主标题：如何一键完成信息自动提取与填写？（2）

副标题：用 OpenVINO™ 工具包轻松实现 PaddleOCR 实时推理

作者：武卓，英特尔 AI 软件布道师

简介：

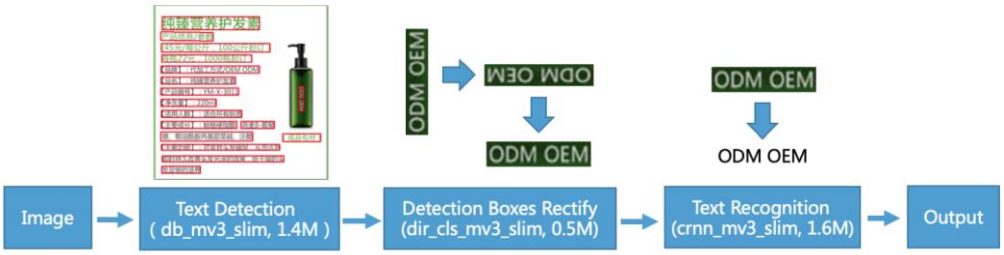
在上篇文章中我们介绍过，光学字符识别（OCR）技术可以将文件、图片或自然场景中的文字信息进行识别并提取，与一系列的自然语言处理技术联合使用，能够完成诸如文档票据的文字信息自动化处理、实时图片文字翻译等任务。通过机器的自动化处理，可以帮助财务人员在处理票据时省却大量手工输入的工作量，也能够方便我们在出国旅游时随时对异域中的外国文字信息进行实时翻译、减少语言不通带来的不便。

既然 OCR 技术如此实用，有没有什么方法能让我们利用自己手边的设备，随时使用到这项技术呢？答案当然是肯定的。接下来，我们将以百度开源的 PaddleOCR【1-2】技术为例，具体介绍如何利用英特尔开源的 OpenVINO™ 工具套件，仅使用我们手边都有的 CPU 就能轻松实现对 PaddleOCR 的实时推理。

本篇是用 OpenVINO™ 实现基于 OCR 及 NLP 轻松实现信息自动化提取的系列博客中的第二篇。我们将简要介绍 PaddleOCR 的原理，以及利用 OpenVINO™ 实现 PaddleOCR 推理加速的工作流程。同样只需利用一页 Jupyter notebook，依照简单的三个步骤，即可利用 CPU 实现基于 PaddleOCR 的实时文字信息提取。

PaddleOCR 原理简介

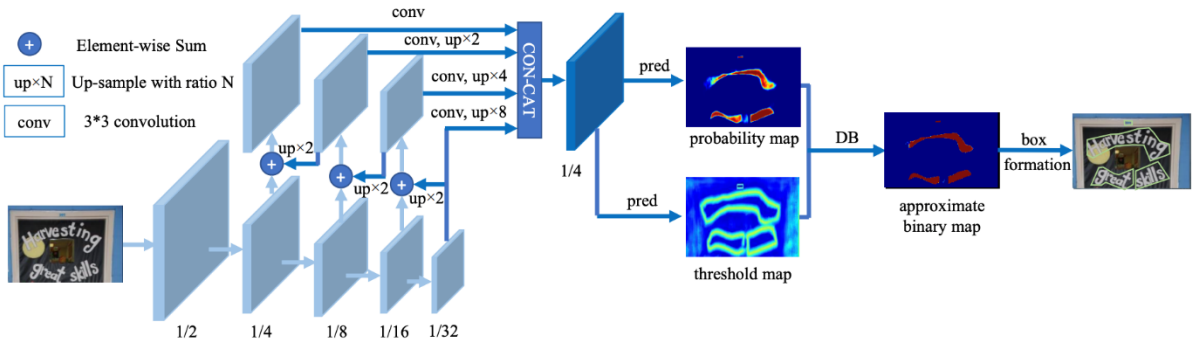
PaddleOCR 是基于深度学习框架 PaddlePaddle 的一项 OCR 技术，具有超轻、模型小、便于移动端及服务器端部署等特点。整个 PaddleOCR 技术的工作流程如下图所示，主要包括文本检测、方向分类、以及文本识别三部分。



