VIETNAM NATIONAL UNIVERSITY

UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Authors:

Phạm Vũ Duy – 21020179

Đỗ Mạnh Dũng – 21020611

Phạm Thu Trang – 21020248

Trần Phương Linh – 21020214

Sino-nom Character Localization

HANOI - 2024

VIETNAM NATIONAL UNIVERSITY

UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Authors:

Phạm Vũ Duy – 21020179

Đỗ Mạnh Dũng – 21020611

Phạm Thu Trang – 21020248

Trần Phương Linh – 21020214

Sino-nom Character Localization

Course: Image Processing

Group: 5

Supervisor: Assoc. Prof. Lê Thanh Hà

BS Lê Công Thương

HANOI – 2024

Abstract

In the realm of document analysis, the precise localization of Sino-nom characters is crucial for the digital preservation and interpretation of East Asian texts. This report describes our multifaceted approach to the Sino-nom character localization challenge as part of a mid-term assessment for a image processing course. Utilizing a combination of provided and externally sourced datasets, our team employs advanced preprocessing techniques to enhance the training effectiveness of several state-of-the-art models. These models are known for their robust performance in object detection tasks and are adapted to the specific nuances of Sino-nom character localization. We evaluate these models based on the mean average precision (mAP) metric across various intersection over union (IoU) thresholds to identify the most effective model. Our methodology achieved mAP@[0.5,0.95] scores of 0.93 and 0.85 on the validation and test sets, respectively. The results underscore the significant potential of our approach in improving the automation of text recognition processes for historical documents.

Keywords: *Sino-nom character localization, document analysis, mean average precision (mAP), Advanced preprocessing techniques*

Table of Contents

Abstract . . . . **ii**

Table of Contents . . . . **iii**

List of Figures . . . . **iv**

List of Tables . . . . **iv**

1 Introduction **1**

2 Materials and Methods **3**

2.1 Dataset 3

2.2 Data Collection and Preprocessing, Augmentation 4

2.3 Model Selection and Training 9

2.4 Model Evaluation 11

3 Experiment and Results **12**

3.1 Experiments setup 12

3.2 Results and Discussion 20

4 Conclusion **26**

References **27**

List of Figures

3.1 Example of an image after preprocessing and drawing bounding boxes. . 15
3.2 Example of training YOLO process 19
3.3 The output image of Faster R-CNN 21
3.4 The output image of EAST 22
3.5 Sample output of model trained with default dataset 23
3.6 Sample output of model trained with augmented dataset 23
3.7 Confusion matrix of our final model 24

List of Tables

3.1 The results of Sino-nom character localization task 20
3.2 Comparison of different training configuration with mAP@[0.5,0.95] metric 22

Chapter 1

Introduction

Sino-nom documents are precious ancient resources closely associated with the long history of the Vietnamese nation. An important way to understand and protect these documents is to convert their content into other media using digital technology. In this process, the Sino-nom character detection and localization task emerges to identify and locate Sino-nom characters in ancient documents, facilitating recognition and retrieval tasks.

Unlike Latin characters, Sino-nom characters are much more complex. Sino-nom characters often have more strokes, and their shapes can vary depending on the text and era. In addition, many ancient Sino-nom documents are hundreds of years old, so the image quality is often degraded over time, leading to blurring, fading, and stains. Lighting and camera angle can also affect image quality, making it even harder to distinguish Sino-nom characters. Furthermore, the amount of data is limited.

To address the challenge of accurately locating characters in images, we propose a two-step approach that utilizes data manipulation and deep learning techniques. Firstly, we will focus on enhancing the quality and quantity of the datasets employed to train our models. This entails data preprocessing steps to ensure consistency, eliminate noise, and standardize the image format. Additionally, we will employ data augmentation techniques to artificially expand the dataset by generating variations of existing images. This process may involve operations such as image rotation, flipping, and brightness adjustments. By enriching the datasets in this manner, we aim to make them more diverse and representative of real-world scenarios. Following data preparation, we will then leverage the power of deep learning. Three distinct deep learning models: Faster R-CNN[2][3], EAST[4] and YOLO[1] will be trained on the enhanced datasets. These models will be specifically designed to identify and localize the positions of individual characters within the images. By comparing the performance of these different models, we can identify the most effective approach for this task.

The remaining chapters of the paper are structured as follows. In Section 2, our proposed methods to the tasks are described in details. Then, Section 4 focuses on evaluating the performance of the proposed models. In addition, the advantage and disadvantage of our proposed models are also discussed. The conclusion is provided in Section 5.

Chapter 2

Materials and Methods

2.1 Dataset

Our primary dataset for the Sino-nom character localization project comprises 70 high-resolution image files of ancient Sino-nom documents. These images, integral to a competition aimed at advancing digital preservation techniques, are meticulously curated to include detailed annotations. Each annotation specifies the exact locations of Sino-nom characters within the documents, providing a reliable ground truth for model training and validation.

To further improve our model's robustness, we supplemented the primary dataset with additional external datasets. These datasets feature historical documents bearing Sino-nom characters and undergo a rigorous preprocessing and annotation process. This process ensures uniformity in data quality, mirroring the precision of the primary dataset's annotations.

2.1.1 Features of the Combined Dataset

- **Image Resolution:** Each image is of high resolution, capturing the intricate details of Sino-nom characters necessary for precise localization.
- **Annotations:** The datasets include bounding box annotations for each character, detailing *x_center*, *y_center*, *bbox_width*, *bbox_height*, and a *confidence score*. These comprehensive annotations are crucial for accurate character localization during the training of our models.
- **Diversity in Text Layout and Style:** The datasets encompass a diverse range of

document types and character styles. This variety challenges our model, enhancing its ability to adapt and perform accurately across various historical document contexts.

