# CV180x CV181x Yolo Series Algorithm Deployment Guide

Version: 1.0.1

Release date: 2023-07-25

# directory

** Revision record **

| Revision | Date | Description |
|---|---|---|
| 1.0.0 | 2023/7/25 | First Draft |
| 1.0.1 | 2023/10/25 | Markdown to HTML |

# 1 Disclaimer

**Terms and Conditions**

The document and all information contained herein remain the CVITEK Co., Ltd' s ( "CVITEK" ) confidential information, and should not disclose to any third party or use it in any way without CVITEK' s prior written consent. User shall be liable for any damage and loss caused by unauthority use and disclosure.

CVITEK reserves the right to make changes to information contained in this document at any time and without notice.

All information contained herein is provided in "AS IS" basis, without warranties of any kind, expressed or implied, including without limitation mercantability, non-infringement and fitness for a particular purpose. In no event shall CVITEK be liable for any third party' s software provided herein, User shall only seek remedy against such third party. CVITEK especially claims that CVITEK shall have no liable for CVITEK' s work result based on Customer' s specification or published shandard.

**Contact Us**

**Address** Building 1, Yard 9, FengHao East Road, Haidian District, Beijing, 100094, China

Building T10, UpperCoast Park, Huizhanwan, Zhancheng Community, Fuhai Street, Baoan District, Shenzhen, 518100, China

**Phone** +86-10-57590723 +86-10-57590724

**Website** https://www.sophgo.com/

**Forum** https://developer.sophgo.com/forum/index.html

# 2 Functional Overview

## 2.1 Objective

The integrated YOLO series algorithm C++interface provided by the computing end, To shorten the time required for external developers to customize the deployment of YOLO series models.

The TDLSDK internally implements the Yolo series of algorithms to encapsulate its pre and post processing and inference, Provide a unified and convenient programming interface.

Currently, the TDL SDK includes but is not limited to Yolov5, yolov6, yolov7, yolov8, yolox, ppyoloe

# 3 Deployment of YOLOv5 Model for General Use

## 3.1 Introduction

This document introduces the operation process of deploying the YOLOv5 architecture model on the CV181x development board. The main steps include:

- Convert YOLOv5 model Pytorch version to ONNX model
- Convert onnx model to cvi model format
- Finally, write a calling interface to obtain the inference results

## 3.2 Convert pt Model to onnx

1. Firstly, you can download the official Yolov5 warehouse code at the following address: [ultralytics/yolov5: YOLOv5] In PyTorch>ONNX>CoreML>TFLite](https://github.com/ultralytics/yolov5)

```
git clone https://github.com/ultralytics/yolov5.git
```

2. Obtain the YOLOv5 model in. pt format, such as the address for downloading the YOLOv5S model: [yolov5s](https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.pt)

3. It is necessary to modify the forward function in the Detect class in the yolov5/models/yolo .py file, Let the RISC-V do the latter part of YOLOv5 and output nine branches, which will be referred to as the TDLSDK export method in the future

The original output is a result and the post-processing is done by the model, which is the official export result.

The reason is that the output contains a relatively large coordinate range, which is prone to quantization failure or poor performance.

```
def forward(self, x):
      z = []  # inference output
      for i in range(self.nl):
          x[i] = self.m[i](x[i])  # conv
```

(continues on next page)

(continued from previous page)

```
            bs, _, ny, nx = x[i].shape  # x(bs,255,20,20) to x(bs,3,20,20,85)
            x[i] = x[i].view(bs, self.na, self.no, ny, nx).permute(0, 1, 3, 4, 2).
↪contiguous()

            xywh, conf, score = x[i].split((4, 1, self.nc), 4)
            z.append(xywh[0])
            z.append(conf[0])
            z.append(score[0])
        return z
#The output of the modified model is divided into 9 different branches:
# (3, 20, 20, 80) - class
# (3, 20, 20, 4)  - box
# (3, 20, 20, 1)  - conf
# (3, 40, 40, 80) - class
# (3, 40, 40, 4)  - box
# (3, 40, 40, 1)  - conf
# (3, 40, 40, 80) - class
# (3, 40, 40, 4)  - box
# (3, 40, 40, 1)  - conf
```

4.Export the onnx model using the officially provided export.py

```
#Where - weights represents the relative path of the weight file, and - include␣
↪represents the conversion format as onnx
python export.py --weights ./yolov5s.pt --include onnx
#The generated onnx model is in the current directory
```

## 3.3 Preparing the Environment for Model Conversion

Converting onnx to cvi model requires a TPU-MLIR release package. TPU-MLIR is a TPU compiler project that can calculate TDL processors.

TPU-MLIR Toolkit Download TPU-MLIR code path https://github.com/sophgo/tpu-mlir Interested parties can be referred to as open source developers who jointly maintain the open source community. And we only need the corresponding toolkit, which can be downloaded from the TPU-MLIR forum on the official website of Quanneng, later referred to as the toolchain toolkit: (https://developer.sophgo.com/thread/473.html)

The TPU-MLIR project provides a complete toolchain that can transform pre trained neural networks under different frameworks into files that can be efficiently executed on computational TPUs. Currently, it supports direct conversion of onnx and Caffe models, while models from other frameworks need to be converted to onnx models and then converted through the TPU-MLIR tool.

The conversion model needs to be executed in the specified docker, and the main steps can be divided into two steps:

- The first step is to use the model_transform.py converts the original model into an mlir file
- The second step is to use the model_deploy. py converts the mlir file into a cvi model

>If you need to convert to INT8 model, you also need to call run before the second step_Calibration. py generates a calibration table and passes it to the model_Deploy.py

Docker Configuration

TPU-MLIR needs to be developed in the Docker environment, and the Docker image can be directly downloaded (which is relatively slow). Please refer to the following command:

```
docker pull sophgo/tpuc_dev:latest
```

Alternatively, you can download the Docker image from the TPU Toolchain Toolkit (which is faster) and load the Docker.

```
docker load -i  docker_tpuc_dev_v2.2.tar.gz
```

If using Docker for the first time, you can execute the following commands for installation and configuration (only for the first time):

```
sudo apt install docker.io
sudo systemctl start docker
sudo systemctl enable docker
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
```

Enter Docker Environment

Ensure that the installation package is in the current directory, and then create a container in the current directory as follows:

```
docker run --privileged --name myname -v $PWD:/workspace -it sophgo/tpuc_dev:v2.
↪2
```

The following steps assume that the user is currently in the/workspace directory in the Docker

####Loading tpu mlir toolkit&preparing working directory

The following operations need to be performed on the Docker container

[Decompression tpu_mlr toolkit] The following folders are mainly created for the convenience of subsequent management, and you can also classify files according to your preferred management method

Create a new folder tpu_Mlir, extract the new toolchain to tpu_Under the mlir/directory, and set the environment variables:

```
##Among them, tpu mlir_ Xxx in xxx.tar.gz is the version number, determined by␣
↪the corresponding file name
mkdir tpu_mlir && cd tpu_mlir
cp tpu-mlir_xxx.tar.gz ./
tar zxf tpu-mlir_xxx.tar.gz
source tpu_mli_xxx/envsetup.sh
```

[Copy Onnx model] Create a folder, using yolov5s as an example, create a folder yolov5s, and place the onnx model in the yolov5s/onnx/path

```
mkdir yolov5s && cd yolov5s
##Copy the yolov5 onnx model transferred from the previous section to the␣
↪yolov5s directory
cp yolov5s.onnx ./
##Copy the dog.jpg from the official website and come over for verification.
cp dog.jpg ./
```

After the above preparation work is completed, you can start converting the model

## 3.4 Onnx to MLIR

If the model is image input, we need to understand the preprocessing of the model before converting it.

If the model uses preprocessed npz files as input, there is no need to consider preprocessing.

In this example, the image of yolov5 is rgb, with mean and scale corresponding to:

- mean: 0.0, 0.0, 0.0
- scale: 0.0039216, 0.0039216, 0.0039216

The command for model conversion is as follows:

```
model_transform.py \
--model_name yolov5s \
--model_def yolov5s.onnx \
--input_shapes [[1,3,640,640]] \
--mean 0.0,0.0,0.0 \
--scale 0.0039216,0.0039216,0.0039216 \
--keep_aspect_ratio \
--pixel_format rgb \
--test_input ./dog.jpg \
--test_result yolov5s_top_outputs.npz \
--mlir yolov5s.mlir
```

Among them, model_For details of the transform.py parameter, please refer to the [tpu_mlr_xxxxx/doc/TPU-MILIR Quick Start Guide]

After converting to an mlir file, a yolov5s will be generated In_F32.npz file, which is the input file for the model

## 3.5 MLIR to INT8 Model

[Generate Calibration Table]

Before converting to the INT8 model, it is necessary to run a calibration to obtain the calibration table; The quantity of input data is prepared to be around 100-1000 sheets according to the situation.

