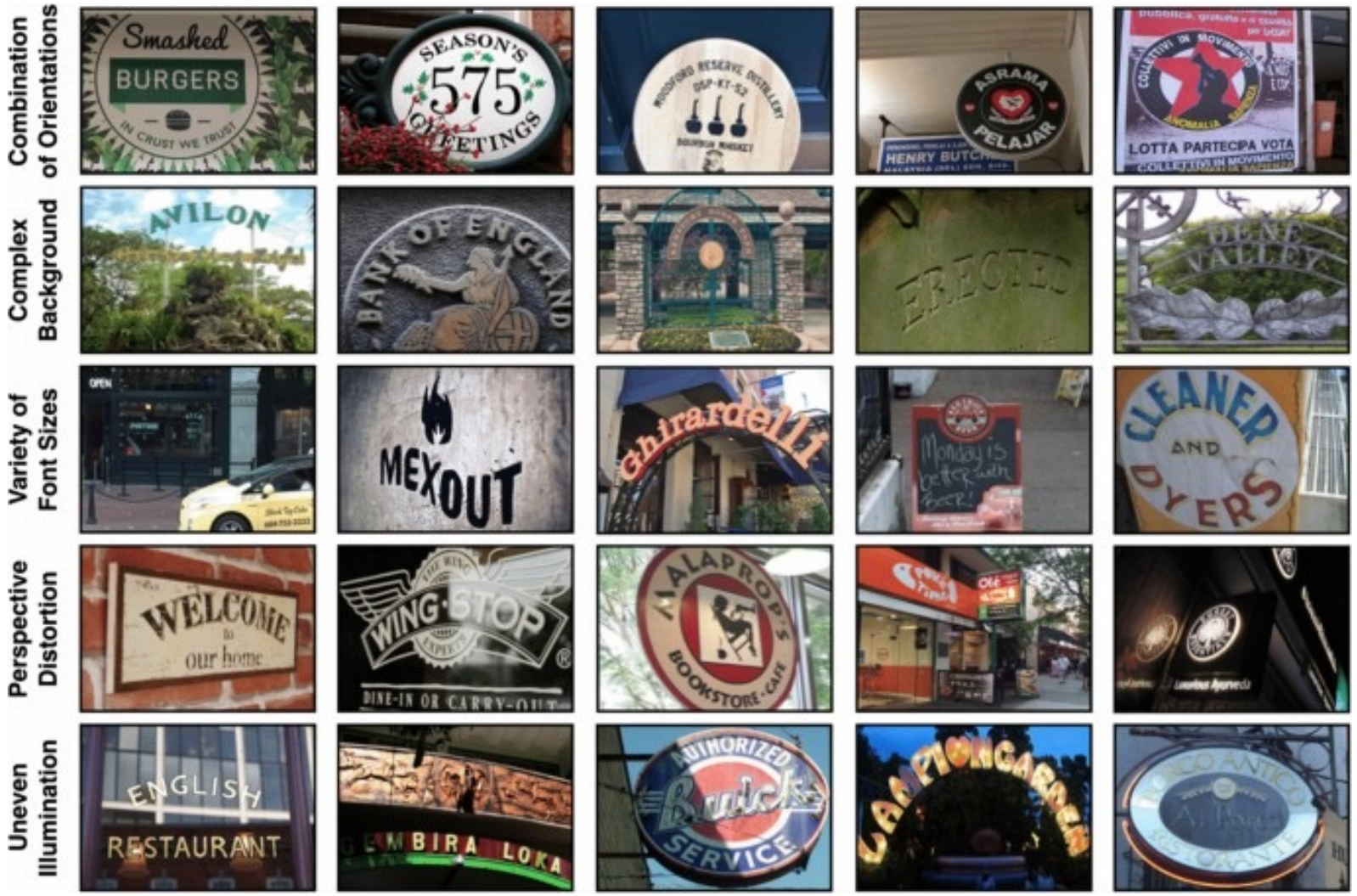


Figure 3: Overall framework of the proposed SPTS. The visual and contextual features are first extracted by a series of CNN and Transformer encoders. Then, the features are auto-regressively decoded into a sequence that contains both localization and recognition information, which is subsequently translated into point coordinates and text transcriptions. Only a point-level annotation is required for training.

SPTS: https://arxiv.org/abs/2112.07917

Introduction

❖ Challenges



- Variety of font sizes
- Complex Background
- Combination of Orientations: Horizontal, Multi-oriented, Curve.
- …

TotalText: https://arxiv.org/abs/1710.10400

Introduction

❖ Challenges

Incidental Scene Text refers to text that appears in the scene without the user having taken any specific prior action to cause its appearance or improve its positioning / quality in the frame.

Incidental Scene Text Challenge: https://rrc.cvc.uab.es/?ch=4

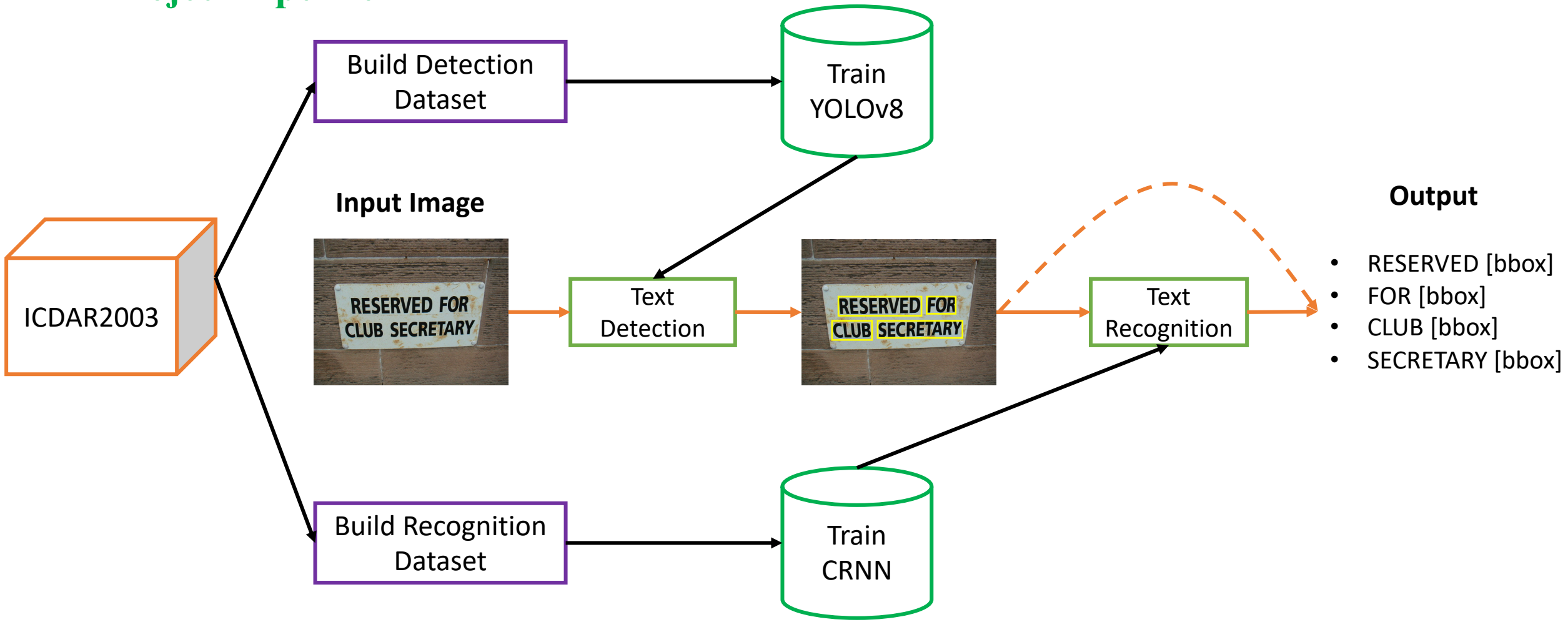
Introduction

❖ Project Description

Description: Given ICDAR2003 dataset (download here), build a Scene Text Recognition model using YOLOv8 and CRNN.

Introduction

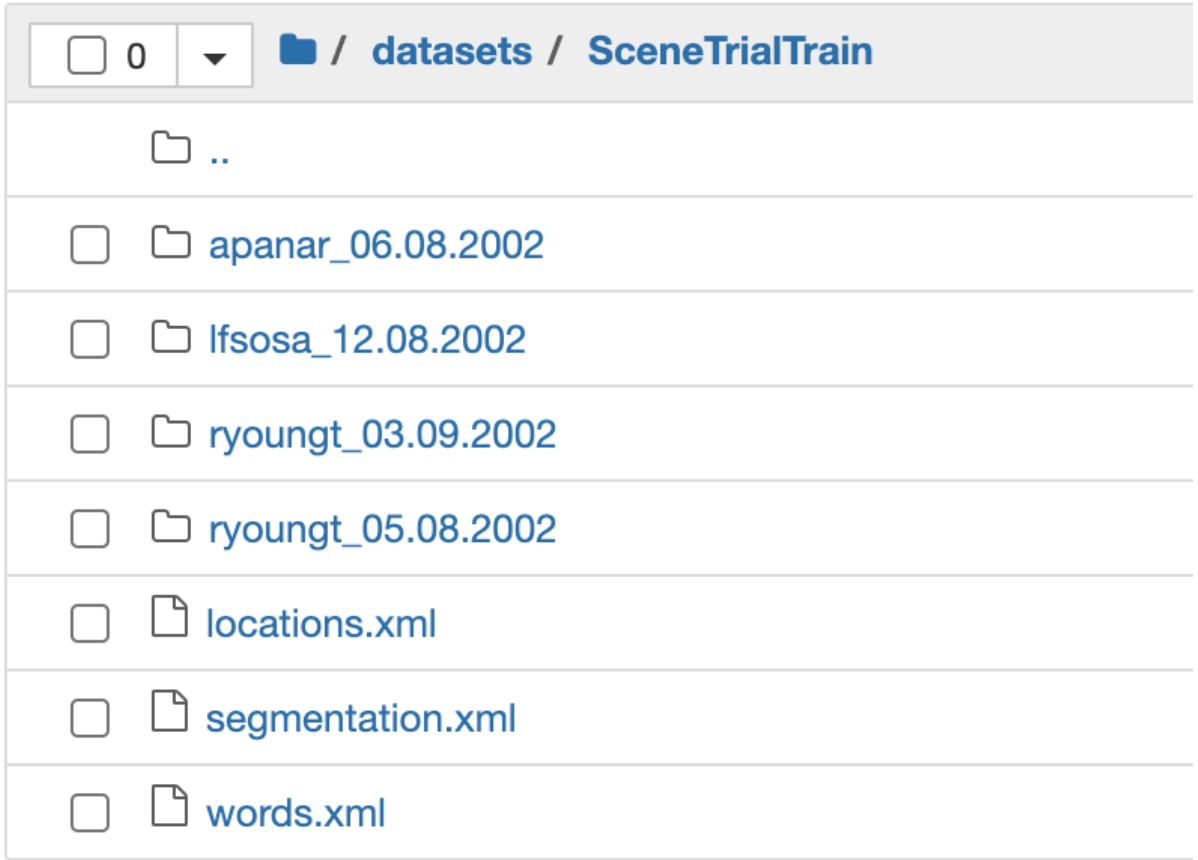
❖ Project Pipeline



Prepare datasets

❖ ICDAR2003

ICDAR2003 Datasets: Created for the ICDAR 2003 Robust Reading competitions organised by Prof Simon Lucas and his team.

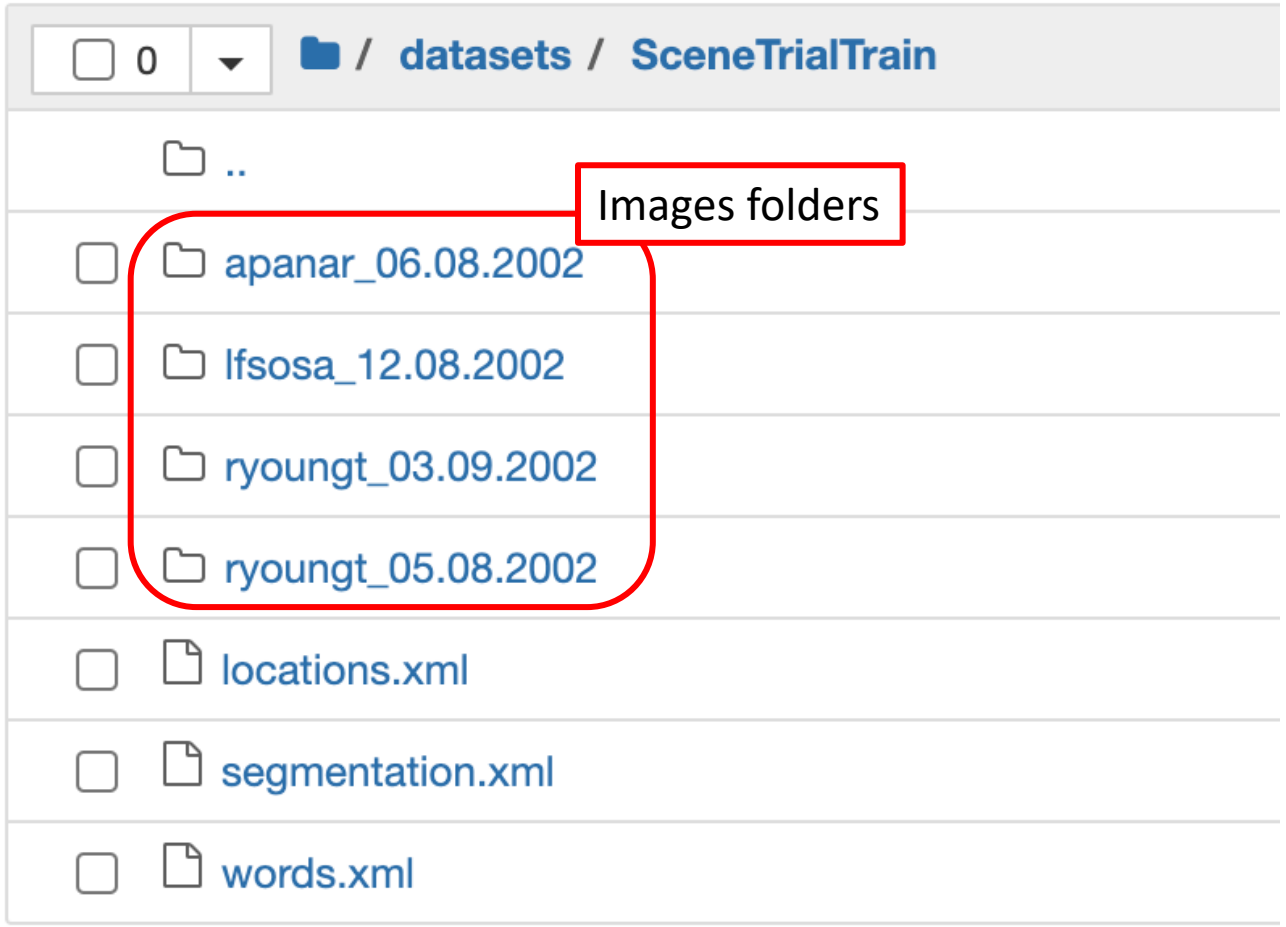


Dataset structure

ICAD2003: http://www.iapr-tc11.org/mediawiki/index.php/ICDAR_2003_Robust_Reading_Competitions

Prepare datasets

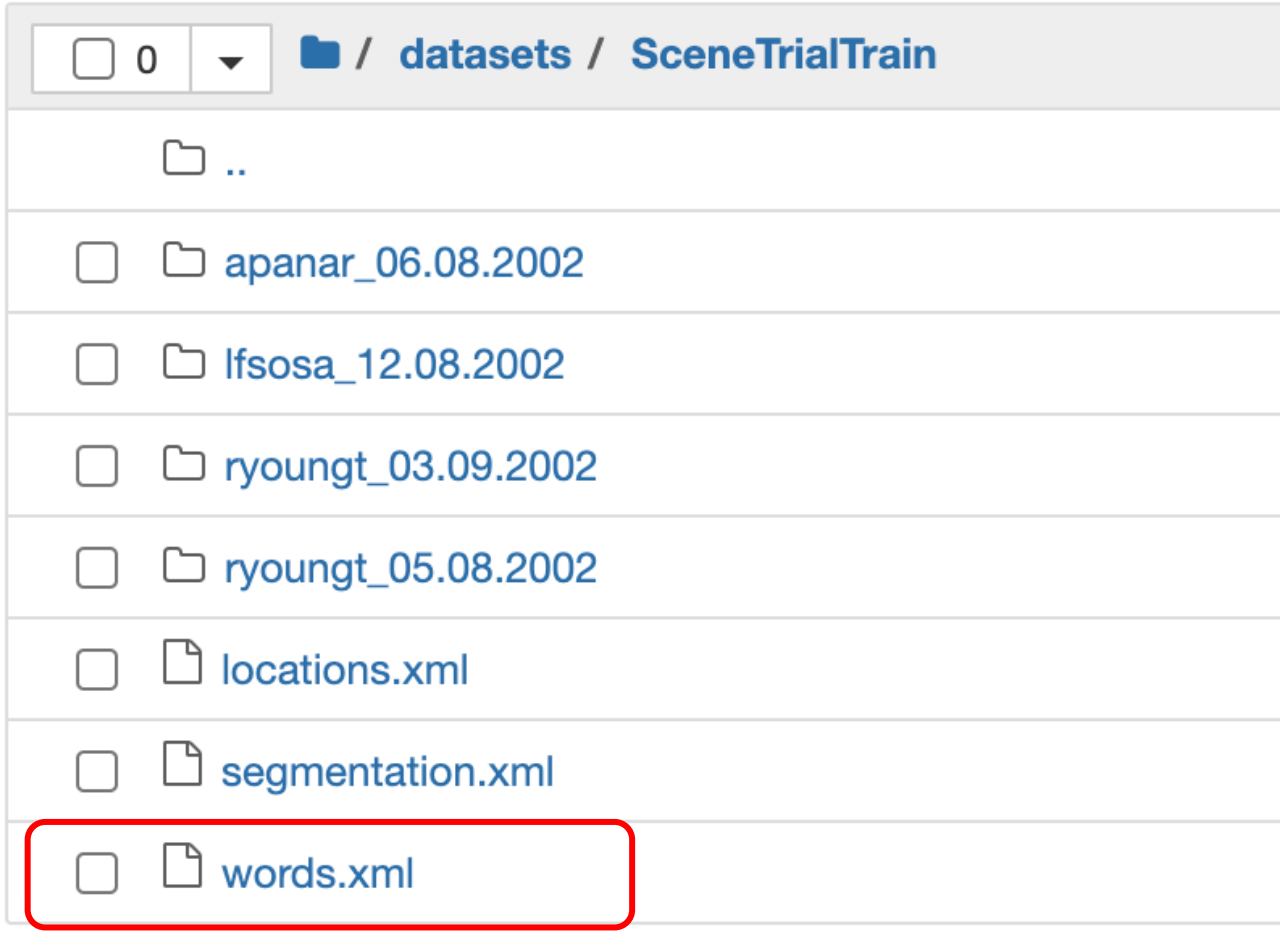
❖ ICDAR2003



- **locations.xml** is for the Text Locating problem.
- **words.xml** is for the Robust Reading competition - this tags each image with the bounding rectangles of each word in the image together with the text in each rectangle.
- **segmentation.xml** - like words.xml, except that each word is also given its segmentation points.