文本检测任务是找出图像或视频中的文字位置。不同于目标检测任务，目标检测不仅要解决定位问题，还要解决目标分类问题。但是，文本检测也面临一些难点，比如：自然场景中的文本具有多样性，文字大小、方向、长度、形状、语言都会有不同。有的时候，文字重叠或者密度较高，

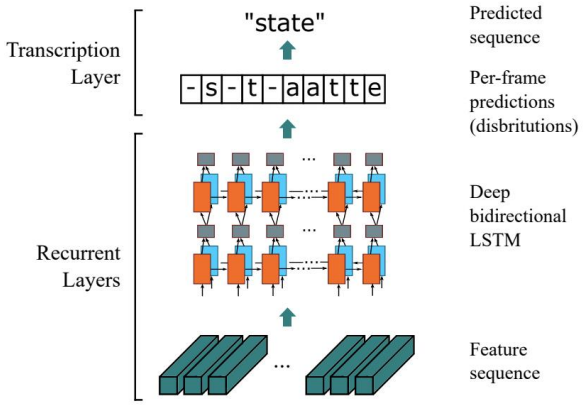
这些都会影响最终文本检测的效果。目前常用的文本检测方法有基于回归以及基于分割的方法。而在 PaddleOCR 中，我们选取的是基于分割的 DBNet【3】方法。

DBNet 的工作原理如下图所示。针对基于分割的方法需要使用阈值进行二值化处理而导致后处理耗时的问题，DBNet 提出了一种可学习阈值的 方法，并巧妙地设计了一个近似于阶跃函数的二值化函数，使得分割网络在训练的时候能端对端的学习文本分割的阈值。自动调节阈值不仅带来精度的提升，同时简化了后处理，提高了文本检测的性能。



方向分类指的是针对图片中某些经文本检测得到的 bounding box 中的文字方向为非水平排列的情况，对 bounding box 的方向进行检测。如果发现 bounding box 中的文字方向为非水平排列，则对该 bounding box 的方向进行纠正，使其旋转为文字水平排列的方向，方便下一步的文本识别。

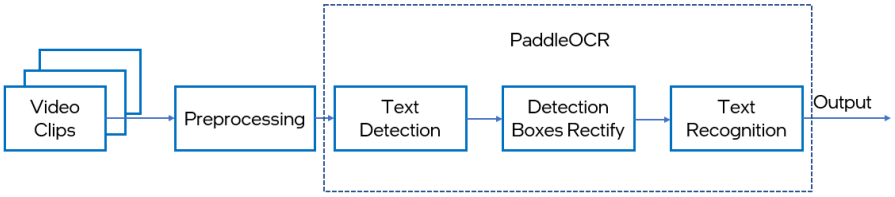
文本识别的任务是将文本检测得到的 bounding box 中的具体的文字内容识别出来。文本识别的算法有针对规则文本以及不规则文本识别的算法。对于规则文本，主流的算法有 CTC(Conectionist Temporal Classification) 和基于 Sequence2Sequence 的方法。在本文 demo 中，我们采用的是基于 CTC 的方法。由于文本识别任务的特殊性，输入数据中存在大量的上下文信息，卷积神经网络的卷积核特性使其更关注于局部信息，缺乏长依赖的建模能力，因此仅使用 CNN 很难挖掘到文本之间的上下文联系。为了解决这一问题，首先通过使用 CRNN (Convolutional Recurrent Neural Network)【4】，利用卷积网络提取图像特征，并同时引入了双向 LSTM(Long Short-Term Memory) 用来增强上下文建模。最终将输出的特征序列输入到 CTC 模块， 通过 ctc 归纳字符间的连接特性，直接解码序列结果。该结构被验证有效，并广泛应用在文本识别任务中， 如下图所示。



