Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://sps.ewi.tudelft.nl/

SPS-2023-00

M.Sc. Thesis

Small end-to-end OCR model

Jingwen Dun B.Sc.

Abstract

Optical Character Recognition (OCR) extracts text from images and is widely used in document digitization and medical records. Traditional OCR systems have two stages, but end-to-end models offer superior data efficiency. Offline models are crucial in areas with limited internet access or strict data privacy.

Our project refines end-to-end OCR models, reducing the size to 19MB, allowing the model to run on mobile devices. Through database utilization and fine-tuning, the model achieves a 47.3% precision rate and 45.3% f-score. An Android demo showcases the model's mobile prowess, processing images in 433ms on average.

Keywords: OCR, end-to-end, mobile device

Small end-to-end OCR model

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Electrical Engineering

by

Jingwen Dun B.Sc.
born in Shenyang, China

This work was performed in:

Circuits and Systems Group
Department of Signals & Systems
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Delft University of Technology
Department of
Signals & Systems

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Small end-to-end OCR model"** by **Jingwen Dun B.Sc.** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 2023-09-27

Chairman: _____

dr.ir. Justin Dauwels

Committee Members: _____

dr. David M.J. Tax

_____

dr. Dragos Datcu

Abstract

Optical Character Recognition (OCR) is a pivotal technology used to extract text information from images, finding wide-ranging applications in document digitization and medical records management. The integration of machine learning has ushered in an era of swift and precise OCR models. Broadly, OCR comprises two key components: detecting the bounding boxes around text instances and recognizing the characters within them. Presently, prevailing OCR models are primarily intricate two-stage systems necessitating real-time operation on remote servers. Nevertheless, end-to-end models exhibit superior performance from a data utilization perspective. There exist scenarios where offline models prove indispensable, such as in environments with restricted internet access or locales with stringent data privacy and security requirements.

This project delves into various end-to-end models, leveraging the PaddleOCR end-to-end model as a foundational reference to devise a compact OCR model tailored for edge devices. Through meticulous optimization of the backbone architecture and the introduction of diverse Feature Pyramid Network (FPN) structures within the stem network, we achieved a remarkable reduction in model size, down to 19MB. This represents a substantial advancement, constituting merely one-tenth of the original PaddleOCR end-to-end model's footprint.

By leveraging an extensive database and conducting a series of fine-tuning experiments specifically tailored for end-to-end OCR tasks involving curved text images, the model exhibits an impressive precision rate of 47.3% and an f-score of 45.3%. This achievement highlights the effectiveness of the customized loss function relative to the original model, despite its reduced size. Notably, this performance is comparable to certain end-to-end models with larger backbones. Furthermore, an Android demo has been carefully developed to demonstrate the model's capabilities on mobile devices, achieving an average processing time of 433 milliseconds per image.

Keywords: OCR, end-to-end, mobile device

Acknowledgments

First and foremost, I am grateful to the company for affording me the opportunity to engage in this work and gain insights into the fields of Computer Vision and machine learning. My colleagues were not only friendly but also generously offered their assistance, which greatly facilitated my work. The knowledge I acquired from the company has served as invaluable guidance for my future endeavors.

I extend my deepest gratitude to my supervisor, Dr. Dragos Datcu, for his invaluable guidance and unwavering support throughout this project. His insightful meetings and readiness to provide creative solutions in the face of challenges were pivotal to the success of this research. Additionally, I would like to express my appreciation to Dr. Ir. Justin Dauwels for granting me the opportunity to be a part of this project. I am thankful for his valuable contributions during our meetings and for offering insightful revision suggestions. I would also like to extend my thanks to Dr. David Tax for graciously accepting the invitation to join the thesis committee.

I want to express my heartfelt gratitude to my family(including our family cat) as well as my friends both at TU Delft and my bachelor university, SUSTech. Their unwavering support has been instrumental throughout my journey at TU Delft. They have consistently provided assistance with my studies and offered fresh perspectives on life. Having them by my side has truly been a source of great comfort and strength.

My two years at TU Delft have constituted a profoundly enriching journey. Transitioning from a major in Physics during my Bachelor's to Electrical Engineering was not without its initial challenges and uncertainties. However, the experience has proven to be immensely rewarding. In contrast to my studies in Physics, which emphasized 'logical reasoning', my time at TU Delft, particularly during my thesis, has provided me with a deeper understanding of 'utilizing the tools' and the significance of effective collaboration with others.

Jingwen Dun B.Sc.
Delft, The Netherlands
2023-09-27

Contents

List of Figures

List of Tables

Introduction 1

1.1 OCR

The technology of recognizing text from images, known as OCR(Optical Character recognition), has been a hot research topic with a long research history and wide applications, for example automatically labeling the license plate of passing vehicles and recognizing the texts from scanned texts. An example of OCR is shown in figure OCR is widely utilized in the field of education, transportation and banking. A example of applying OCR technology is the Google translator application, which can recognize text from images in almost real-time [2].

Figure 1.1: OCR demonstration

Due to the complex nature of OCR recognition which will be introduced in Chapter 2, most current OCR models have a size of several hundred megabytes, using relatively large backbones such as ResNet (around several hundred megabytes) and transformers (around several hundred gigabytes). In these cases, OCR models require high-performance hardware and are executed on the server or computer CPU. However, since OCR needs to be embedded in various devices, limiting the size of OCR models has become a worthwhile topic of study. Speaking more abstractly, trying to reduce the size of a model while minimizing the decrease in its performance has always been a subject worth researching.

The advent of neural networks optimized for mobile CPU such as ShuffleNet, SqueezeNet and MobileNet have shown that, even though simplifying the neural network structure may result in a slight performance loss, this loss is relatively small and acceptable compared to the reduction in model size [3, 4, 1]. The work of PaddleOCR works on thresholding OCR models [5], and have succeeded in thresholding an OCR model from 200MB to 10MB with an acceptable performance loss by using methods including changing backbones and other techniques including PACT normalization and changing input resolution.

1.2 The main problem

Current OCR models are mostly 2-stage models, which does the OCR jobs in several models separately. Apart from 2-stage models, end-to-end models OCR models can also accomplish the task as the 2-stage models, where it does all the tasks within one model. Compared to 2-stages OCR models, end-to-end OCR models have the advantage in the better utilization of shared information, but due to the complexity of the model, its level of research is not as extensive as 2-stage models. In this work, we consider and test the feasibility for building small end-to-end OCR models that can eventually run on the CPU of portable devices for example mobile phone.

1.3 thesis pipeline

In this paper we give a thorough investigation of current OCR model algorithms for the 2-stage and end-to-end OCR models, and will show the performance of thresholding the end-to-end OCR model from PaddleOCR. The rest of the paper is organized as follows. In Chapter 2, the current state of art of OCR technology is introduced. In Chapter 3, we give insight to thresholding the model, introducing light backbones to replace the original one and building the FPN structure aimed for connection and feature extraction. In chapter 4, we give a thorough look to the OCR algorithm we use in this model, both the preprocessing and postprocessing steps. After that in chapter 5, we introduce the experimental settings for the training, including the platform and hardware for training, selection of datasets and also techniques used in training the model. After introducing the basic settings, we show the result and give comprehensive analysis and comparison with the original PaddleOCR end-to-end model in chapter 6. For the demonstration of the model, we further implement an android demo to show its performance, and related information is written in chapter 7. Finally, we conclude all the results in chapter 8.

Current State of art 2

This chapter provides an overview of the current state of the art in OCR technology, as well as an introduction to end-to-end models.

2.1 End-to-end model

End-to-end learning, within the realm of Artificial Intelligence (AI) and Machine Learning (ML), is a technique wherein the model comprehensively learns all the steps from the initial input phase to the ultimate output result. This process is rooted in deep learning, where every distinct component is trained concurrently, as opposed to a sequential training approach.

2.2 OCR technology

OCR(Optical character recognition) is the technology that extracts texts from non-text media such as paper documents, scanned images, or photographs. It is widely used in document digitization, handwritten text conversion, etc.

Although OCR has undergone extensive research for an extended period, it still has research opportunities mainly due to the challenges of image and text variety and computational efficiency.

Shown in Figure 2.1, in different scenes, the challenging cases in image and text variety include:

- Image blur: blurred images cause the edges of characters to be fuzzy or unclear, making it difficult for OCR algorithms to identify character shapes and boundaries accurately. The blurring may also cause loss of detail or confusion, making it difficult for OCR algorithms to segment and recognize characters correctly.
- Illumination and chromatic aberration: the uneven illumination and chromatic aberration of the image may lead to shadow or variation of brightness across the image, making OCR algorithm difficult to recognize characters.
- Irregular boundaries: if the characters are not in a line but are curved or the bounding boxes have irregular boundaries, OCR algorithms will have difficulty detecting bounding boxes and segmenting the characters. Moreover, irregular borders can cause OCR algorithms to misinterpret character shapes.

- Various font styles: the variety in font style brings difficulties for OCR algorithms to map different styles to one character in the dictionary.
- Long text: in long texts, the spacing between characters may be smaller, and there may be no obvious separator between characters. This can cause character segmentation errors. Although long text seems to be an easy problem to solve compared to other problems, it remains in nearly all the current OCR algorithms.

Image blur | Curved texts

Illumination | Chromatic aberration

Figure 2.1: Challenging cases in OCR

There is also a difficulty in computational efficiency. Due to algorithm characteristics, the algorithm runs on the CPU instead of the GPU. This restricts the algorithm's efficiency in utilizing computation resources.

2.3 OCR models

Current OCR models can be divided into two types according to the number of sub-models used for the text extraction task.

2.3.1 2-stage models

As its name shows, the two-stage model does the text extraction job separately in two steps: text detection and text recognition. In the text detection step, the model detects the boundary box of text, and the result is sent into the text recognition step. The output of the text recognition step is the text on the image.

the The 2-stage model is the mainstream model of current OCR applications such as Google Translator and Baidu Translator. This separate design allows each task to focus on its optimization, and specific models best suited for each task can be chosen separately. In addition, the text detection stage can help filter out non-text regions in the image, thus reducing the amount of subsequent text recognition calculations and increasing overall processing speed and efficiency.

2.3.1.1 Text detection part

Commonly used models include:

- DB(Differentiable Binarization): The DB method, short for image binarization, is tailored to convert grayscale images into binary ones. It accomplishes this by assessing the contrast between pixels, determining whether they should be rendered as black or white [6]. This approach hinges on the fundamental concept of comparing a pixel's intensity to that of its neighboring pixels.

  Moreover, the advanced iteration of the DB algorithm employs a more refined and intelligent strategy for discerning pixel values. When integrated into the segmentation network for text detection, the DB module proves invaluable. Through joint optimization with this module, the segmentation network gains the capacity to dynamically establish the binarization threshold. This not only streamlines post-processing but also elevates the overall performance of text detection.

- EAST(Efficient and Accurate Scene Text Detector): EAST presents a streamlined and robust approach for swiftly and accurately detecting text in real-world scenes. This method operates directly on full images, enabling the direct prediction of words or text lines with diverse orientations and quadrilateral shapes. It achieves this without the need for superfluous intermediary processes like candidate aggregation and word partitioning. Remarkably, this entire task is accomplished using a singular neural network [7].

- SAST(Single-shot Arbitrarily-Shaped Text detector): SAST stands out as a text detection method that excels in identifying text of various shapes, including both arbitrary and rectangular forms. When compared to alternative text detection approaches, SAST demonstrates superior performance in discerning adjacent and lengthy texts [8].

  In this model, after extracting image features through stem networks, four crucial maps are learned as part of a multi-task framework. These encompass the text center line (TCL), text border offset (TBO), text center offset (TCO), and text vertex offset (TVO). The TCL map represents a condensed

version of the text region, serving as a one-channel segmentation map to distinguish text from non-text areas. The remaining label maps, namely TCO, TVO, and TBO, encode per-pixel offsets related to the pixels in the TCL map. Specifically, the TCO map captures the offset between pixels in the TCL map and the center of the bounding box, while TVO represents the offset between the four vertices of the bounding box and the pixels in the TCL map.

To illustrate this process, Figure 2.3 depicts the Label Generation stage: (a) The annotation of the text center region for a curved text, highlighted in red; (b) The generation of the TBO map; (c) The four vertices of the bounding box denoted as red stars; and (d) The center point of the bounding box also indicated as a red star, which the TVO and TCO maps reference.

2.3.2 Text recognition part

According to Jeonghun et al., the algorithms CRNN and Rosetta have the best overall performance considering the accuracy and operating time among all the word recognition algorithms [9].

- CRNN: the CRNN (Convolutional Recurrent Neural Network) architecture comprises three essential components. Firstly, the convolutional layers extract a feature sequence from the input image using a standard CNN model. Secondly, the recurrent layers employ the Long Short-Term Memory (LSTM) structure to predict a label distribution for each frame, which is particularly effective for processing sequential data. Finally, the transcription layer, integrated with the Connectionist Temporal Classification (CTC) layer, translates the per-frame predictions into the final label sequence. The CTC layer plays a crucial role in accurately transcribing sequences of varying lengths. By seamlessly integrating these three components, CRNN is capable of efficiently processing sequential data like text, making it a powerful tool for OCR [10].

- Rosetta: Rosetta is a comprehensive system designed for large-scale image text detection and recognition. In its recognition module, two distinct models are employed. The first, known as the CHAR Model (Character Sequence Encoding Model), comprises a sequence of convolutional layers followed by k independent multiclass classification heads. Each classification head predicts a character, including a NULL character, at a specific position, assuming uniform image dimensions (32×100). This model employs k parallel losses during training. The second model is the CTC Model (Connectionist Temporal Classification Model), which is fully convolutional and adopts a ResNet-18 architecture as its convolutional backbone. This model directly outputs a

character sequence and utilizes a sequence-to-sequence CTC loss for training. Notably, during inference, the CTC model exhibits a 6% improvement in speed compared to the CHAR model [11].