Prepare datasets

❖ ICDAR2003

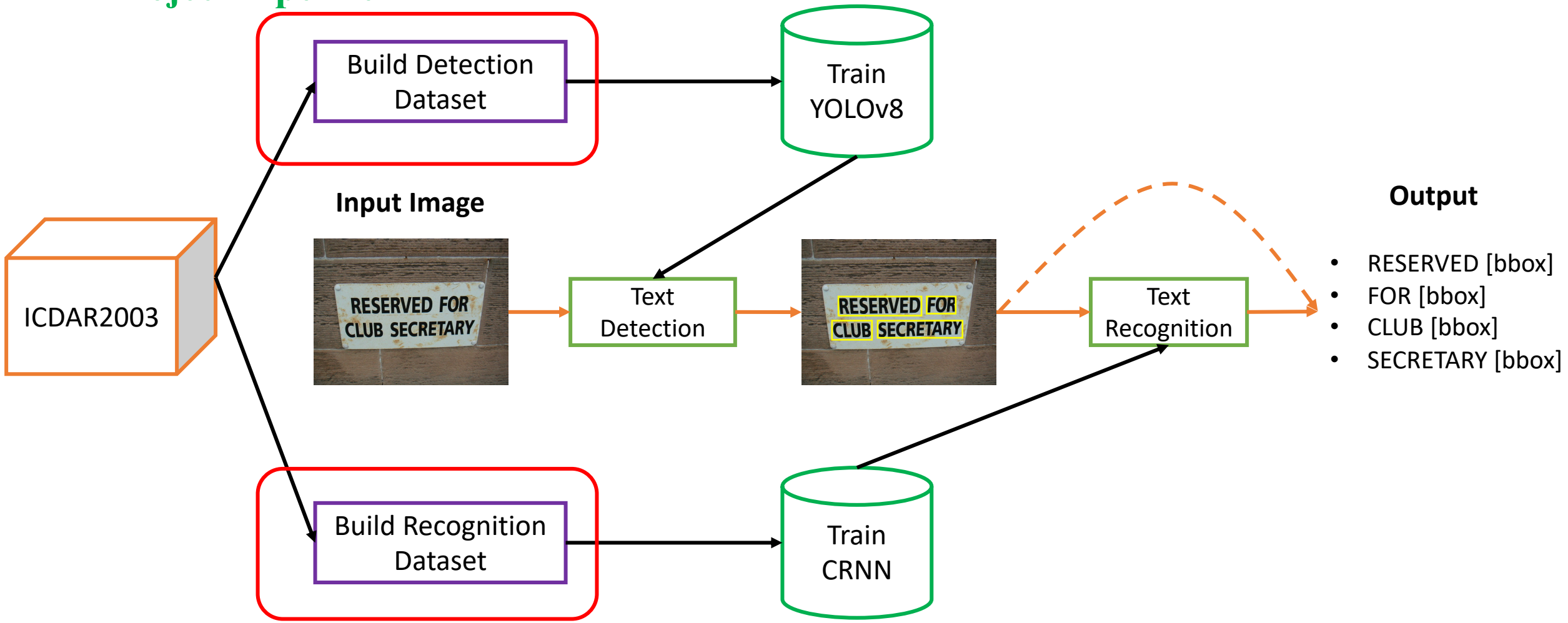


This is the file we need

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<tagset>
  <image>
    <imageName>apanar_06.08.2002/IMG_1261.JPG</imageName>
    <resolution x="1600" y="1200" />
    <taggedRectangles>
      <taggedRectangle x="174.0" y="392.0" width="274.0"
        <tag>self</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="512.0" y="391.0" width="679.0"
        <tag>adhesive</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="184.0" y="612.0" width="622.0"
        <tag>address</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="863.0" y="599.0" width="446.0"
        <tag>labels</tag>
        <segmentation />
      </taggedRectangle>
```

Prepare datasets

❖ Project Pipeline



Prepare datasets

❖ Create prepare dataset function

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<tagset>
  <image>
    <imageName>apanar_06.08.2002/IMG_1261.JPG</imageName>
    <resolution x="1600" y="1200" />
    <taggedRectangles>
      <taggedRectangle x="174.0" y="392.0" width="274.0" height="195.0" offset="0.0" rotation="0.0" userName="admin">
        <tag>self</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="512.0" y="391.0" width="679.0" height="183.0" offset="0.0" rotation="0.0" userName="admin">
        <tag>adhesive</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="184.0" y="612.0" width="622.0" height="174.0" offset="-2.0" rotation="0.0" userName="admin">
        <tag>address</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="863.0" y="599.0" width="446.0" height="187.0" offset="0.0" rotation="0.0" userName="admin">
        <tag>labels</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="72.0" y="6.0" width="95.0" height="87.0" offset="0.0" rotation="0.0" userName="admin">
        <tag>36</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="247.0" y="2.0" width="197.0" height="88.0" offset="0.0" rotation="0.0" userName="admin">
        <tag>89m</tag>
        <segmentation />
      </taggedRectangle>
```

Prepare datasets

❖ Create prepare dataset function

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<tagset>
  <image>
    <imageName>apanar_06.08.2002/IMG_1261.JPG</imageName>
    <resolution x="1600" y="1200" />
    <taggedRectangles>
      <taggedRectangle x="174.0" y="392.0" width="274.0"
        <tag>self</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="512.0" y="391.0" width="679.0"
        <tag>adhesive</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="184.0" y="612.0" width="622.0"
        <tag>address</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="863.0" y="599.0" width="446.0"
        <tag>labels</tag>
        <segmentation />
      </taggedRectangle>
```

<tagset>: root tag cotaning <image> tag

<image>: image tag containing label information.

<imageName>: name of the image.

<resolution>: resolution of the image.

<taggedRectangles>: containing each bounding box of a text within the image.

<taggedRectangle>: text detection label.

<tag>: text recognition label.

Prepare datasets

❖ Create prepare dataset function

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<tagset>
  <image>
    <imageName>apanar_06.08.2002/IMG_1261.JPG</imageName>
    <resolution x="1600" y="1200" />
    <taggedRectangles>
      <taggedRectangle x="174.0" y="392.0" width="274.0" h
        <tag>self</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="512.0" y="391.0" width="679.0" h
        <tag>adhesive</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="184.0" y="612.0" width="622.0" h
        <tag>address</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="863.0" y="599.0" width="446.0" h
        <tag>labels</tag>
        <segmentation />
      </taggedRectangle>
```

```python
import xml.etree.ElementTree as ET

words_xml_path = 'datasets/SceneTrialTrain/words.xml'
tree = ET.parse(words_xml_path)
root = tree.getroot()

print(root)
```

```
<Element 'tagset' at 0x7f0cc3306980>
```

Prepare datasets

❖ Create prepare dataset function

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<tagset>
  <image>
    <imageName>apanar_06.08.2002/IMG_1261.JPG</imageName>
    <resolution x="1600" y="1200" />
    <taggedRectangles>
      <taggedRectangle x="174.0" y="392.0" width="274.0" h
        <tag>self</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="512.0" y="391.0" width="679.0" h
        <tag>adhesive</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="184.0" y="612.0" width="622.0" h
        <tag>address</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="863.0" y="599.0" width="446.0" h
        <tag>labels</tag>
        <segmentation />
      </taggedRectangle>
```

```python
import xml.etree.ElementTree as ET

words_xml_path = 'datasets/SceneTrialTrain/words.xml'
tree = ET.parse(words_xml_path)
root = tree.getroot()

for image in root:
    print(image)
```

```
<Element 'image' at 0x7f0c55688450>
<Element 'image' at 0x7f0c55689080>
<Element 'image' at 0x7f0c55689300>
<Element 'image' at 0x7f0c55689df0>
<Element 'image' at 0x7f0c5568b060>
```

Prepare datasets

❖ Create prepare dataset function

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<tagset>
  <image>
    <imageName>apanar_06.08.2002/IMG_1261.JPG</imageName>
    <resolution x="1600" y="1200" />
    <taggedRectangles>
      <taggedRectangle x="174.0" y="392.0" width="274.0" h
        <tag>self</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="512.0" y="391.0" width="679.0" h
        <tag>adhesive</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="184.0" y="612.0" width="622.0" h
        <tag>address</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="863.0" y="599.0" width="446.0" h
        <tag>labels</tag>
        <segmentation />
      </taggedRectangle>
```

<image> is like a list:
<image>[0]: <imageName>
<image>[1]: <resolution>
<image>[2]: <taggedRectangles>

```python
import xml.etree.ElementTree as ET

words_xml_path = 'datasets/SceneTrialTrain/words.xml'
tree = ET.parse(words_xml_path)
root = tree.getroot()

for image in root:
    print(image[0].text)
    print(image[1].attrib['x'], image[1].attrib['y'])
    break
```

```
apanar_06.08.2002/IMG_1261.JPG
1600 1200
```

Prepare datasets

❖ Create prepare dataset function

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<tagset>
  <image>
    <imageName>apanar_06.08.2002/IMG_1261.JPG</imageName>
    <resolution x="1600" y="1200" />
    <taggedRectangles>
      <taggedRectangle x="174.0" y="392.0" width="274.0"
        <tag>self</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="512.0" y="391.0" width="679.0"
        <tag>adhesive</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="184.0" y="612.0" width="622.0"
        <tag>address</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="863.0" y="599.0" width="446.0"
        <tag>labels</tag>
        <segmentation />
      </taggedRectangle>
```

```python
import xml.etree.ElementTree as ET

words_xml_path = 'datasets/SceneTrialTrain/words.xml'
tree = ET.parse(words_xml_path)
root = tree.getroot()

for image in root:
    img_name = image[0].text
    for bbs in image.findall('taggedRectangles'):
        for bb in bbs:
            bbox = [
                bb.attrib['x'],
                bb.attrib['y'],
                bb.attrib['width'],
                bb.attrib['height'],
            ]
            print(bb[0].text, bbox)

        break
    break
```

```
self ['174.0', '392.0', '274.0', '195.0']
adhesive ['512.0', '391.0', '679.0', '183.0']
address ['184.0', '612.0', '622.0', '174.0']
labels ['863.0', '599.0', '446.0', '187.0']
36 ['72.0', '6.0', '95.0', '87.0']
89m ['247.0', '2.0', '197.0', '88.0']
cls ['792.0', '0.0', '115.0', '81.0']
250 ['200.0', '848.0', '228.0', '139.0']
on ['473.0', '878.0', '165.0', '109.0']
a ['684.0', '878.0', '71.0', '106.0']
roll ['806.0', '844.0', '218.0', '141.0']
```

Prepare datasets

❖ Create prepare dataset function

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<tagset>
  <image>
    <imageName>apanar_06.08.2002/IMG_1261.JPG</imageName>
    <resolution x="1600" y="1200" />
    <taggedRectangles>
      <taggedRectangle x="174.0" y="392.0" width="274.0" h
        <tag>self</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="512.0" y="391.0" width="679.0" h
        <tag>adhesive</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="184.0" y="612.0" width="622.0" h
        <tag>address</tag>
        <segmentation />
      </taggedRectangle>
      <taggedRectangle x="863.0" y="599.0" width="446.0" h
        <tag>labels</tag>
        <segmentation />
      </taggedRectangle>
```

```python
image_paths = []
image_sizes = []
image_labels = []
bounding_boxes = []

for image in root:
    bbs_of_image = []
    labels_of_image = []

    for bbs in image.findall('taggedRectangles'):
        for bb in bbs:
            # check non-alphabet and non-number
            if not bb[0].text.isalnum():
                continue

            if 'é' in bb[0].text.lower() or 'ñ' in bb[0].text.lower():
                continue

            bbs_of_image.append(
                [
                    float(bb.attrib['x']),
                    float(bb.attrib['y']),
                    float(bb.attrib['width']),
                    float(bb.attrib['height'])
                ]
            )
            labels_of_image.append(bb[0].text.lower())

    # Store
    image_paths.append(image[0].text)
    image_sizes.append((int(image[1].attrib['x']), int(image[1].attrib['y'])))
    bounding_boxes.append(bbs_of_image)
    image_labels.append(labels_of_image)
```

Prepare datasets

❖ Create prepare dataset function

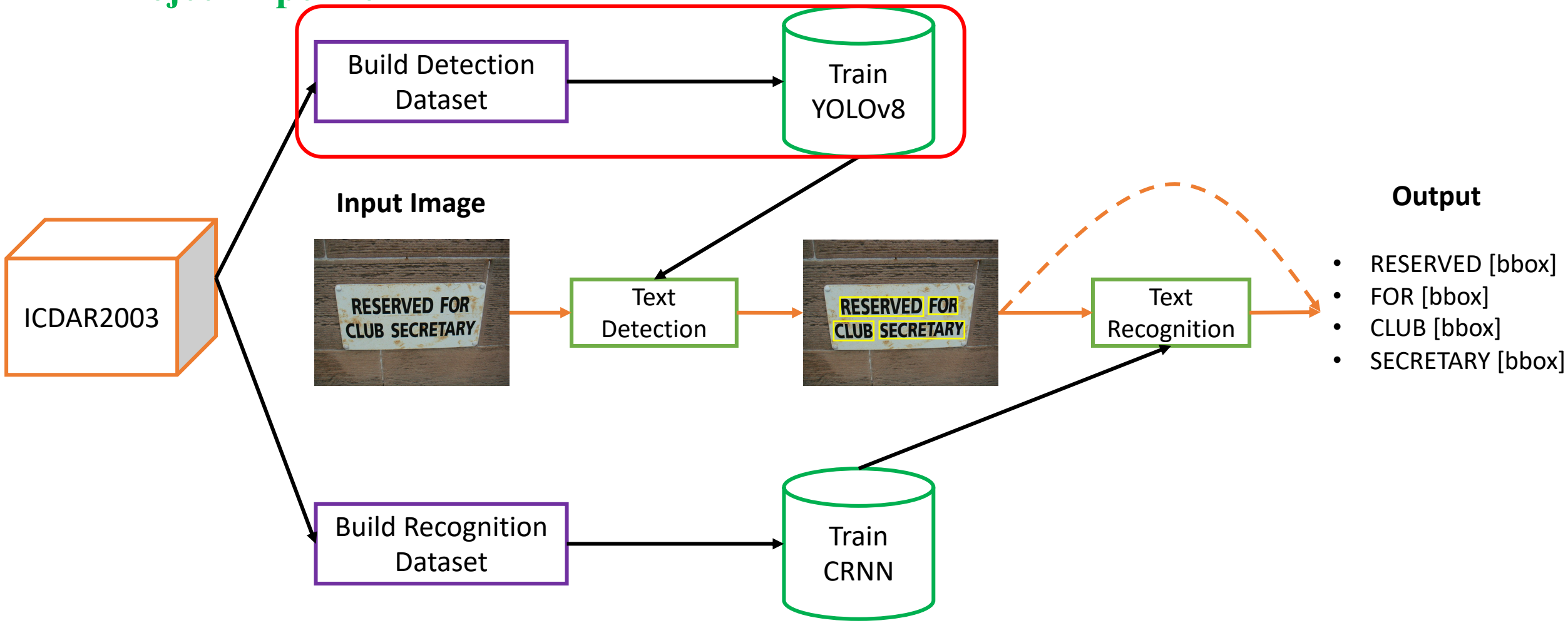
```
dataset_dir = 'datasets/SceneTrialTrain'
words_xml_path = os.path.join(
    dataset_dir,
    'words.xml'
)
image_paths, image_sizes, image_labels, bounding_boxes = extract_data_from_xml(words_xml_path)
```

```
for path, size, label, bbox in zip(image_paths, image_sizes, image_labels, bounding_boxes):
    print(path, size, label, bbox)
    break
```

```
apanar_06.08.2002/IMG_1261.JPG (1600, 1200) ['self', 'adhesive', 'address', 'labels', '36', '89m', 'cls', '250', 'o
n', 'a', 'roll'] [[174.0, 392.0, 274.0, 195.0], [512.0, 391.0, 679.0, 183.0], [184.0, 612.0, 622.0, 174.0], [863.0,
599.0, 446.0, 187.0], [72.0, 6.0, 95.0, 87.0], [247.0, 2.0, 197.0, 88.0], [792.0, 0.0, 115.0, 81.0], [200.0, 848.0,
228.0, 139.0], [473.0, 878.0, 165.0, 109.0], [684.0, 878.0, 71.0, 106.0], [806.0, 844.0, 218.0, 141.0]]
```