5 分钟 3 步骤快速实现 PaddleOCR 实时推理

在最新版本的 OpenVINO™ 2022.1 中，已经实现了对基于 PaddlePaddle 深度学习框架的深度学习模型的支持。而 PaddleOCR 作为一项深受广大开发者喜爱的开源技术，其中开源的预训练模型已经可以在 OpenVINO™ 2022.1 版本中直接进行模型读取以及加速推理。

接下来，我们将通过代码示例，介绍如何按照简单的三个步骤，实现 OpenVINO™ 对 PaddleOCR 的加速推理。整个工作流程如下图所示：



其中 OpenVINO™ 会对 PaddleOCR 中的文本检测以及文本识别模型进行读取以及推理加速。本次 demo 中我们展示的是利用自己的网络摄像头，将实时获取的视频流中的文字信息利用 PaddleOCR 进行提取。当然，开发者也可以上传图片，利用 OpenVINO™ 对 PaddleOCR 的推理实现对图片中的文字信息进行提取。

- **步骤一：下载需要使用的 PaddleOCR 预训练模型，并完成模型的读取与加载**
  在导入需要使用到的相应 Python 包后，首先需要对将要使用的 PaddleOCR 开源预训练模型进行下载。本次 demo 中使用到的是轻量化的"Chinese and English ultra-lightweight PP-OCR model (9.4M)"模型。由于 PaddleOCR 中包含了文本检测及文本识别两个深度学习模型，因此，我们首先定义一个模型下载函数，如下图所示。

```
# Define the function to download text detection and recognition models from PaddleOCR resources

def run_model_download(model_url, model_file_path):
    """
    Download pre-trained models from PaddleOCR resources

    Parameters:
        model_url: url link to pre-trained models
        model_file_path: file path to store the downloaded model
    """
    model_name = model_url.split("/")[-1]

    if model_file_path.is_file():
        print("Model already exists")
    else:
        # Download the model from the server, and untar it.
        print("Downloading the pre-trained model... May take a while...")

        # create a directory
        os.makedirs("model", exist_ok=True)
        urllib.request.urlretrieve(model_url, f"model/{model_name} ")
        print("Model Downloaded")

        file = tarfile.open(f"model/{model_name} ")
        res = file.extractall("model")
        file.close()
        if not res:
            print(f"Model Extracted to {model_file_path}.")
        else:
            print("Error Extracting the model. Please check the network.")
```

接下来，完成文本检测模型的下载，

Download the Model for Text Detection

```
# Directory where model will be downloaded

det_model_url = "https://paddleocr.bj.bcebos.com/dygraph_v2.0/ch/ch_ppocr_mobile_v2.0_det_infer.tar"
det_model_file_path = Path("model/ch_ppocr_mobile_v2.0_det_infer/inference.pdmodel")

run_model_download(det_model_url, det_model_file_path)
```

以及推理引擎的初始化、文本检测模型的读取以及在 CPU 上面的加载。

Load the Model for Text Detection

```
# initialize inference engine for text detection
core = Core()
det_model = core.read_model(model=det_model_file_path)
det_compiled_model = core.compile_model(model=det_model, device_name="CPU")

# get input and output nodes for text detection
det_input_layer = det_compiled_model.input(0)
det_output_layer = det_compiled_model.output(0)
```

再然后，完成文本识别模型的下载，

Download the Model for Text **Recognition**

```
rec_model_url = "https://paddleocr.bj.bcebos.com/dygraph_v2.0/ch/ch_ppocr_mobile_v2.0_rec_infer.tar"
rec_model_file_path = Path("model/ch_ppocr_mobile_v2.0_rec_infer/inference.pdmodel")

run_model_download(rec_model_url, rec_model_file_path)
```