2.2 Data Collection and Preprocessing, Augmentation

2.2.1 Data Collection

The primary dataset for our Sino-nom character localization project consists of 70 high-resolution images of ancient Sino-nom documents. These images have been supplemented with additional datasets containing historical documents to enhance model robustness. Each document is annotated with precise character locations to serve as ground truth for training.

2.2.2 Preprocessing Techniques

To prepare our data for effective training, we employ a series of preprocessing steps. Each step is crucial for ensuring data uniformity and enhancing model performance.

Grayscale Conversion

We convert color images to grayscale to reduce computational demands and focus on structural features. Converting to grayscale simplifies the image data by eliminating color variance, focusing on the intensity of the pixels, which is crucial for feature extraction.

Implementation and Explanation:

```
def preprocess_image(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    resized_image = cv2.resize(gray_image, (256, 256))
    normalized_image = resized_image / 255.0
    return normalized_image
```

- **cv2.cvtColor(image, cv2.COLOR_BGR2GRAY):**
  - *image:* The original image input into the function.

  - *cv2.COLOR_BGR2GRAY:* This constant represents the conversion of the color space from BGR (Blue, Green, Red) to grayscale. This conversion reduces computational costs and focuses on the essential information in the image, also reducing noise.

Image Resizing

Standardizing image size to 256x256 pixels ensures consistent input dimensions for our model:

Code Implementation:

```
resized_image = cv2.resize(gray_image, (256, 256))
return resized_image
```

Explanation:

- *gray_image:* The image that has been converted to grayscale.
- *(256, 256):* The target dimensions to which the image will be resized. This resizing helps in reducing the image size to fixed dimensions, easier to handle during model training.

Normalization

Normalizing pixel values to the range [0, 1] stabilizes the neural network training by providing consistent input scales:

Code Implementation:

```
normalized_image = resized_image / 255.0
return normalized_image
```

Explanation:

- After resizing, this function normalizes the pixel values of the image by dividing all the pixel values by 255.0, resulting in each pixel value ranging from 0 to 1. This makes the training process more stable as pixel values are normalized to the same range.

Label Reading and Bounding Box Drawing

Properly annotated data is crucial for the successful training of accurate models. In our project, we meticulously read bounding box labels from files, which specify the locations of Sino-nom characters within the images. This process is vital to ensure the correctness of annotations and to assess the effectiveness of our preprocessing techniques:

```
def read_labels(label_path):
    with open(label_path, 'r') as file:
        lines = file.readlines()
        labels = [list(map(float, line.strip().split()))
        for line in lines]
    return labels

def draw_boxes(image, labels):
    image_with_boxes = image.copy()
    image_height, image_width = image.shape[:2]
    for label in labels:
        class_id, x_center, y_center, width, height = label
        x1 = int((x_center - width / 2) * image_width)
        y1 = int((y_center - height / 2) * image_height)
        x2 = int((x_center + width / 2) * image_width)
        y2 = int((y_center + height / 2) * image_height)
        cv2.rectangle(image_with_boxes, (x1, y1), (x2, y2),
        (255, 0, 0), 1)
    return image_with_boxes
```

Detailed Function Descriptions:

- **read_labels(label_path):** This function is used to read labels from a text file and returns a list of labels.
  - *label_path:* Path to the file containing labels.
  - The function reads each line in the file, splits the line into numeric values, and converts them into a list of labels.
- **draw_boxes(image, labels):** This function is used to draw bounding boxes around characters on an image.

  - *image:* The original image on which bounding boxes are to be drawn.
  - *labels:* A list of labels for characters on the image.
  - The function iterates over the list of labels, calculates the absolute coordinates of bounding boxes based on relative coordinates and image dimensions, and then draws these bounding boxes on the original image.

- **convert_to_uint8(image):** This function is used to convert an image to uint8 data type if necessary.
  - *image:* Image to be converted.
  - Checks if the image data type is not uint8 (values from 0 to 255), then it converts the pixel values of the image to uint8 by multiplying them with 255 and converting to uint8.

By integrating bounding box annotations directly into the image data, we enable the model to learn the spatial context of Sino-nom characters, which is essential for tasks such as character recognition and document analysis. These comprehensive preprocessing and augmentation strategies discussed form the backbone of our approach to developing a robust model, capable of accurately localizing Sino-nom characters in various document contexts.

2.2.3 Data Augmentation

To increase the diversity of our training set and improve model robustness against various transformations, we implement several augmentation techniques:

Rotation

Images are rotated by specified angles to simulate different document orientations:

```
def rotate_image(image, angle):
    height, width = image.shape[:2]
    rotation_matrix = cv2.getRotationMatrix2D
    ((width / 2, height / 2), angle, 1)
    rotated_image = cv2.warpAffine(image, rotation_matrix,
    (width, height))
    return rotated_image
```

Rotating images helps the model learn to recognize characters from different angles, improving its ability to handle real-world variability.

This function uses `cv2.getRotationMatrix2D` to create a rotation matrix based on the provided center and angle of rotation. It then uses `cv2.warpAffine` to perform the image rotation by applying this rotation matrix.

Translation

Shifts the image in the x and y directions:

```
def translate_image(image, x, y):
    translation_matrix = np.float32([[1, 0, x], [0, 1, y]])
    translated_image = cv2.warpAffine(image, translation_matrix,
    (image.shape[1], image.shape[0]))
    return translated_image
```

Translation mimics the effect of characters appearing in different parts of the image, which is common in scanned documents. This function utilizes a translation matrix to perform image shifting based on the provided x and y values.