Then use a calibration table to generate a cvi model. The image of the generated calibration table should be as similar as the distribution of the training data as possible

```
## This dataset is extracted from COCO2017 by 100 for calibration, and other␣
↪images are also acceptable. There is no mandatory requirement here.
run_calibration.py yolov5s.mlir \
--dataset COCO2017 \
--input_num 100 \
-o yolov5s_cali_table
```

After the operation is completed, a file named yolov5 will be generated_Cali_Table file, which is used as the input file for subsequent compilation of the cvimode model

[Generate cvi-model]

Then generate the int8 symmetric quantization cvi model and execute the following command:

Among them - quantum the output parameter indicates that the output layer is also quantified as int8, and if this parameter is not added, the output layer is kept as float32.

From the subsequent test results, quantifying the output layer to int8 can reduce some ions and improve inference speed,

And the model detection accuracy has not decreased significantly. It is recommended to add - quantum Output parameter

```
model_deploy.py \
--mlir yolov5s.mlir \
--quant_input \
--quant_output \
--quantize INT8 \
--calibration_table yolov5s_cali_table \
--processor cv181x \
--test_input yolov5s_in_f32.npz \
--test_reference yolov5s_top_outputs.npz \
--tolerance 0.85,0.45 \
--model yolov5_cv181x_int8_sym.cvimodel
```

Among them, model For the main parameters of deploy.py, please refer to the [tpu_mlr_xxxxx/doc/TPU-MILIR Quick Start Guide]

After compilation, a file named yolov5 will be generated_Cv181x_Int8_File for sym.cvimodel

After successfully running the above steps, the step of compiling the cvi model is completed, and then TDL can be used The SDK calls the exported CVIModel for YOLOv5 target detection and inference.

## 3.6 TDL SDK Interface Description

The TDLSDK toolkit needs to be contacted to obtain it.

The integrated YOLOv5 interface opens up pre processing settings, anchor, conf confidence, and NMS confidence settings for YOLOv5 model algorithms

The structure set for preprocessing is Yolov5PreParam

```
/** @struct YoloPreParam
*  @ingroup core_cvitdlcore
*  @brief Config the yolov5 detection preprocess.
*  @var YoloPreParam::factor
*  Preprocess factor, one dimension matrix, r g b channel
*  @var YoloPreParam::mean
*  Preprocess mean, one dimension matrix, r g b channel
*  @var YoloPreParam::rescale_type
*  Preprocess config, vpss rescale type config
*  @var YoloPreParam::pad_reverse
*  Preprocess padding config
*  @var YoloPreParam::keep_aspect_ratio
*  Preprocess config quantize scale
*  @var YoloPreParam::use_crop
*  Preprocess config, config crop
*  @var YoloPreParam:: resize_method
*  Preprocess resize method config
*  @var YoloPreParam::format
*  Preprocess pixcel format config
*/
typedef struct {
  float factor[3];
  float mean[3];
  meta_rescale_type_e rescale_type;
  bool pad_reverse;
  bool keep_aspect_ratio;
  bool use_quantize_scale;
  bool use_crop;
  VPSS_SCALE_COEF_E resize_method;
  PIXEL_FORMAT_E format;
} YoloPreParam;
```

Here is a simple setup example: * Initialize preprocessing settings YoloPreParam and Yolov5 model settings YoloAlgParam, using CVI_TDL_Set_YOLOV5_Param passes in the set parameters * Yolov5 is a detection algorithm for **anchor based**. For ease of use, anchor customization settings have been opened. In setting YoloAlgParam, it is important to note that the order of anchors and structures needs to be one-to-one, otherwise it may cause errors in the inference results * Additionally, it supports modifying the custom classification quantity. If the output classification quantity of the model is modified, YolovAlgParam.cls needs to be set as the modified classification quantity * The properties that appear in YoloPreParam and YoloAlgParam in the following code cannot be empty * Open the model again CVI_TDL_OpenModel * After opening the model again, the corresponding confidence level and nsm threshold can

be set: * CVI_TDL_SetModelThreshold sets the confidence threshold, which defaults to 0.5 * CVI_TDL_SetModelNmsThreshold sets the nsm threshold to 0.5 by default

```
// yolo preprocess setup
YoloPreParam p_preprocess_cfg;
for (int i = 0; i < 3; i++) {
    p_preprocess_cfg.factor[i] = 0.003922;
    p_preprocess_cfg.mean[i] = 0.0;
}
p_preprocess_cfg.use_quantize_scale = true;
p_preprocess_cfg.format = PIXEL_FORMAT_RGB_888_PLANAR;

// setup yolov5 param
YoloAlgParam p_yolov5_param;
uint32_t p_anchors[3][3][2] = {{{10, 13}, {16, 30}, {33, 23}},
                               {{30, 61}, {62, 45}, {59, 119}},
                               {{116, 90}, {156, 198}, {373, 326}}};
p_yolov5_param.anchors = &p_anchors;
uint32_t strides[3] = {8, 16, 32};
p_yolov5_param.strides = &strides;
p_yolov5_param.anchor_len = 3;
p_yolov5_param.stride_len = 3;
p_yolov5_param.cls = 80;

printf("setup yolov5 param \n");
ret = CVI_TDL_Set_YOLOV5_Param(tdl_handle, &p_preprocess_cfg, &p_yolov5_param);
if (ret != CVI_SUCCESS) {
    printf("Can not set Yolov5 parameters %#x\n", ret);
    return ret;
}

ret = CVI_TDL_OpenModel(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV5, model_path.
↪c_str());
if (ret != CVI_SUCCESS) {v c
    printf("open model failed %#x!\n", ret);
    return ret;
}

// set thershold for yolov5
CVI_TDL_SetModelThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV5, 0.5);
CVI_TDL_SetModelNmsThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV5, 0.5);
```

## 3.7 Compilation Instructions

1. Obtain the cross-compilation tools

```
wget https://sophon-file.sophon.cn/sophon-prod-s3/drive/23/03/07/16/
↪host-tools.tar.gz
tar xvf host-tools.tar.gz
cd host-tools
export PATH=$PATH:$(pwd)/gcc/riscv64-linux-musl-x86_64/bin
```

2. Download the TDL SDK

The download site for the tdlsdk toolkit: sftp://218.17.249.213. Account: cvitek_mlir_2023. Password: 7&2Wd%cu5k.

We download the cvitek_tdl_sdk_1227.tar.gz file.

3. Compile the TDL SDK

We enter the sample directory under cvitek_tdl_sdk.

```
chmod 777 compile_sample.sh
./compile_sample.sh
```

4. After compilation, connect to the development board and execute the program:
   - Connect the development board to the network, ensuring that the board and computer are on the same gateway.
   - Connect the computer to the development board via serial port, set the baud rate to 115200, and enter ifconfig on the computer's serial port to obtain the development board's IP address.
   - Connect to the development board using an SSH remote tool to the corresponding IP address, with the default username: root, and the default password: cvitek_tpu_sdk.
   - **After connecting to the development board, you can mount an SD card or a computer folder:**
     - The command to mount the SD card is:

```
mount /dev/mmcblk0 /mnt/sd
# or
mount /dev/mmcblk0p1 /mnt/sd
```

     - The command to mount a computer folder is:

```
mount -t nfs 10.80.39.3:/sophgo/nfsuser ./admin1_data -o nolock
```

     Be sure to change the IP address to your computer's IP and modify the path to your own path accordingly.

5. Export the Dynamic Dependency Libraries

The main dynamic dependency libraries required are:

- lib under the ai_sdk directory

- lib under the tpu_sdk directory
- middlewave/v2/lib
- middleware/v2/3rd
- lib under the ai_sdk/sample/3rd directory

Example as follows:

```
export LD_LIBRARY_PATH=/tmp/lfh/cvitek_tdl_sdk/lib:\
                       /tmp/lfh/cvitek_tdl_sdk/sample/3rd/opencv/
↪lib:\
                       /tmp/lfh/cvitek_tdl_sdk/sample/3rd/tpu/lib:\
                       /tmp/lfh/cvitek_tdl_sdk/sample/3rd/ive/lib:\
                       /tmp/lfh/cvitek_tdl_sdk/sample/3rd/middleware/
↪v2/lib:\
                       /tmp/lfh/cvitek_tdl_sdk/sample/3rd/lib:\
                       /tmp/lfh/cvitek_tdl_sdk/sample/3rd/middleware/
↪v2/lib/3rd:
```

**Caution:** Be sure to change /tmp/lfh to a path accessible by the development board. If you are using an SD card mount, you can copy all the necessary files from the lib directories into one folder in advance, and then export the corresponding path on the SD card.