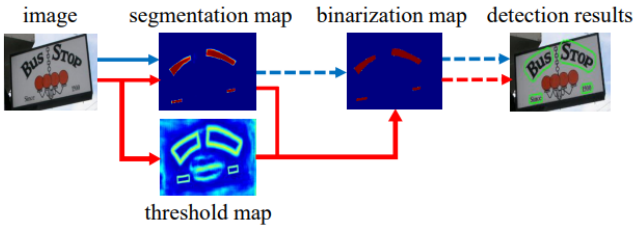


Figure 2.2: DB model pipeline

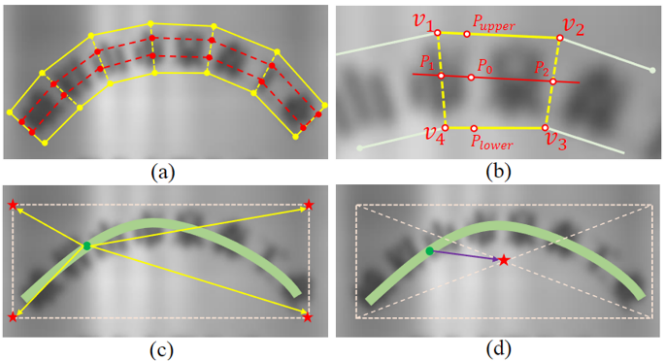


Figure 2.3: Different feature maps of the SAST model

2.3.3 End-to-end models and paddle end-to-end model

2.3.3.1 End-to-end models

End-to-end models do the text extraction task in one model. Figure 2.7 shows the current end-to-end models with a related structure to the PaddleOCR end-to-end model. Within the GT (ground-truth) enclosure, annotations are denoted by 'W' and 'C,' signifying word-level and character-level distinctions. The symbols 'H', 'Q,' and 'A' indicate the method's capabilities in identifying horizontally aligned, quadrilateral, and freely contoured text, respectively.

In Michal et al.'s model [12], text region proposals are initiated through a Region Proposal Network. Regions exhibiting substantial textual confidence are subsequently transformed into variable-width feature tensors using bilinear sampling. Ultimately, each part is linked to a character sequence or dismissed as non-textual. The ABCNet proposed by Liu et al. [13] has a similar structure. The

model can recognize curved-shape text instances using cubic Bezier curves and the Bezier Align technique, as shown in figure 2.4.

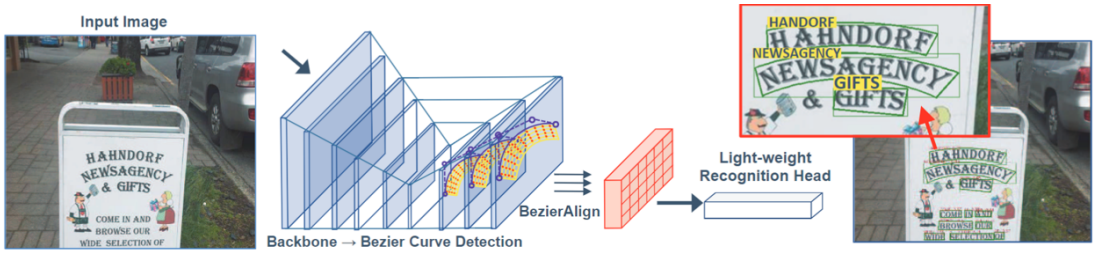


Figure 2.4: Pipeline of ABCNet

Besides being an end-to-end model, the TextDragon model from Feng et al. [14, 15]. The CharNet model from Xing et al. is more similar to the PaddleOCR model, where they output several features from one network, using the advantage of the end-to-end model of sharing information between different parts. In the TextDragon model, the output feature from the convolutional neural network contains information for the bounding box regression value for predicting the location of bounding boxes and center line segmentation for predicting the center line of text instances. The bounding boxes are grouped to depict the ROI region and finally put into the CTC decoder for recognition; the pipeline is shown in figure 2.5.

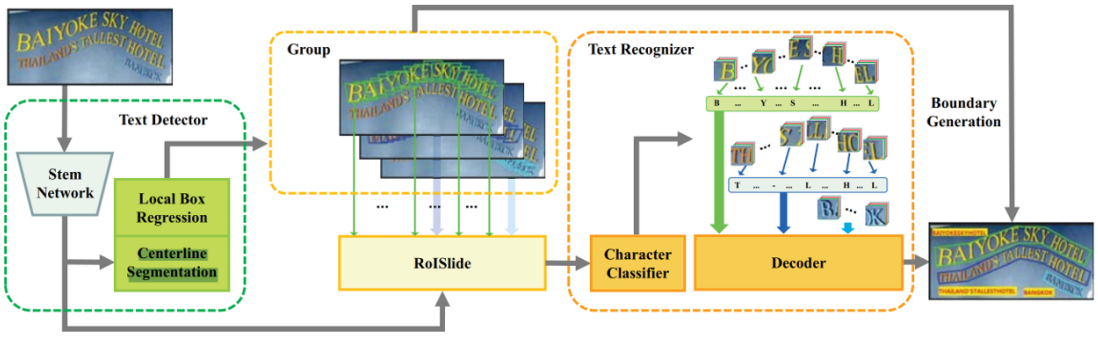


Figure 2.5: Pipeline of TextDragon

In the model from Xing et al., the output from the convolutional neural network contains two branches working in parallel: the character branch and the detection branch. The character branch follows the same schedule as the EAST method, and the detection branch first detects the bounding boxes for the characters, then groups them to be the is. This model is not limited by RoI cropping and pooling, and by using character as the detection unit, the model is naturally suitable for recognizing curved texts.

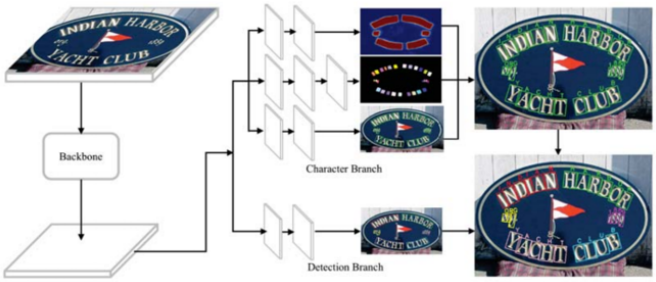


Figure 2.6: Pipeline of CharNet

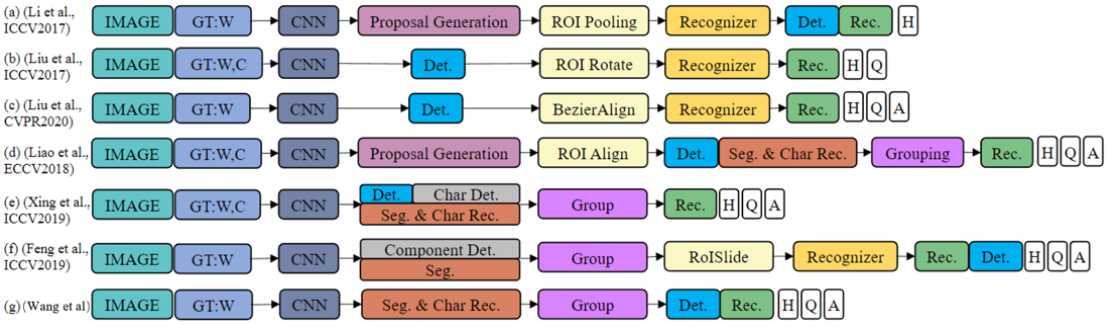


Figure 2.7: Overview of current end-to-end models

2.3.4 Paddle end-to-end model

The paddle end-to-end model is currently the only end-to-end model that is commercially available and is used in license plate recognition. Compared to other end-to-end models, the paddle end-to-end models consider the size and computation time and make more optimization to the computation process. Moreover, the algorithm of the paddle end-to-end model has the following advantages:

- Free of POI(point of interest): The end-to-end model is free of POI operation. Some end-to-end models first detect the text region and then recognize it based on the region; this operation for narrowing down the region will cause complex calculations that will cost computing resources and time. However, the PaddleOCR end-to-end model is free of POI operations, meaning less time to run the model.

- Free of character-level annotation: Some of the end-to-end models, for example, the TextDragon model, require both word and character-level annotations for training, which makes training difficult because there aren't enough OCR datasets with character-level annotations, and those character-level annotations are usually not so accurate. Since the PaddleOCR end-to-end model only needs word-level annotation for training, it avoids such

problems in finding suitable datasets.

The structure of the paddle end-to-end model is shown in Figure 2.8. For a given image, the image matrix with the size of $height \times weight \times 3$, where the 3 represent the 3 RGB channel , first go into the stem network, consisting of backbone, neck, and head part. In the original end-to-end model, the backbone is a standard ResNet50 model, the neck part is FPN structure, and the output of the head part is the visual features, consisting of four parts: the TCL(text center line) showing the center line of instances, text border offset (TBO) that gives information about the bounding boxes, text direction offset (TDO) indicating knowledge of the direction of text instances and text character classification map (TCC) which is the information for the text strings. Since the features are only raw information of OCR, the features go into the detection branch and PG-CTC decoder to give the final result for detecting bounding boxes and recognizing characters.

The detection branch and the related maps are inherited from the SAST algorithm from the two-stage model, and the PG-CTC decoder and associated maps are mainly inherited from the CRNN algorithm from the two-stage model, which will be further explained in Chapter 4.

In this project, to threshold the PaddleOCR model and make it able to run on a portable device, eventually, we will change its backbone and neck part in the stem network, which will be further explained in Chapter 3.

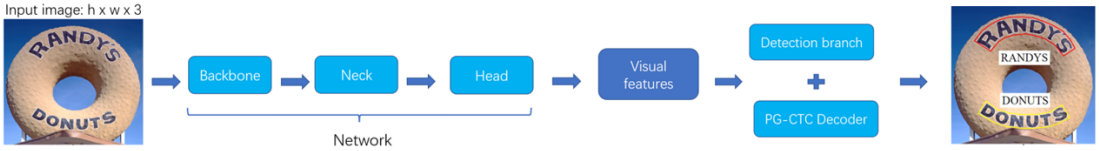


Figure 2.8: Paddle end-to-end model

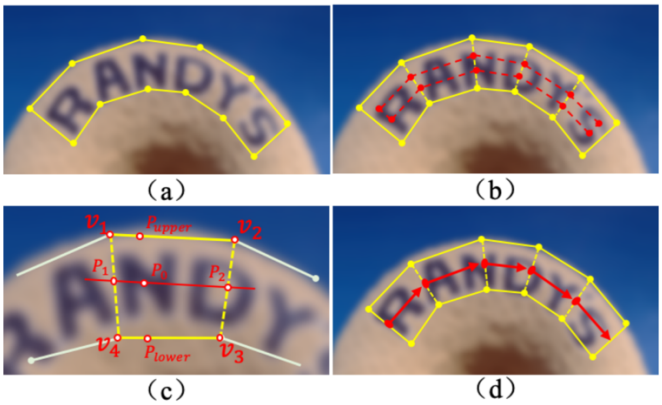


Figure 2.9: Paddle end-to-end maps: this figure shows the different feature maps output by the head part of PaddleOCR network

3 Model threshold

3.1 Overall approach

The stem network of the model is illustrated in Figure 2.8. The emphasis of the model is on minimizing the size of the backbone and neck components while keeping the head part of the network unchanged. This chapter explains the original structure and the rationale behind reconstructing the backbone and neck segments.

3.2 Backbone

The original backbone is ResNet-50, which has a size of approximately 100 MB [16]. The backbone is slated to be substituted to optimize the model with a customized MobileNetV3 variant, which is considerably more compact at around 5 MB.

3.2.1 MobileNet

MobileNet network was proposed by the Google team in 2017, focusing on lightweight CNN networks for mobile or embedded devices. Compared to traditional convolutional neural networks, MobileNet significantly reduces model parameters and computational requirements while achieving a slight decrease in accuracy [1]. Compared to the VGG network, the ImageNet accuracy of MobileNet only decreased by 0.9% but is 32 times smaller and 27 times less compute intensive. Table 3.1 compares MobileNet and VGG 16.

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Table 3.1: Model comparison between MobileNet and VGG 16 [1].

Compared to traditional convolutional neural networks, MobileNet mainly utilized depthwise separable convolution to threshold the size. Compared to traditional convolutional neural networks, depthwise separable convolution has the following advantages [17]:

- Reducing parameters: compared to traditional convolution process, depthwise separable convolution separates the convolution into two parts: The depthwise convolution and the pointwise convolution, and this reduces the number of parameters.
- Reducing calculation: since the convolution is separated into two parts in depthwise separable convolution, and each part is much smaller than the traditional convolution, the overall calculation is thus reduced.
- Light weight feature extraction: since MobileNet is designed for circumstances with limited computation resources, the depthwise separable convolution enables the MobileNet models to reduce model size while maintaining high recognition accuracy significantly

3.2.2 MobileNetV3

MobileNetV3 is a convolutional neural network designed to suit mobile phone CPUs and is the latest MobileNet model [18]. Compared with previous MobileNet models. MobileNetV3 is optimized in several aspects:

- Adding SE(Squeeze and excitation) module: The SE module enhances the feature representation capabilities of deep convolutional neural networks (CNN)
- Redesign the activation function: in MobileNetV3 the activation function "Hard Swish" is introduced:

$$HardSwish(x) = x * ReLU6(x+3)/6 \tag{3.1}$$

- Introducing NAS(Neural Architecture Search) algorithm: when applied to MobileNetV3, it contributes to its success by creating lightweight, efficient, and high-performing models that are well-suited for deployment on mobile devices and other edge computing scenarios.