以及文本识别模型的读取以及在 CPU 上面的加载。其中，有一步需要特别说明的是，**动态输入的处理**。由于文本识别模型的输入是文本检测得到的一系列 bounding box 图像，而图像中的字体由于大小和文字长短程度不一，就造成了文本识别模型的输入是动态输入的。与以往版本需要对图像尺寸进行重调整（resize）而将模型输入尺寸固定、从而可能引起性能损失的处理方法不同的是，OpenVINO™ 2022. 1 版本已经可以很好的支持模型的动态输入。在 CPU 上进行文本识别模型加载之前，只需要对于输入的若干维度中具有动态输入的维度赋值-1 或申明动态输入尺寸的上限值，比如 Dimension（1，512），即可完成对模型动态输入的处理。接下来，即可按常规步骤完成在 CPU 上加载文本识别模型。

```
# read the model and corresponding weights from file
rec_model = core.read_model(model=rec_model_file_path)

# assign dynamic shapes to every input layer on the last dimension
for input_layer in rec_model.inputs:
    input_shape = input_layer.partial_shape
    input_shape[3] = -1
    rec_model.reshape({input_layer: input_shape})

rec_compiled_model = core.compile_model(model=rec_model, device_name="CPU")

# get input and output nodes
rec_input_layer = rec_compiled_model.input(0)
rec_output_layer = rec_compiled_model.output(0)
```

- **步骤二：为文本检测及文本识别定义必要的前处理及后处理函数。**

  为文本检测模型定义必要的前处理函数，如下图所示

```
# Preprocess for text detection
def image_preprocess(input_image, size):
    """
    Preprocess input image for text detection

    Parameters:
        input_image: input image
        size: value for the image to be resized for text detection model
    """
    img = cv2.resize(input_image, (size, size))
    img = np.transpose(img, [2, 0, 1]) / 255
    img = np.expand_dims(img, 0)
    # NormalizeImage: {mean: [0.485, 0.456, 0.406], std: [0.229, 0.224, 0.225], is_scale: True}
    img_mean = np.array([0.485, 0.456, 0.406]).reshape((3, 1, 1))
    img_std = np.array([0.229, 0.224, 0.225]).reshape((3, 1, 1))
    img -= img_mean
    img /= img_std
    return img.astype(np.float32)
```

为文本识别模型定义必要的前处理函数，如下图所示

```python
# Preprocess for text recognition
def resize_norm_img(img, max_wh_ratio):
    """
    Resize input image for text recognition

    Parameters:
        img: bounding box image from text detection
        max_wh_ratio: value for the resizing for text recognition model
    """
    rec_image_shape = [3, 32, 320]
    imgC, imgH, imgW = rec_image_shape
    assert imgC == img.shape[2]
    character_type = "ch"
    if character_type == "ch":
        imgW = int((32 * max_wh_ratio))
    h, w = img.shape[:2]
    ratio = w / float(h)
    if math.ceil(imgH * ratio) > imgW:
        resized_w = imgW
    else:
        resized_w = int(math.ceil(imgH * ratio))
    resized_image = cv2.resize(img, (resized_w, imgH))
    resized_image = resized_image.astype('float32')
    resized_image = resized_image.transpose((2, 0, 1)) / 255
    resized_image -= 0.5
    resized_image /= 0.5
    padding_im = np.zeros((imgC, imgH, imgW), dtype=np.float32)
    padding_im[:, :, 0:resized_w] = resized_image
    return padding_im
```

```python
def batch_text_box(dt_boxes, frame):
    """
    Batch the detected bounding boxes for text recognition

    Parameters:
        dt_boxes: detected bounding boxes from text detection
        frame: original input frame
    """

    ori_im = frame.copy()
    img_crop_list = []
    for bno in range(len(dt_boxes)):
        tmp_box = copy.deepcopy(dt_boxes[bno])
        img_crop = processing.get_rotate_crop_image(ori_im, tmp_box)
        img_crop_list.append(img_crop)

    img_num = len(img_crop_list)
    # Calculate the aspect ratio of all text bars
    width_list = []
    for img in img_crop_list:
        width_list.append(img.shape[1] / float(img.shape[0]))
    # Sorting can speed up the recognition process
    indices = np.argsort(np.array(width_list))
    rec_res = [['', 0.0]] * img_num
    batch_num = 6
```

```
    # For each detected text box batch, run inference for text recognition
    for beg_img_no in range(0, img_num, batch_num):
        end_img_no = min(img_num, beg_img_no + batch_num)

        norm_img_batch = []
        max_wh_ratio = 0
        for ino in range(beg_img_no, end_img_no):
            h, w = img_crop_list[indices[ino]].shape[0:2]
            wh_ratio = w * 1.0 / h
            max_wh_ratio = max(max_wh_ratio, wh_ratio)
        for ino in range(beg_img_no, end_img_no):
            norm_img = resize_norm_img(img_crop_list[indices[ino]], max_wh_ratio)
            norm_img = norm_img[np.newaxis, :]
            norm_img_batch.append(norm_img)

    norm_img_batch = np.concatenate(norm_img_batch)
    norm_img_batch = norm_img_batch.copy()
    return norm_img_batch, rec_res, indices, beg_img_no
```