Scaling

Adjusts the scale of the image to help the model learn to recognize characters of different sizes:

```
def scale_image(image, scale_factor):
    scaled_image = cv2.resize(image, None, fx=scale_factor,
    fy=scale_factor, interpolation=cv2.INTER_LINEAR)
    return scaled_image
```

- scale_factor: The scaling factor. If the scale_factor is greater than 1, the image will be enlarged; if less than 1, the image will be reduced.
- This function uses the cv2.resize function to perform the scaling of the image according to the given factor.

Flipping

Flips the image horizontally or vertically to further augment the dataset:

```
def flip_image(image, flip_code):
    flipped_image = cv2.flip(image, flip_code)
    return flipped_image
```

- flip_code: The flip code specifies the direction of the flip. If flip_code is 0, the image is flipped vertically (around the x-axis); if 1, it is flipped horizontally (around the y-axis); if -1, the image is flipped both horizontally and vertically (around both axes).
- This function uses the cv2.flip function to perform the flipping of the image as specified by the flip_code.

Flipping images enhances the model's ability to understand and process characters irrespective of their orientation.

2.3 Model Selection and Training

This section details the advanced deep learning models selected for the Sino-nom character localization task. Each model is chosen based on its ability to effectively detect and localize objects within images, which is critical for the accurate identification of characters in historical documents.

2.3.1 Faster R-CNN

Faster R-CNN is a object detection algorithm that enhances the speed and accuracy of object detection. It integrates a Region Proposal Network (RPN) with a Fast R-CNN detector into a single model:

- **Region Proposal Network (RPN):** Generates object proposals by scanning the convolutional feature map obtained from the image.
- **RoI Pooling:** Extracts and scales the features corresponding to each proposal to ensure a fixed size input for the classifier.

- **Classifier and Bounding Box Regressor:** Determines the class of each object and refines its bounding box coordinates.

Faster R-CNN's precision in localizing objects makes it highly suitable for detailed character recognition tasks within dense and complex document images.

2.3.2 YOLO (You Only Look Once)

YOLO applies a radically different approach as a one-stage detector that treats object detection as a simple regression problem from image pixels to bounding box coordinates and class probabilities:

- **Unified Detection:** Utilizes a single convolutional network to predict multiple bounding boxes and their class probabilities for the entire image simultaneously.
- **Speed and Efficiency:** Achieves remarkable detection speeds, which is advantageous for real-time applications.

YOLO's unique architecture enables it to perform fast, making it an excellent choice for applications where both speed and accuracy are required.

2.3.3 EAST

EAST was introduced in the paper "EAST: An Efficient and Accurate Scene Text Detector" by Xinyu Zhou, Cong Yao, He Wen, Yuzhi Wang, Shuchang Zhou, Weiran He, and Jiajun Liang. This algorithm aims to find text in real-world environments, where text can appear in various sizes, orientations, and perspectives. The key principles of EAST are as follows:

- **Feature Extraction:** EAST employs a deep convolutional neural network (CNN) for feature extraction from the input image. This network is typically pre-trained on a large dataset and fine-tuned for text detection (such as VGG, ResNet,....).
- **Anchors and Region Proposal Network (RPN):** Like object detectors, EAST relies on anchors and an RPN to pinpoint areas containing text. But unlike them, EAST modifies the RPN to predict axis-aligned quadrilaterals (instead of rectangles) that closely hug the text regions.
- **Geometry Predictions:** EAST predicts the coordinates of the four vertices of the quadrilateral bounding the text region, along with a confidence score indicating the

presence of text. This capability allows EAST to handle text in arbitrary orientations and shapes.

- **Non-Maximum Suppression (NMS):** After predictions, NMS is applied to eliminate duplicate and overlapping text region proposals.

2.4 Model Evaluation

This section discusses the metrics used to evaluate the performance of the classification models utilized in our Sino-nom character localization project. For detection tasks, the standard metric to measure performance is Mean Average Precision (mAP) at certain IoU (Intersection-over-Union) thresholds. Intersection over Union (IoU) is a metric that quantifies the degree of overlap between two regions.

$$IoU = \frac{Area of Overlap}{Area of Union} \tag{2.1}$$

While precision and recall are single-valued metrics that measure performance at a given threshold, Average Precision (AP) takes into account the order of the predictions as well. Average Precision (AP) measures the area under the precision-recall curve for a given IoU threshold interval.

$$AP = \sum_n (R_n - R_{n-1}) P_n \tag{2.2}$$

where P_n and R_n are the precision and recall at the $n^t h$ score threshold. Mean Average Precision (mAP) is calculated as average of AP's across the whole test set. For this task, we report mAP@[0.5,0.95] as the primary evaluation metric for quality of text detection model. mAP@[0.5,0.95] is calculated as average of set of mAP's over different IoU thresholds, from 0.5 to 0.95 with step 0.05 (0.5, 0.55, 0.6 ..., 0.95). Higher mAP value for the detection model is desirable.

Chapter 3

Experiment and Results

3.1 Experiments setup

3.1.1 Data Collection

For the Sino-nom character localization project, our initial step involved gathering the necessary data and setting up the environment for development and testing. We utilized Google Colab for its powerful cloud-based capabilities, which allowed us to access high-performance computing resources essential for training deep learning models.