3.2.3 MobileNetV3 rebuild

To align with the specifications of the original end-to-end model, the input image dimensions are set to $512 \times 512 \times 3$ (width $\times$ height $\times$ channel). Consequently, adjustments are made to the input size of the MobileNetV3 network to match the new image input size. This alteration leads to changes in both the input and output matrix sizes for each stage.

Furthermore, to seamlessly integrate with the FPN (neck) component, we now output the results from every stage of the MobileNetV3 network instead of solely passing along the final stage. For this implementation, we have opted for the

MobileNetV3 large model with a size factor of 0.5, resulting in differences in the stages compared to the original model.

The reconstructed structure is illustrated in Figure 3.1. The restructured MobileNetV3 model consists of six stages: the input stage, the stage following convolution, and subsequent sequential blocks. Each sequential block comprises a convolution layer, a batch normalization layer, and an activation function. These six stages are then fed into the next segment of the stem network: the neck part.

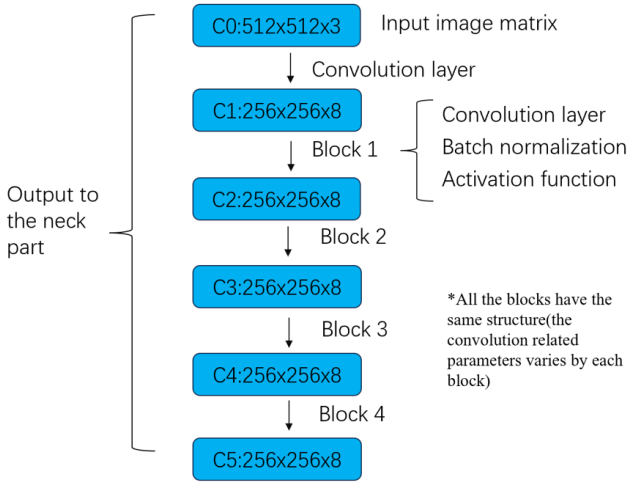


Figure 3.1: Rebuilt MobileNetV3 structure

3.3 Neck

3.3.1 FPN structure

The FPN structure, also known as Feature Pyramid Network structure, is an architecture aiming for better image feature extraction and classification, proposed by Tsung-Yi Lin et al [19]. The difference between FPN structure and other feature extraction is its ability to combine the features of all the extraction steps with accuracy and speed in mind.

The original FPN (Feature Pyramid Network) structure proposed by Tsung-Yi Lin et al. is depicted in the image labeled 3.2. This architecture comprises two key components: the bottom-up and top-down pathways.

The bottom-up pathway builds the foundation of the ResNet network structure. Features are successively extracted from stages C2 through C5, and the output size is halved at each stage. Each stage yields an output feature matrix.

Moving to the top-down pathway, it initiates from the output matrix of stage C5. This matrix undergoes upsampling using the nearest neighbor method, in-

creasing its dimensions to match the size of the output matrix of stage C4. These matrices are then merged together. This process is repeated for all adjacent outputs in the bottom-up pathway: upsample the smaller one to match the size and merge them together. Subsequently, the outputs of the top-down pathway are directed into the same head part for further processing.

The advantage of FPN is that it simultaneously preserves the feature of stages with different semantic values instead of keeping only the semantic value of the final stage for classification.

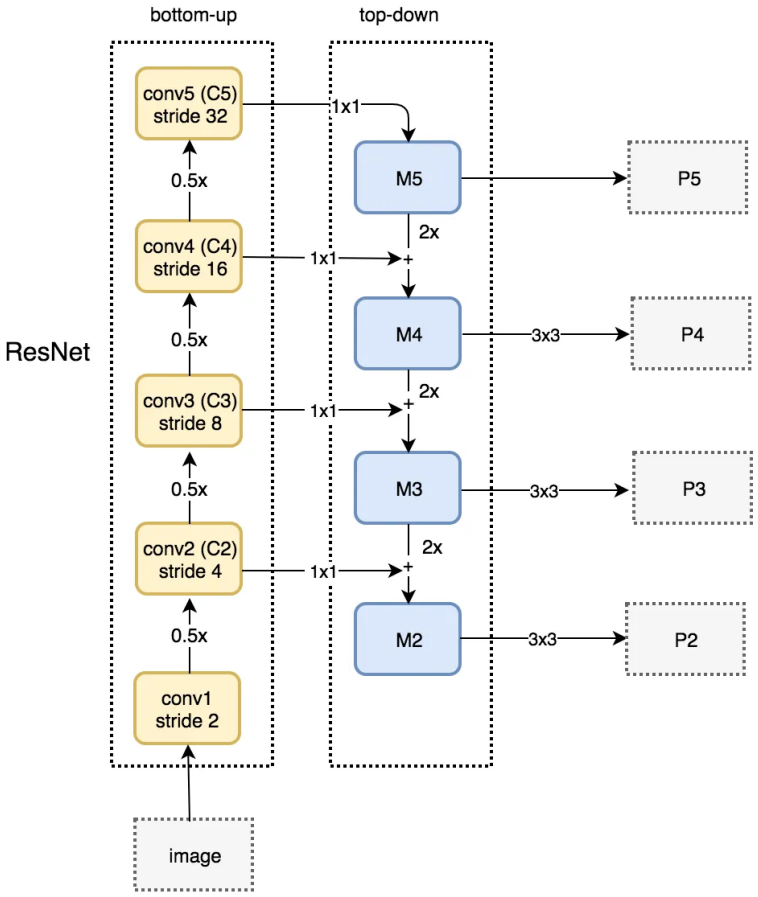


Figure 3.2: Original FPN structure

3.3.2 FPN structure rebuild

The neck component in the end-to-end model adopts the FPN structure, serving as both a feature extractor and a connector between the backbone and the stem network's neck part. The dimensions of the FPN neck are comparable to those of the MobileNetV3 portion in the backbone. Therefore, reducing the size of the FPN

section would lead to an overall reduction in the model's size. With this in mind, we propose two types of FPN structures: one with only the top-down portion and another with both bottom-up and top-down components. As the required input size for the head part is $(128 \times 128 \times 128)$, the output of the FPN part must adhere to this size specification.

3.3.2.1 Only top-down

Learning from the FPN part of the DB algorithm model, this design leverages the output of the preceding backbone network as the output for the bottom-up component of the FPN structure while constructing only the top-down segment. This structure is visually represented in Figure 3.3. Consequently, the overall model size with this FPN configuration is approximately 9MB.

As depicted in the figure, this structure adopts the 'C5 to C2' stages from the MobileNetV3 output. Starting from the C5 stage, it undergoes an up-sampling process followed by a convolution operation combined with the previous stage, C4. This operation pipeline of up-sampling, convolution, and addition is repeated for the upper layers, culminating in the final layer, C2. The resulting output size for the head part is $128 \times 128 \times 128$, following the height, width, and depth sequence.

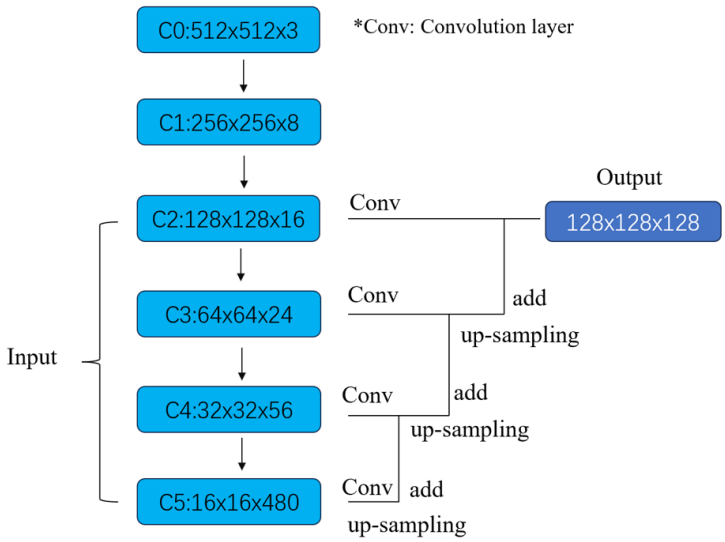


Figure 3.3: Top-down FPN structure

3.3.2.2 Bottom-up and top-down

Although focusing on reducing the structure of the FPN can significantly reduce the model's size, the impact on the final accuracy is not negligible. Therefore, we designed another FPN structure using the output of the backbone for both the bottom-up and top-down parts, shown in figure 3.4.

In Figure 3.4, we can see that the bottom-up component makes use of the C2 to C0 stages from the MobileNetV3 output. Starting with the C2 stage, it undergoes down-sampling, followed by batch normalization, and is then integrated with the C1 layer. This process is repeated for the upper layers. The top-down part remains unchanged from the previous section.

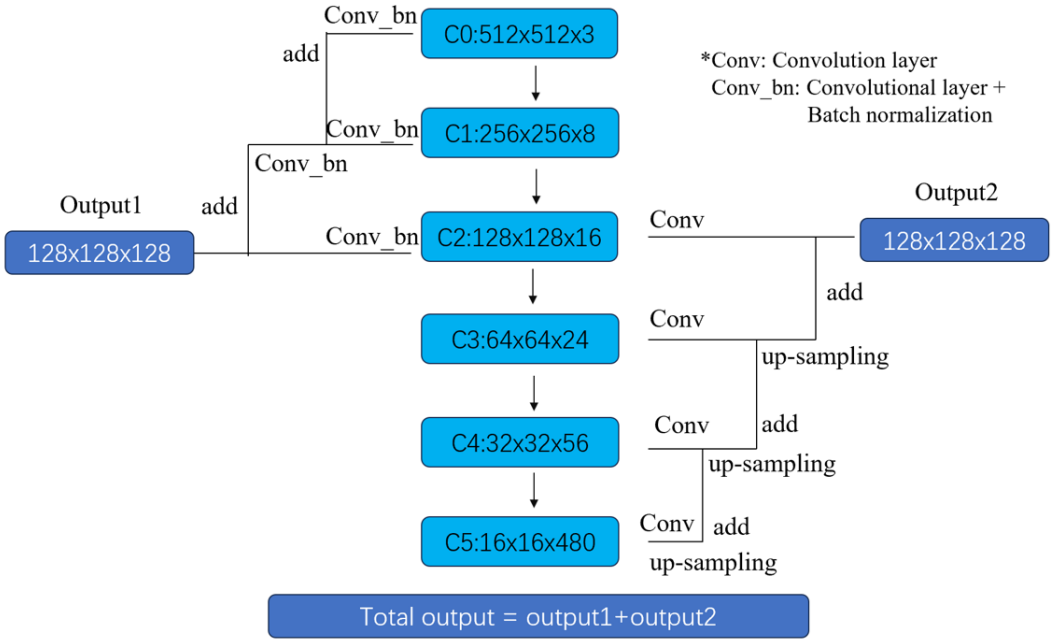


Figure 3.4: Bottom-up and top-down FPN structure

4 E2E algorithm

4.1 General

To attain a more profound comprehension of the algorithm, I meticulously scrutinized the codebase, and a comprehensive elucidation of the processing intricacies will be expounded upon in the ensuing chapter.

4.2 Preprocess

In the preprocessing, the image is processed with the following procedures:

- Image Resize: to enhance computational efficiency and ensure that the input image is of manageable size, the maximum side length of the input image is capped at 512. This value corresponds to the preferred input side length for the MobileNetV3 network. The image is resized accordingly while preserving the aspect ratio of its height and width. Additionally, the maximum stride is designated as 128. Consequently, both the height and width are rounded up to the nearest multiple of this maximum stride value. This adjustment facilitates seamless processing within the network, preventing potential issues associated with non-divisibility during upsampling convolutions.

- Image Normalization: this process involves four essential steps:

  1.Mean Subtraction: The pixel values of each channel are subtracted by the mean value of their respective channels. This operation centers the distribution of pixel values around zero mean;

  2.Standard Deviation Normalization: Subsequently, the pixel values of each channel are divided by the standard deviation of that channel. This step ensures unit variance normalization.

  3.Scaling: This step is employed to transform pixel values from the [0, 255] range to the [0, 1] range. Scaling the pixel values to [0, 1] aligns them with the range commonly utilized in neural networks.

  4.Dimension Order: This parameter indicates the arrangement of dimensions within the image.

4.3 Features

There are four features output by : TCL(text center line),TBO(text boundary offset), TDO(text direction offset), TCC(text character classification). The map illustration with the original image is shown in figure 4.1.