为文本检测模型定义后处理函数，将文本检测模型的推理结果转为 bounding box 形式，作为文本识别模型的输入，如下图所示。

```
def post_processing_detection(frame, det_results):
    """
    Postprocess the results from text detection into bounding boxes

    Parameters:
        frame: input image
        det_results: inference results from text detection model
    """
    ori_im = frame.copy()
    data = {'image': frame}
    data_resize = processing.DetResizeForTest(data)
    data_list = []
    keep_keys = ['image', 'shape']
    for key in keep_keys:
        data_list.append(data_resize[key])
    img, shape_list = data_list

    shape_list = np.expand_dims(shape_list, axis=0)
    pred = det_results[0]
    if isinstance(pred, paddle.Tensor):
        pred = pred.numpy()
    segmentation = pred > 0.3

    boxes_batch = []
    for batch_index in range(pred.shape[0]):
        src_h, src_w, ratio_h, ratio_w = shape_list[batch_index]
        mask = segmentation[batch_index]
        boxes, scores = processing.boxes_from_bitmap(pred[batch_index], mask, src_w, src_h)
        boxes_batch.append({'points': boxes})
    post_result = boxes_batch
    dt_boxes = post_result[0]['points']
    dt_boxes = processing.filter_tag_det_res(dt_boxes, ori_im.shape)
    return dt_boxes
```

- **步骤三：利用 OpenVINO™ 推理引擎（Runtime）针对摄像头采集视频进行实时推理**

```
def run_paddle_ocr(source=0, flip=False, use_popup=False, skip_first_frames=0):
    """
    Main function to run the paddleOCR inference:
    1. Create a video player to play with target fps (utils.VideoPlayer).
    2. Prepare a set of frames for text detection and recognition.
    3. Run AI inference for both text detection and recognition.
    4. Visualize the results.

    Parameters:
        source: the webcam number to feed the video stream with primary webcam set to "0", or the video path.
        flip: to be used by VideoPlayer function for flipping capture image
        use_popup: False for showing encoded frames over this notebook, True for creating a popup window.
        skip_first_frames: Number of frames to skip at the beginning of the video.
    """
```

定义运行 PaddleOCR 模型推理的主函数，主要包括以下四个部分：

1. 运行网络摄像头，将捕捉到的视频流作为 paddleOCR 的输入

```
    # create video player to play with target fps
    player = None
    try:
        player = utils.VideoPlayer(source=source, flip=flip, fps=30, skip_first_frames=skip_first_frames)
        # Start video capturing
        player.start()
        if use_popup:
            title = "Press ESC to Exit"
            cv2.namedWindow(winname=title, flags=cv2.WINDOW_GUI_NORMAL | cv2.WINDOW_AUTOSIZE)
```