6. Run the Sample Program

- Switch to the mounted cvitek_tdl_sdk/bin directory.
- Then run the following test case:

```
./sample_yolov5 /path/to/yolov5s.cvimodel /path/to/test.jpg
```

Be mindful to select your own cvimodel and the mounted path for the test image when running the above command.

**Reasoning and result acquisition**

Obtain images locally or through streaming, and use CVI_TDL_ReadImage function reads the picture, and then calls Yolov5 inference interface CVI_TDL_Yolov5.

The results of reasoning are stored in obj In the meta structure, traverse to obtain the coordinates of the upper left and lower right corners of the bounding box bbox, as well as the object score (x1, y1, x2, y2, score), as well as the classification classes

```
VIDEO_FRAME_INFO_S fdFrame;
ret = CVI_TDL_ReadImage(img_path.c_str(), &fdFrame, PIXEL_FORMAT_RGB_888);
std::cout << "CVI_TDL_ReadImage done!\n";

if (ret != CVI_SUCCESS) {
    std::cout << "Convert out video frame failed with :" << ret << ".file:" <<␣
↪str_src_dir
              << std::endl;
```

(continues on next page)

(continued from previous page)

```
}

cvtdl_object_t obj_meta = {0};

CVI_TDL_Yolov5(tdl_handle, &fdFrame, &obj_meta);

for (uint32_t i = 0; i < obj_meta.size; i++) {
    printf("detect res: %f %f %f %f %f %d\n", obj_meta.info[i].bbox.x1,
                                obj_meta.info[i].bbox.y1,
                                obj_meta.info[i].bbox.x2,
                                obj_meta.info[i].bbox.y2,
                                obj_meta.info[i].bbox.score,
                                obj_meta.info[i].classes);
}
```

The following are the results of testing the official YOLOv5 model after conversion on the Coco2017 dataset, using CV1811h as the testing platform_Wevb_0007a_Spinor

## 3.8 Test Result

The threshold used for the following tests is:

- Conf Threshold: 0.001
- Nms Threshold: 0.65

Input resolutions are all 640 x 640

The official export method of the YOLOv5S model onnx performance:

| platform | Inference time (ms) | band-width (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| pytorch | N/A | N/A | N/A | 56.8 | 37.4 |
| cv181x | 92.8 | 100.42 | 16.01 | Quantification failure | Quantification failure |
| cv182x | 69.89 | 102.74 | 16 | Quantification failure | Quantification failure |
| cv183x | 25.66 | 73.4 | N/A | Quantification failure | Quantification failure |

TDL of yolov5s model_SDK export method onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| onnx | N/A | N/A | N/A | 55.4241 | 36.6361 |
| cv181x | 87.76 | 85.74 | 15.8 | 54.204 | 34.3985 |
| cv182x | 65.33 | 87.99 | 15.77 | 54.204 | 34.3985 |
| cv183x | 22.86 | 58.38 | 14.22 | 54.204 | 34.3985 |

The official export method of the YOLOv5m model onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| pytorch | N/A | N/A | N/A | 64.1 | 45.4 |
| cv181x | ion allocation failure | ion allocation failure | 35.96 | Quantification failure | Quantification failure |
| cv182x | 180.85 | 258.41 | 35.97 | Quantification failure | Quantification failure |
| cv183x | 59.36 | 137.86 | 30.49 | Quantification failure | Quantification failure |

TDL of yolov5m model SDK export method onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| onnx | N/A | N/A | N/A | 62.770 | 44.4973 |
| cv181x | N/A | N/A | 35.73 | ion allocation failure | ion allocation failure |
| cv182x | 176.04 | 243.62 | 35.74 | 61.5907 | 42.0852 |
| cv183x | 56.53 | 122.9 | 30.27 | 61.5907 | 42.0852 |

# 4 Deployment of Yolov6 Model for General Use

## 4.1 Introduction

This document introduces the operation process of deploying the YOLOv6 architecture model on the CV181x development board. The main steps include:

- Convert the YOLOv6 model Pytorch version to the ONNX model
- Convert onnx model to cvi model format
- Finally, write a calling interface to obtain the inference results

## 4.2 Convert pt Model to onnx

Download the official Yolov6 repository [meituan/YOLOv6](https://github.com/meituan/YOLOv6)Download the yolov6 weight file, create a new directory called weights in the yolov6 folder, and place the downloaded weight file in the directory yolov6 main/weights/

Modify yolov6 main/deploy/export_Onnx. py file, and then add a function

```
def detect_forward(self, x):

  final_output_list = []
  for i in range(self.nl):
      b, _, h, w = x[i].shape
      l = h * w
      x[i] = self.stems[i](x[i])
      cls_x = x[i]
      reg_x = x[i]
      cls_feat = self.cls_convs[i](cls_x)
      cls_output = self.cls_preds[i](cls_feat)
      reg_feat = self.reg_convs[i](reg_x)
      reg_output_lrtb = self.reg_preds[i](reg_feat)

      final_output_list.append(cls_output.permute(0, 2, 3, 1))
```

(continues on next page)

(continued from previous page)

```
        final_output_list.append(reg_output_lrtb.permute(0, 2, 3, 1))

    return final_output_list
```

Then use dynamic binding to modify the forward of the YOLOv6 model. You need to first import types and then add the following code before onnx export

```
print("==================")
print(model)
print("==================")

# Dynamic binding to modify the forward function of model detect
model.detect.forward = types.MethodType(detect_forward, model.detect)

y = model(img)  # dry run
# ONNX export
```

Then enter the following command in the yolov6 main/directory, where:

- Weights is the path to the pytorch model file
- IMG is the input size for the model
- Batch model input batch
- Simplify the ONNX model

```
python ./deploy/ONNX/export_onnx.py \
  --weights ./weights/yolov6n.pt \
  --img 640 \
  --batch 1
```

And then we get the onnx model

## 4.3 Onnx Model Conversion cvi model

The cvi model conversion operation can refer to the onnx model conversion cvi model section in the Yolo v5 porting chapter.

## 4.4 Yolov6 Interface Description

Provide preprocessing parameters and algorithm parameter settings, including parameter settings:

- YoloPreParam input preprocessing settings
- The reciprocal of factor preprocessing variance
- Mean preprocessing mean
- Use_Quantify_Scale Preprocessing Image Size

- Format Image Format
- YoloAlgParam
- Cls sets the classification of yolov6 models

>Yolov6 is an anchor free object detection network that does not require an anchor to be passed in

Additionally, there are two parameter settings for yolov6:

- CVI_TDL_SetModelThreshold sets the confidence threshold, which defaults to 0.5
- CVI_TDL_SetModelNmsThreshold sets the nms threshold to 0.5 by default

```
// setup preprocess
YoloPreParam p_preprocess_cfg;

for (int i = 0; i < 3; i++) {
    printf("asign val %d \n", i);
    p_preprocess_cfg.factor[i] = 0.003922;
    p_preprocess_cfg.mean[i] = 0.0;
}
p_preprocess_cfg.use_quantize_scale = true;
p_preprocess_cfg.format = PIXEL_FORMAT_RGB_888_PLANAR;

printf("start yolov algorithm config \n");
// setup yolov6 param
YoloAlgParam p_yolov6_param;
p_yolov6_param.cls = 80;

ret = CVI_TDL_Set_YOLOV6_Param(tdl_handle, &p_preprocess_cfg, &p_yolov6_param);
printf("yolov6 set param success!\n");
if (ret != CVI_SUCCESS) {
    printf("Can not set Yolov6 parameters %#x\n", ret);
    return ret;
}

ret = CVI_TDL_OpenModel(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV6, model_path.
↪c_str());
if (ret != CVI_SUCCESS) {
    printf("open model failed %#x!\n", ret);
    return ret;
}
// set thershold
CVI_TDL_SetModelThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV6, 0.5);
CVI_TDL_SetModelNmsThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV6, 0.5);

CVI_TDL_Yolov6(tdl_handle, &fdFrame, &obj_meta);

for (uint32_t i = 0; i < obj_meta.size; i++) {
    printf("detect res: %f %f %f %f %f %d\n",
           obj_meta.info[i].bbox.x1,
```