Text Detection

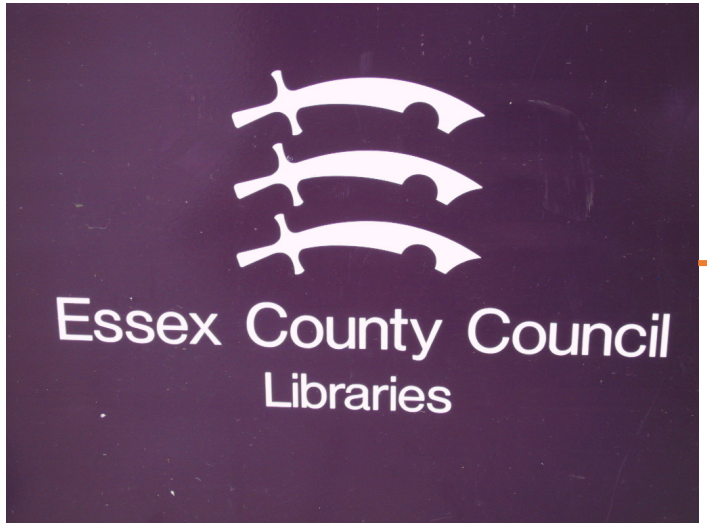
❖ Project Pipeline

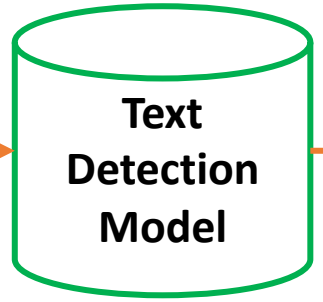


Text Detection

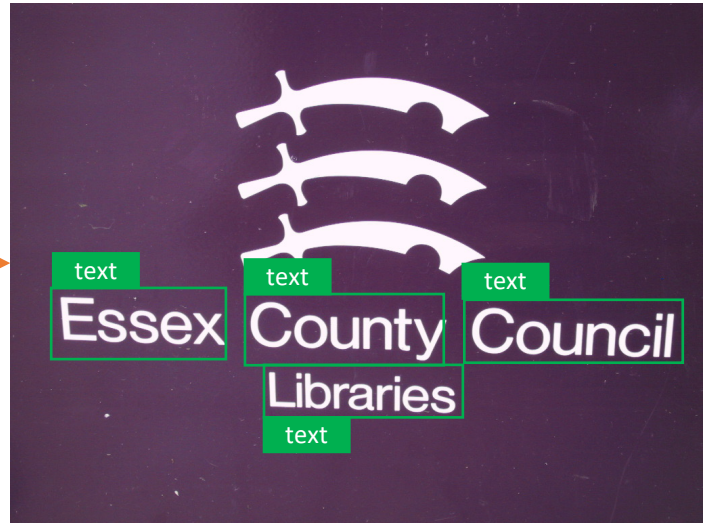
❖ Text Detection I/O

Input





Output



Objective: Get the coodinate (x1, y1, x2, y2) of all text on input image.

Text Detection

❖ YOLOv8

ultralytics
YOLOv8

Text Detection

❖ Step 1: Import libraries

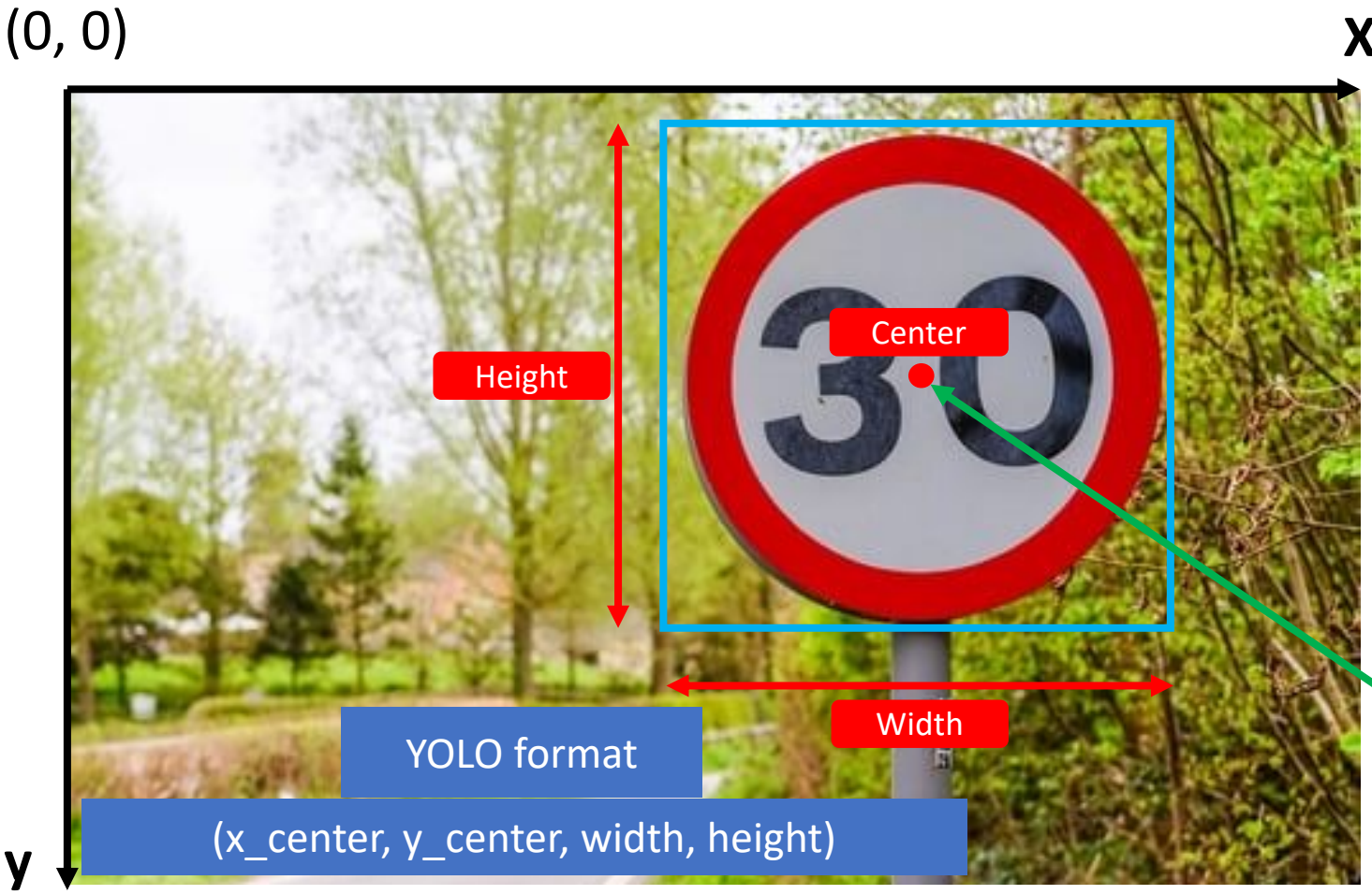
```
%pip install ultralytics
import ultralytics
ultralytics.checks()
```

```
Ultralytics YOLOv8.0.222 🚀 Python-3.11.5 torch-2.1.0 CUDA:0 (NVIDIA GeForce RTX 3060, 12036MiB)
Setup complete ✅ (20 CPUs, 31.1 GB RAM, 385.9/915.3 GB disk)
```

```
import os
import shutil
import yaml
import xml.etree.ElementTree as ET

from sklearn.model_selection import train_test_split
```

Text Detection

❖ Step 2: Normalize bounding box




```
<taggedRectangles>
  <taggedRectangle x="174.0" y="392.0" width="274.0" height="195.0"
    <tag>self</tag>
    <segmentation />
  </taggedRectangle>
  <taggedRectangle x="512.0" y="391.0" width="679.0" height="183.0"
    <tag>adhesive</tag>
    <segmentation />
  </taggedRectangle>
  <taggedRectangle x="184.0" y="612.0" width="622.0" height="174.0"
    <tag>address</tag>
    <segmentation />
  </taggedRectangle>
  <taggedRectangle x="863.0" y="599.0" width="446.0" height="187.0"
    <tag>labels</tag>
    <segmentation />
  </taggedRectangle>
```

(x_center, y_center)

Text Detection

❖ Step 2: Normalize bounding box

```
<taggedRectangles>
  <taggedRectangle x="174.0" y="392.0" width="274.0" height="195.0"
    <tag>self</tag>
    <segmentation />
  </taggedRectangle>
  <taggedRectangle x="512.0" y="391.0" width="679.0" height="183.0"
    <tag>adhesive</tag>
    <segmentation />
  </taggedRectangle>
  <taggedRectangle x="184.0" y="612.0" width="622.0" height="174.0"
    <tag>address</tag>
    <segmentation />
  </taggedRectangle>
  <taggedRectangle x="863.0" y="599.0" width="446.0" height="187.0"
    <tag>labels</tag>
    <segmentation />
  </taggedRectangle>
```

```
0 0.565 0.1375 0.2275 0.058333333333333334
0 0.5378125 0.25125 0.175625 0.0725
0 0.4853125 0.35 0.075625 0.058333333333333334
0 0.548125 0.5716666666666667 0.2025 0.07166666666666667
0 0.4559375 0.8791666666666667 0.028125 0.056666666666666664
0 0.5053125 0.8791666666666667 0.029375 0.05333333333333334
0 0.67 0.8766666666666667 0.25 0.05333333333333334
```

Format: [class_id, x_center, y_center, width, height]

Current bounding box is not been normalized yet

Text Detection

❖ Step 2: Normalize bounding box

```
0 0.565 0.1375 0.2275 0.05833333333333334
0 0.5378125 0.25125 0.175625 0.0725
0 0.4853125 0.35 0.075625 0.05833333333333334
0 0.548125 0.5716666666666667 0.2025 0.07166666666666667
0 0.4559375 0.8791666666666667 0.028125 0.05666666666666664
0 0.5053125 0.8791666666666667 0.029375 0.05333333333333334
0 0.67 0.8766666666666667 0.25 0.05333333333333334

```

Format: [class_id, x_center, y_center, width, height]

```python
for bbox in bboxes:
    x, y, w, h = bbox

    # Calculate normalized bounding box coordinates
    center_x = (x + w / 2) / image_width
    center_y = (y + h / 2) / image_height
    normalized_width = w / image_width
    normalized_height = h / image_height

    # Because we only have one class, we set class_id to 0
    class_id = 0

    # Convert to YOLOv8 format
    yolov8_label = f"{class_id} {center_x} {center_y} {normalized_width} {normalized_height}"
    yolov8_labels.append(yolov8_label)
```

Text Detection

❖ Step 2: Normalize bounding box

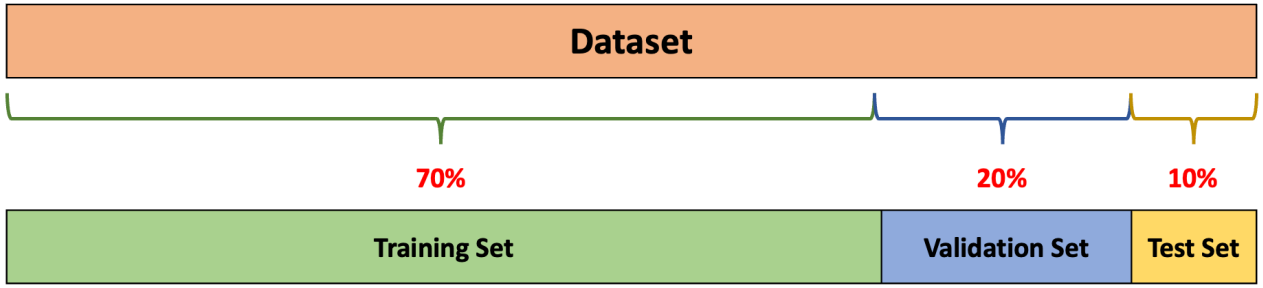
```python
def convert_to_yolov8_format(image_paths, image_sizes, bounding_boxes):
    yolov8_data = []

    for image_path, image_size, bboxes in zip(image_paths, image_sizes, bounding_boxes):
        image_width, image_height = image_size

        yolov8_labels = []

        for bbox in bboxes:
            x, y, w, h = bbox

            # Calculate normalized bounding box coordinates
            center_x = (x + w / 2) / image_width
            center_y = (y + h / 2) / image_height
            normalized_width = w / image_width
            normalized_height = h / image_height

            # Because we only have one class, we set class_id to 0
            class_id = 0

            # Convert to YOLOv8 format
            yolov8_label = f"{class_id} {center_x} {center_y} {normalized_width} {normalized_height}"
            yolov8_labels.append(yolov8_label)

        yolov8_data.append((image_path, yolov8_labels))

    return yolov8_data
```

```python
# Define class labels
class_labels = ['text']

# Convert data into YOLOv8 format
yolov8_data = convert_to_yolov8_format(
    image_paths,
    image_sizes,
    bounding_boxes
)
```

```
('apanar_06.08.2002/IMG_1261.JPG',
 ['0 0.194375 0.40791666666666665 0.17125 0.1625',
  '0 0.5321875 0.40208333333333335 0.424375 0.1525',
  '0 0.309375 0.5825 0.38875 0.145',
  '0 0.67875 0.5770833333333333 0.27875 0.15583333333333332',
  '0 0.0746875 0.04125 0.059375 0.0725',
  '0 0.2159375 0.03833333333333333 0.123125 0.07333333333333333',
  '0 0.5309375 0.03375 0.071875 0.0675',
  '0 0.19625 0.7645833333333333 0.1425 0.11583333333333333',
  '0 0.3471875 0.7770833333333333 0.103125 0.09083333333333334',
  '0 0.4496875 0.7758333333333334 0.044375 0.08833333333333333',
  '0 0.571875 0.7620833333333333 0.13625 0.1175'])
```

Text Detection

❖ Step 3: Train, val, test split




```python
seed = 0
val_size = 0.2
test_size = 0.125
is_shuffle = True
train_data, test_data = train_test_split(
    yolov8_data,
    test_size=val_size,
    random_state=seed,
    shuffle=is_shuffle
)
test_data, val_data = train_test_split(
    test_data,
    test_size=test_size,
    random_state=seed,
    shuffle=is_shuffle
)
```

Text Detection

❖ Step 4: Prepare YOLO data structure

In order to train YOLOv8, we have to organize a folder structure like this:

- train: for train data
- val: for validation data
- test: for test data
- data.yml: for metadata

Text Detection

❖ Step 4: Prepare YOLO data structure


```python
def save_data(data, src_img_dir, save_dir):
    # Create folder if not exists
    os.makedirs(save_dir, exist_ok=True)

    # Make images and labels folder
    os.makedirs(os.path.join(save_dir, 'images'), exist_ok=True)
    os.makedirs(os.path.join(save_dir, 'labels'), exist_ok=True)

    for image_path, yolov8_labels in data:
        # Copy image to images folder
        shutil.copy(
            os.path.join(src_img_dir, image_path),
            os.path.join(save_dir, 'images')
        )

        # Save labels to labels folder
        image_name = os.path.basename(image_path)
        image_name = os.path.splitext(image_name)[0]

        with open(os.path.join(save_dir, 'labels', f"{image_name}.txt"), 'w') as f:
            for label in yolov8_labels:
                f.write(f"{label}\n")
```

Text Detection

❖ Step 4: Prepare YOLO data structure


```
save_data(
    train_data,
    dataset_dir,
    save_train_dir
)
save_data(
    test_data,
    dataset_dir,
    save_val_dir
)
save_data(
    val_data,
    dataset_dir,
    save_test_dir
)
```

Text Detection

❖ Step 4: Prepare YOLO data structure

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
8
```

- **names:** list of class names.
- **nc:** number of classes.
- **path:** path to root folder.
- test: path to test folder.
- **train:** path to train folder.
- **val:** path to val folder.

Text Detection

❖ Step 4: Prepare YOLO data structure

data.yml

```
names:
- text
nc: 1
path: yolo_data
test: test/images
train: train/images
val: val/images

```

```python
data_yaml = {
    'path': 'yolo_data',
    'train': 'train/images',
    'test': 'test/images',
    'val': 'val/images',
    'nc': 1,
    'names': class_labels
}

yolo_yaml_path = os.path.join(
    save_yolo_data_dir,
    'data.yml'
)
with open(yolo_yaml_path, 'w') as f:
    yaml.dump(data_yaml, f, default_flow_style=False)
```

Text Detection

❖ Step 5: Training

```
from ultralytics import YOLO

# Load a model
model = YOLO('yolov8s.yaml').load('yolov8s.pt')

# Train the model
epochs = 200
imgsz = 1024
results = model.train(
    data=yolo_yaml_path,
    epochs=epochs,
    imgsz=imgsz,
    project='models',
    name='yolov8/detect/train'
)
```

Load pretrained model: yolov8s

Start training

Text Detection

❖ Step 5: Training