2. 准备进行文本检测和文本识别的视频帧

```
        processing_times = collections.deque()
        while True:
            # grab the frame
            frame = player.next()
            if frame is None:
                print("Source ended")
                break
            # if frame larger than full HD, reduce size to improve the performance
            scale = 1280 / max(frame.shape)
            if scale < 1:
                frame = cv2.resize(src=frame, dsize=None, fx=scale, fy=scale,
                                   interpolation=cv2.INTER_AREA)
```

3. 针对文本检测进行推理

```
            # preprocess image for text detection
            test_image = image_preprocess(frame, 640)

            # measure processing time for text detection
            start_time = time.time()
            # perform the inference step
            det_results = det_compiled_model([test_image])[det_output_layer]
            stop_time = time.time()

            # Postprocessing for Paddle Detection
            dt_boxes = post_processing_detection(frame, det_results)

            processing_times.append(stop_time - start_time)
            # use processing times from last 200 frames
            if len(processing_times) > 200:
                processing_times.popleft()
            processing_time_det = np.mean(processing_times) * 1000
```

根据文本检测得到的 bounding box，进行文本识别推理

```
            # Preprocess detection results for recognition
            dt_boxes = processing.sorted_boxes(dt_boxes)
            if dt_boxes:
                # Recognition starts from here
                norm_img_batch, rec_res, indices, beg_img_no = batch_text_box(dt_boxes, frame)

                # Run inference for text recognition
                rec_results = rec_compiled_model([norm_img_batch])[rec_output_layer]

                # Postprocessing recognition results
                postprocess_op = processing.build_post_process(processing.postprocess_params)
                rec_result = postprocess_op(rec_results)
                for rno in range(len(rec_result)):
                    rec_res[indices[beg_img_no + rno]] = rec_result[rno]

                # Text recognition results, rec_res, include two parts:
                # txts are the recognized text results, scores are the recognition confidence level
                if rec_res:
                    image = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
                    boxes = dt_boxes
                    txts = [rec_res[i][0] for i in range(len(rec_res))]
                    scores = [rec_res[i][1] for i in range(len(rec_res))]

                    # draw text recognition results beside the image
                    draw_img = processing.draw_ocr_box_txt(
                        image,
                        boxes,
                        txts,
                        scores,
                        drop_score=0.5)
            else:
                draw_img = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
```

4. 将文本提取的结果可视化

```
            # Visualize PaddleOCR results
            f_height, f_width = draw_img.shape[:2]
            fps = 1000 / processing_time_det
            cv2.putText(img=draw_img, text=f"Inference time: {processing_time_det:.1f}ms ({fps:.1f} FPS)",
                        org=(20, 40),fontFace=cv2.FONT_HERSHEY_COMPLEX, fontScale=f_width / 1000,
                        color=(0, 0, 255), thickness=1, lineType=cv2.LINE_AA)

            # use this workaround if there is flickering
            if use_popup:
                draw_img = cv2.cvtColor(draw_img, cv2.COLOR_RGB2BGR)
                cv2.imshow(winname=title, mat=draw_img)
                key = cv2.waitKey(1)
                # escape = 27
                if key == 27:
                    break
            else:
                # encode numpy array to jpg
                draw_img = cv2.cvtColor(draw_img, cv2.COLOR_RGB2BGR)
                _, encoded_img = cv2.imencode(ext=".jpg", img=draw_img,
                                              params=[cv2.IMWRITE_JPEG_QUALITY, 100])
                # create IPython image
                i = display.Image(data=encoded_img)
                # display the image in this notebook
                display.clear_output(wait=True)
                display.display(i)
```

整个 demo 实现的完整代码请您参考这里，（https://github.com/zhuo-yoyowz/openvino_notebooks/tree/zhuo-notebook-submission/notebooks/405-paddle-ocr-webcam ），**下载使用。**