(continues on next page)

(continued from previous page)

```
            obj_meta.info[i].bbox.y1,
            obj_meta.info[i].bbox.x2,
            obj_meta.info[i].bbox.y2,
            obj_meta.info[i].bbox.score,
            obj_meta.info[i].classes);
    }
```

## 4.5 Test Result

Converted yolov6n and yolov6s provided by the official yolov6 warehouse, with a test dataset of COCO2017

The threshold parameter is set to:

- Conf_Threshold: 0.03
- Nms_Threshold: 0.65

Resolution is 640x640

The official export method of the YOLOv6N model onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| pytorch | N/A | N/A | N/A | 53.1 | 37.5 |
| cv181x | ion allocation failure | ion allocation failure | 11.58 | Quantification failure | Quantification failure |
| cv182x | 39.17 | 47.08 | 11.56 | Quantification failure | Quantification failure |
| cv183x | Quantification failure | Quantification failure | Quantification failure | Quantification failure | Quantification failure |

TDL of yolov6n model SDK export method onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| onnx | N/A | N/A | N/A | 51.6373 | 36.4384 |
| cv181x | 49.11 | 31.35 | 8.46 | 49.8226 | 34.284 |
| cv182x | 34.14 | 30.53 | 8.45 | 49.8226 | 34.284 |
| cv183x | 10.89 | 21.22 | 8.49 | 49.8226 | 34.284 |

The official export method of the yolov6s model onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| pytorch | N/A | N/A | N/A | 61.8 | 45 |
| cv181x | ion allocation failure | ion allocation failure | 27.56 | Quantification failure | Quantification failure |
| cv182x | 131.1 | 115.81 | 27.56 | Quantification failure | Quantification failure |
| cv183x | Quantification failure | Quantification failure | Quantification failure | Quantification failure | Quantification failure |

TDL of yolov6s model SDK export method onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| onnx | N/A | N/A | N/A | 60.1657 | 43.5878 |
| cv181x | ion allocation failure | ion allocation failure | 25.33 | ion allocation failure | ion allocation failure |
| cv182x | 126.04 | 99.16 | 25.32 | 56.2774 | 40.0781 |
| cv183x | 38.55 | 57.26 | 23.59 | 56.2774 | 40.0781 |

# 5 Deployment of Yolov7 Model for General Use

## 5.1 Introduction

This document introduces the operation process of deploying the YOLOV7 architecture model on the CV181x development board. The main steps include:

- Convert YOLOV7 model Pytorch version to ONNX model
- Convert onnx model to cvi model format
- Finally, write a calling interface to obtain the inference results

## 5.2 Convert pt Model to onnx

Download official [yolov7](https://github.com/WongKinYiu/yolov7)Warehouse code

```
Git clone https://github.com/WongKinYiu/yolov7.git
```

Create a new folder called weights in the directory where you downloaded the code above, and then move the model that needs to be exported to onnx to yolov7/weights

```
cd yolov7&mkdir weights

cp path/to/nnx/ Weights/
```

Create a new file onnx in the yolov7/directory_Export.py, add the following code

```
import torch
import torch.nn as nn
import onnx
import models
from models.common import Conv
from utils.activations import Hardswish, SiLU
import types
```

(continues on next page)

(continued from previous page)

```
pt_path = "path/to/yolov7-tiny.pt"
save_path = pt_path.replace(".pt", ".onnx")

ckpt = torch.load(pt_path, map_location="cpu")
model = ckpt["model"].float().fuse().eval()

# Compatibility updates
for m in model.modules():
    if type(m) in [nn.Hardswish, nn.LeakyReLU, nn.ReLU, nn.ReLU6, nn.SiLU]:
        m.inplace = True  # pytorch 1.7.0 compatibility
    elif type(m) is nn.Upsample:
        m.recompute_scale_factor = None  # torch 1.11.0 compatibility
    elif type(m) is Conv:
        m._non_persistent_buffers_set = set()

# Update model
    for k, m in model.named_modules():
        m._non_persistent_buffers_set = set()  # pytorch 1.6.0 compatibility
        if isinstance(m, models.common.Conv):  # assign export-friendly␣
↪activations
            if isinstance(m.act, nn.Hardswish):
                m.act = Hardswish()
            elif isinstance(m.act, nn.SiLU):
                m.act = SiLU()

def forward(self, x):
        # x = x.copy()  # for profiling
        z = []  # inference output

        for i in range(self.nl):
            x[i] = self.m[i](x[i])  # conv

            bs, _, ny, nx = x[i].shape  # x(bs,255,20,20) to x(bs,3,20,20,85)
            x[i] = x[i].view(bs, self.na, self.no, ny, nx).permute(0, 1, 3, 4,␣
↪2).contiguous()

            xywh, conf, score = x[i].split((4, 1, self.nc), 4)

            z.append(xywh[0])
            z.append(conf[0])
            z.append(score[0])

        return z

model.model[-1].forward = types.MethodType(forward, model.model[-1])
img = torch.zeros(1, 3, 640, 640)
torch.onnx.export(model, img, save_path, verbose=False,
                  opset_version=12, input_names=['images'])
```

## 5.3 Onnx Model Conversion cvi model

The cvi model conversion operation can refer to the onnx model conversion cvi model section in the Yolo v5 porting chapter.

## 5.4 TDL SDK Interface Description

The detection and decoding process of the YOLOv7 model is basically similar to that of the YOLOv5 model, so the interface of YOLOv5 can be directly used

> ** Pay attention to modifying anchors to anchors of yolov7 ** > > > anchors: > - [12,16, 19,36, 40,28] *# P3/8* > - [36,75, 76,55, 72,146] *# P4/16* > - [142,110, 192,243, 459,401] *# P5/32* >

## 5.5 Test Result

Tested the indicators of various versions of the YOLOV7-TINY model, with the test data being COCO2017, where the threshold was set to:

- Conf_Threshold: 0.001
- Nms_Threshold: 0.65

All resolutions are 640 x 640

The official export method of the YOLOV7-TINY model onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| pytorch | N/A | N/A | N/A | 56.7 | 38.7 |
| cv181x | 75.4 | 85.31 | 17.54 | Quantification failure | Quantification failure |
| cv182x | 56.6 | 85.31 | 17.54 | Quantification failure | Quantification failure |
| cv183x | 21.85 | 71.46 | 16.15 | Quantification failure | Quantification failure |

TDL of yolov7-tiny model SDK export method onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| onnx | N/A | N/A | N/A | 53.7094 | 36.438 |
| cv181x | 70.41 | 70.66 | 15.43 | 53.3681 | 32.6277 |
| cv182x | 52.01 | 70.66 | 15.43 | 53.3681 | 32.6277 |
| cv183x | 18.95 | 55.86 | 14.05 | 53.3681 | 32.6277 |

# 6 Deployment of Yolov8 Model for General Use

## 6.1 Introduction

This document introduces the operation process of deploying the YOLOv8 architecture model on the CV181x development board. The main steps include:

- Convert YOLOv8 model Pytorch version to ONNX model
- Convert onnx model to cvi model format
- Finally, write a calling interface to obtain the inference results

## 6.2 Convert pt Model to onnx

Firstly, obtain the official warehouse code for YOLOv8 [ultra tics/ultra tics: NEW - YOLOv8 In PyTorch>ONNX>OpenVINO>CoreML>TFLite (github. com)](https://github.com/ultralytics/ultralytics) .. code-block:: shell

git clone https://github.com/ultralytics/ultralytics.git

Download the corresponding yolov8 model file again to [yolov8n](https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n.pt)For example, then place the downloaded yolov8n. pt in the ultra tics/weights/directory, as shown on the following command line

cd ultralytics & mkdir weights cd weights wget https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n.pt

Adjust the output branch of YOLOv8, remove the decoding part of the forward function, and separate the boxes and cls of three different feature maps to obtain six branches

Specifically, you can create a new file in the Ultratics/directory and paste the following code

```
from ultralytics import YOLO
import types

input_size = (640, 640)
```

(continues on next page)

(continued from previous page)

```
def forward2(self, x):
    x_reg = [self.cv2[i](x[i]) for i in range(self.nl)]
    x_cls = [self.cv3[i](x[i]) for i in range(self.nl)]
    return x_reg + x_cls

model_path = "./weights/yolov8s.pt"
model = YOLO(model_path)
model.model.model[-1].forward = types.MethodType(forward2, model.model.model[-
↪1])
model.export(format='onnx', opset=11, imgsz=input_size)
```

After running the above code, you can run it in/ Obtain the yolov8n. onnx file in the weights/directory, and then convert the onnx model to the cvi model

## 6.3 Onnx Model Conversion cvi model

The CVIModel conversion operation can refer to the onnx model conversion CVIModel section in the YOLO-V5 porting section.