Installation and Configuration

Necessary tools for Optical Character Recognition and image processing are set up as follows:

```
!pip install pytesseract
!apt-get install tesseract-ocr
!apt-get install libtesseract-dev
```

Importing Libraries

The Python libraries required for executing image processing and machine learning tasks are imported:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import pytesseract
import os
from sklearn.metrics import accuracy_score,
 precision_score, recall_score, f1_score
```

Reading and Preprocessing the Data

Each image is processed to enhance its features suitable for the training model. This includes conversion to grayscale, normalization, and resizing:

```
def preprocess_image(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)
    contrast_image = cv2.equalizeHist(blurred_image)
    resized_image = cv2.resize(contrast_image, (64, 64))
    normalized_image = resized_image / 255.0
    return normalized_image
```

3.1.2 Data Visualization

Visualizing the preprocessed images along with their respective annotations provides insights into how well the model will potentially perform. This process helps to visually verify the accuracy of character localization.

Displaying Preprocessed Images with Annotations Using matplotlib to showcase the preprocessed images with bounding boxes drawn around localized characters:

```
for image_file in os.listdir(train_dir):
    image_path = os.path.join(train_dir, image_file)
    image = cv2.imread(image_path)
    preprocessed_image = preprocess_image(image)
    labels = read_labels(os.path.join(train_label_dir,
                                  f"{os.path.splitext(image_fi
    image_with_boxes = draw_boxes(preprocessed_image, labels)
    plt.imshow(cv2.cvtColor(image_with_boxes, cv2.COLOR_BGR2RGB))
    plt.title("Preprocessed Image with Bounding Boxes")
    plt.axis('off')
    plt.show()
```

3.1.3 Data Exploration

In-depth analysis of the dataset is crucial to understand the characteristics and variations within the images, which aids in refining the preprocessing techniques and improving model accuracy.

Analyzing Character Distributions

Investigating variations in character sizes and styles provides crucial feedback for adjusting the bounding box algorithms, optimizing conditions for model training.

Additional Preprocessing Techniques

We implement several preprocessing and augmentation techniques to ensure data uniformity and enhance model performance:

Grayscale Conversion, Image Resizing, and Normalization

```
def preprocess_image(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    resized_image = cv2.resize(gray_image, (256, 256))
    normalized_image = resized_image / 255.0
    return normalized_image
```

Data Augmentation: Rotation, Translation, Scaling, Flipping

The preprocessing and augmentation strategies, detailed extensively in Section 2, are essential for developing a robust model. These methods enable accurate localization of Sino-nom characters across diverse document contexts.

Visualization of Preprocessing and Augmentation

Examples of preprocessing and augmentation can be visualized using matplotlib to demonstrate the transformations applied to the images. This visualization aids in understanding the potential impact of preprocessing on the training process. These comprehensive preprocessing and augmentation strategies are integral to developing a robust model capable of accurately localizing Sino-nom characters in various document contexts.

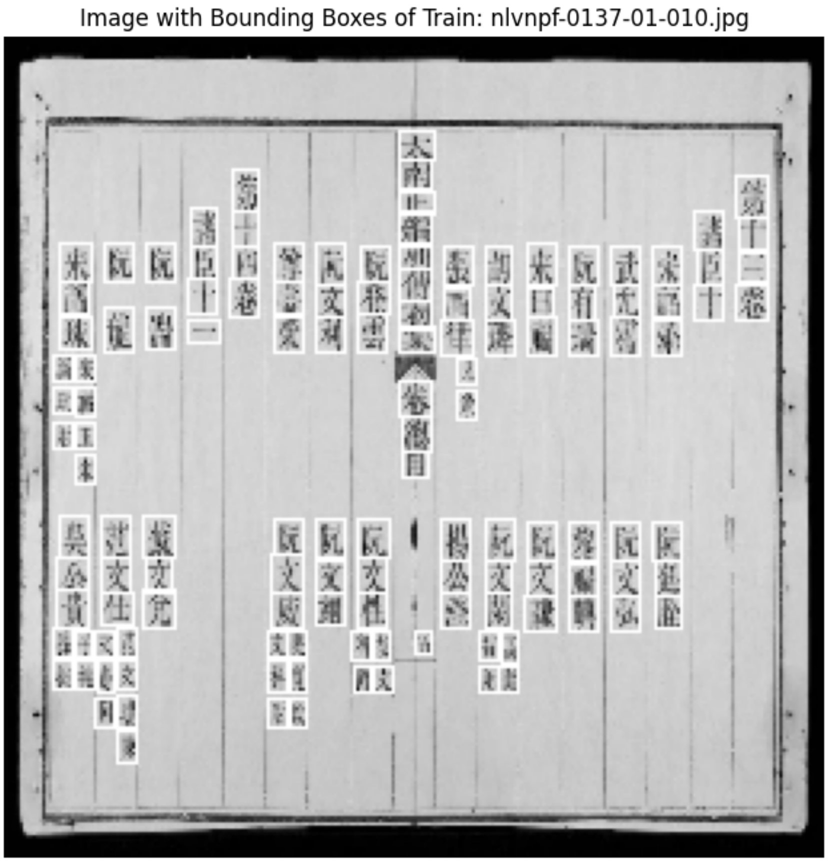


Figure 3.1: Example of an image after preprocessing and drawing bounding boxes.

3.1.4 Faster R-CNN

Data Preparation

The original data is formatted in the YOLO standard, with labels containing coordinates of bounding box centers and corresponding sizes for each object in the image. To use this data with Detectron2, we converted these labels to the format required by Faster R-CNN. The `yolo_to_detectron2` function is used to convert bounding boxes from YOLO format to Detectron2 format.

```
def yolo_to_detectron2(bbox, img_width, img_height):
    x_center, y_center, width, height = bbox
    x_center *= img_width
    y_center *= img_height
    width *= img_width
    height *= img_height
    x_min = x_center - width / 2
    y_min = y_center - height / 2
    return [x_min, y_min, width, height]
```

Model Configuration

We configure the Faster R-CNN model using Detectron2's `DefaultTrainer` class. This class handles the training loop, including loading data, forward and backward passes, and updating the model parameters.

```
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file
("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("wb_localization_train",)
cfg.DATASETS.TEST = ("wb_localization_val",)
cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url
("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml")
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.00025
cfg.SOLVER.MAX_ITER = 50
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
cfg.MODEL.DEVICE = "cuda"

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
```

Training the Model

We train the Faster R-CNN model using the `DefaultTrainer` class provided by Detectron2.