```
Plotting labels to models/yolov8/detect/train/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0'
and 'momentum' automatically...
optimizer: AdamW(lr=0.002, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bi
as(decay=0.0)
Image sizes 1024 train, 1024 val
Using 8 dataloader workers
Logging results to models/yolov8/detect/train
Starting training for 200 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size

      1/200      10.7G      2.038       5.46      1.827         61       1024: 100%|██████████| 13/13 [00:05<00:00,
2.42it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:
00<00:00,  4.28it/s]
                   all         43        200      0.312      0.455      0.307       0.15
```

Text Detection

❖ Step 6: Evaluation

```python
from ultralytics import YOLO

model_path = 'models/yolov8/detect/train/weights/best.pt'
model = YOLO(model_path)

metrics = model.val(
    project='models',
    name='yolov8/detect/val'
)
```

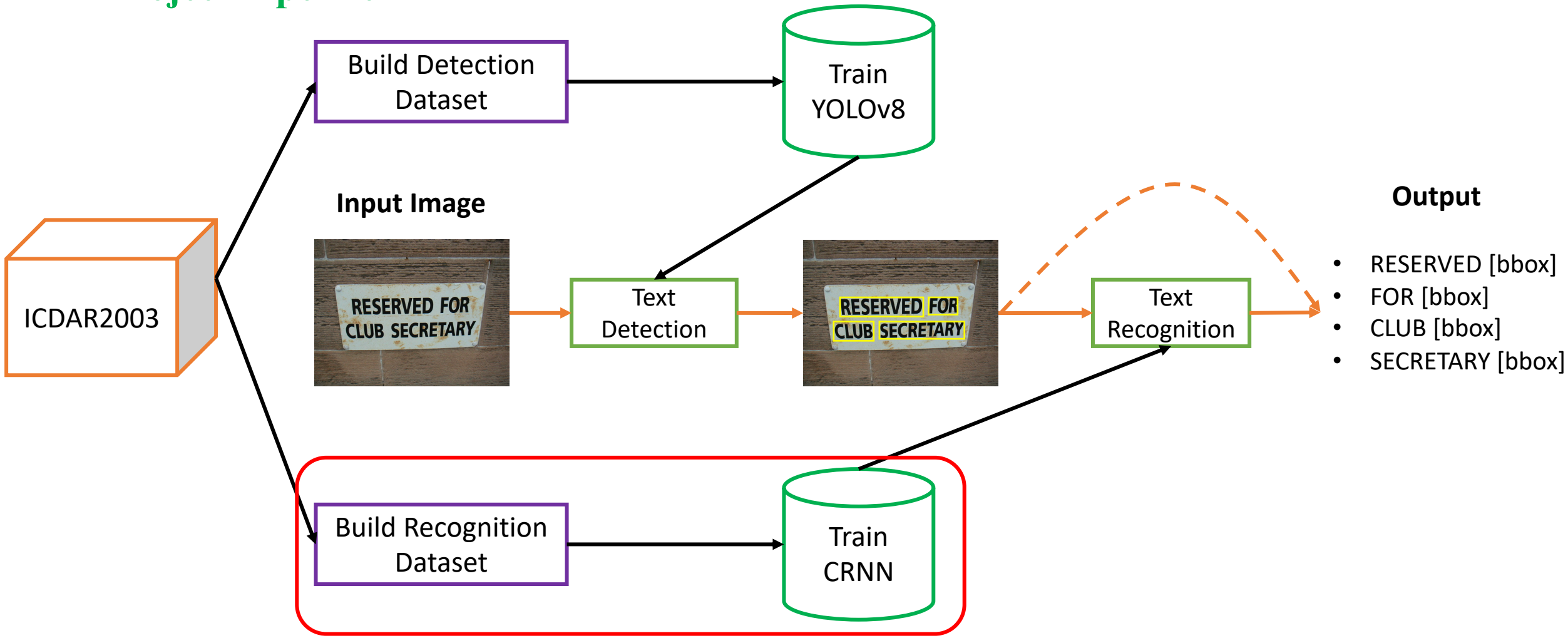
```
Ultralytics YOLOv8.0.222 🚀 Python-3.11.5 torch-2.1.0 CUDA:0 (NVIDIA GeForce RTX 3060, 12036MiB)
YOLOv8s summary (fused): 168 layers, 11125971 parameters, 0 gradients, 28.4 GFLOPs
val: Scanning /home/aivn12s1/thangdd/project_scene_text_detection/datasets/yolo_data/val/labels.cache... 43 images,
2 backgrounds, 0 corrupt: 100%|██████████| 43/43 [00:00<?
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:
00<00:00,  3.77it/s]
                   all         43        200      0.933      0.895      0.941      0.748
Speed: 0.3ms preprocess, 12.3ms inference, 0.0ms loss, 1.1ms postprocess per image
Results saved to models/yolov8/detect/val
```

Text Detection

❖ Step 6: Evaluation

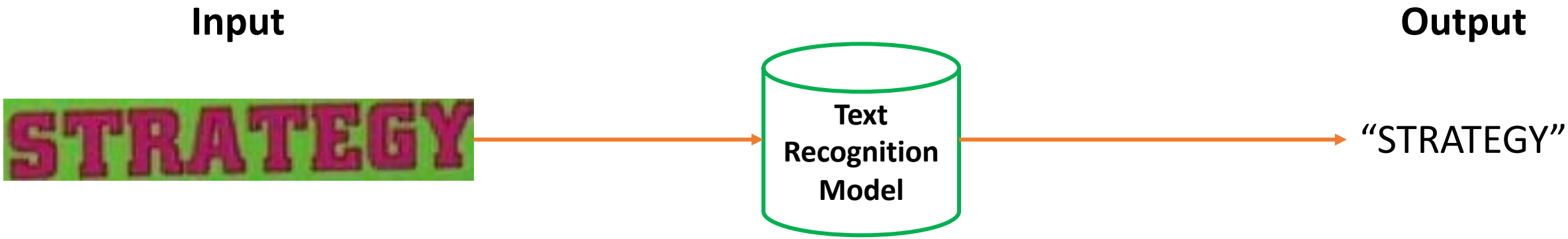
Text Recognition

❖ Project Pipeline



Text Recognition

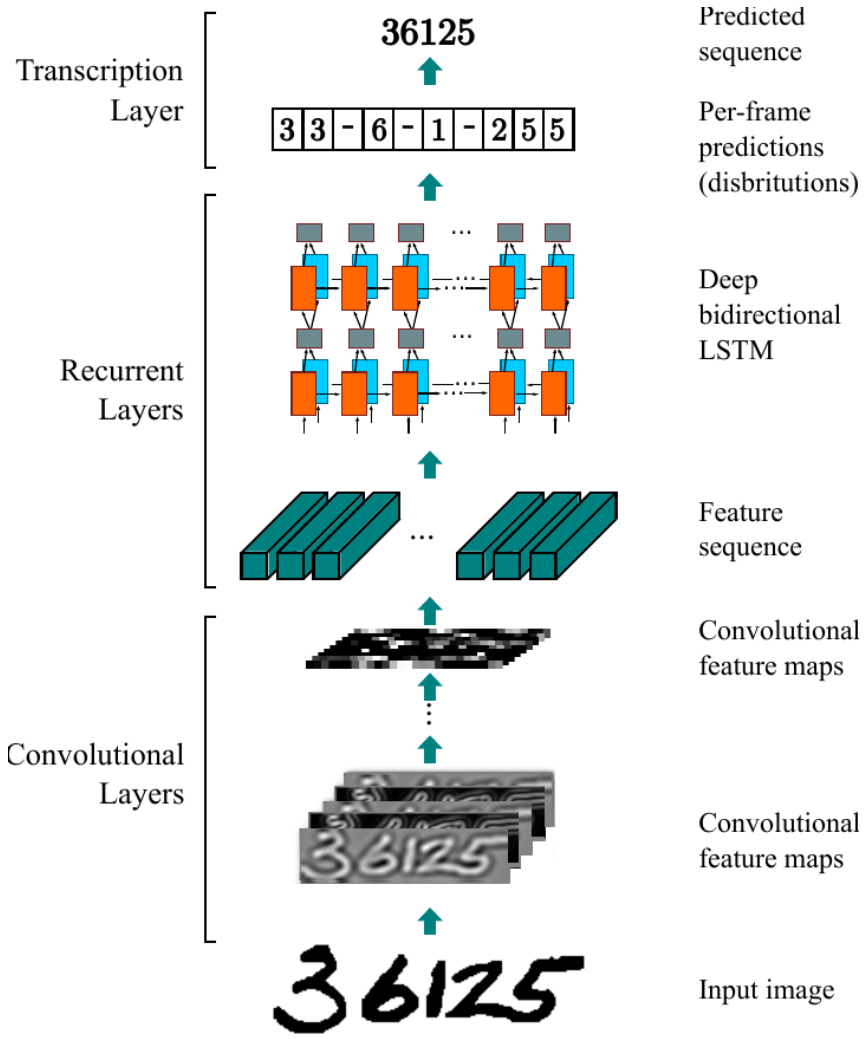
❖ Text Recognition I/O



Objective: Transcribe the text within input image into string.

Text Recognition

❖ CRNN



An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition

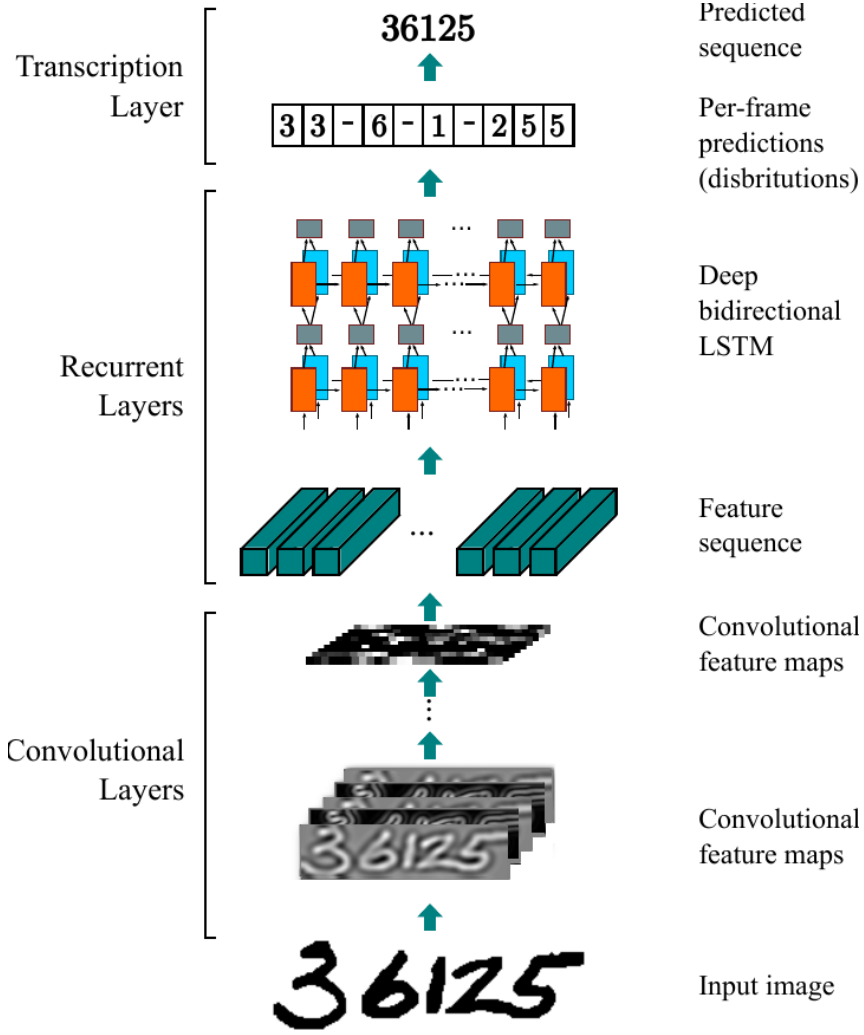
Baoguang Shi, Xiang Bai and Cong Yao
School of Electronic Information and Communications
Huazhong University of Science and Technology, Wuhan, China
{shibaoguang,xbai}@hust.edu.cn, yaocong2010@gmail.com

CRNN (Convolutional Recurrent Neural Network): A type of neural network that combines CNNs and RNNs to process images with sequential data, particularly effective for tasks like STR and OCR.

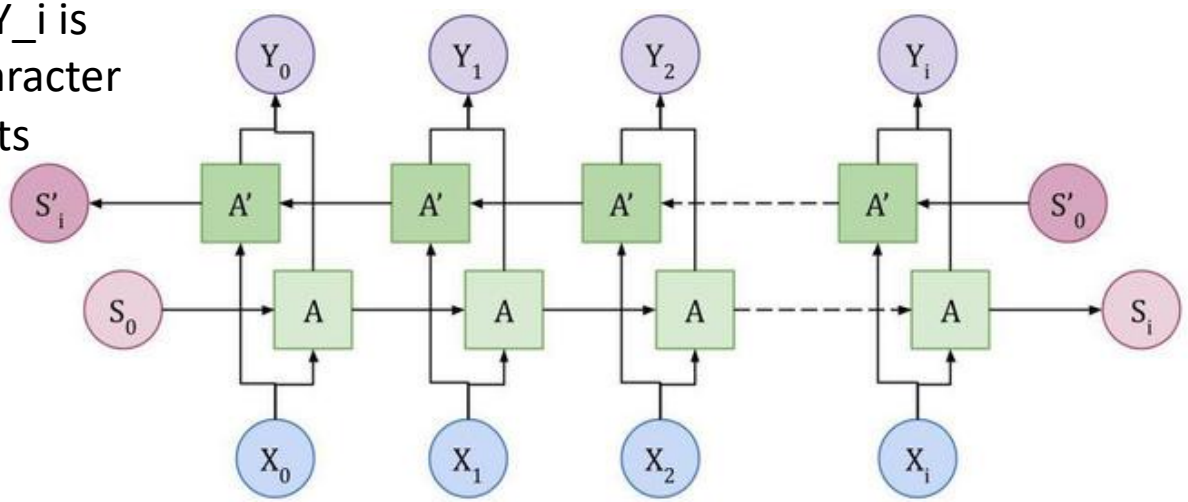
CRNN: https://arxiv.org/pdf/1507.05717.pdf

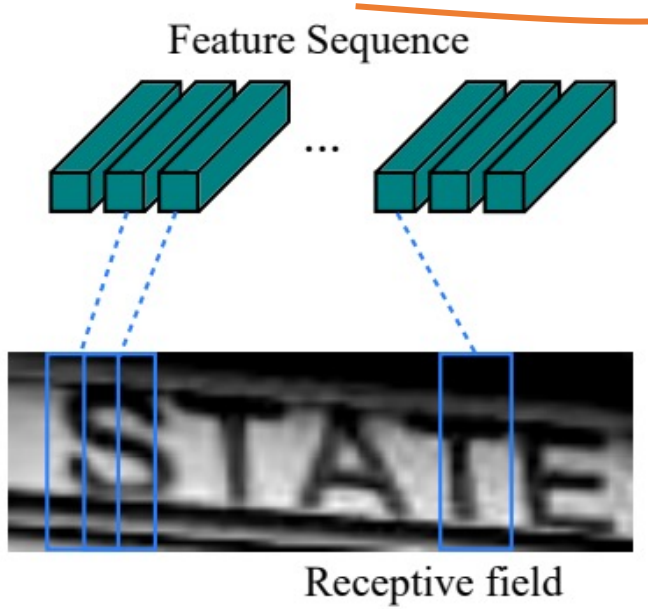
Text Recognition

❖ CRNN



Each predicted Y_i is represented character inside X_i if exists

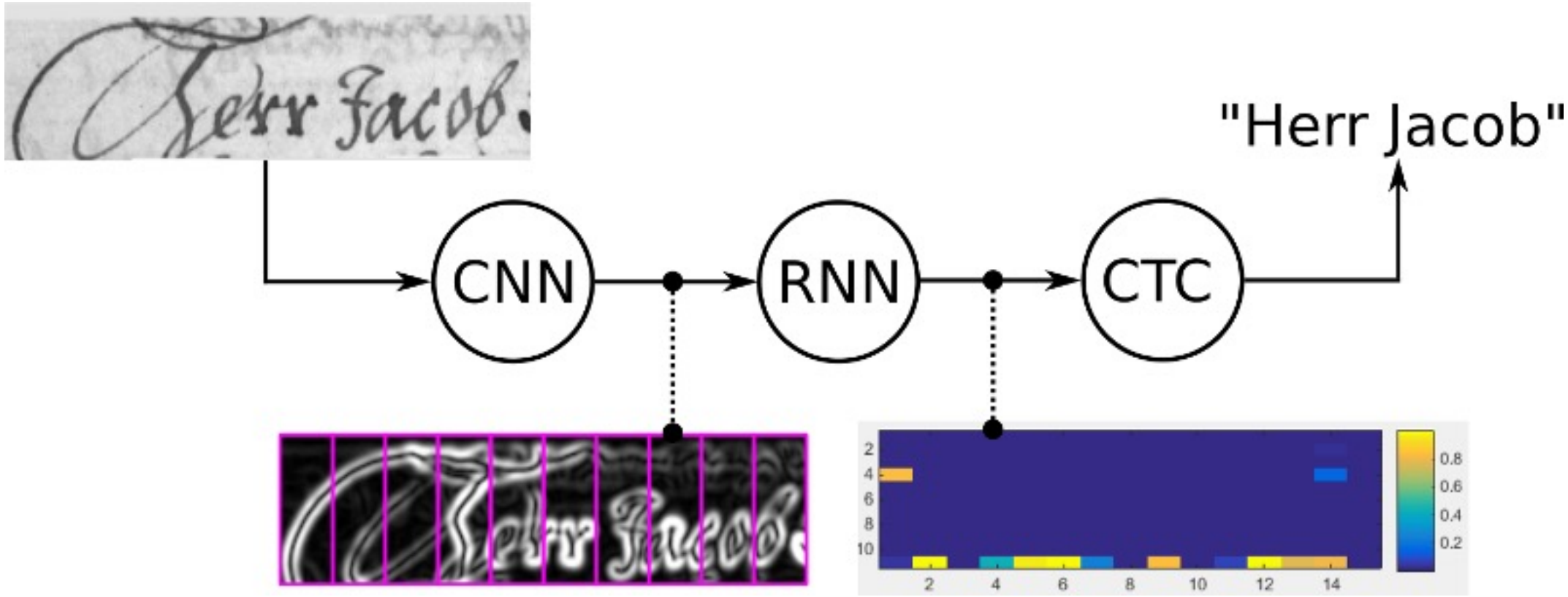




Consider each feature vector as a token for RNN model.

Text Recognition

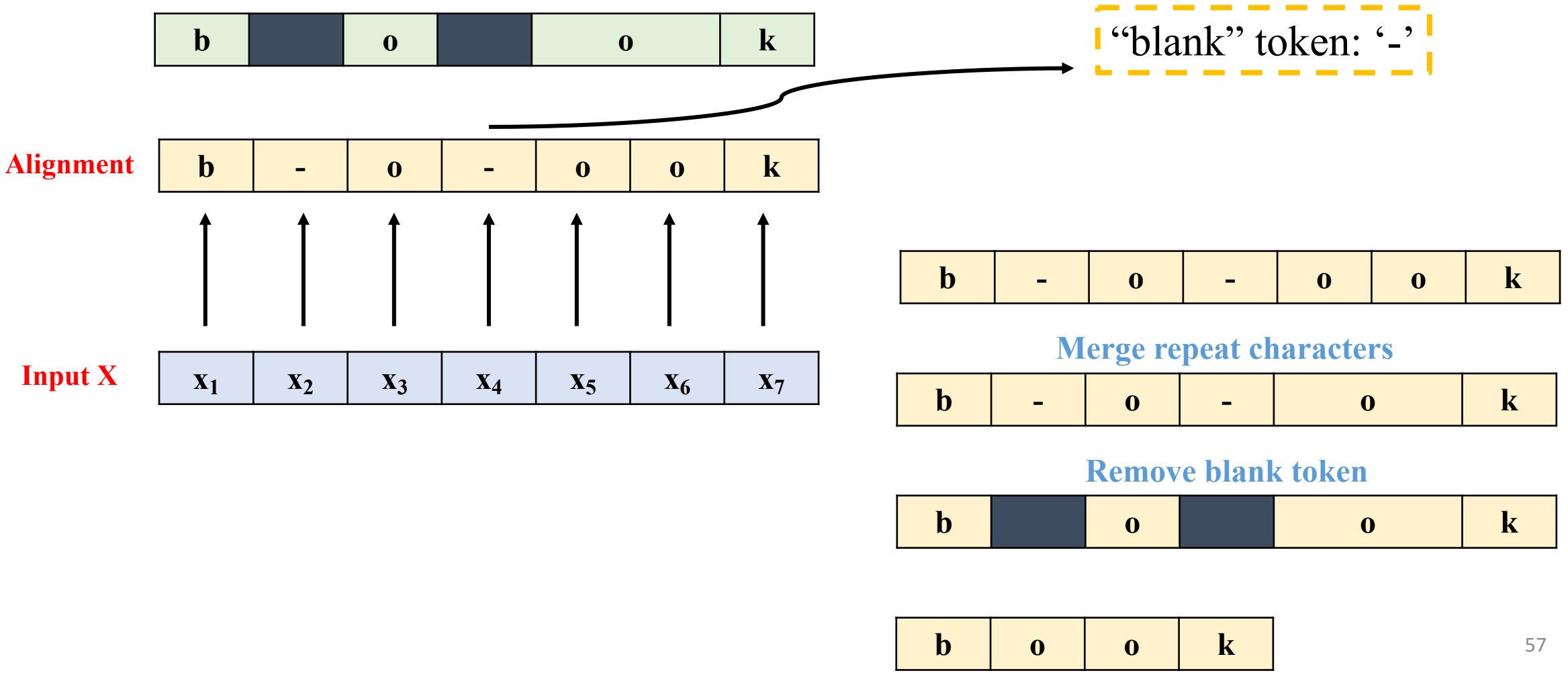
❖ CRNN



https://viblo.asia/p/tim-hieu-bai-toan-ocr-voi-crnn-va-ctc-loss-ocr-from-scratch-with-pytorch-p1-OeVKBA905kW

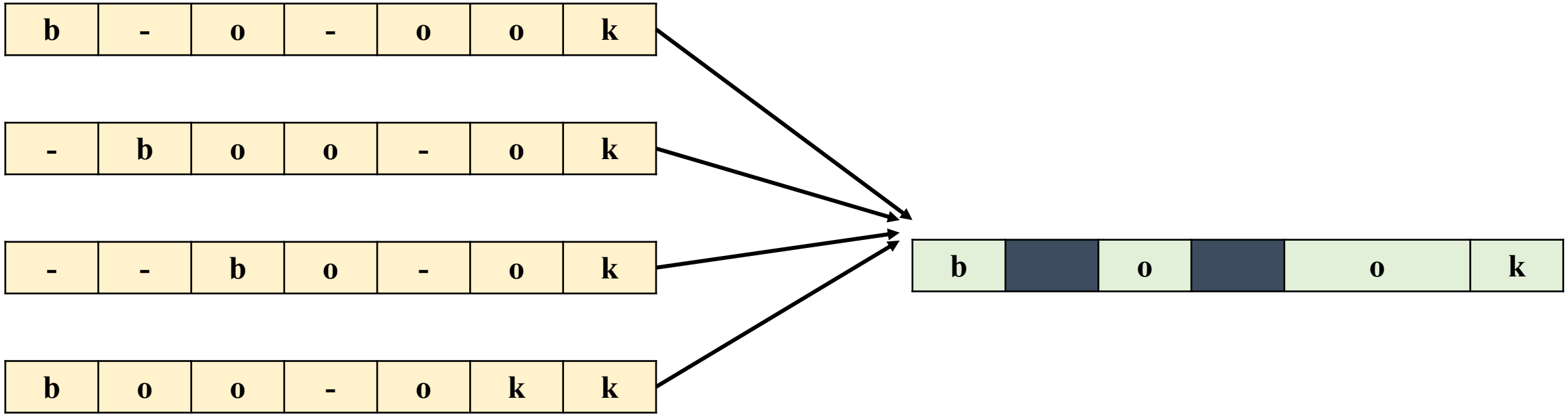
CTC Loss

❖ Connectionist Temporal Classification (CTC)



CTC Loss

❖ Connectionist Temporal Classification (CTC)

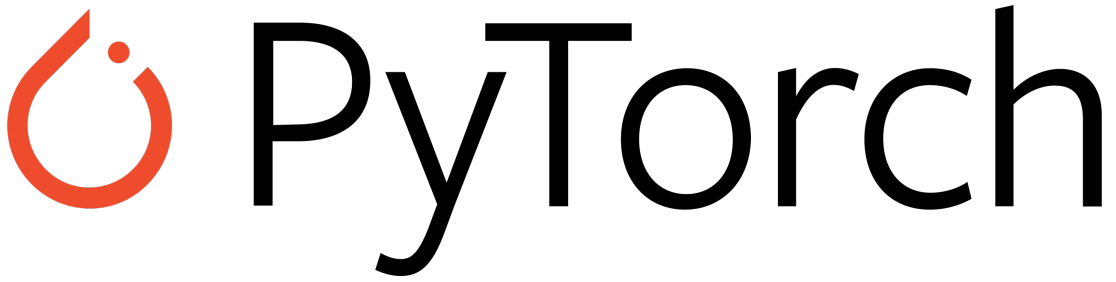


Many alignments lead to the same output

Text Recognition

❖ Step 1: Import libraries