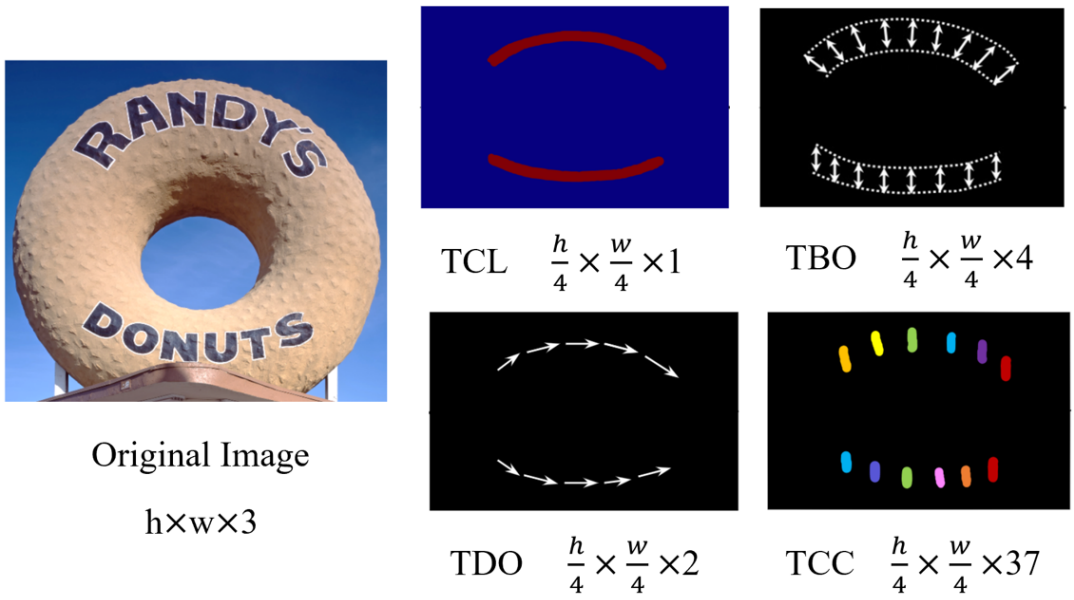


Figure 4.1: maps

4.3.1 TCL

The TCL(text center line) gives a rough sketch of the center line of text instances. To be specific the map is with values ranging from 0 to 1 which indicate the probability that there is a center line. To be specific the lines given in TCL have width more than 1 pixel and need further treatment.

4.3.2 TBO

The TBO is the offset of the center point and the upper and lower lines of bounding boxes, as seen in figure 4.2, the offset is the vector offset between P_0 and P_{upper}, P_{lower}. For the points with value 0 in TCL, the value of corresponding point is set to $0, 0, 0, 0$ in TBO.

Next we explain how is the offsets defined. For a point P_0 in the center line with a partial bounding box with vertices V_1, V_2, V_3, V_4(the points are arranged in

clockwise starting from the top left corner), we first determine the intersection points P_1, P_2 of the line through P_0: slope of the line is the average slope of line V_1V_2 and line V_4V_3:

$$slope_{P_0} = \frac{slope_{V_1V_2} + slope_{V_4V_3}}{2} \tag{4.1}$$

and combining the slope and the coordinate of P_0 the line can be determined. After that, we calculate the intersection point of the line with line V_1V_4 and line V_2V_3, and name the intersection points as P_1 and P_2 respectively.

Since P_0 in line P_1P_2 is at the same relative position as P_{upper} in line V_1V_2 and P_{lower} in line line V_4V_3, we can write the equation:

$$\frac{P_0 - P_1}{P_2 - P_1} = \frac{P_{upper} - V_1}{V_2 - V_1} = \frac{P_{lower} - V_4}{V_3 - V_4} \tag{4.2}$$

In this way P_{upper} and P_{lower} can be determined, and the offset value is the location difference between them and the point P_0.

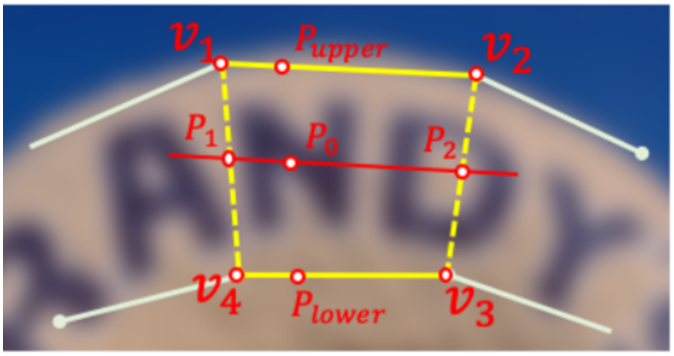


Figure 4.2: TBO

4.3.3 TDO

The TDO(text direction offset) suggests the direction of the text. Each point is given a vector x, y as the direction vector's x and y weight. The It is worth notifying that, the vector is not normalized, because in the actual calculation we only need the ratio of y and x.

4.3.4 TCC

The TCC give the probability of the character for pixels in the feature map. For each pixel, a probability vector of 1×37 is given, where 37 is the length of the dictionary of English OCR, including 26 letters, 10 Arabic numerals, and one background class.

4.4 Postprocess

The postprocess consists of two parts. In the first part, it generate the center line and the corresponding text strings, namely `pivot`; in the second part, it generate the bounding boxes based on the center lines generated in the first step, namely `restore_poly`.

4.4.1 Pivot

4.4.1.1 Processing TCL

Since the TCL is just a rough result feature map with number in the range of [0,1], it first binarized with a threshold value of 0.5. After that, the map is skeletonized, meaning that the width of the lines are eroded to be 1 pixel. Here I optimized the algorithm using the gau algorithm instead of the zhang algorithm, to get a clearer skeleton of the line map. Then the number of lines(also known as the number of text instances) is calculated using the function `ConnectedComponets`: this function returns the number of instances and the label map of instances. For example if there are 2 instances, in the label map the background points will be 0, and the points in the first instance will be 1, the points in the second instance will be 2, the returned number will be 3.

4.4.1.2 Generate instance coordinates and its direction

The TDO maps are initially transposed from dimensions $2 \times h \times w$ to $h \times w \times 2$ to facilitate subsequent calculations. Pixel coordinates corresponding to the instance centerlines are then generated. For each instance centerline, the corresponding vectors from the TBO map are computed.

Subsequently, to establish the sequence of pixels along each instance centerline, we employ the following method:

As depicted in (a) in Figure 4.3, we compute the average direction of the TBO vectors and determine the sequence of points along the instance centerline by comparing the projections of the point vectors onto the TBO vectors. However, as illustrated in (b), if the centerline is lengthy and contains turning points with large angles, the sequencing algorithm may result in points appearing disorderly, even though the sequence of most points would be correct. Therefore, for centerlines with more than 16 points, the sequence is recalculated by dividing the line into two parts—referred to as the 'left' and 'right' segments in the image. The sequence is then determined separately for each part and subsequently combined.

In the next step, the center line is averagely cut into 3 parts, as seen in figure 4.4 (a), and the following parameters are calculated for the left and right parts:

- Average direction: the average TDO vector of the part.

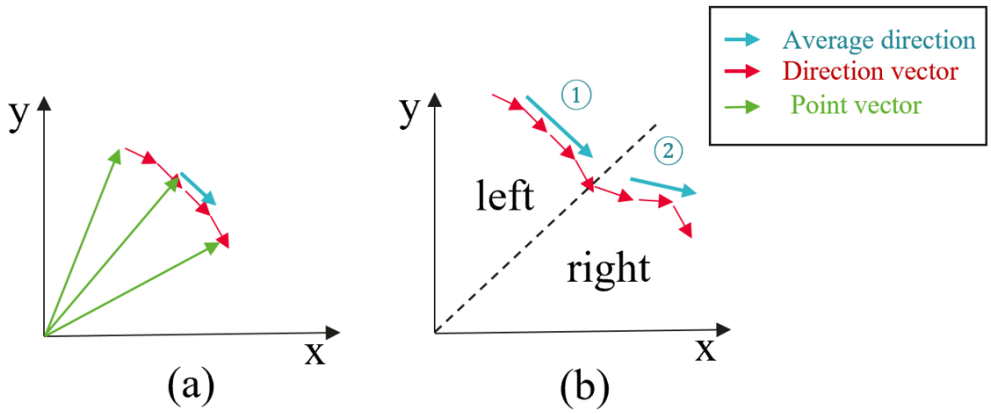


Figure 4.3: sequence

- Average length: the average length of TDO vector of the part.
- Start point: the point on the two sides of the center line.
- Step: average direction/average length.

  As there might be points excluded during the 'skeletonize' phase that actually pertain to the centerline, this step is intended to supplement those omitted points on either side. As illustrated in Figure 4.4, once the average direction of the left and right segments is established, points are appended in both the negative left average and right average directions. The step size has already been defined, commencing from the endpoints of the centerline.

  With these procedures followed, precise centerlines of the text instances are now ascertained with accurate sequencing. The subsequent step involves identifying the character instances based on this centerline, employing the CTC decoder.

4.4.2 CTC decoder

The characters are predicted based on the pixels along the centerline by comparing the probability values in the corresponding pixels of the TCC map. Background pixels are assigned a value of 0, while the remaining values represent characters. Connected characters that repeat are consolidated into a single character and separated by the background class. Ultimately, the numbers are clustered together to align with the characters in the dictionary, forming the output string.

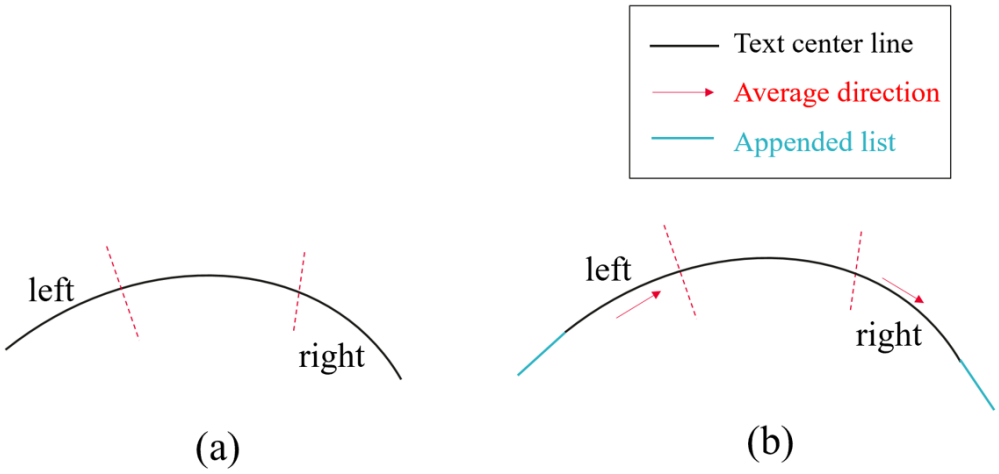


Figure 4.4: append_sequence

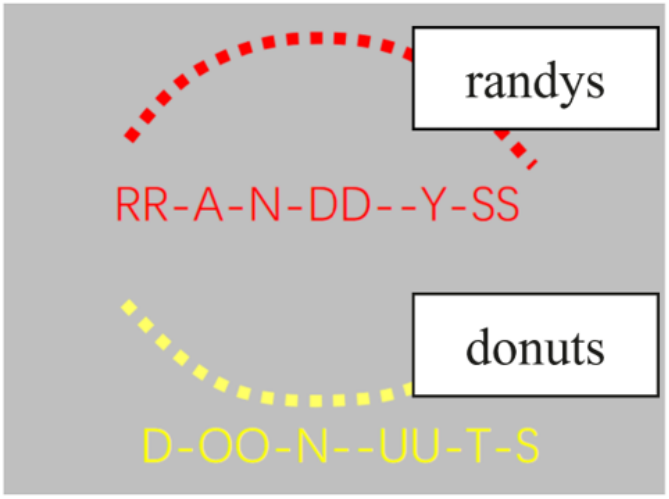


Figure 4.5: CTC decoder

4.4.3 Restore_poly

In this step, the bounding boxes are determined based on the text centerline established in the preceding stage.

Although the centerline points persist as an extensive list from the prior step, only a select few key points are essential. Therefore, for the centerline, only seven points are retained: the points flanking either side, along with five equidistantly spaced points in between. In the case of rectangular-shaped texts, solely the points on the two sides are preserved.

For every point within the chosen list, the corresponding border point offset is

extracted from the TBO map. By adding this offset to the point, the corresponding upper and lower border points can be ascertained. Since we require the bounding boxes to be drawn in a specific sequence (clockwise), the points are reorganized accordingly. This resequencing is illustrated in Figure 4.6, where (a) depicts the initial point sequence, while (b) displays the resequenced points in a clockwise manner.

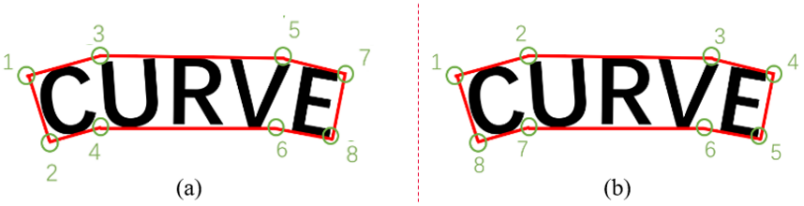


Figure 4.6: The rearranging of the border points

4.5 Drawing

The output comprises both the coordinates for the bounding boxes and the corresponding instance text string. These bounding boxes, along with their associated texts, are visualized on the image using the `draw` function provided by OpenCV. In each instance, the text is positioned at the upper-left corner of its respective bounding box.

5 Experimental setting

5.1 Deep learning platform

This project is developed on the PaddlePaddle deep learning platform. Compared to other prominent platforms like PyTorch, PaddlePaddle offers similar functionalities, including robust support for industrial-grade applications and the capacity for large-scale data processing. Additionally, PaddlePaddle excels in providing strong support for dynamic graph construction during training, enhancing its flexibility and adaptability in various scenarios.

5.1.1 Model types

Three different types of models serve various purposes:

- Inference model: this model is utilized for inference through the Paddle inference engine, enabling the generation of output results.
- Trained model, pre-trained model: these models serve as checkpoint models during training. Each model comprises three files, denoted by *.pdparams, *.pdopt, and *.states. These files store the model's parameters, primarily used for model evaluation and continuous training.
- Nb model: this model is optimized using the Paddle-Lite framework, making it suitable for deployment in mobile-side scenarios (Paddle-Lite is required for nb model deployment).