```
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

Running Inference

The `DefaultPredictor` class simplifies the process of running inference on images with a trained model. It handles pre-processing, model inference, and post-processing.

```
# Running
cfg.MODEL.WEIGHTS = model_weights_path
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5
cfg.MODEL.DEVICE = "cuda"
predictor = DefaultPredictor(cfg)

# Initialize visualizer
    v = Visualizer(im, metadata=metadata, scale=0.8)
    v = v.draw_instance_predictions(outputs["instances"].to("cpu")
    cv2_imshow(v.get_image()[:, :, ::-1])
```

3.1.5 EAST

Training the model

We used Pytorch to bulid model based on code on https://github.com/SakuraRiven/EAST. We used dataset provided, data augumentation and NomNaOCR dataset to train model. We employ pretrained VGG16 from torchvision as the backbone model. During training, we use the Adam optimizer. The learning rates is 1e-3. The batch size is set to 32. The image size is set to 256x256 for all datasets. The number of training epochs is set to 15 for all datasets. We used GPU on Kaggle and Google Colab to train model quickly and effectively.

Running inference

During inference, we just retain bounding boxes which have the confidence score > 0.9. In addition, after analyzing the data, we realized that characters are usually written relatively neatly, so the bounding box will usually be a vertical rectangle. Therefore, the output of the model will eliminate all tilted rectangles. Intersection Over Union (IoU) threshold for Non-Maximum Suppression (NMS) is set to 0.2.

3.1.6 YOLO

Setup

Our approach was transfer learning using pre-trained YOLOv8 model, in order to do this we used Ultralytics YOLOv8. Every other step including training, validation and prediction, we will be using Ultralytics YOLOv8 provided interface which will be easier and cost us less time.

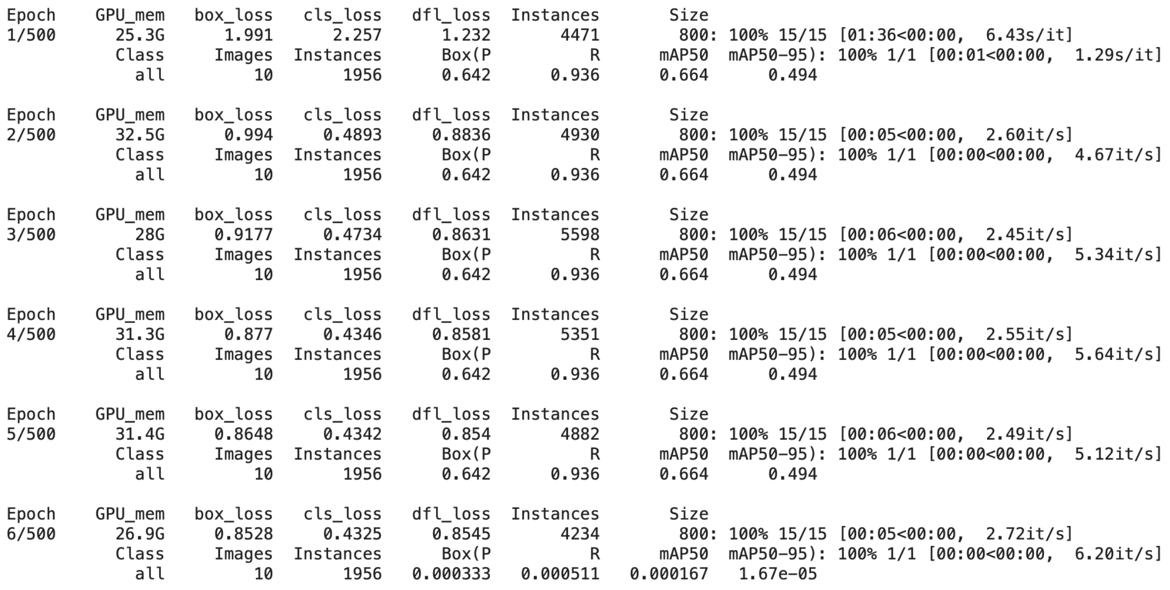
Installing Ultralytics YOLOv8 can be done in one simple command:

```
pip install ultralytics
```

Training the model

At first, we trained our model on the provided dataset only, after achieving less than ideal result, we augmented our data with blurring and increasing contrast to the images. During training the model, we tried changing a few parameters including: image size, batch size, iteratively increasing epochs and adjusting weight of loss function. The model is trained using Ultralytics YOLOv8, and the pretrained model we used was YOLOv8x. We trained multiple time producing multiple models with different training dataset and configurations.

Ultralytics YOLO provided yolo command which we can easily use to train. The training process takes a long time and a lot of computation unit. All of our training was done on Google Colab.



```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
1/500      25.3G      1.991      2.257      1.232       4471        800: 100% 15/15 [01:36<00:00,  6.43s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:01<00:00,  1.29s/it]
             all         10       1956      0.642      0.936      0.664      0.494

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
2/500      32.5G      0.994     0.4893     0.8836       4930        800: 100% 15/15 [00:05<00:00,  2.60it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  4.67it/s]
             all         10       1956      0.642      0.936      0.664      0.494

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
3/500        28G     0.9177     0.4734     0.8631       5598        800: 100% 15/15 [00:06<00:00,  2.45it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  5.34it/s]
             all         10       1956      0.642      0.936      0.664      0.494

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
4/500      31.3G      0.877     0.4346     0.8581       5351        800: 100% 15/15 [00:05<00:00,  2.55it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  5.64it/s]
             all         10       1956      0.642      0.936      0.664      0.494

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
5/500      31.4G     0.8648     0.4342      0.854       4882        800: 100% 15/15 [00:06<00:00,  2.49it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  5.12it/s]
             all         10       1956      0.642      0.936      0.664      0.494

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
6/500      26.9G     0.8528     0.4325     0.8545       4234        800: 100% 15/15 [00:05<00:00,  2.72it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  6.20it/s]
             all         10       1956   0.000333   0.000511   0.000167   1.67e-05
```

Figure 3.2: Example of training YOLO process

In our training process, we focus mainly on `box_loss`, `mAP0.5`, `mAP@[0.5,0.95]` to evalute our current model efficiency and accuracy.