```
import os
import shutil
import xml.etree.ElementTree as ET
import numpy as np
import pandas as pd
import random
import cv2
import timm
import matplotlib.pyplot as plt
from PIL import Image

import torch
import torch.nn as nn
import torchvision
from torch.nn import functional as F
from torchvision import models, transforms
from torch.utils.data import Dataset, DataLoader

from sklearn.model_selection import train_test_split
```



Text Recognition

❖ Step 2: Prepare dataset structure

Cropped Images

Text Recognition

❖ Step 2: Prepare dataset structure

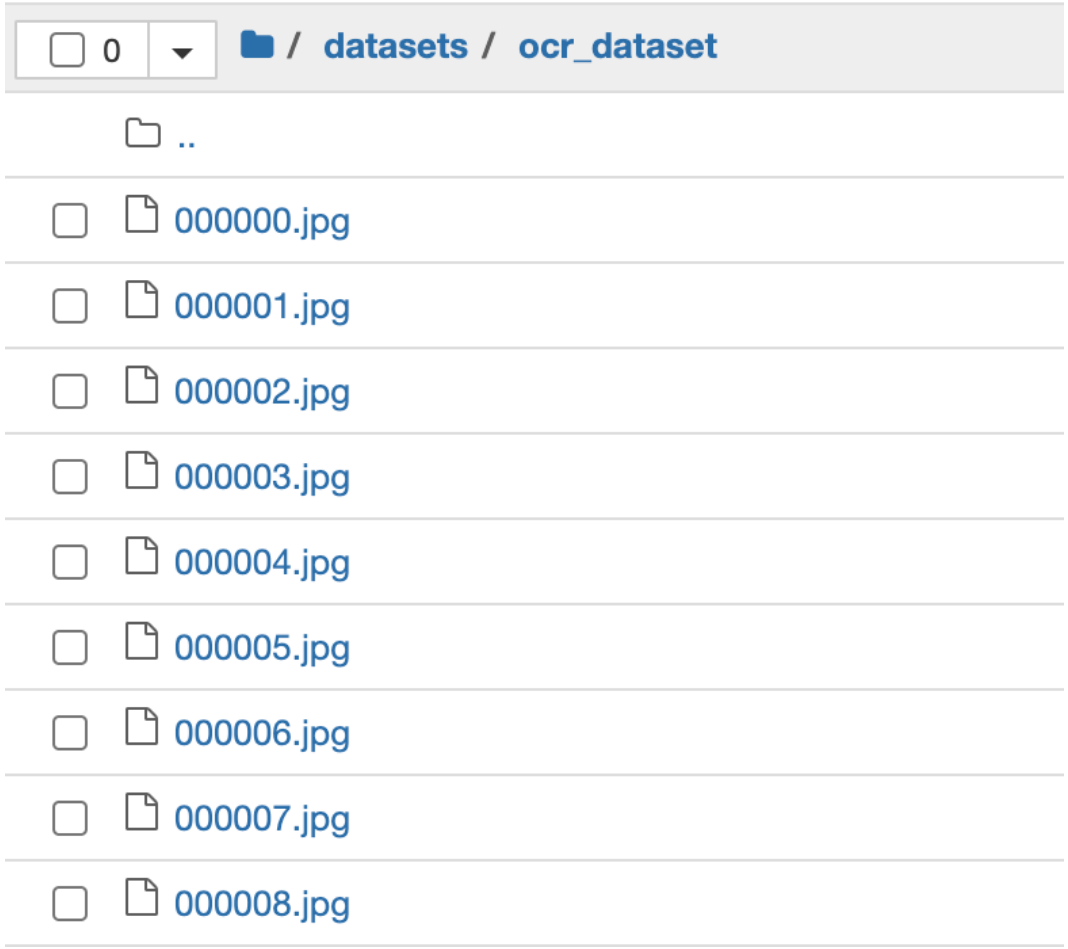


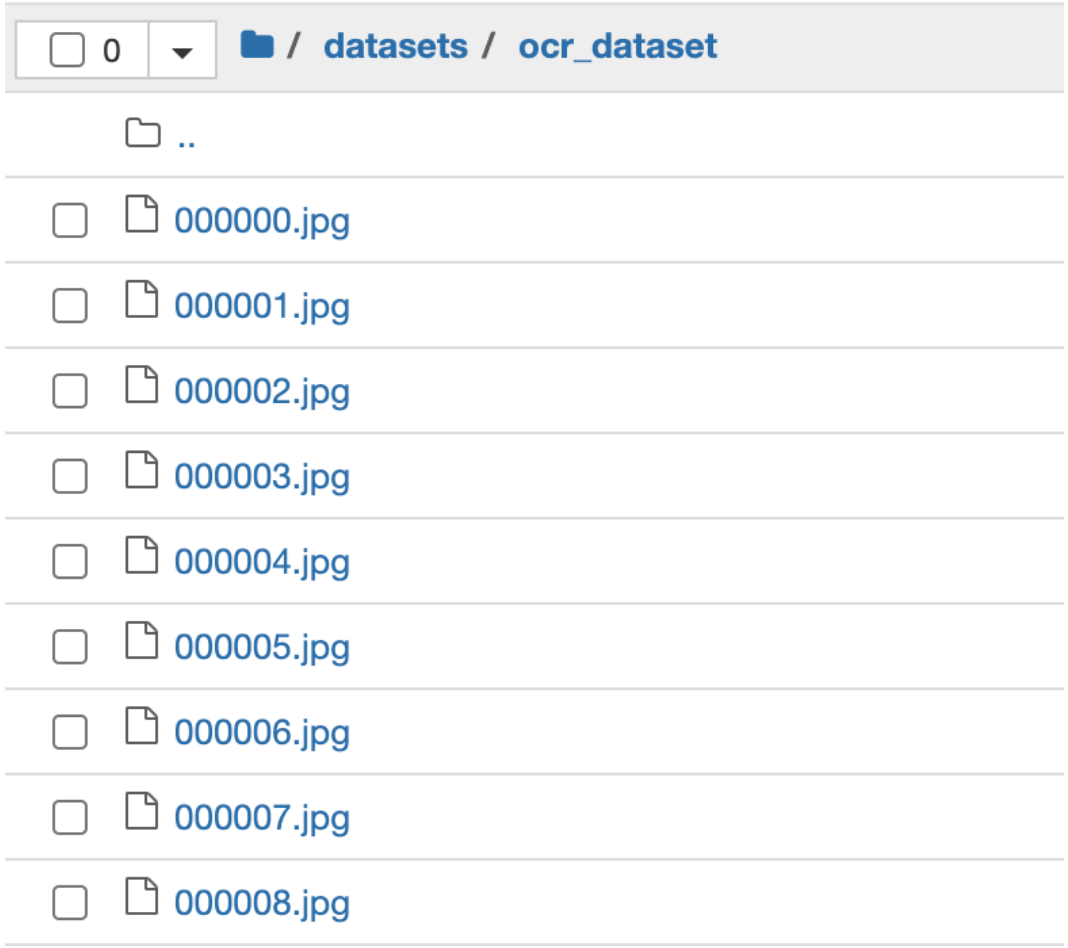
Image samples

```
datasets/ocr_dataset/000000.jpg self
datasets/ocr_dataset/000001.jpg adhesive
datasets/ocr_dataset/000002.jpg address
datasets/ocr_dataset/000003.jpg labels
datasets/ocr_dataset/000004.jpg 36
datasets/ocr_dataset/000005.jpg 250
datasets/ocr_dataset/000006.jpg on
datasets/ocr_dataset/000007.jpg a
datasets/ocr_dataset/000008.jpg roll
datasets/ocr_dataset/000009.jpg greek
datasets/ocr_dataset/000010.jpg gastronomy
datasets/ocr_dataset/000011.jpg cookery
datasets/ocr_dataset/000012.jpg wines
datasets/ocr_dataset/000013.jpg local
datasets/ocr_dataset/000014.jpg toubrs
datasets/ocr_dataset/000015.jpg specialties
datasets/ocr_dataset/000016.jpg festive
datasets/ocr_dataset/000017.jpg recipes
datasets/ocr_dataset/000018.jpg illustrated
datasets/ocr_dataset/000019.jpg 83
```

Labels.txt file

Text Recognition

❖ Step 2: Prepare dataset structure




```python
def split_bounding_boxes(img_paths, img_labels, bboxes, save_dir):
    os.makedirs(save_dir, exist_ok=True)

    count = 0
    labels = []  # List to store labels

    for img_path, img_label, bbs in zip(img_paths, img_labels, bboxes):
        img = Image.open(img_path)

        for label, bb in zip(img_label, bbs):
            # Crop image
            cropped_img = img.crop((bb[0], bb[1], bb[0] + bb[2], bb[1] + bb[3]))

            # filter out if 90% of the cropped image is black or white
            if np.mean(cropped_img) < 35 or np.mean(cropped_img) > 220:
                continue

            if cropped_img.size[0] < 10 or cropped_img.size[1] < 10:
                continue

            if len(img_label) < 3:
                continue

            # Save image
            filename = f"{count:06d}.jpg"
            cropped_img.save(os.path.join(save_dir, filename))
```

Text Recognition

❖ Step 2: Prepare dataset structure