The three types of models are in distinct formats, each requiring a specific engine provided by PaddlePaddle to function. Once trained with datasets, the Trained model can be converted to an Inference model, and the Inference model can further be transformed into an Nb model. However, it's important to note that these conversions are unidirectional and cannot be reversed, as illustrated in Figure 5.1.

5.2 Hardware and version

The hardware used for training the model includes Quadro RTX 6000 *3 and Quadro RTX 8000*3. The CPU utilized is the Intel(R) Xeon(R) Gold 6226 CPU @ 2.70GHz.

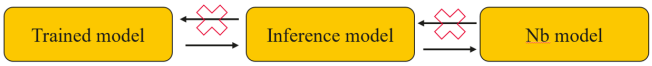


Figure 5.1: Model Transition in the PaddlePaddle System: Models can progress forward in the pipeline but cannot revert backward.

The project is based on Paddlepaddle-gpu 2.5.1, CUDA10.2, and python3.8 for the platform and software version.

5.3 Datasets and label format

5.3.1 Datasets

While OCR is well-researched, real-world scene images suitable for training OCR models are relatively limited. Therefore, a combination of synthetic datasets and real scene datasets are employed for training. Generally, two types of databases are used, each aiming to train the model for rectangular and curved text respectively: those with rectangular bounding boxes and those with curved/irregular bounding boxes. According to the official documentation of PaddleOCR, recognizing images with English text instances enclosed in rectangular bounding boxes requires tens of thousands of images for accurate recognition and thousands of images for effective detection.

For rectangular bounding boxes, the utilized database is as follows:

- **SynthText:** This dataset encompasses 800 thousand images, comprising approximately 8 million synthetic word instances. Each text instance is meticulously annotated with its corresponding text-string and word-level and character-level bounding boxes [20]. While many images share the same background, the word instances vary in content and style.
- **ICDAR2015:** This dataset comprises 1500 images featuring real scene texts captured using wearable devices. The provided text instances are outlined by rectangular bounding boxes accompanied by coordinate and character information. Lexicons of words, namely 'strong', 'weak', and 'general', are also provided. It's worth noting that the images in this dataset tend to be more challenging, often characterized by blurriness and smaller text size, which typically leads to lower OCR performance [21].
- **ICDAR2017:** This dataset encompasses 7200 images in the training set and 1800 images in the test set [22]. The images predominantly feature real scenes, with word instances of medium size and in multiple languages.

For curved or irregular bounding boxes, the utilized database is as follows:

- **Total-text:** This dataset comprises 1500 images featuring texts in real-world scenes. The texts exhibit arbitrary shapes, and coordinate information along with character details are provided [23]. Labels are assigned at the word level, with word instances typically larger than the norm.
- **ICDAR2019-ArT:** This dataset encompasses approximately 5500 images in the training set and around 1000 images in the test set, showcasing text instances in multiple languages. Labels are provided at the word level, along with information about the language of each word.

5.3.2 PaddleOCR label format

To prepare the datasets for training, all label files from the datasets above are converted to the required data format for PaddleOCR, which is in text form:

```
" Image file name             Image annotation information encoded by json.dumps"
ch4_test_images/img_61.jpg    [{"transcription": "MASA", "points": [[310, 104], [416, 141], [418, 216], [312, 179]]}, {...}]
```

Figure 5.2: PaddleOCR annotation file format

5.4 Dataset processing

This section shows the processing methods for the images and the dataset's label file.

5.4.1 SynthText

Actual training has shown that low-resolution images adversely affect OCR performance. Thus, images in the SynthText dataset with a resolution below 300×300 are omitted. The label file is available in Matlab's .mat format. Furthermore, it's worth noting that many images in the dataset share repeated backgrounds, which are selectively avoided during training.

5.4.2 ICDAR2019-ArT

The label file of the ICDAR2019-ArT dataset is provided in JSON format. The label file contains information about each image's instance, language type, coordinates, and illegibility. To prepare the data for PaddleOCR, words with the language type 'Latin' are selected and transformed into the PaddleOCR training format.

An additional step is required to enable PaddleOCR to recognize the label file. A specific set of 14 points is needed for curved texts, arranged in clockwise order starting from the left upper point. However, in the ICDAR2019 dataset, words

are given with either 5, 6, 8, 10, or 12-point bounding boxes. For bounding boxes with only 5 points, determining where to insert points is challenging, and since they represent a small portion of the dataset, these instances are discarded.

For other types of bounding boxes, the process of inserting points is illustrated in Figure 5.3. In this figure, the methods for inserting points in bounding boxes with 6, 8, 10, and 12 points are depicted in (a), (b), (c), and (d), respectively. The green points represent the original points, and the blue points are the inserted ones evenly distributed between the nearby original points.

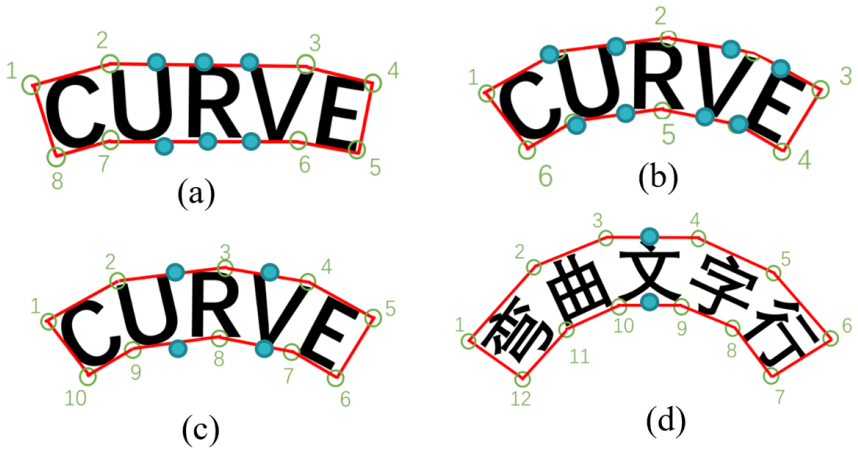


Figure 5.3: ICDAR2019: inserting points for labels with different numbers of points

5.4.3 ICDAR2017

The label files for the ICDAR2017 dataset are organized within a folder, with each TXT file corresponding to an image. These TXT files contain information such as bounding box coordinates, word transcriptions, and language labels. Specifically, only words labeled with the 'Latin' language are processed and adapted for training.

5.4.4 ICDAR2015 & Total-Text

The PaddleOCR format label files are already provided by PaddleOCR officially, so they are directly used in training without further processing.

5.5 Image augmentation

Due to the lack of data and to make the model more robust, the images are augmented in the following ways:

- Blur
- Jitter
- Gaussian noise
- random crop
- Disturbance method

Each disturbance method is selected with 40% of the total batch size during the training process.

5.6 Base model and fine tuning

While real scene data tends to yield superior results compared to synthetic data, it's worth noting that, based on previous OCR studies, achieving a satisfactory model performance for recognition requires hundreds of thousands of images. Unfortunately, the availability of real-scene data is limited.

Additionally, if all the data were combined into a single training set, the synthetic data would disproportionately influence the results, given its significantly larger proportion than real-scene data. To address this, our training approach involves initially training a base model using the synthetic dataset, followed by fine-tuning this base model using the real-scene data.

5.6.1 Base model

In the base model training, 10,000 images and 20,000 images in the SynthText dataset are used respectively, and parameters are set to be:

- Batch Size: Given the substantial volume of data, the batch size is set at 32, representing the maximum size compatible with GPU processing capabilities.
- Warm-up Epoch: The warm-up epoch is set to 10 for pretraining.
- Learning rate: the learning rate is 0.001.

5.6.2 Freezing layers

Layer freezing involves keeping the weights of trained model layers unchanged during reuse in subsequent downstream tasks. These frozen layers remain static. In the training process, the head layers are frozen after a few training epochs to enhance performance [24].

5.6.3 Warm up steps

A "warm-up step" in machine learning involves initially using a lower learning rate that gradually increases over training iterations or epochs, as shown in the left part of figure 5.4. This helps the model adapt to the data more steadily before higher learning rates are applied, leading to more stable and efficient training.

5.6.4 Cosine

In the training phase, the learning rate is configured to follow a Cosine decay pattern, as illustrated in Figure 5.4. This decay scheme adheres to the following functional behavior:

- Initially, it slows down the rate of decrease.
- Gradually, it accelerates the rate of decrease.
- Ultimately, it approaches the final value.

Compared to other decay methods like linear decay, which reduces the learning rate at a constant rate, and exponential decay, where the learning rate decreases exponentially with the number of iterations, Cosine decay provides a unique advantage. It ensures a smooth transition from the initial to the final learning rate values. This gradual shift helps prevent sudden spikes or drops in the learning rate, ultimately contributing to a more stable and robust training process.

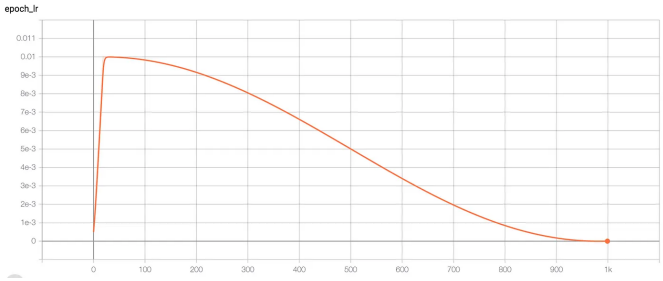


Figure 5.4: Warm up and Cosine decay

5.6.5 Fine tuning

During the fine-tuning phase, the model undergoes training with a reduced learning rate and increased epochs. This phase involves the utilization of actual datasets capturing real-world scenes.

- Batch size: given the substantial volume of data, a batch size of 14 is employed, as it has demonstrated optimal performance in training the PaddleOCR model.

- Warm-up epoch: the warm-up epoch is set to 50, facilitating a gradual increase in the learning rate.
- Learning rate: the learning rate is 0.0005.

5.6.6 Ratio of datasets

PaddlePaddle enables the utilization of two distinct datasets in the training process, along with the flexibility to set the proportion of images used in each dataset. During the fine-tuning step, various dataset ratios are assessed to identify the configuration that yields the highest performance.

5.7 Loss function

The loss function is set as:

$$L = \lambda_1 L_{tcl} + \lambda_2 L_{tbo} + \lambda_3 L_{tdo} + \lambda_4 L_{tcc} \tag{5.1}$$

In this function, L_{tcl},L_{tbo},L_{tdo},L_{tcc}, represent the loss of TCL, TBO, TDO and TCC maps. The loss weights are set as $1.0, 1.0, 1.0, 5.0$ respectively, according to the original PaddleOCR end-to-end model.

5.7.1 TCL loss

For the TCL, the Dice loss is applied [25]. The full name of Dice Loss is the Sørensen-Dice coefficient, also known as the F1 score. It is defined as follows:

$$\text{DiceLoss} = 1 - \frac{2 \times \text{Intersection}(A, B)}{\text{Cardinality } (A) + \text{Cardinality}(B)} \tag{5.2}$$

In this equation, A is the binarized version of the model's prediction result, and B is the binarized version of the real label. Intersection(A, B) is the size of the intersection of A and B, and Cardinality (A) and Cardinality (B) is the size of A and B respectively.

The value of Dice Loss ranges from 0 to 1. The closer the value is to 1, the more similar the model's prediction results are to the real labels

5.7.2 TBO and TDO loss

For the TBO and TDO map, the smooth L1 loss is adapted. For a batch size of N, the loss is defined as:

$$\ell(x, y) = L = \{l_1, \ldots, l_N\}^T \tag{5.3}$$

where

$$l_n = \begin{cases} 0.5\,(x_n - y_n)^2 \,/\, \text{beta}\ , & \text{if } |x_n - y_n| < \text{ beta} \\ |x_n - y_n| - 0.5 * \text{ beta}\ , & \text{otherwise} \end{cases} \tag{5.4}$$

In this equation, β determines the threshold at which the transition between L1 and L2 loss occurs. It exhibits a gradual increase for minor errors while demonstrating a linear growth rate for more significant errors.

5.7.3 TCC loss

As stated in chapter 4, the TCC maps are maps of 37 characters. The point gathering operation aims to format the TCC map as follows:

$$P_\pi = \text{ gather } (TCC, \pi) \tag{5.5}$$

In this equation, $\pi = \{p_1, p_2, \ldots, p_N\}$ is the center point sequence, with a length N, and $p_i = (x_i, y_i)$ is the point coordinate. On the left of the equation, P_π is the gathered sequence with the size of $N \times 37$.

For an image with M text instances, the TCC loss is defined as:

$$L_{TCC} = \sum_{i=1}^{M} CTC\_loss\,(P_{\pi_i}, L_i) \tag{5.6}$$

Where CTC_loss is the classic CTC loss, and $\{\pi_1, \pi_2, \ldots, \pi_M\}$ are the center point coordinate sequences of the M text instances, and $\{L_1, L_2, \ldots, L_M\}$ are the corresponding transcript labels.

6 Results

6.1 Evaluating parameters

Our evaluation considers the following key parameters: precision, recall, F-score, and FPS (frames per second). We will comprehensively compare the performance between the threshold end-to-end and original models with a ResNet50 backbone.

6.1.1 Precision

In general definition, precision in machine learning is defined as:

$$Precision = \frac{TP}{TP + FP} \tag{6.1}$$

where:

- TP (True Positives): The count of samples correctly identified as positive categories among the positive category samples.
- FP (False Positives): The count of negative category samples incorrectly identified as positive categories.

n the precision evaluation of the end-to-end OCR model, two precision values will be assessed: detection precision and end-to-end precision. These will be discussed below.