Running inference

Ultralytics YOLO provided predict mode with detection task for image prediction. Using this we can easily predict bounding box of every images in folder on the commandline using yolo.

In this step, we tried experimenting with 2 parameters: `iou` and `conf`.

`iou`: Intersection Over Union (IoU) threshold for Non-Maximum Suppression (NMS). Lower values result in fewer detections by eliminating overlapping boxes, useful for reducing duplicates.

`conf`: Sets the minimum confidence threshold for detections. Objects detected with confidence below this threshold will be disregarded. Adjusting this value can help reduce false positives.

3.2 Results and Discussion

3.2.1 Results of Sino-nom character localization task:

We evaluated the performance of our models using the mAP@[0.5,0.95] metric on 10 validation image files provided. Figure 1 shows the results achieved for each model. As can be seen, YOLO outperforms the other models with an mAP@[0.5,0.95] score of 0.93. EAST and Faster RCNN have similar results. This indicates that in addition to correctly identifying the characters in the images, the YOLO model can also generate bounding boxes that closely enclose the characters. Furthermore, with this model, we achieved an mAP@[0.5,0.95] score of 0.85 on the test set.

Table 3.1: The results of Sino-nom character localization task

Model	**mAP@[0.5,0.95]**
Faster RCNN	0.34
EAST	0.34
YOLO	**0.93**

3.2.2 Discussion

In this section, we will analyze the outputs of the models in more detail. We will also discuss the advantages and disadvantages of each model.

Faster R-CNN

The example output of Faster R-CNN shown in firgue 3.3 indicate that the model is capable of accurately detecting objects within the images. The bounding boxes are well-aligned with the ground truth, and the confidence scores reflect the model's certainty in its predictions. However, the model still does not detect all the characters in the image. In addition, we encountered several challenges when processing this dataset. The primary issue was the incompatibility of the label formats between YOLO and Detectron2. The conversion process added complexity and potential sources of error, which could impact the model's performance. Additionally, Faster R-CNN tends to be computationally intensive and slower compared to YOLO, making it less suitable for real-time

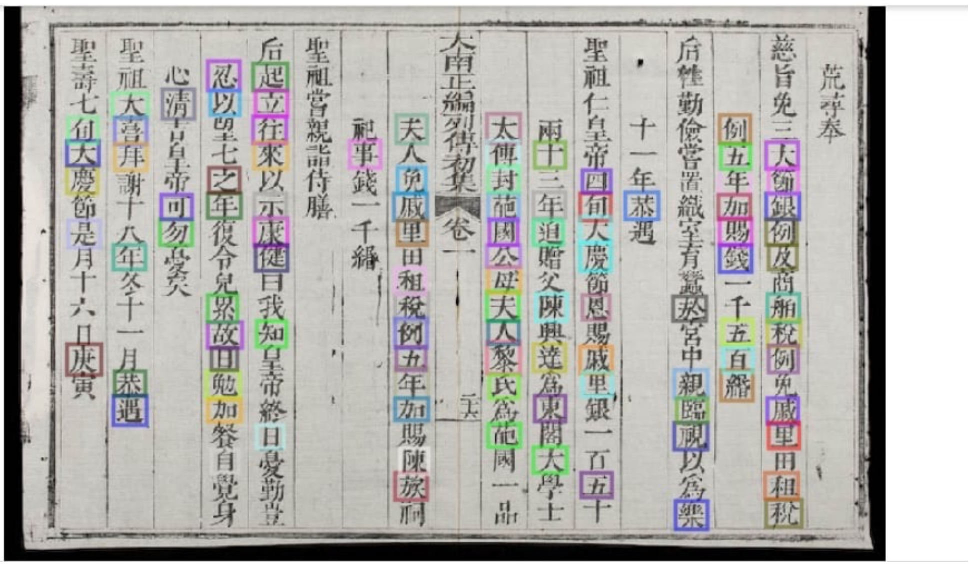


Figure 3.3: The output image of Faster R-CNN

applications. The model also showed slow learning, non-optimal performance, and low accuracy on this specific dataset.

EAST

An example of the model's output is shown in firgue 3.4. From the output, it can be seen that the model is able to detect more characters than the Faster R-CNN model. However, characters that are too close together can be difficult for the model to predict correctly. In addition, the model also detects bounding boxes that do not contain any characters.

In addition to using the available dataset, we also used the NomNaOCR dataset for training. However, the labels of this dataset instead of identifying each character separately, identify them by a column of characters. We have tried to split the bounding boxes based on the number of characters in each column. However, for some image files, this splitting is not effective and the bounding boxes do not fully enclose the characters. This can lead to noise during the training process of the model. In addition, the limitation of free GPUs on platforms like Kaggle and Google Colab leads to the need to choose a small image size and a small number of epochs, which results in the model not achieving the desired results.