结果讨论：

下面我们来看看运行结果吧：

我们可以看到，对于网络摄像头采集的视频流中的文字提取效果还是很不错的。仅仅利用 CPU 进行推理，也可以得到 30FPS 以上的性能，可以说能够达到实时的推理效果！当然，除了视频流作为输入，开发者还可以上传图片，进行文本信息提取。以下是针对上传图片中印刷体文字和手写体文字信息提取的一些测试效果。

[('2021 年', 0.9998825),

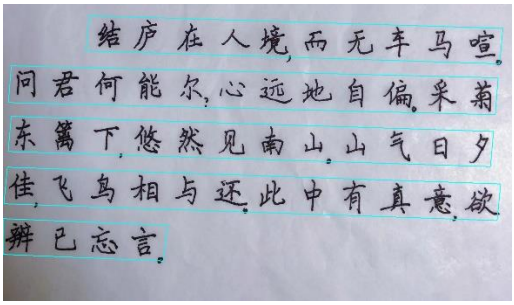
('上海市初中英语', 0.99546415),

('考纲词汇用法手册', 0.9261546),

('配套综合练习', 0.9912483),

('本书编写组', 0.98184997),

('上海译文出版社', 0.99167067)]



[('结庐在人境品无车马喧', 0.7947353),

('问君何能尔心远地自偏采前', 0.78279585),

('东鹤下您然见南山山气日夕', 0.88027453),

('佳飞鸟相与还此中有真意欲', 0.86756784),

('辩已忘言', 0.9859764)]

你还在等什么，快来根据我们提供的源代码，在自己的个人电脑上尝试一下吧！

小结：

OCR 具有将图片、扫描文档或自然场景中的文字信息识别转化为数字化、机器编码方式存储的优势。将 OCR 进行文字识别的结果与自然语言处理中的 NLP 技术相结合，能够实现自动化的信息提取，为我们免去手动输入信息填写的麻烦，并有助于信息的结构化存储与查找。在本次系列博客的第二篇中，我们简要介绍了 PaddleOCR 的工作原理，并提供了一个基于 OpenVINO™ 实现 PaddleOCR 的 Jupyter notebook demo。可以方便读者在阅读的同时，下载源码并在自己的电脑端利用 CPU 来轻松实现 PaddleOCR 的加速推理。

关于英特尔 OpenVINO™ 开源工具套件的详细资料，包括其中我们提供的三百多个经验证并优化的预训练模型的详细资料，请您点击
https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html

除此之外，为了方便大家了解并快速掌握 OpenVINO™ 的使用，我们还提供了一系列开源的 Jupyter notebook demo。运行这些 notebook，就能快速了解在不同场景下如何利用 OpenVINO™ 实现一系列、包括 OCR 在内的、计算机视觉及自然语言处理任务。OpenVINO™ notebook 的资源可以在 Github 这里下载安装：https://github.com/openvinotoolkit/openvino_notebooks 。

参考文献：

【1】 Du Yuning, Chenxia Li, Guo Ruoyu, Xiaoting Yin, Weiwei Liu, Jun Zhou, Yifan Bai, Yu Zilin, Yehua Yang, Qingqing Dang and Haoshuang Wang. PP-OCR: A Practical Ultra Lightweight OCR System. In arXiv: Computer Vision and Pattern Recognition (2020): n. pag.

【2】 Yuning Du, Chenxia Li, Ruoyu Guo, Cheng Cui, Weiwei Liu, Jun Zhou, Bin Lu, Yehua Yang, Qiwen Liu, Xiaoguang Hu, Dianhai Yu and Yanjun Ma. PP-OCRv2: Bag of Tricks for Ultra Lightweight OCR System. In arXiv: Computer Vision and Pattern Recognition (2021): n. pag.

【3】 Minghui Liao, Zhaoyi Wan, Cong Yao, Kai Chen and Xiang Bai. Real-Time Scene Text Detection with Differentiable Binarization. In National Conference on Artificial Intelligence (2020).

【4】 Baoguang Shi, Xiang Bai and Cong Yao. An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition. In IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (2017): 2298-2304.