```
datasets/ocr_dataset/000000.jpg	self
datasets/ocr_dataset/000001.jpg	adhesive
datasets/ocr_dataset/000002.jpg	address
datasets/ocr_dataset/000003.jpg	labels
datasets/ocr_dataset/000004.jpg	36
datasets/ocr_dataset/000005.jpg	250
datasets/ocr_dataset/000006.jpg	on
datasets/ocr_dataset/000007.jpg	a
datasets/ocr_dataset/000008.jpg	roll
datasets/ocr_dataset/000009.jpg	greek
datasets/ocr_dataset/000010.jpg	gastronomy
datasets/ocr_dataset/000011.jpg	cookery
datasets/ocr_dataset/000012.jpg	wines
datasets/ocr_dataset/000013.jpg	local
datasets/ocr_dataset/000014.jpg	toubrs
datasets/ocr_dataset/000015.jpg	specialties
datasets/ocr_dataset/000016.jpg	festive
datasets/ocr_dataset/000017.jpg	recipes
datasets/ocr_dataset/000018.jpg	illustrated
datasets/ocr_dataset/000019.jpg	83
```

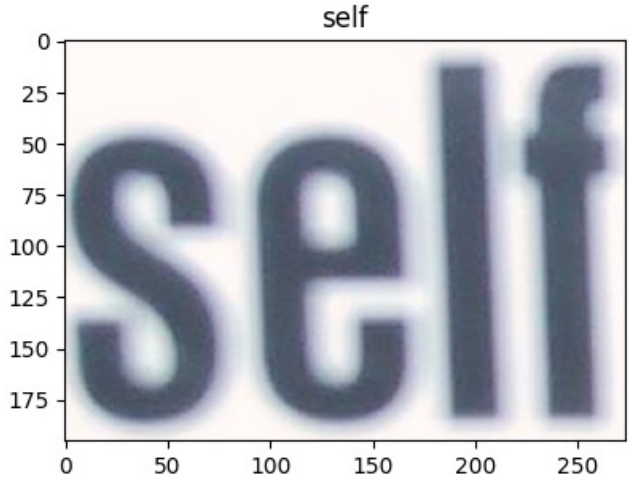
```
            new_img_path = os.path.join(save_dir, filename)

            label = new_img_path + '\t' + label

            labels.append(label)  # Append label to the list

            count += 1

print(f"Created {count} images")

# Write labels to a text file
with open(os.path.join(save_dir, 'labels.txt'), 'w') as f:
    for label in labels:
        f.write(f"{label}\n")
```

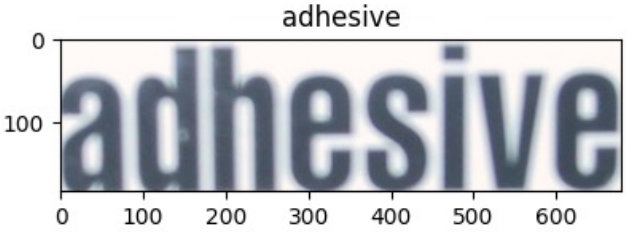
Text Recognition

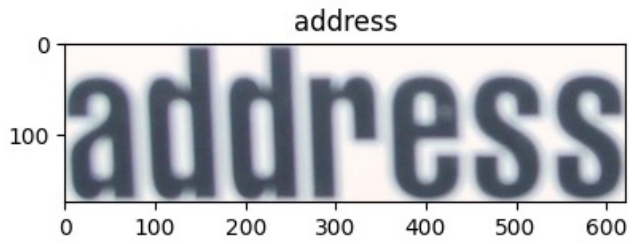
❖ Step 3: Read dataset

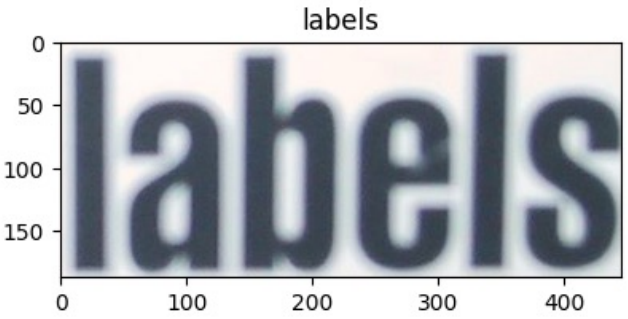
```
root_dir = save_dir

img_paths = []
labels = []

# Read labels from text file
with open(os.path.join(root_dir, 'labels.txt'), 'r') as f:
    for label in f:
        labels.append(label.strip().split('\t')[1])
        img_paths.append(label.strip().split('\t')[0])

print(f"Total images: {len(img_paths)}")
```

```
Total images: 1088
```









Text Recognition

❖ Step 4: Create vocabulary

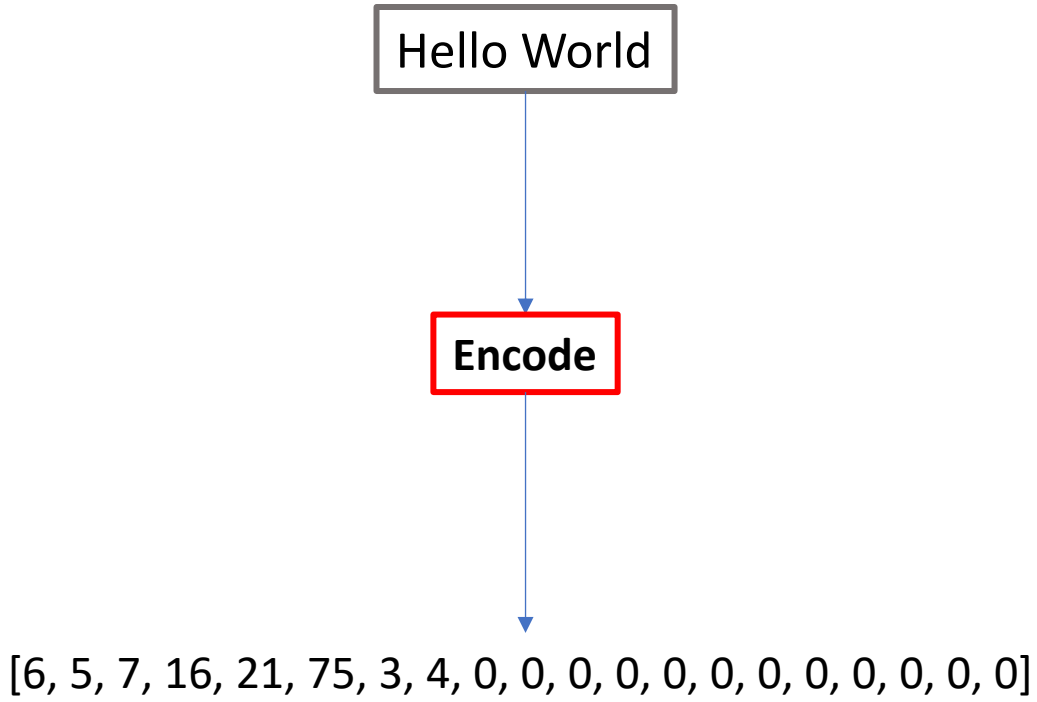
```
letters = [char.split(".")[0].lower() for char in labels]
letters = "".join(letters)
letters = sorted(list(set(list(letters))))

# create a string of all characters in the dataset
chars = "".join(letters)

# for "blank" character
blank_char = '-'
chars += blank_char
vocab_size = len(chars)

print(f'Vocab: {chars}')
print(f'Vocab size: {vocab_size}')
```

```
Vocab: 0123456789abcdefghijklmnopqrstuvwxyz-
Vocab size: 37
```

Text Recognition

❖ Step 5: Create encode/decode function

```
char_to_idx = {char: idx + 1 for idx, char in enumerate(sorted(chars))}
max_label_len = max([len(label) for label in labels])

def encode(label, char_to_idx, max_label_len):
    encoded_labels = torch.tensor(
        [char_to_idx[char] for char in label],
        dtype=torch.long
    )
    label_len = len(encoded_labels)
    lengths = torch.tensor(
        label_len,
        dtype=torch.long
    )
    padded_labels = F.pad(
        encoded_labels,
        (0, max_label_len - label_len),
        value=0
    )

    return padded_labels, lengths
```

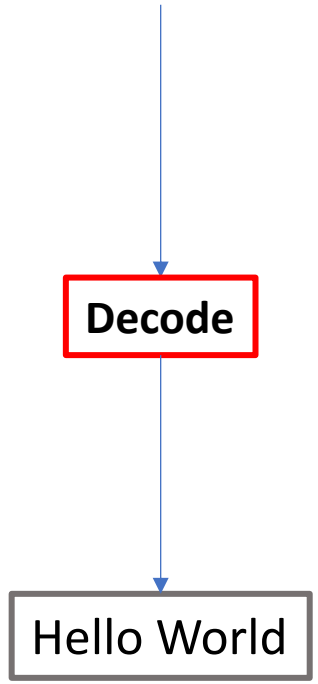


Text Recognition

❖ Step 5: Create encode/decode function

```
idx_to_char = {idx: char for char, idx in char_to_idx.items()}

def decode(encoded_sequences, idx_to_char, blank_char='-'):
    decoded_sequences = []

    for seq in encoded_sequences:
        decoded_label = []
        for idx, token in enumerate(seq):
            if token != 0:
                char = idx_to_char[token.item()]
                if char != blank_char:
                    decoded_label.append(char)

        decoded_sequences.append(''.join(decoded_label))

    return decoded_sequences
```

[6, 5, 7, 16, 21, 75, 3, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]



Text Recognition

❖ Step 6: Create data preprocessing function

Our dataset is complicated, so we should do data augmentation for better generalization

Text Recognition

❖ Step 6: Create data preprocessing function

```
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize((100, 420)),
        transforms.ColorJitter(
            brightness=0.5,
            contrast=0.5,
            saturation=0.5
        ),
        transforms.Grayscale(num_output_channels=1),
        transforms.GaussianBlur(3),
        transforms.RandomAffine(degrees=1, shear=1),
        transforms.RandomPerspective(
            distortion_scale=0.3,
            p=0.5,
            interpolation=3
        ),
        transforms.RandomRotation(degrees=2),
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,)),
    ]),
    'val': transforms.Compose([
        transforms.Resize((100, 420)),
        transforms.Grayscale(num_output_channels=1),
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,)),
    ]),
}
```

Training data after transfomation (data augmentaiton)

Text Recognition

❖ Step 7: Create pytorch datasets

```
class STRDataset(Dataset):
    def __init__(
        self,
        X, y,
        char_to_idx,
        max_label_len,
        label_encoder=None,
        transform=None
    ):
        self.transform = transform
        self.img_paths = X
        self.labels = y
        self.char_to_idx = char_to_idx
        self.max_label_len = max_label_len
        self.label_encoder = label_encoder

    def __len__(self):
        return len(self.img_paths)
```

```
def __getitem__(self, idx):
    label = self.labels[idx]
    img_path = self.img_paths[idx]
    img = Image.open(img_path).convert("RGB")

    if self.transform:
        img = self.transform(img)

    if self.label_encoder:
        encoded_label, label_len = self.label_encoder(
            label,
            self.char_to_idx,
            self.max_label_len
        )
    return img, encoded_label, label_len
```

Text Recognition

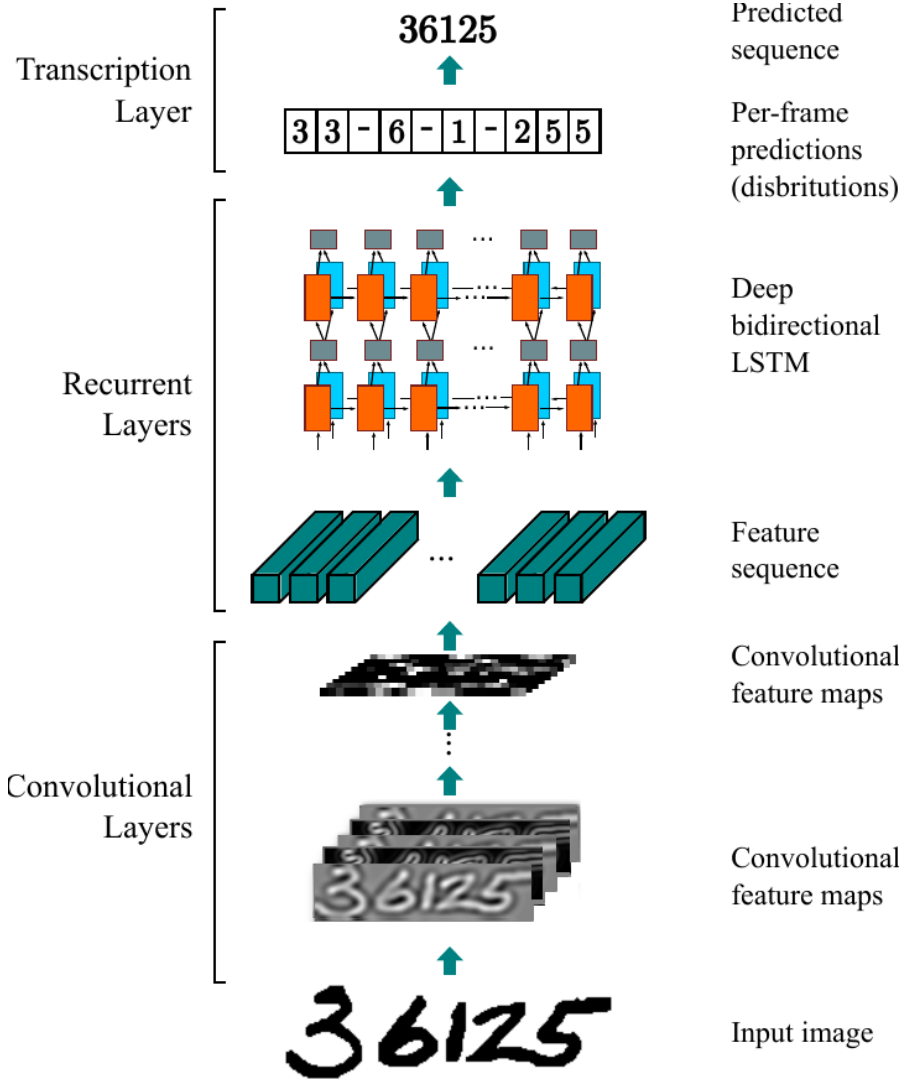
❖ Step 8: Create dataloader

```
train_dataset = STRDataset(
    X_train, y_train,
    char_to_idx=char_to_idx,
    max_label_len=max_label_len,
    label_encoder=encode,
    transform=data_transforms['train']
)
val_dataset = STRDataset(
    X_val, y_val,
    char_to_idx=char_to_idx,
    max_label_len=max_label_len,
    label_encoder=encode,
    transform=data_transforms['val']
)
test_dataset = STRDataset(
    X_test, y_test,
    char_to_idx=char_to_idx,
    max_label_len=max_label_len,
    label_encoder=encode,
    transform=data_transforms['val']
)
```

```
train_batch_size = 32
test_batch_size = 8

train_loader = DataLoader(
    train_dataset,
    batch_size=train_batch_size,
    shuffle=True
)
val_loader = DataLoader(
    val_dataset,
    batch_size=test_batch_size,
    shuffle=False
)
test_loader = DataLoader(
    test_dataset,
    batch_size=test_batch_size,
    shuffle=False
)
```

Text Recognition

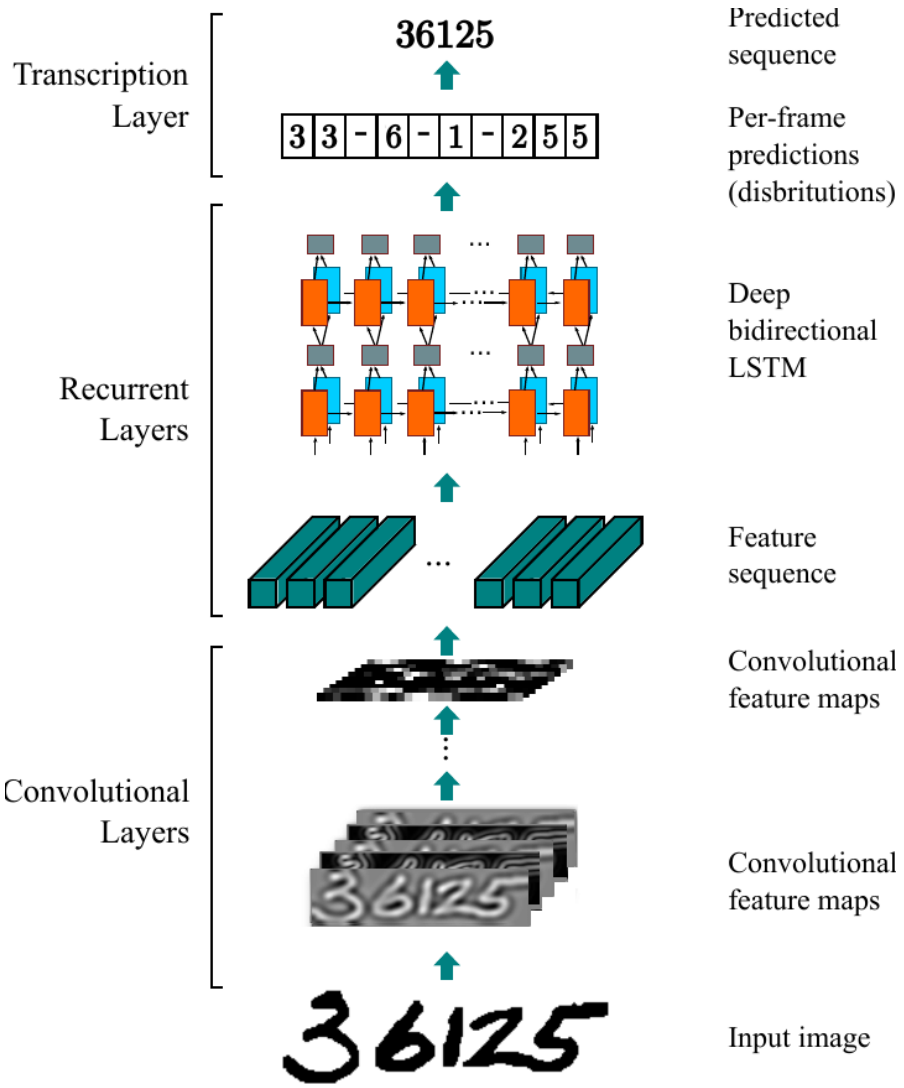
❖ Step 9: Create model