6.1.1.1 Detection precision

In OCR, detection refers to identifying the bounding boxes of text in images. For both arbitrary-shaped and rectangular texts, true positives in detection refer to the successful identification of coordinates, be they 4 points or 14 points, that correspond closely to the actual bounding boxes. The '(TP + FP)' represents the total number of all detected bounding boxes.

In the evaluation code file `Deteval.py`, the detected number of bounding boxes is referred to as `total_num_text`, while true positives are denoted by `global_accumulative_precision`. Detection precision is defined as:

$$\text{detection\_precision} = \frac{\text{global\_accumulative\_precision}}{\text{total\_num\_det}} \tag{6.2}$$

6.1.1.2 end-to-end precision

OCR includes both detection and recognition processes. Recognition involves identifying the characters within the detected bounding boxes. In end-to-end precision, true positives pertain to instances with correct words and bounding boxes and are denoted as `hit_str_count`.

As end-to-end precision is a global measure, the variable '(TP + FP)' still represents the number of detected bounding boxes, labeled as `total_num_det`. Consequently, end-to-end precision is defined as:

$$\text{end-to-end\_precision} = \frac{\text{hit\_str\_count}}{\text{total\_num\_det}} \tag{6.3}$$

6.1.2 Recall

In general definition, recall in machine learning is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{6.4}$$

, where:

- TP (True Positives): The number of samples that are actually positive categories are correctly identified as positive categories.
- FN (False Negatives): The number of samples that are positive categories that are incorrectly identified as negative categories.

6.1.2.1 Detection recall

For OCR detection, the `(TP+FN)` variable is defined as the number of all the bounding boxes, named `total_num_gt`, where 'gt' refers to the ground truth, and true positives are still the number of correctly detected bounding boxes global_accumulative_precision. The detection recall is defined as:

$$\text{detection\_recall} = \frac{\text{global\_accumulative\_precision}}{\text{total\_num\_gt}} \tag{6.5}$$

6.1.2.2 End-to-end recall

Similarly, the `TP` value in end-to-end recall is same as the number of correctly recognized instances `hit_str_count` as defined in 6.1.1.2, and the end-to-end recall is defined as:

$$\text{end-to-end\_recall} = \frac{\text{hit\_str\_count}}{\text{total\_num\_gt}} \tag{6.6}$$

6.1.3 F-score

F-score is defined as:

$$F - score = 2 * \frac{precision * recall}{precision + recall} \tag{6.7}$$

F-score is used to show a balanced result between precision and recall. There is a trade-off between precision and recall because they correspond to different prediction targets, and optimizing one metric may affect the performance of the other. It is necessary to decide whether to pay more attention to precision or recall according to specific application scenarios and needs. For example, the recall rate may be more critical in an OCR job for identifying disease names in medical reports since you do not want to miss important conditions. As a result, the precision will decrease because the system may identify text that isn't a disease name as a disease.

6.1.4 Frame per second(FPS)

In the evaluation process, the FPS is defined as the number of images the OCR model can process in one second. Higher FPS means the system can process images faster, thus improving overall recognition efficiency.

6.2 Ablation study

6.2.1 TDO map

In this ablation study, we aim to assess the effectiveness of the TDO feature map, which is employed for discerning the text direction, particularly in non-traditional reading sequences, and for enhancing the center line. The experiment is conducted on the test sets of Total-Text and ICDAR2015, and the results are presented in Table 6.1. The evaluation metric used is the end-to-end F-score. For the ICDAR 2015 test set, a generic lexicon is utilized.

An example illustrating the impact of using TDO is shown in Figure 6.1. In this image, (a) displays the result utilizing the TDO map, while (b) presents the result without it. Evidently, without the TDO map, the model struggles to provide the correct word sequence and bounding box coordinates. The table illustrates that the TDO map distinctly influences the OCR outcome, particularly for images containing curved texts. As depicted in the figure, negligible alteration is observed in the left half of the word. However, an accurate result cannot be obtained for the right half without the correct orientation of the word instance's direction.

(a) (b)

Figure 6.1: Sample images for TDO ablation. The evaluation parameter is the f-score.

Data Set	Rectangular texts		Curved texts	
Task	Detection	End-to-end	Detection	End-to-end
w/o TDO	50.1	32.4	52.0	21.9
with TDO	60.5	43.6	72.7	45.3
Gain	+10.4	+11.2	+20.7	+23.8

Table 6.1: Ablation study for TDO. The evaluating parameter is the F-score.

6.2.2 FPN structure

The training results on the FPN structure with only the top-down part and both bottom-up and top-down parts are compared, as shown in table 3.2. As seen in the table, although the FPN structure with only the top-down part saves nearly half of the size, but the results are not worth it, as for detection and end-t-end result it showed an unsatisfying performance compared to the FPN with both bottom-up and top-down parts.

Data Set	Rectangular texts		Curved texts	
Task	Detection	End-to-end	Detection	End-to-end
Full FPN	60.5	43.6	72.7	45.3
Top-down	42.9	21.6	54.3	22.1
Gain	+17.6	+22.0	+18.4	+23.2

Table 6.2: Ablation study for FPN structure. The evaluating parameter is the F-score. This table compares the model performance in using FPN structure with only top-down pat and with both top-down and bottom-up parts, depicted as 'Full FPN'.

6.3 Detection results

The detection performance was assessed using the test sets from ICDAR2015 and Total-text. Initially, a base model was trained on 10,000 images from SynthText, achieving an f-score of 52.2% on the ICDAR2015 test set. Subsequently, by expanding the training data to 20,000 images from SynthText, the detection result saw significant improvement, yielding an f-score of 57.8% on the same test set. The detailed results are presented in the table 6.3.

6.3.1 Rectangular texts

The example image displaying the detection results can be observed in Figure 6.2. As depicted, most images in the ICDAR2015 dataset originate from real-world scenes, often characterized by low light conditions and smaller text instances. The model undergoes initial fine-tuning using the training set of the ICDAR2015 dataset. Subsequently, a combined dataset comprising ICDAR2017 and ICDAR2015 is utilized for further fine-tuning, with a ratio of 7:3. Other ratio of the datasets were also tested, but there was no significant change. While the f-score for detection sees minimal change, there is a notable improvement in the recognition performance. Detailed results are presented in the table 6.3. Compared to the PPOCR-V2 model of PaddleOCR, the f-score of the DB OCR detection model using the same MobileNetV3 backbone is 61.3%, our end-to-end model is very close to this result, showing a satisfying performance [5].

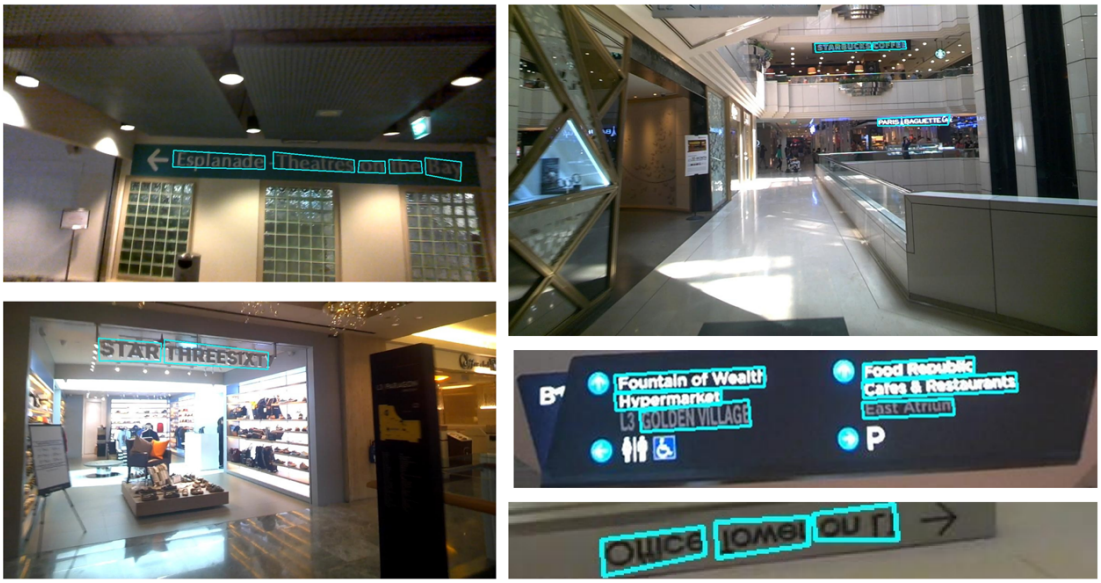


Figure 6.2: Sample images for detection results. The images show different circumstances for curved text detection

	F-score	Precision	Recall
SynthText1	51.8	51.5	52.2
SynthText2	58.8	59.9	52.2
ICDAR2015	60.6	61.5	59.7
7:3	60.5	65.3	56.4

Table 6.3: Detection result for rectangular texts. The table shows the performance for experiments using different data from the SynthText, using the ICDAR2015 dataset and using both the ICDAR2015 and the ICDAR2017 dataset.

Data Set	F-score	Precision	Recall
Total-text	71.3	74.5	68.3
6:4	71.7	75.6	67.1
7:3	72.7	75.6	70.0
8:2	71.9	74.3	70.0

Table 6.4: Detection result for the curved-texts. The table shows the performance for experiments using different data from the Total-text dataset and using both the Total-text and ICDAR2019 datasets.

6.3.2 Curved texts

The model designed for handling curved texts undergoes an initial fine-tuning process on the training set of the Total-text dataset. Subsequently, the model is trained on the ICDAR2019-ArT dataset for further refinement. Different ratios of images from these two datasets are experimented with to achieve an optimized result, as illustrated in Table 6.4. Ultimately, the best ratio is determined to be 3:7 between the ICDAR2019 and Total-text datasets.

The example images showcasing the detection results are presented in Figure 6.3. The images demonstrate the model's capability to accurately provide text bounding box information across varying light conditions and font styles.

6.4 End-to-end results

As described in previous sections, the model is firstly trained on Synthetic dataset to get a base model and then fine tuned on real scene datasets. The end-to-end result of the based model is shown in table 6.5.

Figure 6.3: Sample images for detection result

	End-to-end result				
	F-score	Precision	Recall	Weak lexicon	Generic lexicon
SynthText1	22.9	23.9	22	31.4	29.5
SynthText2	24.3	25.9	22.9	36.1	32.2
ICDAR2015	23.2	25.1	21.6	33.3	31.0
7:3	34.5	38.1	31.4	48.5	43.6

Table 6.5: End-to-end result for rectangular texts. The table shows the performance for experiments using different data from the SynthText, using the ICDAR2015 dataset, and using both the ICDAR2015 and the ICDAR2017 dataset.

6.4.1 Rectangular texts

The end-to-end result of training for rectangular texts is shown in table 6.5. Sample result images are shown in 6.4. As seen in the images, the model is robust to scenes with poor light conditions, high color contrasts and blurred images.

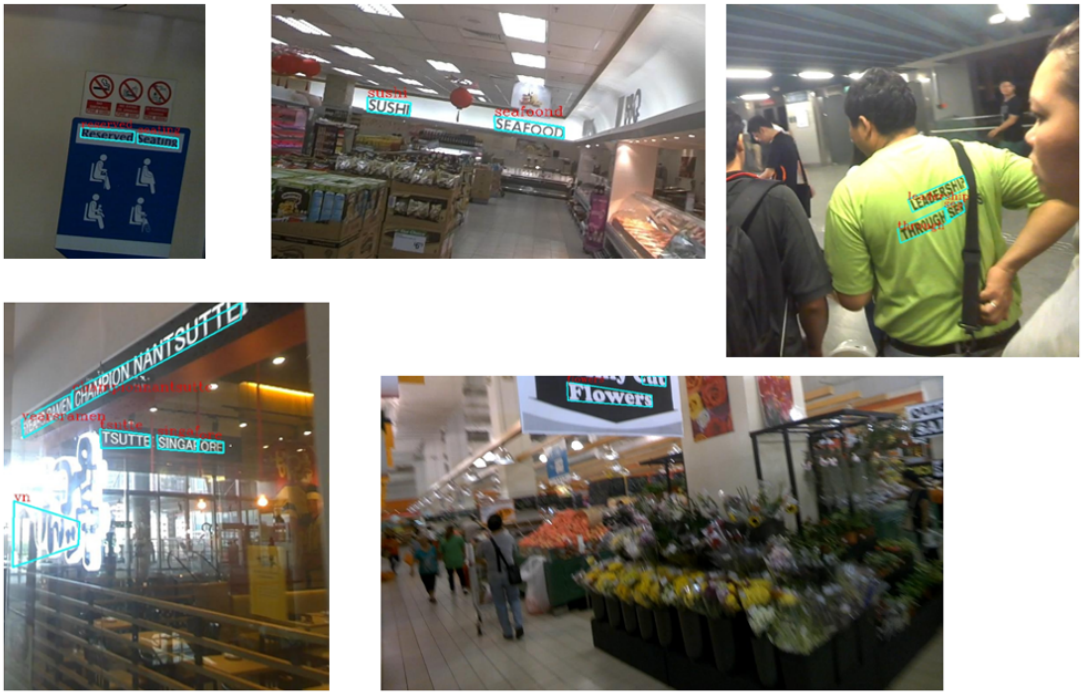


Figure 6.4: Sample end-to-end OCR result on the ICDAR2015 dataset.

6.4.2 Curved texts

For images with curved texts, the end-to-end result of out model is shown in table 6.6. Shown in the table, when the ratio of the Total-text dataset and the ICDAR2019 dataset is 7:3 the highest f-score of end-to-end is achieved. Figure 6.5 shown some sample images of end-to-end result of the images with curved texts.