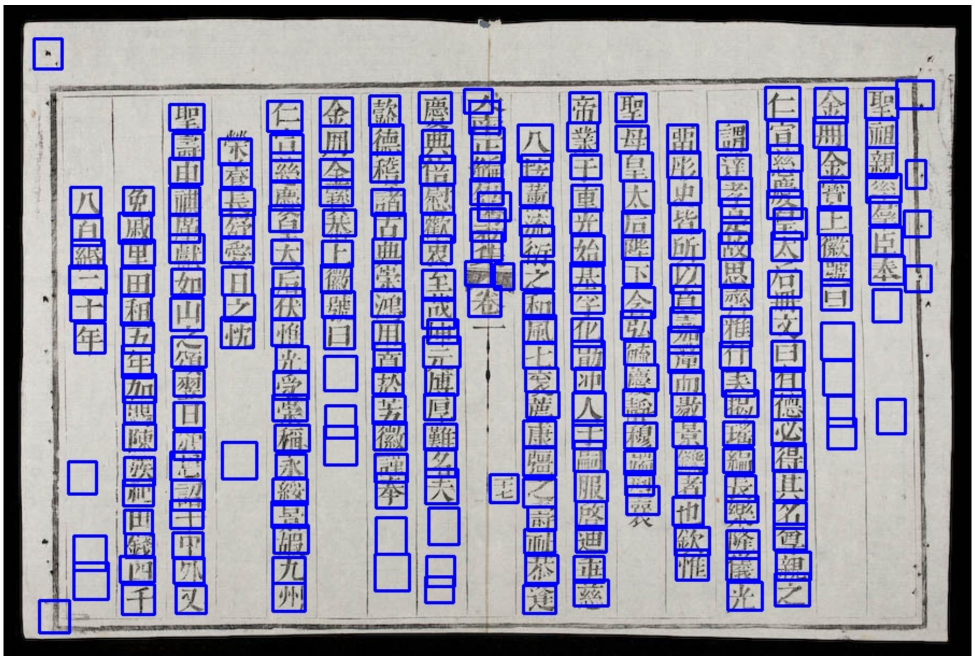


Figure 3.4: The output image of EAST

YOLO

The table 3.2 show the result of models that we trained using YOLO with different dataset and configuration.

Table 3.2: Comparison of different training configuration with mAP@[0.5,0.95] metric

Model	Training setting[a]				Metric (mAP@[0.5,0.95])
	imgsz	epochs	box	data	
YOLOv8m	640	300	7.5	default	0.85
YOLOv8x	640	300	7.5	default	0.87
YOLOv8x	800	300	7.5	default	0.9
YOLOv8x	1000	300	7.5	default	0.9
YOLOv8x	800	300	10	default	0.9
YOLOv8x	800	300	7.5	augmented	0.93

[a] imgsz: Target image size for training; epochs: Total numbers of training epochs; box: Weight of the box loss component in the loss function; data: source of dataset used to train, augmented or default; . . .

In our training process we have 2 key findings:

- Image size settings, data and larger pretrained model has the most influence on the final model accuracy.

- Our proposed method of changing box loss function does not have a large impact

In our data analysis phase, we recognize that the majority of image in the dataset was around 800x800 pixel size. With this discovery, we tried to train the dataset with this configuration and achieved some improvement. However increasing image size even more has diminishing returns as proved with our imgsz 1000 in the table.

We chose to use the largest model YOLOv8x due to higher accuracy, however this cost us larger training time, and an increase in the inference process. Due to our large preparation time, we decided to use the large model instead.

Our proposed augmenting data method improved our model accuracy by a large margin as seen in the table. We tried the 2 models with different images from Nom-NaOCR dataset. The results of the two models are presented in Figures 3.5 and 3.6, respectively.

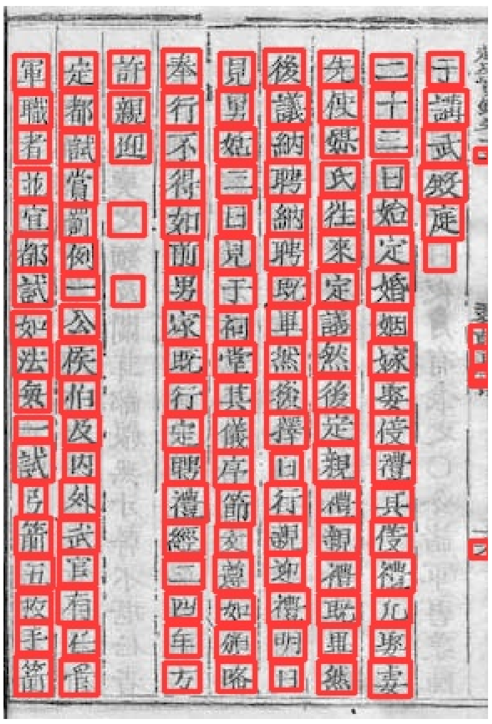


Figure 3.5: Sample output of model trained with default dataset

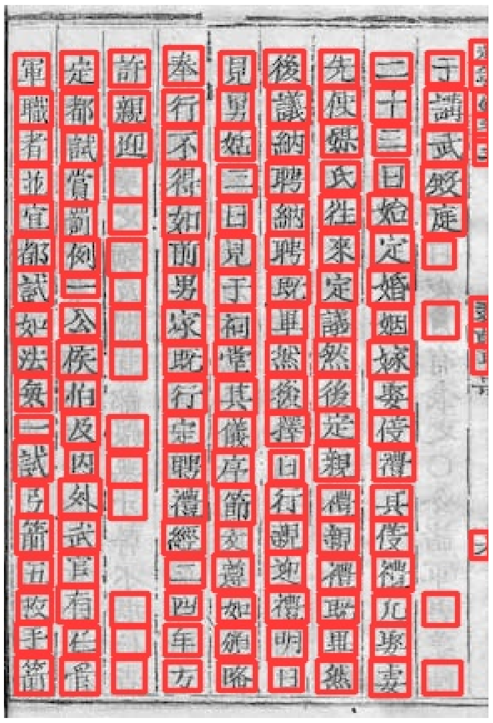


Figure 3.6: Sample output of model trained with augmented dataset

We can clearly see that with augmented data which included blurring and contrasting images helped our model with detecting images with artifacts like blurring. Meanwhile, the results also addressed the character missing issue of RCNN and achieved less noisy bounding boxes compared to EAST. After augmenting data, we see some improvement in our confusion matrix. The model predict more correctly and does not often confuse between background and word.