## 6.4 TDL SDK Interface Description

First, create a cvitdl_handle, and then open the corresponding cvi model. Before running the inference interface, you can set two thresholds for your own model

- CVI_TDL_SetModelThreshold
- CVI_TDL_SetModelNmsThreshold

The final inference result is analyzed through cvtdl_object_t. Information acquisition

```
// create handle
cvitdl_handle_t tdl_handle = NULL;
ret = CVI_TDL_CreateHandle(&tdl_handle);
  if (ret != CVI_SUCCESS) {
    printf("Create tdl handle failed with %#x!\n", ret);
    return ret;
  }

// read image
VIDEO_FRAME_INFO_S bg;
ret = CVI_TDL_ReadImage(strf1.c_str(), &bg, PIXEL_FORMAT_RGB_888_PLANAR);

// open model and set conf & nms threshold
ret = CVI_TDL_OpenModel(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV8_DETECTION,␣
↪path_to_model);
CVI_TDL_SetModelThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV8_DETECTION,␣
↪0.5);
```

(continues on next page)

(continued from previous page)

```
CVI_TDL_SetModelNmsThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOV8_
↪DETECTION, 0.5);
if (ret != CVI_SUCCESS) {
  printf("open model failed with %#x!\n", ret);
    return ret;
}

// start infer
cvtdl_object_t obj_meta = {0};
CVI_TDL_YOLOV8_Detection(tdl_handle, &bg, &obj_meta);

// analysis result
std::stringstream ss;
ss << "boxes=[";
for (uint32_t i = 0; i < obj_meta.size; i++) {
ss << "[" << obj_meta.info[i].bbox.x1 << "," << obj_meta.info[i].bbox.y1 << ","
  << obj_meta.info[i].bbox.x2 << "," << obj_meta.info[i].bbox.y2 << ","
  << obj_meta.info[i].classes << "," << obj_meta.info[i].bbox.score << "],";
}
```

## 6.5 Test Result

The Yolov8n and Yolov8s models on the official website were converted and tested on the COCO2017 dataset, with the threshold set to:

- Conf: 0.001
- Nms_Threshold: 0.6

All resolutions are 640 x 640

The official export method of the YOLOV8n model onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| pytorch | N/A | N/A | N/A | 53 | 37.3 |
| cv181x | 54.91 | 44.16 | 8.64 | Quantification failure | Quantification failure |
| cv182x | 40.21 | 44.32 | 8.62 | Quantification failure | Quantification failure |
| cv183x | 17.81 | 40.46 | 8.3 | Quantification failure | Quantification failure |

TDL of yolov8n model_SDK export method onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| onnx | N/A | N/A | N/A | 51.32 | 36.4577 |
| cv181x | 45.62 | 31.56 | 7.54 | 51.2207 | 35.8048 |
| cv182x | 32.8 | 32.8 | 7.72 | 51.2207 | 35.8048 |
| cv183x | 12.61 | 28.64 | 7.53 | 51.2207 | 35.8048 |

The official export method of the yolov8s model onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| pytorch | N/A | N/A | N/A | 61.8 | 44.9 |
| cv181x | 144.72 | 101.75 | 17.99 | Quantification failure | Quantification failure |
| cv182x | 103 | 101.75 | 17.99 | Quantification failure | Quantification failure |
| cv183x | 38.04 | 38.04 | 16.99 | Quantification failure | Quantification failure |

TDL of yolov8s model SDK export method onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| onnx | N/A | N/A | N/A | 60.1534 | 44.034 |
| cv181x | 135.55 | 89.53 | 18.26 | 60.2784 | 43.4908 |
| cv182x | 95.95 | 89.53 | 18.26 | 60.2784 | 43.4908 |
| cv183x | 32.88 | 58.44 | 16.9 | 60.2784 | 43.4908 |

# 7 Deployment of Yolox Model for General Use

## 7.1 Introduction

This document introduces the operation process of deploying the YOLOX architecture model on the CV181x development board. The main steps include:

- Convert the YOLOX model Pytorch version to the ONNX model
- Convert onnx model to cvi model format
- Finally, write a calling interface to obtain the inference results

## 7.2 Convert pt Model to onnx

Firstly, you can download the official code of YOLOX on Github: [Megvii BaseDetection/YOLOX: YOLOX is a high performance anchor free YOLO, excepting YOLOv3~v5 with MegEngine, ONNX, TensorRT, ncnn, and OpenVINO supported. Documentation: https://yolox.readthedocs.io/ (github. com) (https://github.com/Megvii-BaseDetection/YOLOX/tree/main )

Install YOLOX from source code using the following command

```
git clone git@github.com : Megvii BaseDetection/YOLOX.git

cd YOLOX

pip3 install - v - e# Or Python 3 setup.py development
```

##Onnx model export

You need to switch to the YOLOX repository path you just downloaded, and then create a weights directory to move the pre trained. pth file here

```
Cd YOLOX&mkdir weights

cp path/to/pth/ Weigths/
```

###Official export onnx

Switch to the tools path

```
cd tools
```

Export method for decoding in onnx

```
python \
export_onnx.py \
--output-name ../weights/yolox_m_official.onnx \
-n yolox-m \
--no-onnxsim \
-c ../weights/yolox_m.pth \
--decode_in_inference
```

The meanings of relevant parameters are as follows:

- –output-name Represents the path and name of the exported onnx model
- -n Represents the model name, which can be selected * yolox-s, m, l, x * yolo-nano * yolox-tiny * yolov3
- -c Path to the. pth model file representing pre training
- –decode_in_inference Indicates whether to decode in onnx

###TDL_SDK version export onnx

To ensure the accuracy of quantization, it is necessary to divide the YOLOX decoded head into three different branch outputs, rather than the official version of the merged output

Export the heads of three different branches through the following scripts and commands:

Create a new file export in the YOLOX/tools/directory_Onnx_TDL_Sdk.py and attach the following code

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-
# Copyright (c) Megvii, Inc. and its affiliates.

import argparse
import os
from loguru import logger

import torch
from torch import nn

import sys
sys.path.append("..")

from yolox.exp import get_exp
from yolox.models.network_blocks import SiLU
from yolox.utils import replace_module
import types
```

(continues on next page)

(continued from previous page)

```
def make_parser():
    parser = argparse.ArgumentParser("YOLOX onnx deploy")
    parser.add_argument(
        "--output-name", type=str, default="yolox.onnx", help="output name of␣
↪models"
    )
    parser.add_argument(
        "--input", default="images", type=str, help="input node name of onnx␣
↪model"
    )
    parser.add_argument(
        "--output", default="output", type=str, help="output node name of onnx␣
↪model"
    )
    parser.add_argument(
        "-o", "--opset", default=11, type=int, help="onnx opset version"
    )
    parser.add_argument("--batch-size", type=int, default=1, help="batch size")
    parser.add_argument(
        "--dynamic", action="store_true", help="whether the input shape should␣
↪be dynamic or not"
    )
    parser.add_argument("--no-onnxsim", action="store_true", help="use onnxsim␣
↪or not")
    parser.add_argument(
        "-f",
        "--exp_file",
        default=None,
        type=str,
        help="experiment description file",
    )
    parser.add_argument("-expn", "--experiment-name", type=str, default=None)
    parser.add_argument("-n", "--name", type=str, default=None, help="model name
↪")
    parser.add_argument("-c", "--ckpt", default=None, type=str, help="ckpt path
↪")
    parser.add_argument(
        "opts",
        help="Modify config options using the command-line",
        default=None,
        nargs=argparse.REMAINDER,
    )
    parser.add_argument(
        "--decode_in_inference",
        action="store_true",
        help="decode in inference or not"
    )
```

(continues on next page)

(continued from previous page)