	F-score	Precision	Recall
Total-text	40.0	42.5	41.2
6:4	42.4	45.3	40.0
7:3	45.3	47.3	43.6
8:2	44.2	45.9	42.6

Table 6.6: End-to-end result for curved texts. The table shows performance for using the Total-text dataset and both the Total-text dataset and the ICDAR2019 dataset with different ratios.

Figure 6.5: Sample end-to-end OCR result on the Total-text dataset.

6.5 Freezing layer training

To test the effect of freezing layers, the layers for the backbone part of the model are set as 'stop gradient', and the model is further trained. Since there are seven outputs of the backbone part, their gradients are frozen, respectively. However, there is no significant improvement based on the original performance.

6.6 Comparison with the original PaddleOCR end-to-end model

In this section, we compare the performance of our model and the original model on the test set of the Total-text and ICDAR2015 dataset for rectangular and curved texts respectively.

	Rectangular	Curved	Size
Our	41.6	37.1	19MB
Original	30.8	20.52	198MB

Table 6.7: FPS and size comparison of our model and the original PaddleOCR end-to-end model.

	Rectangular texts			Curved texts		
	F-score	Precision	Recall	F-score	Precision	Recall
Our	60.5	65.3	56.4	72.7	75.6	70.0
Original	79.6	82.3	77.1	85.4	87.6	82.6

Table 6.8: Comparison with the original PaddleOCR model in detection results.

6.6.1 FPS and size comparison

The FPS comparison is shown in table 6.7. The images from the total-text dataset are averagely larger, so the processing time is longer. It can be seen in the table that, due to the decrease in the model size, the FPS of our model is higher, meaning a faster model processing speed.

6.6.2 Detection result comparison

The comparison of the detection result is shown in table 6.8. For images with rectangular texts, the F-score of our model is 60.5%, showing a 19.1% f-score loss compared to the original model. For images with curve texts, the f-score is 72.7 % and the

6.6.3 End-to-end result comparison

The comparison of the end-to-end result is shown in table 6.9. For rectangular texts, when no lexicon is used, the f-score is 34.5%, showing a 12.2% loss to the original model's f-score. Using lexicons, the f-score showed 16.1% and 16.5% loss respectively. For curved texts, the f-score is 45.3%, showing a 13.9% loss to the original value.

	Rectangular texts					Curved texts		
	F-score	Precision	Recall	W	G	F-score	Precision	Recall
Our	34.5	38.1	31.4	48.5	43.6	45.3	47.3	43.6
Original	46.7	47.6	45.9	64.6	60.1	59.2	61.2	57.4

Table 6.9: Comparison with the original PaddleOCR model in end-to-end results. The W and G represent using weak and generic lexicons.

7 Android application

7.1 General

This project also prepares an Android demo for the model to show its performance on portable devices. While the original code for training and pre/post-processing is in Python, the android demo utilizes the nb model(introduced in chapter 5), and the code is written in C++ and Java.

7.2 Model conversion

The nb model is converted from the inference model using PaddleLite. Different PaddleLite versions are tried for the project. Based on the precision of the TCL map by comparing to the TCL map produced by the TCL, the PaddleLite 2.9.1c is selected for the conversion.

7.3 original demo

PaddleOCR has officially released an Android demo for its PPOCR-v2 model, known for its lightweight two-stage architecture. The demo features an interface and scripts for function calls implemented in Java files, while the code for preprocessing, model inference, and postprocessing is written in C++ files.

7.4 Code debugging method

The debugging process for the Android demo necessitates either a physical mobile device or an emulator (virtual machine) within Android Studio. This means that reinstalling the app on the phone is required for each debugging or running session. It's important to note that while Android Studio claims to support emulators with both x86 and arm-64 frameworks, in practice, only the x86 framework is compatible. As PaddlePaddle exclusively supports the arm-64 framework, debugging is limited to physical devices, adding complexity to the process.

While Android Studio's logcat function can be employed to output information during the running process, it is limited to printing short text files and is not conducive for retrieving results like vectors or matrices from the C++ code.

Given that the core functionality of the demo resides in the C++ functions, the codes are rigorously tested in Visual Studio. Additionally, as some of the functions wrapped up in Python lack corresponding counterparts in C++, they are written and tested separately to ensure seamless integration.

7.5 Visual Studio implementation details

7.5.1 Function in getting instance center line

To test the performance of the nb model, the model is run in Android Studio, and the TCL map is "outputted", as seen in (a) in Figure 7.1. As stated in the previous section, Android Studio can only output text files for debugging, so the locations of the pixels with TCL value higher than 0.5 are outputted and reformatted to be (a) in figure 7.1. Because of the output problem of Android Studio itself, the below pixels are mixed in (a), but in actual inference, they exist, and (a) is only used for testing in Visual Studio. Since the thinning functions used in Python do not have a corresponding one in C++, the thinningIteration function from OpenCV is used. Since the Paddlepaddle framework does not provide support for extra part in the OpenCV library where the `thinningIteration` function belongs, the source codes are directly added to the Android demo, and some changes to the data type are made to make the input and output both in the format of CV_8UC1, this aims at saving spaces for memory and prevent formatting problems.

The thinningIteration supports two thinning types: the Zhang-Suen Thinning algorithm and the Guohall thinning algorithm [26]. For this project, the skeletonize step aims to preserve the frame of the original TCC map with a width of one pixel. After testing the two algorithms, the Guohall thinning algorithm behaves better performance, and after 4 iterations of applying the algorithm, the frame with one-pixel width is achieved. The iterations are shown in figure7.1 (b) to (e).

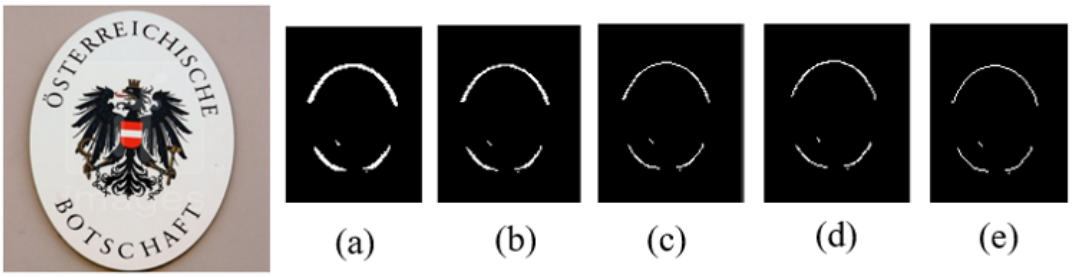


Figure 7.1: The skeletonize process

7.5.2 Function in determining corresponding characters

For character determination, the Python code utilizes the `np_where` and `np_argsort` functions. The former is employed to identify the locations of instances within the output map generated by the connected components function, while the latter is used to arrange the sequence of pixels in instances based on information from the TDO map. Since there are no direct equivalents in C++, these functions are implemented with the same names in Visual Studio.

In cases where an instance has an extended length and needs to be divided into halves and sorted according to the algorithm outlined in Chapter 4, the operations required in C++ are more intricate than in Python. Therefore, a function named `np_argsort_again` is introduced in C++. This function handles the division of the instance, recursively calls `np_argsort` to sort the two halves, and ultimately combines them.

Furthermore, the output of the `np_where` function is a 2-dimensional vector with a shape of $2 \times n$, where n represents the number of pixels. To facilitate subsequent operations, such as obtaining the coordinates of a point, another function named `transpose` is developed to transpose the vector. This function is called within the `np_where` function, resulting in a final output shape of $n \times 2$.

The Python function `join`, responsible for merging identical character instances and eliminating background instances, lacks a direct counterpart in C++. To address this, it has been integrated into the `processLabels` function. This function takes the rough character instance vector as input and produces the combined character instance vector as output.

7.6 C++ implementation

The C++ implementation mainly follows the algorithm described in Chapter 4. Some functions are tested in Visual Studio, and the details are described in the previous section. In the Android demo, the C++ part is mostly implemented in the file `ocr_ppredictor.cpp` and `native.cpp`.

The function for running is in the file `ocr_ppredictor.cpp`, namely `infer_ocr()`. It first calls the preprocess, then the inference part, then calls the post-process part to get the result in the form of a 1-dimensional vector of `OCRPredictResult`. The variable is further converted into the form that Java can process.

7.6.1 Pre-process

The Pre-process follows the same schedule as described in the algorithm chapter, doing resize and normalization to the image.

7.6.2 Inference

After the preprocess, the nb model is run with the function `infer()` from Paddlepaddle, and the result is a list of the references to the 4 feature maps. The pointer to the float data in the maps are got from the function `get_float_data()` applying on the references. To copy the data from the original map to `cv::Mat` form variable in OpenCV, the function `memcpy` is applied instead of assigning element by element.

7.6.3 Post-process

As introduced in Chapter 4, the postprocess step can be roughly divided into the generate text step and the generate bounding box step.

7.6.3.1 Generating text

After generating the TCL map in `cv::Mat` form, the matrix is binarized with a threshold of 0.5 using the function `cv::threshold`. The TCL map is skeletonized using the `thinningIteration` function. After the skeleton is calculated, the number and map of the center lines are determined using the function `connectedcomponents` from OpenCV, where the data type of input and output data type are both `CV_8UC1`. The text locations are determined using function `np_where`.

For each text instance with a length of more than 2, it is considered as a true instance. Otherwise, it will be skipped. For a true instance, the pixels' corresponding values in the TDO map will be generated in the form of `std::vector<std::vector<float>>`. Using the information from TCL and TDO, the sequence of the points is sorted utilizing the function `np_argsort`. For the instances with a length higher than 16, the function `np_argsort_again` is applied for better sequence.

Since the sequence of floats in the TCC map is in the sequence of $h \times w \times 37$, and switching dimensions for such a large map is very costly, the largest probability value of the pixel is calculated by comparing the values directly from the corresponding locations from the pointer of the TCC float value. After the character vector is determined, the combined character vector can be calculated using function `processLabels`. For the Paddlepaddle framework for Android, the contents in the vector are added 1 to match the dictionary sequence of Paddlepaddle.

7.6.3.2 Generating bounding box

As stated in chapter 4, the bounding box is generated from the center line and the TBO map. For the center line gained in the previous step, it is first processed with the function `keepLeftRightAndMiddle`, which is implemented to keep the

points at two sides of the center line and another 5 points in the middle, and the output is a vector with 7 points selected from the center line. After that, the offset is added to the points to get the points from the bounding boxes. However, the output TBO map from the nb is not as good as the one from the inference model, reflecting in two aspects: 1. some of the offsets are opposite to the correct ones; 2. most of the offsets appear to be smaller than the correct one2. To fix the above problems, for each point in the center line, the y coordinate of points in the border point pair is compared, and the one with a smaller y coordinate is kept as the upper border point. In addition, if the distance between the upper and lower point is smaller than a specific value, their distance will enlarged.

After getting the border point pair for each point in the center line, they are rearranged to the clockwise sequence, and the point list is pushed into the `OCRPredictResult.point` type variable for the final output.

7.7 Java implementation

The Java codes mainly aim for the transmission of data, building the interface, and displaying the output of the OCR.

7.7.1 Transmission of data

The Java code employs the JNI method to call the C++ function, as depicted in Figure 7.2. The returned values from C++ functions can be transferred back by calling this method from Java codes. Within the PaddleOCR android demo, the `runImage` function in the file `OCRPredictorNative.java` invokes the `forward` function, which is implemented in the file `native.cpp`. This function orchestrates the preprocessing, model execution, and postprocessing steps, which are called from other C++ files. The result is returned in the form of `std::vector::ppredictor::OCRPredictResult`. In the `native.cpp` file, this returned value is serialized into a float array, transmitted to the Java layer, and subsequently deserialized.

7.7.2 Interface building

The interface is constructed in the file `activity_main.xml` in the Android demo. This file specifies the placement of buttons and determines which commands to call through the functions. We incorporated an 'E2E' button in the mode list within this file. When selected, it triggers the functions to execute the end-to-end OCR model.

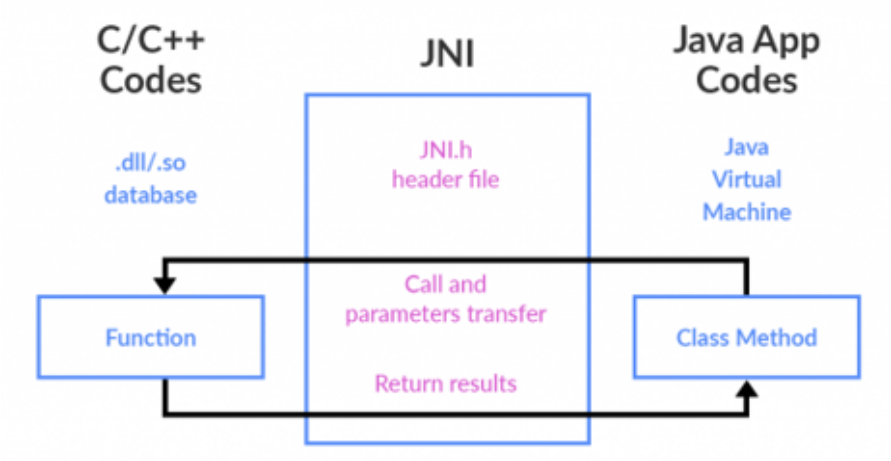


Figure 7.2: JNI

7.7.3 Displaying the output

The function for drawing results is defined in the file `Predictor.java`, where a `Canvas` variable is first created for the input image, and the bounding boxes are drawn on it by calling the function `Canvas.drawPath`.

7.8 Performance test

The fine-tuned mode for the curved-texts is used due to its higher precision. The performance is tested on multiple curved texts, and the performance and the inference time are compared with the original end-to-end model.