```python
class CRNN(nn.Module):
    def __init__(
        self,
        vocab_size,
        hidden_size,
        n_layers,
        dropout=0.2,
        unfreeze_layers=3
    ):
        super(CRNN, self).__init__()

        backbone = timm.create_model(
            'resnet101',
            in_chans=1,
            pretrained=True
        )
        modules = list(backbone.children())[:-2]
        modules.append(nn.AdaptiveAvgPool2d((1, None)))
        self.backbone = nn.Sequential(*modules)

        # Unfreeze the last few layers
        for parameter in self.backbone[-unfreeze_layers:].parameters():
            parameter.requires_grad = True

        self.mapSeq = nn.Sequential(
            nn.Linear(2048, 512),
            nn.ReLU(),
            nn.Dropout(dropout)
        )

        self.lstm = nn.LSTM(
            512, hidden_size,
            n_layers, bidirectional=True, batch_first=True,
            dropout=dropout if n_layers > 1 else 0
        )
        self.layer_norm = nn.LayerNorm(hidden_size * 2)
        self.out = nn.Sequential(
            nn.Linear(hidden_size * 2, vocab_size),
            nn.LogSoftmax(dim=2)
        )
```

Text Recognition

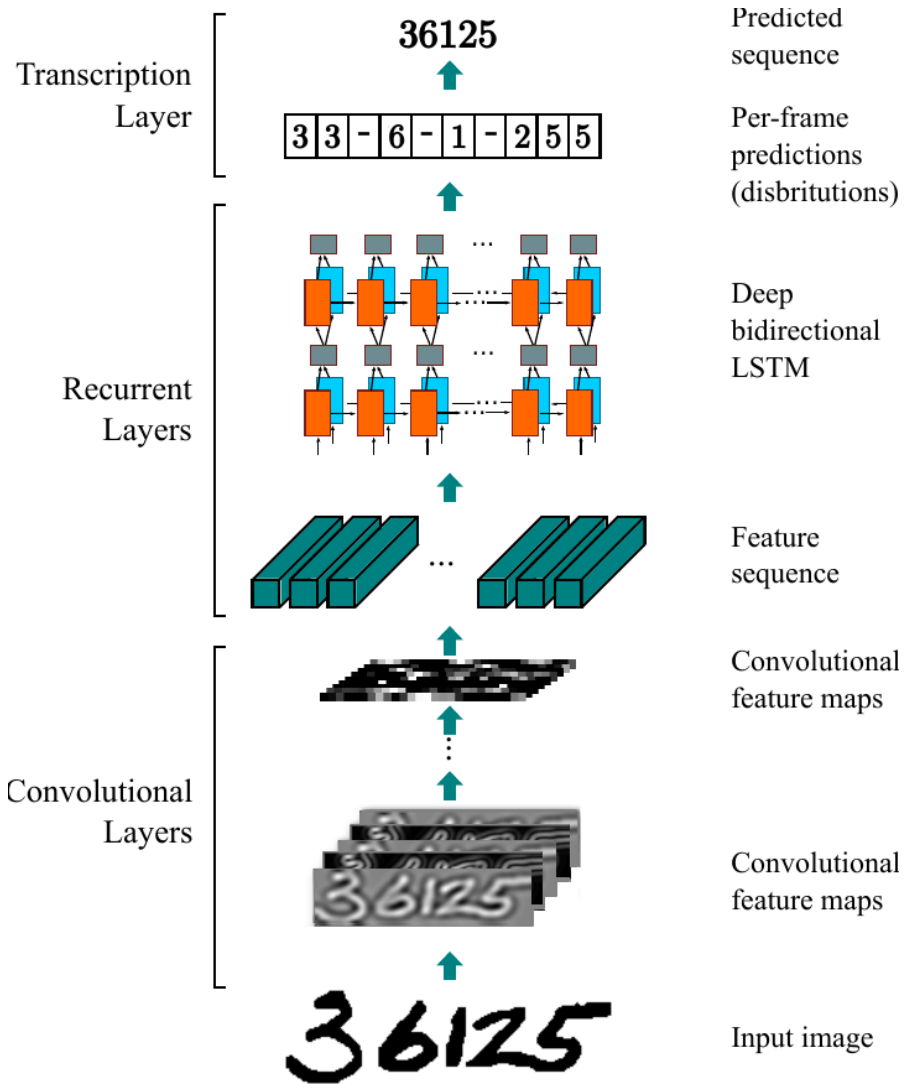
❖ Step 9: Create model




```python
def forward(self, x):
    x = self.backbone(x)
    x = x.permute(0, 3, 1, 2)
    x = x.view(x.size(0), x.size(1), -1)  #
    x = self.mapSeq(x)
    x, _ = self.lstm(x)
    x = self.layer_norm(x)
    x = self.out(x)
    x = x.permute(1, 0, 2) # Based on CTC

    return x
```

Text Recognition

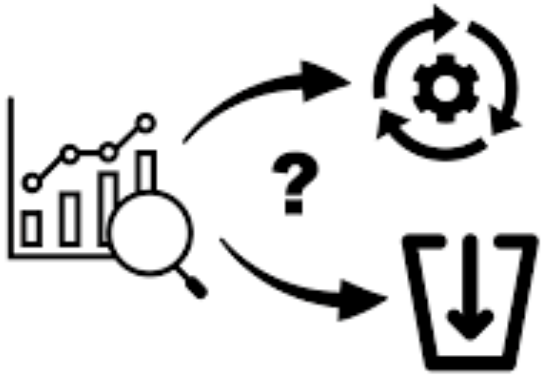
❖ Step 9: Create model

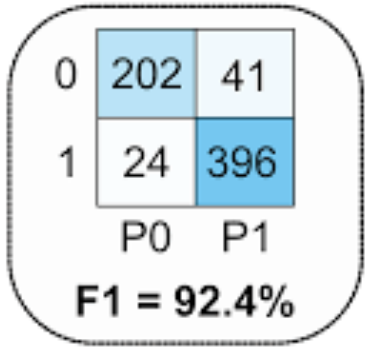



```
hidden_size = 256
n_layers = 3
dropout_prob = 0.2
unfreeze_layers=3
device = 'cuda' if torch.cuda.is_available() else 'cpu'

model = CRNN(
    vocab_size=vocab_size,
    hidden_size=hidden_size,
    n_layers=n_layers,
    dropout=dropout_prob,
    unfreeze_layers=unfreeze_layers
).to(device)
```

Text Recognition

❖ Step 10: Create training and evaluate function






```python
def evaluate(model, dataloader, criterion, device):
    model.eval()
    losses = []
    with torch.no_grad():
        for inputs, labels, labels_len in dataloader:
            inputs = inputs.to(device)
            labels = labels.to(device)
            labels_len = labels_len.to(device)

            outputs = model(inputs)
            logits_lens = torch.full(
                size=(outputs.size(1),),
                fill_value=outputs.size(0),
                dtype=torch.long
            ).to(device)

            loss = criterion(
                outputs, labels, logits_lens, labels_len
            )
            losses.append(loss.item())

    loss = sum(losses) / len(losses)

    return loss
```

Text Recognition

❖ Step 10: Create training and evaluate function

```python
def fit(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    scheduler,
    device,
    epochs
):
    train_losses = []
    val_losses = []

    for epoch in range(epochs):
        batch_train_losses = []

        model.train()
        for idx, (inputs, labels, labels_len) in enumerate(train_loader):
            inputs = inputs.to(device)
            labels = labels.to(device)
            labels_len = labels_len.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)

            logits_lens = torch.full(
                size=(outputs.size(1),),
                fill_value=outputs.size(0),
                dtype=torch.long
            ).to(device)

            loss = criterion(outputs, labels, logits_lens, labels_len)
            loss.backward()
            torch.nn.utils.clip_grad_norm_(
                model.parameters(),
                5
            )
            optimizer.step()

            batch_train_losses.append(loss.item())

        train_loss = sum(batch_train_losses) / len(batch_train_losses)
        train_losses.append(train_loss)

        val_loss = evaluate(
            model, val_loader,
            criterion, device
        )
        val_losses.append(val_loss)

        print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tVal loss: {val_loss:.4f}')

        scheduler.step()

    return train_losses, val_losses
```

Text Recognition

❖ Step 11: Training

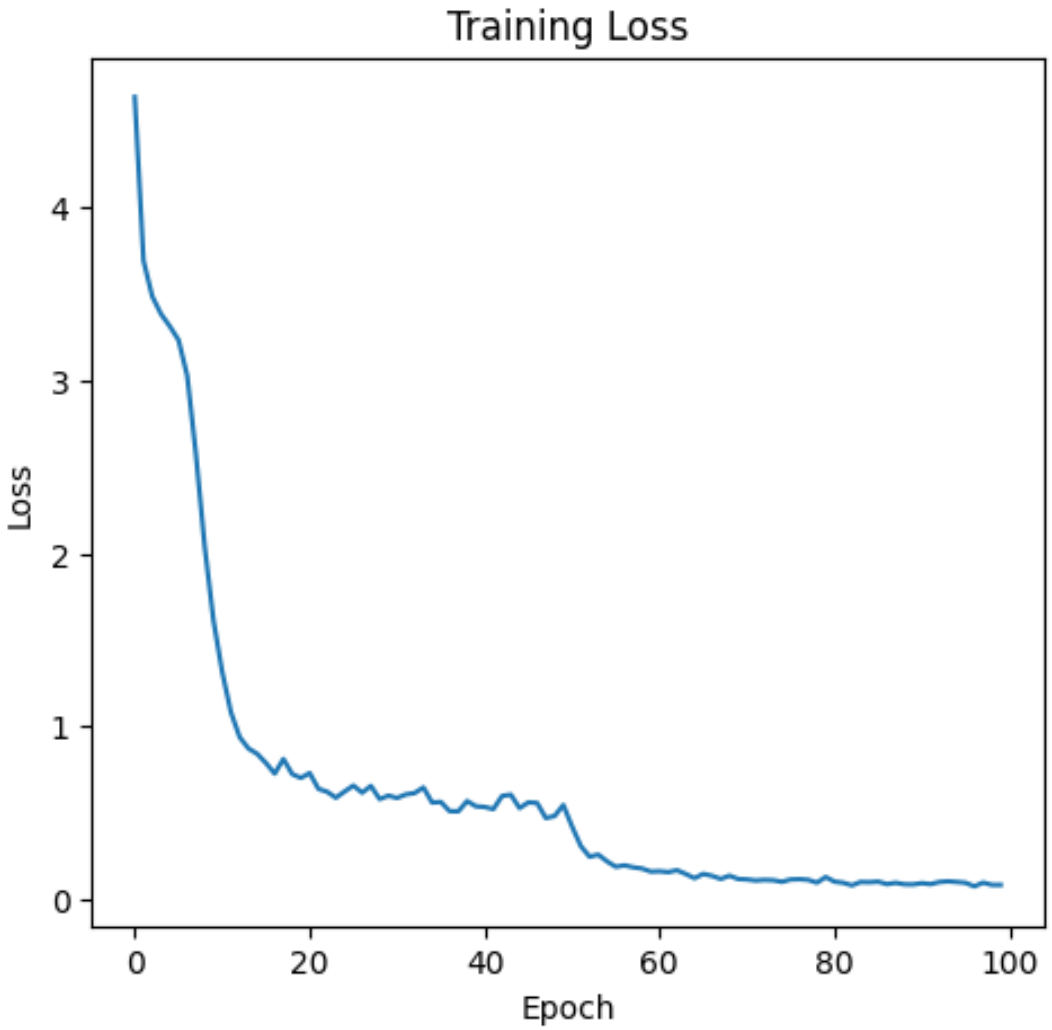
```
epochs = 100
lr = 0.001
weight_decay=1e-5
scheduler_step_size = epochs * 0.4

criterion = nn.CTCLoss(
    blank=char_to_idx[blank_char],
    zero_infinity=True
)
optimizer = torch.optim.Adam(
    model.parameters(),
    lr=lr,
    weight_decay=weight_decay
)
scheduler = torch.optim.lr_scheduler.StepLR(
    optimizer,
    step_size=scheduler_step_size,
    gamma=0.1
)
```

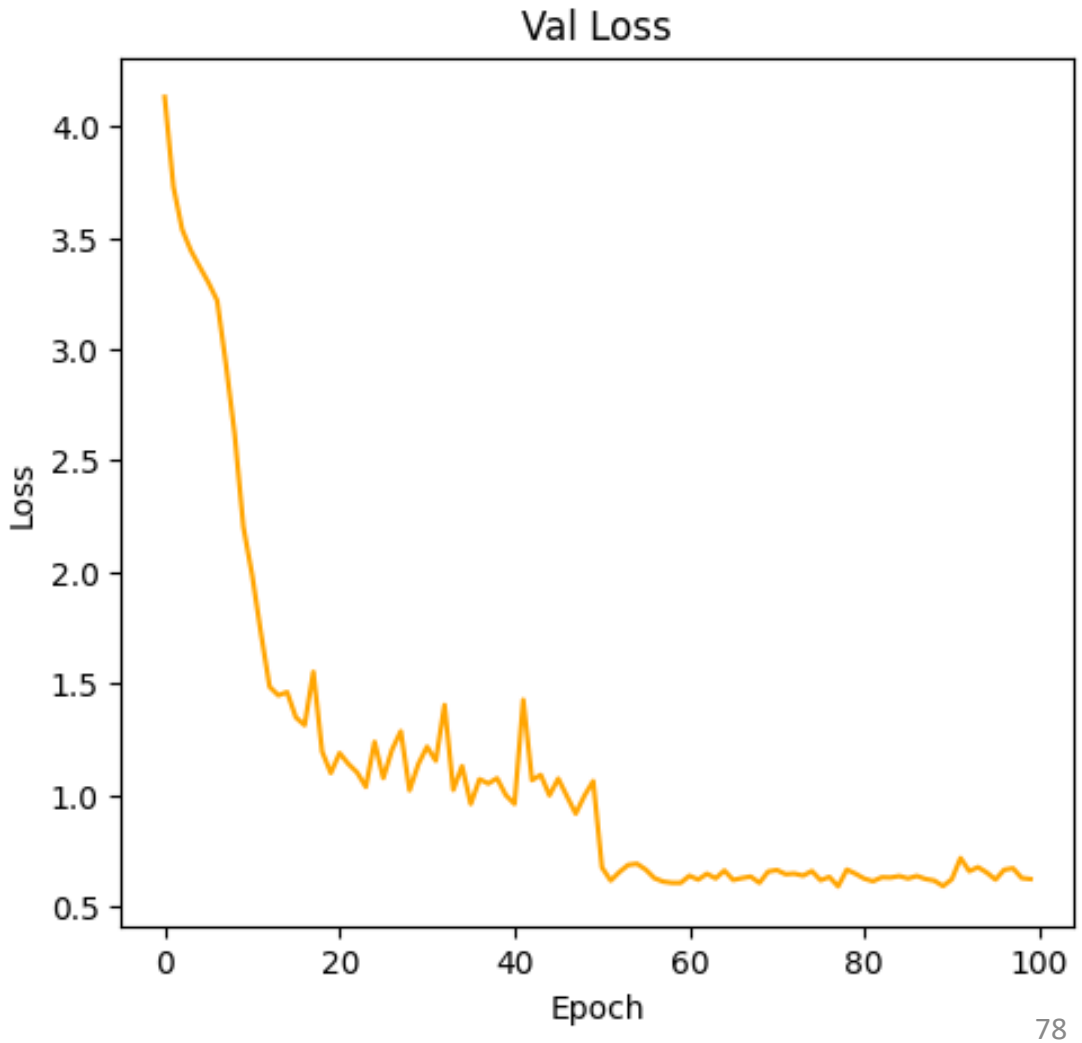
```
train_losses, val_losses = fit(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    scheduler,
    device,
    epochs
)
```

```
EPOCH 1:        Train loss: 4.2350      Val loss: 3.8063
EPOCH 2:        Train loss: 3.5230      Val loss: 3.4833
EPOCH 3:        Train loss: 3.4020      Val loss: 3.4674
EPOCH 4:        Train loss: 3.2737      Val loss: 3.3083
EPOCH 5:        Train loss: 3.0089      Val loss: 3.0442
EPOCH 6:        Train loss: 2.7299      Val loss: 2.7941
EPOCH 7:        Train loss: 2.4106      Val loss: 2.4425
EPOCH 8:        Train loss: 2.0683      Val loss: 2.0739
EPOCH 9:        Train loss: 1.6665      Val loss: 1.7921
EPOCH 10:       Train loss: 1.3355      Val loss: 1.4708
EPOCH 11:       Train loss: 1.0806      Val loss: 1.3984
EPOCH 12:       Train loss: 0.9070      Val loss: 1.2445
EPOCH 13:       Train loss: 0.7567      Val loss: 1.2226
EPOCH 14:       Train loss: 0.7110      Val loss: 1.0825
EPOCH 15:       Train loss: 0.6035      Val loss: 0.9752
EPOCH 16:       Train loss: 0.4918      Val loss: 1.1918
EPOCH 17:       Train loss: 0.4615      Val loss: 1.0801
EPOCH 18:       Train loss: 0.4367      Val loss: 1.0798
EPOCH 19:       Train loss: 0.4071      Val loss: 0.9820
EPOCH 20:       Train loss: 0.3887      Val loss: 0.8523
```

Text Recognition

❖ Step 11: Training





Text Recognition

❖ Step 12: Evaluation