```
    return parser

def forward(self, xin, labels=None, imgs=None):
        outputs = []
        origin_preds = []
        x_shifts = []
        y_shifts = []
        expanded_strides = []

        for k, (cls_conv, reg_conv, stride_this_level, x) in enumerate(
            zip(self.cls_convs, self.reg_convs, self.strides, xin)
        ):
            x = self.stems[k](x)
            cls_x = x
            reg_x = x

            cls_feat = cls_conv(cls_x)
            cls_output = self.cls_preds[k](cls_feat)

            reg_feat = reg_conv(reg_x)
            reg_output = self.reg_preds[k](reg_feat)
            obj_output = self.obj_preds[k](reg_feat)

            outputs.append(reg_output.permute(0, 2, 3, 1))
            outputs.append(obj_output.permute(0, 2, 3, 1))
            outputs.append(cls_output.permute(0, 2, 3, 1))

        return outputs

@logger.catch
def main():
    args = make_parser().parse_args()
    logger.info("args value: {}".format(args))
    exp = get_exp(args.exp_file, args.name)
    exp.merge(args.opts)

    if not args.experiment_name:
        args.experiment_name = exp.exp_name

    model = exp.get_model()
    if args.ckpt is None:
        file_name = os.path.join(exp.output_dir, args.experiment_name)
        ckpt_file = os.path.join(file_name, "best_ckpt.pth")
    else:
        ckpt_file = args.ckpt

    # load the model state dict
    ckpt = torch.load(ckpt_file, map_location="cpu")
```

(continues on next page)

(continued from previous page)

```
    model.eval()
    if "model" in ckpt:
        ckpt = ckpt["model"]
    model.load_state_dict(ckpt)
    model = replace_module(model, nn.SiLU, SiLU)

    # replace official head forward function
    if not args.decode_in_inference:
        model.head.forward = types.MethodType(forward, model.head)

    model.head.decode_in_inference = args.decode_in_inference

    logger.info("loading checkpoint done.")
    dummy_input = torch.randn(args.batch_size, 3, exp.test_size[0], exp.test_
↪size[1])

    torch.onnx._export(
        model,
        dummy_input,
        args.output_name,
        input_names=[args.input],
        output_names=[args.output],
        dynamic_axes={args.input: {0: 'batch'},
                      args.output: {0: 'batch'}} if args.dynamic else None,
        opset_version=args.opset,
    )
    logger.info("generated onnx model named {}".format(args.output_name))

    if not args.no_onnxsim:
        import onnx
        from onnxsim import simplify

        # use onnx-simplifier to reduce reduent model.
        onnx_model = onnx.load(args.output_name)
        model_simp, check = simplify(onnx_model)
        assert check, "Simplified ONNX model could not be validated"
        onnx.save(model_simp, args.output_name)
        logger.info("generated simplified onnx model named {}".format(args.
↪output_name))

if __name__ == "__main__":
    main()
```

然后输入以下命令

```
python \
export_onnx_tdl_sdk.py \
--output-name ../weights/yolox_s_9_branch.onnx \
```

(continues on next page)

(continued from previous page)

```
-n yolox-s \
--no-onnxsim \
-c ../weights/yolox_s.pth
```

## 7.3 Onnx Model Conversion cvi model

The cvi model conversion operation can refer to the onnx model conversion cvi model section in the Yolo v5 porting chapter.

## 7.4 TDL SDK Interface Description

###Preprocessing parameter settings

Preprocessing parameter settings are passed in through a structure to set parameters

```
typedef struct {
  float factor[3];
  float mean[3];
  meta_rescale_type_e rescale_type;

  bool use_quantize_scale;
  PIXEL_FORMAT_E format;
} YoloPreParam;
```

For YOLOX, the following four parameters need to be passed in:

- Factor preprocessing scale parameter
- Mean preprocessing mean parameter
- Use_Quantify_Does scale use the size of the model? The default is true
- Format image format, PIXEL_FORMAT_RGB_888_PLANAR

###Algorithm parameter settings

```
typedef struct {
  uint32_t cls;
} YoloAlgParam;
```

The number of categories that need to be passed in, such as

```
YoloAlgParam p_yolo_param;
p_yolo_param.cls = 80;
```

The additional model confidence parameter settings and NMS threshold settings are as follows:

```
CVI_TDL_SetModelThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOX, conf_
↪threshold);
CVI_TDL_SetModelNmsThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOX, nms_
↪threshold);
```

Among them, conf_Threshold is the confidence threshold; Nms_Threshold is the nms threshold

###Test Code

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <chrono>
#include <fstream>
#include <functional>
#include <iostream>
#include <map>
#include <sstream>
#include <string>
#include <vector>
#include "core.hpp"
#include "core/cvi_tdl_types_mem_internal.h"
#include "core/utils/vpss_helper.h"
#include "cvi_tdl.h"
#include "evaluation/cvi_tdl_media.h"
#include "sys_utils.hpp"

int main(int argc, char* argv[]) {
  int vpssgrp_width = 1920;
  int vpssgrp_height = 1080;
  CVI_S32 ret = MMF_INIT_HELPER2(vpssgrp_width, vpssgrp_height, PIXEL_FORMAT_
↪RGB_888, 1,
                                 vpssgrp_width, vpssgrp_height, PIXEL_FORMAT_RGB_
↪888, 1);
  if (ret != CVI_TDL_SUCCESS) {
    printf("Init sys failed with %#x!\n", ret);
    return ret;
  }

  cvitdl_handle_t tdl_handle = NULL;
  ret = CVI_TDL_CreateHandle(&tdl_handle);
  if (ret != CVI_SUCCESS) {
    printf("Create tdl handle failed with %#x!\n", ret);
    return ret;
  }
  printf("start yolox preprocess config \n");
  // // setup preprocess
  YoloPreParam p_preprocess_cfg;
```

(continues on next page)

(continued from previous page)

```
  for (int i = 0; i < 3; i++) {
    p_preprocess_cfg.factor[i] = 1.0;
    p_preprocess_cfg.mean[i] = 0.0;
  }
  p_preprocess_cfg.use_quantize_scale = true;
  p_preprocess_cfg.format = PIXEL_FORMAT_RGB_888_PLANAR;

  printf("start yolo algorithm config \n");
  // setup yolo param
  YoloAlgParam p_yolo_param;
  p_yolo_param.cls = 80;

  printf("setup yolox param \n");
  ret = CVI_TDL_Set_YOLOX_Param(tdl_handle, &p_preprocess_cfg, &p_yolo_param);
  printf("yolox set param success!\n");
  if (ret != CVI_SUCCESS) {
    printf("Can not set YoloX parameters %#x\n", ret);
    return ret;
  }

  std::string model_path = argv[1];
  std::string str_src_dir = argv[2];

  float conf_threshold = 0.5;
  float nms_threshold = 0.5;
  if (argc > 3) {
    conf_threshold = std::stof(argv[3]);
  }

  if (argc > 4) {
    nms_threshold = std::stof(argv[4]);
  }

  printf("start open cvimodel...\n");
  ret = CVI_TDL_OpenModel(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOX, model_path.
↪c_str());
  if (ret != CVI_SUCCESS) {
    printf("open model failed %#x!\n", ret);
    return ret;
  }
  printf("cvimodel open success!\n");
  // set thershold
  CVI_TDL_SetModelThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOX, conf_
↪threshold);
  CVI_TDL_SetModelNmsThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_YOLOX, nms_
↪threshold);

  std::cout << "model opened:" << model_path << std::endl;
```

(continues on next page)

(continued from previous page)

```
  VIDEO_FRAME_INFO_S fdFrame;
  ret = CVI_TDL_ReadImage(str_src_dir.c_str(), &fdFrame, PIXEL_FORMAT_RGB_888);
  std::cout << "CVI_TDL_ReadImage done!\n";

  if (ret != CVI_SUCCESS) {
    std::cout << "Convert out video frame failed with :" << ret << ".file:" <<␣
↪str_src_dir
              << std::endl;
  }

  cvtdl_object_t obj_meta = {0};

  CVI_TDL_YoloX(tdl_handle, &fdFrame, &obj_meta);

  printf("detect number: %d\n", obj_meta.size);
  for (uint32_t i = 0; i < obj_meta.size; i++) {
    printf("detect res: %f %f %f %f %f %d\n", obj_meta.info[i].bbox.x1, obj_
↪meta.info[i].bbox.y1,
           obj_meta.info[i].bbox.x2, obj_meta.info[i].bbox.y2, obj_meta.info[i].
↪bbox.score,
           obj_meta.info[i].classes);
  }

  CVI_VPSS_ReleaseChnFrame(0, 0, &fdFrame);
  CVI_TDL_Free(&obj_meta);
  CVI_TDL_DestroyHandle(tdl_handle);

  return ret;
}
```

## 7.5 Test Result

Tested the performance indicators of the YOLOX model onnx and various platforms on CV181x/2x/3x, with parameter settings as follows:

- Conf: 0.001
- Nms: 0.65
- Resolution: 640 x 640

The official export method of the YOLOX-S model onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| pytorch | N/A | N/A | N/A | 59.3 | 40.5 |
| cv181x | 131.95 | 104.46 | 16.43 | Quantification failure | Quantification failure |
| cv182x | 95.75 | 104.85 | 16.41 | Quantification failure | Quantification failure |
| cv183x | Quantification failure | Quantification failure | Quantification failure | Quantification failure | Quantification failure |

TDL of yolox-s model_SDK export method onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| onnx | N/A | N/A | N/A | 53.1767 | 36.4747 |
| cv181x | 127.91 | 95.44 | 16.24 | 52.4016 | 35.4241 |
| cv182x | 91.67 | 95.83 | 16.22 | 52.4016 | 35.4241 |
| cv183x | 30.6 | 65.25 | 14.93 | 52.4016 | 35.4241 |

The official export method of the yolox-m model onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| pytorch | N/A | N/A | N/A | 65.6 | 46.9 |
| cv181x | ion allocation failure | ion allocation failure | 39.18 | Quantification failure | Quantification failure |
| cv182x | 246.1 | 306.31 | 39.16 | Quantification failure | Quantification failure |
| cv183x | Quantification failure | Quantification failure | Quantification failure | Quantification failure | Quantification failure |

TDL of yolox-m model SDK export method onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| onnx | N/A | N/A | N/A | 59.9411 | 43.0057 |
| cv181x | ion allocation failure | ion allocation failure | 38.95 | 59.3559 | 42.1688 |
| cv182x | 297.5 | 242.65 | 38.93 | 59.3559 | 42.1688 |
| cv183x | 75.8 | 144.97 | 33.5 | 59.3559 | 42.1688 |

# 8 Deployment of PP YOLOE Model for General Use

## 8.1 Introduction

This document introduces the operation process of deploying the PPYOLOE architecture model on the CV181x development board. The main steps include:

- Convert the Pytorch version of the PPYOLOE model to the ONNX model
- Convert onnx model to cvi model format
- Finally, write a calling interface to obtain the inference results

## 8.2 Convert pt Model to onnx

PP YOLOE is an Anchor free model based on PP Yolov2, and the official warehouse is located in [PaddleDetection](https://github.com/PaddlePaddle/PaddleDetection)

Obtain official warehouse code and install:

```
Git clone https://github.com/PaddlePaddle/PaddleDetection.git
```

#CUDA10.2

```
Python - m pip install paddlepaddle gpu=2.3.2- i https://mirror.baidu.com/pypi/
↪simple
```

For other versions, please refer to the official installation document [Start using_PaddlePaddle-an open source deep learning platform derived from industrial practice (paddlepaddle. org. cn)](https://www.paddlepaddle.org.cn/install/quick?docurl=/documentation/docs/zh/install/pip/linux -Pip. html)

##Onnx export

ONNX export can refer to the official document [PaddleDetection/deploy/EXPORT-ONNX-MODEL.md at release/2.4 · PaddlePaddle/PaddleDetection (github. com)](https://github.com/PaddlePaddle/PaddleDetection/blob/release/2.4/deploy/EXPORT_ONNX_MODEL.md)

This document provides the official version direct export method and the calculation version export method onnx. The calculation version export method needs to remove the decoding part of the detection header for subsequent quantization, and the decoding part is handed over to TDL_SDK implementation

###Official version

```
cd PaddleDetection
python \
tools/export_model_official.py \
-c configs/ppyoloe/ppyoloe_crn_s_300e_coco.yml \
-o weights=https://paddledet.bj.bcebos.com/models/ppyoloe_crn_s_300e_coco.
↪pdparams

paddle2onnx \
--model_dir \
output_inference/ppyoloe_crn_s_300e_coco \
--model_filename model.pdmodel \
--params_filename model.pdiparams \
--opset_version 11 \
--save_file output_inference_official/ppyoloe_crn_s_300e_coco/ppyoloe_crn_s_
↪300e_coco_official.onnx
```

Parameter Description:

- -c weigt file
- -o paddle weight
- –model_dir model outpath
- –model_filename paddle name
- –params_filename paddle config
- –opset_version opset verson config
- –save_file onnx output path

###Calculus version export

In order to better quantify the model, it is necessary to remove the decoded part of the detection header and export the onnx model. Use the following method to export the non decoded onnx model

Create a new file export in the tools/directory_Model_No_Code.py and add the following code

```
# Copyright (c) 2020 PaddlePaddle Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
```

(continues on next page)

(continued from previous page)

```
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import sys

# add python path of PaddleDetection to sys.path
parent_path = os.path.abspath(os.path.join(__file__, *(['..'] * 2)))
sys.path.insert(0, parent_path)

# ignore warning log
import warnings
warnings.filterwarnings('ignore')

import paddle
from ppdet.core.workspace import load_config, merge_config
from ppdet.utils.check import check_gpu, check_version, check_config
from ppdet.utils.cli import ArgsParser
from ppdet.engine import Trainer
from ppdet.slim import build_slim_model
import paddle.nn.functional as F

from ppdet.utils.logger import setup_logger
logger = setup_logger('export_model')
import types

def yoloe_forward(self):
        body_feats = self.backbone(self.inputs)
        neck_feats = self.neck(body_feats, self.for_mot)
        yolo_head_outs = self.yolo_head(neck_feats)
        return yolo_head_outs

def head_forward(self, feats, targets=None, aux_pred=None):

    cls_score_list, reg_dist_list = [], []
    for i, feat in enumerate(feats):
        _, _, h, w = feat.shape
        l = h * w
        avg_feat = F.adaptive_avg_pool2d(feat, (1, 1))
        cls_logit = self.pred_cls[i](self.stem_cls[i](feat, avg_feat) +
                                     feat)
        reg_dist = self.pred_reg[i](self.stem_reg[i](feat, avg_feat))
```

(continues on next page)

(continued from previous page)

```
        reg_dist = reg_dist.reshape(
            [-1, 4, self.reg_channels, 1]).transpose([0, 2, 3, 1])
        reg_dist = self.proj_conv(F.softmax(
                reg_dist, axis=1)).squeeze(1)
        reg_dist = reg_dist.reshape([-1, h, w, 4])
        cls_logit = cls_logit.transpose([0, 2, 3, 1])
        cls_score_list.append(cls_logit)
        reg_dist_list.append(reg_dist)

    return cls_score_list, reg_dist_list

def parse_args():
    parser = ArgsParser()
    parser.add_argument(
        "--output_dir",
        type=str,
        default="output_inference",
        help="Directory for storing the output model files.")
    parser.add_argument(
        "--export_serving_model",
        type=bool,
        default=False,
        help="Whether to export serving model or not.")
    parser.add_argument(
        "--slim_config",
        default=None,
        type=str,
        help="Configuration file of slim method.")
    args = parser.parse_args()
    return args

def run(FLAGS, cfg):
    # build detector
    trainer = Trainer(cfg, mode='test')

    # load weights
    if cfg.architecture in ['DeepSORT', 'ByteTrack']:
        trainer.load_weights_sde(cfg.det_weights, cfg.reid_weights)
    else:
        trainer.load_weights(cfg.weights)

    # change yoloe forward & yoloe-head forward
    trainer.model._forward = types.MethodType(yoloe_forward, trainer.model)
    trainer.model.yolo_head.forward = types.MethodType(head_forward, trainer.
↪model.yolo_head)
    # model.model.model[-1].forward = types.MethodType(forward2, model.model.
↪model[-1])
```

(continues on next page)

(continued from previous page)

```
    # export model
    trainer.export(FLAGS.output_dir)

    if FLAGS.export_serving_model:
        from paddle_serving_client.io import inference_model_to_serving
        model_name = os.path.splitext(os.path.split(cfg.filename)[-1])[0]

        inference_model_to_serving(
            dirname="{}/{}".format(FLAGS.output_dir, model_name),
            serving_server="{}/{}/serving_server".format(FLAGS.output_dir,
                                                         model_name),
            serving_client="{}/{}/serving_client".format(FLAGS.output_dir,
                                                         model_name),
            model_filename="model.pdmodel",
            params_filename="model.pdiparams")

def main():
    paddle.set_device("cpu")
    FLAGS = parse_args()
    cfg = load_config(FLAGS.config)
    merge_config(FLAGS.opt)

    if FLAGS.slim_config:
        cfg = build_slim_model(cfg, FLAGS.slim_config, mode='test')

    # FIXME: Temporarily solve the priority problem of FLAGS.opt
    merge_config(FLAGS.opt)
    check_config(cfg)
    if 'use_gpu' not in cfg:
        cfg.use_gpu = False
    check_gpu(cfg.use_gpu)
    check_version()

    run(FLAGS, cfg)

if __name__ == '__main__':
    main()
```