7.8.1 Hardware

The performance of the demo is tested on a Huawei P40 mobile device with a CPU of Octa-core (2x2.86 GHz Cortex-A76 & 2x2.36 GHz Cortex-A76 & 4x1.95 GHz Cortex-A55).

7.8.2 End-to-end performance

Figure 7.3 showcases diverse outcomes obtained through the execution of our end-to-end OCR model. Take instance (a) for illustration: Upon selecting the `SELECT` button, users can designate an image from the repository for model application. Activation of the `RUN` button initiates the model, presenting bounding boxes superimposed on the image. Below the visual representation are the coordinates of bounding box points for each text instance, concomitant with their corresponding recognition outcomes and the computational time taken for inference. Should the

need arise to reset the results for the same image, clicking the `CLEAR` button will reestablish the model's initial state, enabling further execution.

7.8.3 Comparison to the original model

A straightforward comparison was conducted with the original PPOCR-v2 model, as illustrated in Figure 7.4. (a) displays the result of the original model, while (b) showcases our model's output on the same image. The PPOCR-v2 model, being 9.8MB and smaller in size, boasts a shorter inference time. However, our model demonstrates superior performance for curved texts by accurately detecting bounding boxes and recognizing words. In contrast, the PPOCR-v2 model can only detect isolated characters.

7.8.4 Inference time and size

The inference time was evaluated using 50 images from the Total-text dataset. The model was run ten times for each image to obtain an average. The inference time for our nb model is 433ms, whereas the original nb model requires 925ms. In terms of size, our end-to-end nb OCR model is 19MB, significantly smaller than the original model, which is 198MB.

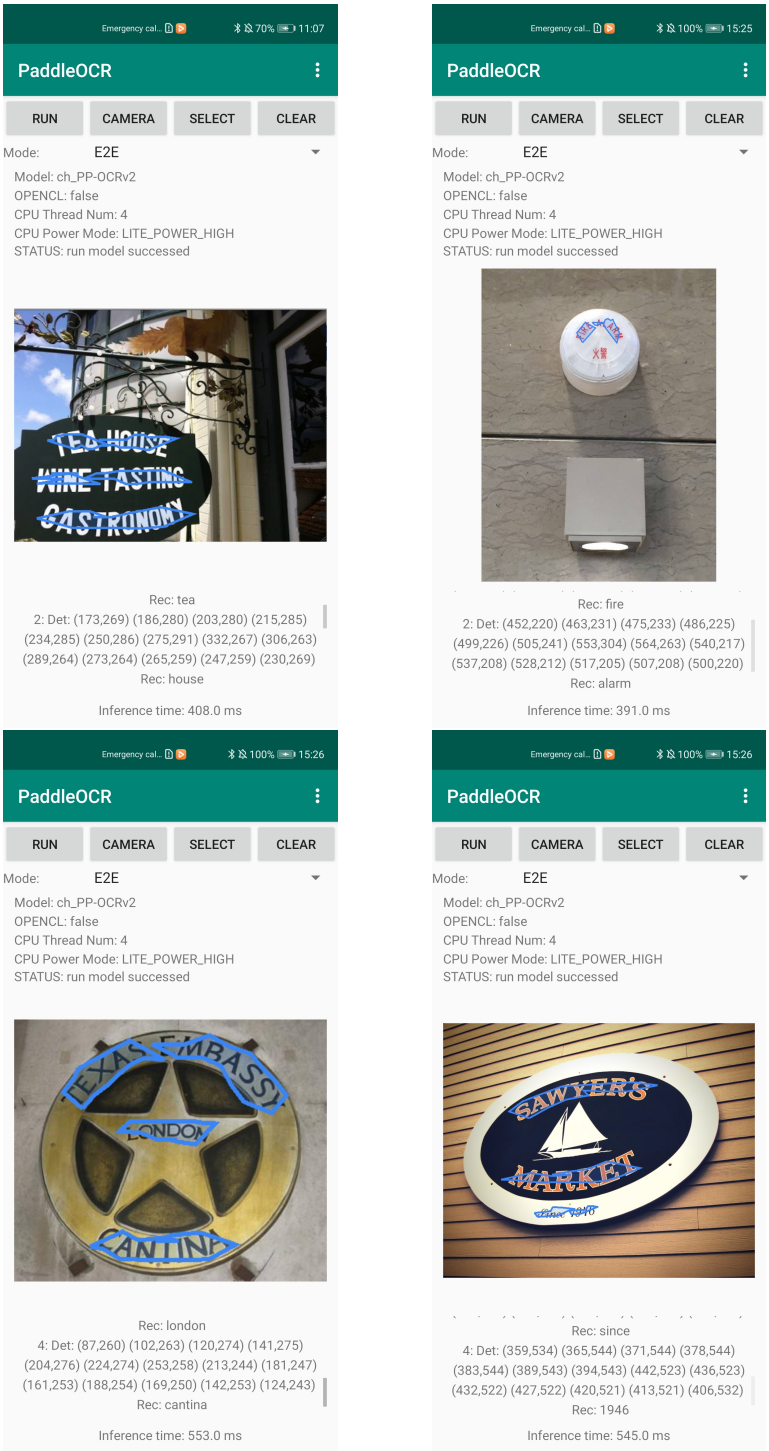
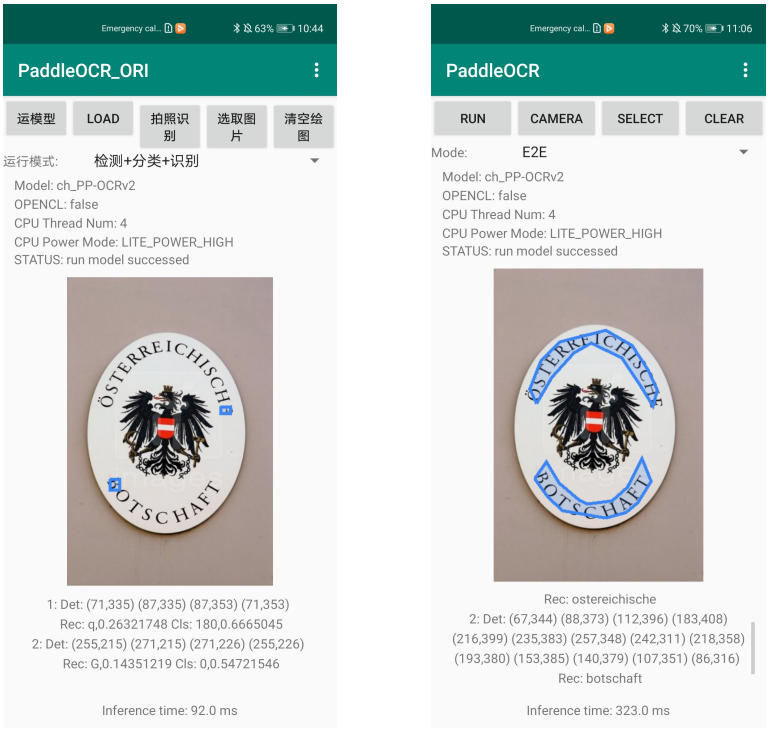


Figure 7.3: Example results of the android demo of the end-to-end OCR model.



(a) OCR original model (b) End-to-end model

Figure 7.4: Comparison between the demo performance and the original one.

8 Conclusion

8.1 Conclusion

In this study, we focused on the development of compact end-to-end OCR models capable of offline operation on mobile devices. By substituting the stem network of the original PaddleOCR end-to-end model with an optimized MobileNetV3 network and introducing a new FPN structure, we successfully created an end-to-end OCR model with a reduced size of 19MB, representing only a tenth of the original model's size. However, this reduction in model size inevitably led to a decrease in performance.

Through extensive training on diverse datasets and conducting fine-tuning experiments, we achieved a 14% reduction in end-to-end precision for curved texts and an 11% reduction for rectangular texts. Additionally, for both rectangular and curved texts, the model's frames per second (FPS) increased by a factor of 10. This level of performance is comparable to some of the previous end-to-end models that utilized larger networks like ResNet50 and VGG19.

Furthermore, we developed an Android demo to showcase the model's performance on mobile devices, highlighting its proficiency in handling OCR tasks for curved texts and its rapid processing speed.

8.2 Further insight

From the perspective of model thresholding, using smaller models may lead to information loss compared to employing larger networks like ResNet50. To mitigate this, maximizing the utilization of all features extracted by the network is crucial. It becomes evident when comparing different Feature Pyramid Network (FPN) structures that the depth of the FPN structure significantly influences the result, despite appearing as a mere 'connection' between the backbone and the head part in the full structure. Additionally, adding layers to the original model can extract even more features.

Turning to the OCR model, despite extensive research in the field, OCR technology still has room for improvement. As most current models are two-stage models, their application is primarily limited to instances with rectangular shapes. Furthermore, recognizing text instances in multiple languages poses a significant challenge. Moreover, for both online and offline models running on mobile devices, speed remains a concern. For real-time OCR tasks, there is currently no model

capable of running on mobile platforms in real-time while maintaining satisfactory accuracy. Hence, additional techniques, such as recognizing similarities in images, need to be incorporated to meet the requirements.

In addition to general OCR models like the Tesseract OCR model used in Google Translator, there are specialized OCR models designed for specific verticals, such as medical text recognition or formula recognition. In such cases, creating more specific datasets for training the model can lead to improved accuracy.

Speaking of datasets, although OCR is a widely discussed topic, sourcing and formatting datasets for this project proved to be a complex task. I experimented with labeling tools like PaddleLabel and SynthText to generate datasets, but the results were not of sufficient quality to use, especially for images with curved-text instances. Therefore, creating flexible and user-friendly labeling tools remains an area with potential for improvement in OCR. In a broader context, achieving clean and accurate datasets remains an open challenge in the field of machine learning.

On a larger scale, given that machine learning is often likened to a black box, there are many aspects of models that remain unclear. Consequently, there is a need for further exploration of methods for interpreting models. While the model I designed is significantly smaller than the original model and may have a potential performance trade-off, it's important to acknowledge that we still have limited understanding of the inner parameters and the potential for further fine-tuning. Additionally, when evaluating parameters, attention should be directed not only towards achieving higher evaluation benchmarks, but also towards considering how the statistics are distributed in the dataset. In specific terms, we should seek ways to qualitatively describe the dataset, which can ultimately enhance our comprehension of the model.

Bibliography

[1] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[2] "Google translator." https://translate.google.com/. Accessed: August 5, 2023.

[3] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," 2017.

[4] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size," 2016.

[5] Y. Du, C. Li, R. Guo, X. Yin, W. Liu, J. Zhou, Y. Bai, Z. Yu, Y. Yang, Q. Dang, and H. Wang, "Pp-ocr: A practical ultra lightweight ocr system," 2020.

[6] M. Liao, Z. Wan, C. Yao, K. Chen, and X. Bai, "Real-time scene text detection with differentiable binarization," 2019.

[7] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang, "East: An efficient and accurate scene text detector," 2017.

[8] P. Wang, C. Zhang, F. Qi, Z. Huang, M. En, J. Han, J. Liu, E. Ding, and G. Shi, "A single-shot arbitrarily-shaped text detector based on context attended multi-task learning," in *Proceedings of the 27th ACM International Conference on Multimedia*, ACM, oct 2019.

[9] J. Baek, G. Kim, J. Lee, S. Park, D. Han, S. Yun, S. J. Oh, and H. Lee, "What is wrong with scene text recognition model comparisons? dataset and model analysis," 2019.

[10] B. Shi, X. Bai, and C. Yao, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," 2015.

[11] F. Borisyuk, A. Gordo, and V. Sivakumar, "Rosetta," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &amp Data Mining*, ACM, jul 2018.

[12] M. Bušta, L. Neumann, and J. Matas, "Deep textspotter: An end-to-end trainable scene text localization and recognition framework," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2223–2231, 2017.

[13] Y. Liu, H. Chen, C. Shen, T. He, L. Jin, and L. Wang, "Abcnet: Real-time scene text spotting with adaptive bezier-curve network," 2020.

[14] W. Feng, W. He, F. Yin, X.-Y. Zhang, and C.-L. Liu, "Textdragon: An end-to-end framework for arbitrary shaped text spotting," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9075–9084, 2019.

[15] L. Xing, Z. Tian, W. Huang, and M. Scott, "Convolutional character networks," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9125–9135, 2019.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[17] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2017.

[18] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," 2019.

[19] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," 2017.

[20] A. Gupta, A. Vedaldi, and A. Zisserman, "Synthetic data for text localisation in natural images," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[21] X. Zhou, S. Zhou, C. Yao, Z. Cao, and Q. Yin, "Icdar 2015 text reading in the wild competition," 2015.

[22] R. Gomez, B. Shi, L. Gomez, L. Numann, A. Veit, J. Matas, S. Belongie, and D. Karatzas, "Icdar2017 robust reading challenge on coco-text," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 01, pp. 1435–1443, 2017.

[23] C. K. Ch'ng, C. S. Chan, and C. Liu, "Total-text: Towards orientation robustness in scene text detection," *IJDAR*, 2019.

[24] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Freezeout: Accelerate training by progressively freezing layers," 2017.

[25] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-net: Fully convolutional neural networks for volumetric medical image segmentation," in *2016 Fourth International Conference on 3D Vision (3DV)*, pp. 565–571, 2016.

[26] W. Chen, L. Sui, Z. Xu, and Y. Lang, "Improved zhang-suen thinning algorithm in binary line drawing applications," in *2012 International Conference on Systems and Informatics (ICSAI2012)*, pp. 1947–1950, 2012.