```
val_loss = evaluate(
    model,
    val_loader,
    criterion,
    device
)
test_loss = evaluate(
    model,
    test_loader,
    criterion,
    device
)

print('Evaluation on val/test dataset')
print('Val loss: ', val_loss)
print('Test loss: ', test_loss)
```

```
Evaluation on val/test dataset
Val loss:  0.6107446316629648
Test loss:  0.7212445609844648
```

```
def evaluate(model, dataloader, criterion, device):
    model.eval()
    losses = []
    with torch.no_grad():
        for inputs, labels, labels_len in dataloader:
            inputs = inputs.to(device)
            labels = labels.to(device)
            labels_len = labels_len.to(device)

            outputs = model(inputs)
            logits_lens = torch.full(
                size=(outputs.size(1),),
                fill_value=outputs.size(0),
                dtype=torch.long
            ).to(device)

            loss = criterion(
                outputs, labels, logits_lens, labels_len
            )
            losses.append(loss.item())

    loss = sum(losses) / len(losses)

    return loss
```

Text Recognition

❖ Step 13: Save model

```
save_model_path = 'models/ocr_crnn_base_best.pt'
torch.save(
    model.state_dict(),
    save_model_path
)
```

Saving & Loading Model for Inference

Save/Load `state_dict` (Recommended)

Save:

```
torch.save(model.state_dict(), PATH)
```

Load:

```
model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load(PATH))
model.eval()
```

Text Recognition

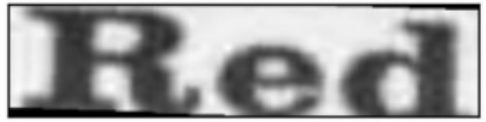
❖ Step 12: Evaluation

Truth: no | Pred: no

Truth: new | Pred: new

Truth: norvig | Pred: norvig

Truth: red | Pred: rred



Truth: caution | Pred: caautioonn

Truth: st | Pred: stt

Truth: willcocks | Pred: willcocks

Truth: labels | Pred: llabeelss

Truth: ltd | Pred: lltd

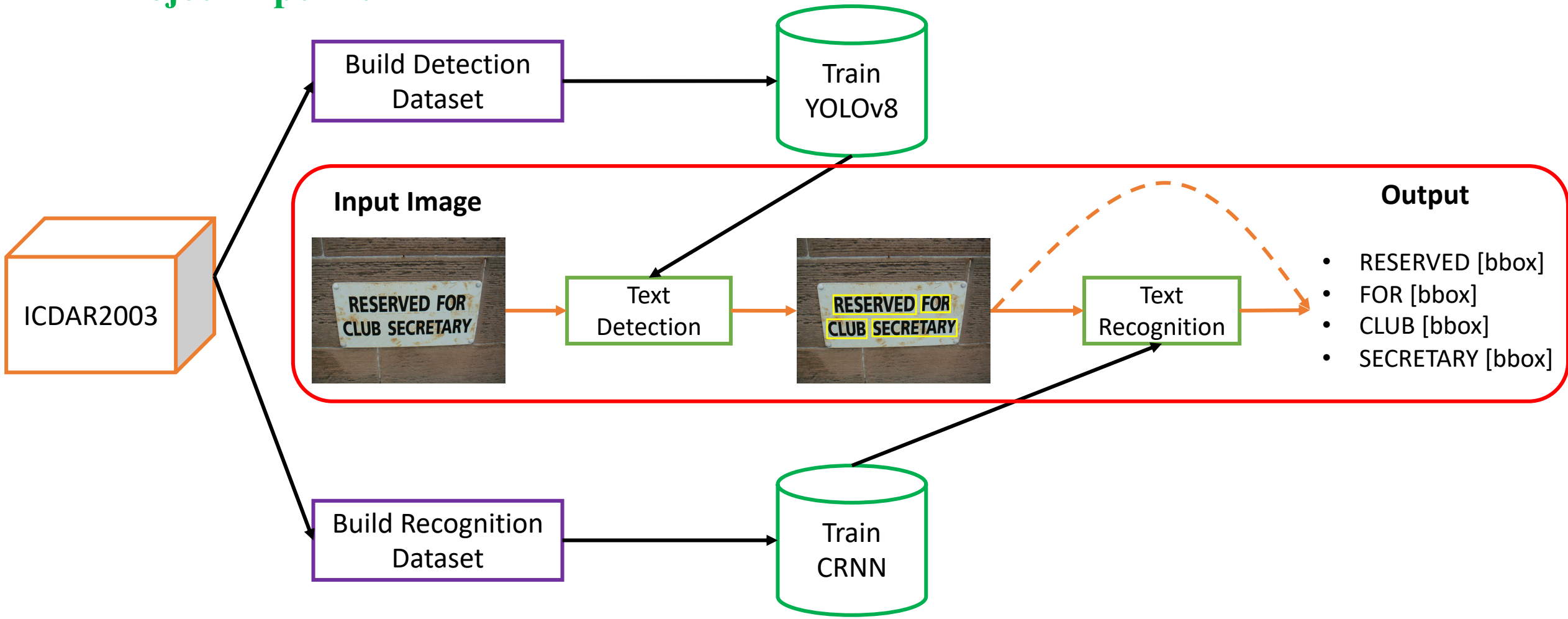
Truth: light | Pred: liighht

Truth: java | Pred: jaavaa

Truth: no | Pred: no

End-to-End Pipeline

❖ Project Pipeline



End-to-End Pipeline

❖ Step 1: Import libaries

```
%pip install ultralytics
import ultralytics
ultralytics.checks()
```

```
Ultralytics YOLOv8.0.222 🚀 Python-3.11.5 torch-2.1.0 CUDA:0 (NVIDIA GeForce RTX 3060, 12036MiB)
Setup complete ✅ (20 CPUs, 31.1 GB RAM, 390.6/915.3 GB disk)
```

```
import os
import numpy as np
import timm
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
from torchvision import models
from torchvision import transforms
from PIL import Image
```

End-to-End Pipeline

❖ Step 2: Load model

2.1 Load YOLO

```
1 from ultralytics import YOLO
2
3 text_det_model_path = 'models/yolov8/detect/train/weights/best.pt'
4 yolo = YOLO(text_det_model_path)
```

End-to-End Pipeline

❖ Step 2: Load model

```
chars = '0123456789abcdefghijklmnopqrstuvwxyz-'
vocab_size = len(chars)
char_to_idx = {char: idx + 1 for idx, char in enumerate(sorted(chars))}
idx_to_char = {idx: char for char, idx in char_to_idx.items()}
```

Saving & Loading Model for Inference

Save/Load `state_dict` (Recommended)

Save:

```
torch.save(model.state_dict(), PATH)
```

Load:

```
model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load(PATH))
model.eval()
```

```
hidden_size = 256
n_layers = 3
dropout_prob = 0.2
unfreeze_layers=3
device = 'cuda' if torch.cuda.is_available() else 'cpu'
model_path = 'models/ocr_crnn_base_best.pt'

crnn_model = CRNN(
    vocab_size=vocab_size,
    hidden_size=hidden_size,
    n_layers=n_layers,
    dropout=dropout_prob,
    unfreeze_layers=unfreeze_layers
).to(device)
crnn_model.load_state_dict(torch.load(model_path))
```

End-to-End Pipeline

❖ Step 2: Load model

```
1 chars = '0123456789abcdefghijklmnopqrstuvwxyz-'
2 vocab_size = len(chars)
3 char_to_idx = {char: idx + 1 for idx, char in enumerate(sorted(chars))}
4 idx_to_char = {idx: char for char, idx in char_to_idx.items()}
```

Saving & Loading Model for Inference

Save/Load `state_dict` (Recommended)

Save:

```
torch.save(model.state_dict(), PATH)
```

Load:

```
model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load(PATH))
model.eval()
```

```
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 model_path = 'models/ocr_crnn_base_best.pt'
7
8 crnn_model = CRNN(
9     vocab_size=vocab_size,
10    hidden_size=hidden_size,
11    n_layers=n_layers,
12    dropout=dropout_prob,
13    unfreeze_layers=unfreeze_layers
14 ).to(device)
15 crnn_model.load_state_dict(torch.load(model_path))
```

Load CRNN model

End-to-End Pipeline

❖ Step 3: Create text detection, text recognition, decode function

```python
def text_detection(img_path, text_det_model):
    text_det_results = text_det_model(img_path, verbose=False)[0]

    bboxes = text_det_results.boxes.xyxy.tolist()
    classes = text_det_results.boxes.cls.tolist()
    names = text_det_results.names
    confs = text_det_results.boxes.conf.tolist()

    return bboxes, classes, names, confs
```

```python
def text_recognition(img, data_transforms, text_reg_model, idx_to_char, device):
    transformed_image = data_transforms(img)
    transformed_image = transformed_image.unsqueeze(0).to(device)
    text_reg_model.eval()
    with torch.no_grad():
        logits = text_reg_model(transformed_image).detach().cpu()
    text = decode(logits.permute(1, 0, 2).argmax(2), idx_to_char)

    return text
```

```python
def decode(encoded_sequences, idx_to_char, blank_char='-'):
    decoded_sequences = []

    for seq in encoded_sequences:
        decoded_label = []
        for idx, token in enumerate(seq):
            if token != 0:
                char = idx_to_char[token.item()]
                if char != blank_char:
                    decoded_label.append(char)

        decoded_sequences.append(''.join(decoded_label))

    return decoded_sequences
```

End-to-End Pipeline

❖ Step 4: Create visualization function

```python
def visualize_detections(img, detections):
    plt.figure(figsize=(12, 8))
    plt.imshow(img)
    plt.axis('off')

    for bbox, detected_class, confidence, transcribed_text in detections:
        x1, y1, x2, y2 = bbox
        plt.gca().add_patch(
            plt.Rectangle(
                (x1, y1), x2 - x1, y2 - y1,
                fill=False, edgecolor='red', linewidth=2)
            )
        plt.text(
            x1, y1 - 10, f"{detected_class} ({confidence:.2f}): {transcribed_text}",
            fontsize=9, bbox=dict(facecolor='red', alpha=0.5)
        )

    plt.show()
```

End-to-End Pipeline

❖ Step 5: Create predict function

```python
def predict(img_path, data_transforms, text_det_model, text_reg_model, idx_to_char, device):
    # Detection
    bboxes, classes, names, confs = text_detection(img_path, text_det_model)

    # Load the image
    img = Image.open(img_path)

    predictions = []

    # Iterate through the results
    for bbox, cls, conf in zip(bboxes, classes, confs):
        x1, y1, x2, y2 = bbox
        confidence = conf
        detected_class = cls
        name = names[int(cls)]

        # Extract the detected object and crop it
        cropped_image = img.crop((x1, y1, x2, y2))

        transcribed_text = text_recognition(
            cropped_image,
            data_transforms,
            text_reg_model,
            idx_to_char,
            device
        )

        predictions.append((bbox, name, confidence, transcribed_text))

    visualize_detections(img, predictions)

    return predictions
```

End-to-End Pipeline

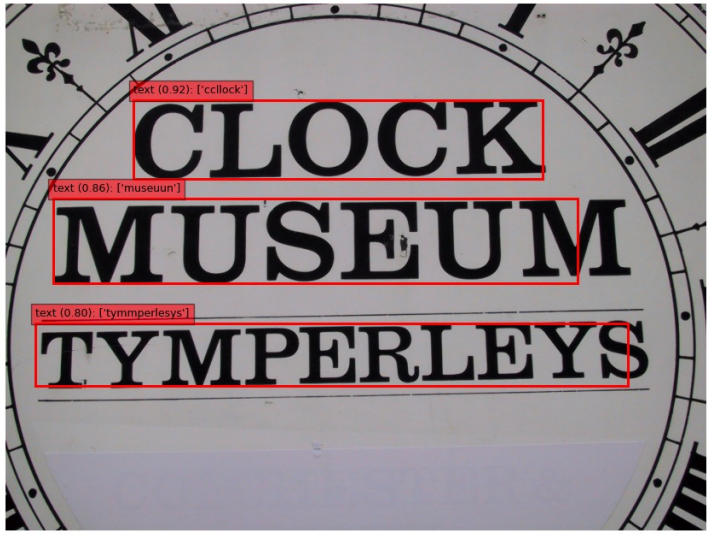
❖ Step 6: Test

```python
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize((100, 420)),
        transforms.ColorJitter(brightness=0.5, contrast=0.5, saturation=0.5),
        transforms.Grayscale(num_output_channels=1),
        transforms.GaussianBlur(3),
        transforms.RandomAffine(degrees=1, shear=1),
        transforms.RandomPerspective(distortion_scale=0.2, p=0.3, interpolation=3),
        transforms.RandomRotation(degrees=2),
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,)),
    ]),
    'val': transforms.Compose([
        transforms.Resize((100, 420)),
        transforms.Grayscale(num_output_channels=1),
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,)),
    ]),
}
```

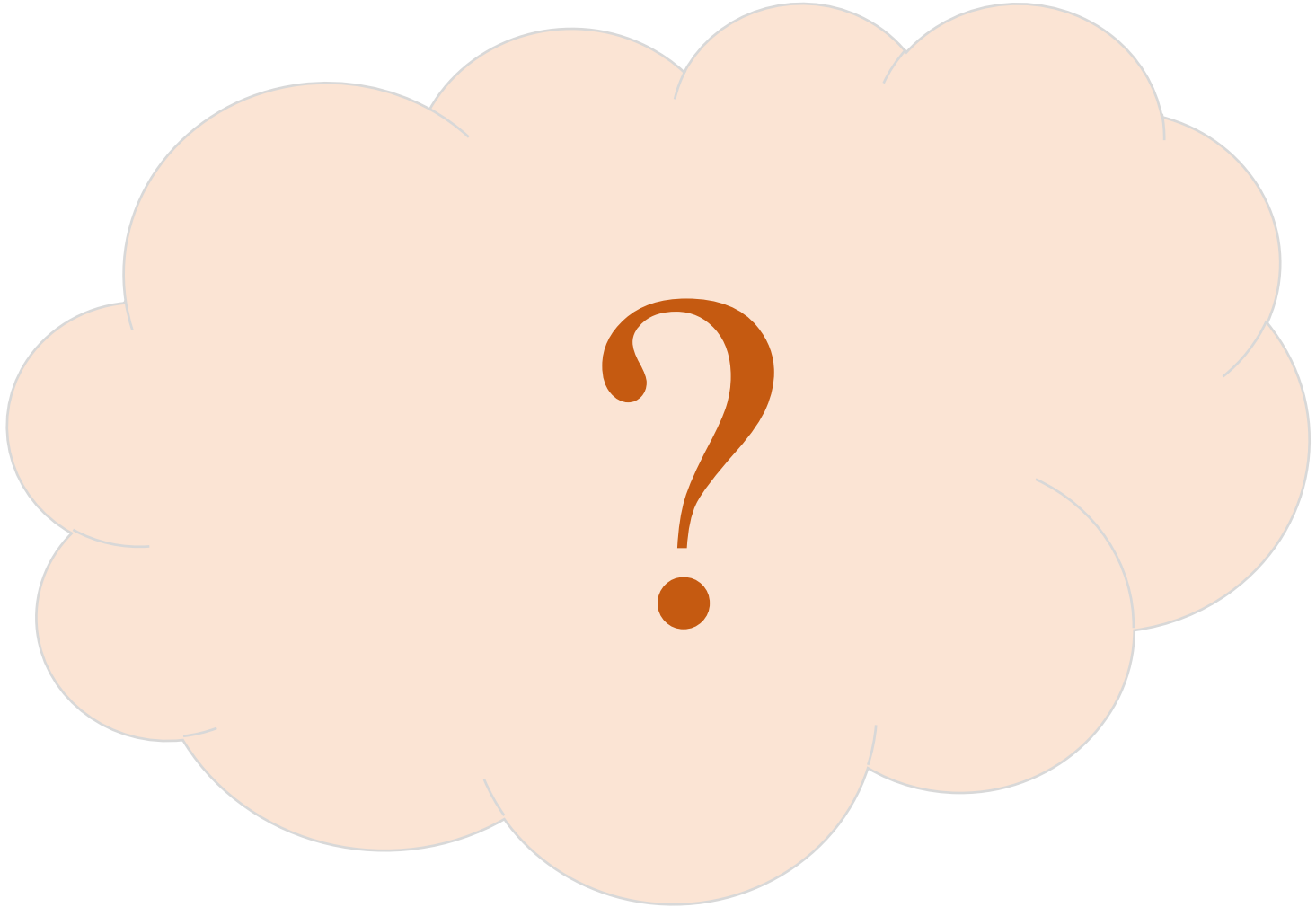
```python
img_dir = 'datasets/SceneTrialTrain/lfsosa_12.08.2002'
inf_transforms = data_transforms['val']

for img_filename in os.listdir(img_dir):
    img_path = os.path.join(img_dir, img_filename)
    predictions = predict(
        img_path,
        data_transforms=inf_transforms,
        text_det_model=yolo,
        text_reg_model=crnn_model,
        idx_to_char=idx_to_char,
        device=device
    )
```

End-to-End Pipeline

❖ Final Results



Question




Thank you!