Then use the following command to export the non decoded onnx model of pp yoloe

```
python \
tools/export_model_no_decode.py \
-c configs/ppyoloe/ppyoloe_crn_s_300e_coco.yml \
-o weights=https://paddledet.bj.bcebos.com/models/ppyoloe_crn_s_300e_coco.
↪pdparams

paddle2onnx \
--model_dir \
output_inference/ppyoloe_crn_s_300e_coco \
```

(continues on next page)

(continued from previous page)

```
--model_filename model.pdmodel \
--params_filename model.pdiparams \
--opset_version 11 \
--save_file output_inference/ppyoloe_crn_s_300e_coco/ppyoloe_crn_s_300e_coco.
↪onnx
```

Parameters refer to the parameter settings exported from the official version

## 8.3 Onnx Model Conversion cvi model

The CVIModel conversion operation can refer to the onnx model conversion CVIModel section in the YOLO-V5 porting section.

## 8.4 TDL SDK Interface Description

###Preprocessing parameter settings

Preprocessing parameter settings are passed in through a structure to set parameters

```
typedef struct {
  float factor[3];
  float mean[3];
  meta_rescale_type_e rescale_type;

  bool use_quantize_scale;
  PIXEL_FORMAT_E format;
} YoloPreParam;
```

For YOLOX, the following four parameters need to be passed in:

- Factor preprocessing scale parameter
- Mean preprocessing mean parameter
- Use_Quantify_Does scale use the size of the model? The default is true
- Format image format, PIXEL_FORMAT_RGB_888_PLANAR

###Algorithm parameter settings

```
typedef struct {
  uint32_t cls;
} YoloAlgParam;
```

The number of categories that need to be passed in, such as

```
YoloAlgParam p_yolo_param;
p_yolo_param.cls = 80;
```

The additional model confidence parameter settings and NMS threshold settings are as follows:

```
CVI_TDL_SetModelThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_PPYOLOE, conf_
↪threshold);
CVI_TDL_SetModelNmsThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_PPYOLOE, nms_
↪threshold);
```

Among them, conf_Threshold is the confidence threshold; Nms_Threshold is the nms threshold

###Test Demo

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <chrono>
#include <fstream>
#include <functional>
#include <iostream>
#include <map>
#include <sstream>
#include <string>
#include <vector>
#include "core.hpp"
#include "core/cvi_tdl_types_mem_internal.h"
#include "core/utils/vpss_helper.h"
#include "cvi_tdl.h"
#include "evaluation/cvi_tdl_media.h"
#include "sys_utils.hpp"

int main(int argc, char* argv[]) {
  int vpssgrp_width = 1920;
  int vpssgrp_height = 1080;
  CVI_S32 ret = MMF_INIT_HELPER2(vpssgrp_width, vpssgrp_height, PIXEL_FORMAT_
↪RGB_888, 1,
                                 vpssgrp_width, vpssgrp_height, PIXEL_FORMAT_RGB_
↪888, 1);
  if (ret != CVI_TDL_SUCCESS) {
    printf("Init sys failed with %#x!\n", ret);
    return ret;
  }

  cvitdl_handle_t tdl_handle = NULL;
  ret = CVI_TDL_CreateHandle(&tdl_handle);
  if (ret != CVI_SUCCESS) {
    printf("Create tdl handle failed with %#x!\n", ret);
    return ret;
  }
  printf("start pp-yoloe preprocess config \n");
  // // setup preprocess
  YoloPreParam p_preprocess_cfg;
```

(continues on next page)

(continued from previous page)

```
  float mean[3] = {123.675, 116.28, 103.52};
  float std[3] = {58.395, 57.12, 57.375};

  for (int i = 0; i < 3; i++) {
    p_preprocess_cfg.mean[i] = mean[i] / std[i];
    p_preprocess_cfg.factor[i] = 1.0 / std[i];
  }

  p_preprocess_cfg.use_quantize_scale = true;
  p_preprocess_cfg.format = PIXEL_FORMAT_RGB_888_PLANAR;

  printf("start yolo algorithm config \n");
  // setup yolo param
  YoloAlgParam p_yolo_param;
  p_yolo_param.cls = 80;

  printf("setup pp-yoloe param \n");
  ret = CVI_TDL_Set_PPYOLOE_Param(tdl_handle, &p_preprocess_cfg, &p_yolo_param);
  printf("pp-yoloe set param success!\n");
  if (ret != CVI_SUCCESS) {
    printf("Can not set PPYoloE parameters %#x\n", ret);
    return ret;
  }

  std::string model_path = argv[1];
  std::string str_src_dir = argv[2];

  float conf_threshold = 0.5;
  float nms_threshold = 0.5;
  if (argc > 3) {
    conf_threshold = std::stof(argv[3]);
  }

  if (argc > 4) {
    nms_threshold = std::stof(argv[4]);
  }

  printf("start open cvimodel...\n");
  ret = CVI_TDL_OpenModel(tdl_handle, CVI_TDL_SUPPORTED_MODEL_PPYOLOE, model_
↪path.c_str());
  if (ret != CVI_SUCCESS) {
    printf("open model failed %#x!\n", ret);
    return ret;
  }
  printf("cvimodel open success!\n");
  // set thershold
  CVI_TDL_SetModelThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_PPYOLOE, conf_
↪threshold);
```

(continues on next page)

(continued from previous page)

```
  CVI_TDL_SetModelNmsThreshold(tdl_handle, CVI_TDL_SUPPORTED_MODEL_PPYOLOE, nms_
↪threshold);

  std::cout << "model opened:" << model_path << std::endl;

  VIDEO_FRAME_INFO_S fdFrame;
  ret = CVI_TDL_ReadImage(str_src_dir.c_str(), &fdFrame, PIXEL_FORMAT_RGB_888);
  std::cout << "CVI_TDL_ReadImage done!\n";

  if (ret != CVI_SUCCESS) {
    std::cout << "Convert out video frame failed with :" << ret << ".file:" <<␣
↪str_src_dir
              << std::endl;
  }

  cvtdl_object_t obj_meta = {0};

  CVI_TDL_PPYoloE(tdl_handle, &fdFrame, &obj_meta);

  printf("detect number: %d\n", obj_meta.size);
  for (uint32_t i = 0; i < obj_meta.size; i++) {
    printf("detect res: %f %f %f %f %f %d\n", obj_meta.info[i].bbox.x1, obj_
↪meta.info[i].bbox.y1,
           obj_meta.info[i].bbox.x2, obj_meta.info[i].bbox.y2, obj_meta.info[i].
↪bbox.score,
           obj_meta.info[i].classes);
  }

  CVI_VPSS_ReleaseChnFrame(0, 0, &fdFrame);
  CVI_TDL_Free(&obj_meta);
  CVI_TDL_DestroyHandle(tdl_handle);

  return ret;
}
```

## 8.5 Test Result

Tested PPYOLOE_Crn_S_300e The performance comparison of the Coco model onnx and cvi model on the cv181x/2x/3x platform, where the threshold parameters are:

- Conf: 0.01
- Nms: 0.7
- Input resolution: 640 x 640

PPYOLOE_Crn_S_300e_Coco model official export method onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| pytorch | N/A | N/A | N/A | 60.5 | 43.1 |
| cv181x | 103.62 | 110.59 | 14.68 | Quantification failure | Quantification failure |
| cv182x | 77.58 | 111.18 | 14.68 | Quantification failure | Quantification failure |
| cv183x | Quantification failure | Quantification failure | Quantification failure | Quantification failure | Quantification failure |

TDL of ppyoloe_crn_s_300e_coco model SDK export method onnx performance:

| platform | Inference time (ms) | bandwidth (MB) | ION(MB) | MAP 0.5 | MAP 0.5-0.95 |
|---|---|---|---|---|---|
| onnx | N/A | N/A | N/A | 55.9497 | 39.8568 |
| cv181x | 101.15 | 103.8 | 14.55 | 55.36 | 39.1982 |
| cv182x | 75.03 | 104.95 | 14.55 | 55.36 | 39.1982 |
| cv183x | 30.96 | 80.43 | 13.8 | 55.36 | 39.1982s |