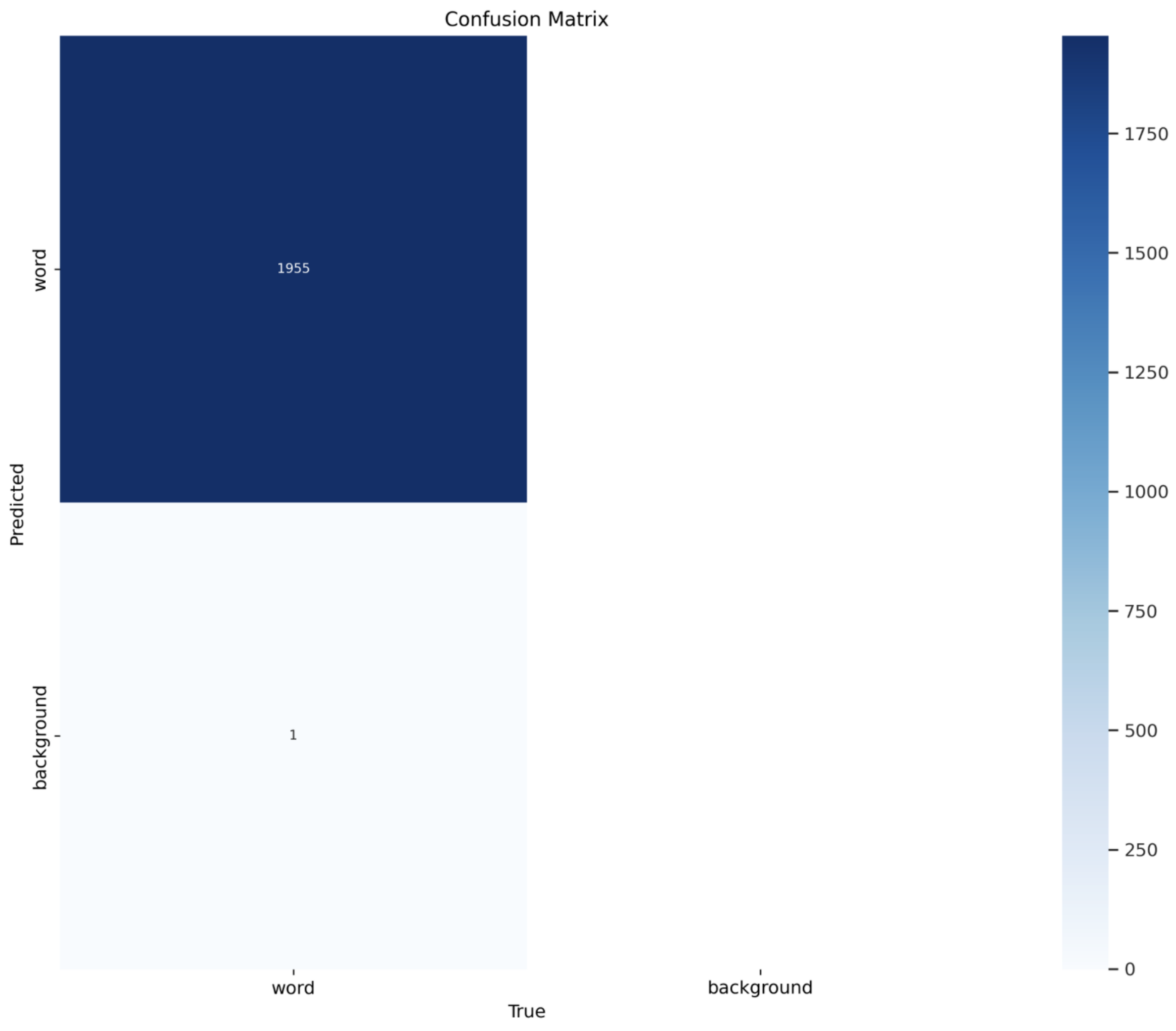


Figure 3.7: Confusion matrix of our final model

With YOLO, our predicted bounding box was not tight enough, which makes our metric on 0.95 IoU less than ideal. Our proposed method was increasing weight of bounding box loss function, which influences how much emphasis is placed on accurately predicting bounding box coordinates. However, in our experiment, modifying this weight does not affect the model accuracy as seen in table.
There are still a lot of improvement we can make, and still a lot of shortcoming we could not overcome:

- Image artifacts like blurring would impact greatly on accuracy.
- The bounding box was not tight enough, which makes our metric on 0.95 IoU less

than ideal.

- We could not achieve diverse training data.

Our method of modifying IoU and confidence threshold in predicting images phase had a problem. While this method can improve the model accuracy and decrease false positive on a few images in some situation, this can also affect our model accuracy on some other images. Therefore, in the end, we decided not to use this method.

Chapter 4

Conclusion

In this midterm report, we proposed methods for handling the Sino-nom character localization task. The first step was image processing and data augmentation. Then, we trained three models, Faster RCNN, EAST, and YOLO, to detect character bounding boxes in images. We achieved mAP scores of 0.93 and 0.85 on the validation and test sets, respectively.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[3] ——, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 6, pp. 1137–1149, 2016.

[4] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang, "East: an efficient and accurate scene text detector," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 5551–5560.