yolox Documentation

Release 0.2.0

yolox contributors

Mar 21, 2022

QUICK RUN


YOLOX
Exceeding YOLO series in 2021

CHAPTER

ONE

GET STARTED

1.1 1.Installation

Step1. Install YOLOX.

```
git clone git@github.com:Megvii-BaseDetection/YOLOX.git
cd YOLOX
pip3 install -U pip && pip3 install -r requirements.txt
pip3 install -v -e .  # or  python3 setup.py develop
```

Step2. Install pycocotools.

```
pip3 install cython; pip3 install 'git+https://github.com/cocodataset/cocoapi.git
↪#subdirectory=PythonAPI'
```

1.2 2.Demo

Step1. Download a pretrained model from the benchmark table.

Step2. Use either -n or -f to specify your detector's config. For example:

```
python tools/demo.py image -n yolox-s -c /path/to/your/yolox_s.pth --path assets/dog.
↪jpg --conf 0.25 --nms 0.45 --tsize 640 --save_result --device [cpu/gpu]
```

or

```
python tools/demo.py image -f exps/default/yolox_s.py -c /path/to/your/yolox_s.pth --
↪path assets/dog.jpg --conf 0.25 --nms 0.45 --tsize 640 --save_result --device [cpu/
↪gpu]
```

Demo for video:

```
python tools/demo.py video -n yolox-s -c /path/to/your/yolox_s.pth --path /path/to/
↪your/video --conf 0.25 --nms 0.45 --tsize 640 --save_result --device [cpu/gpu]
```

1.3 3.Reproduce our results on COCO

Step1. Prepare COCO dataset

```
cd <YOLOX_HOME>
ln -s /path/to/your/COCO ./datasets/COCO
```

Step2. Reproduce our results on COCO by specifying -n:

```
python tools/train.py -n yolox-s -d 8 -b 64 --fp16 -o [--cache]
                      yolox-m
                      yolox-l
                      yolox-x
```

- -d: number of gpu devices
- -b: total batch size, the recommended number for -b is num-gpu * 8
- –fp16: mixed precision training
- –cache: caching imgs into RAM to accelarate training, which need large system RAM.

Weights & Biases for Logging

To use W&B for logging, install wandb in your environment and log in to your W&B account using

```
pip install wandb
wandb login
```

Log in to your W&B account

To start logging metrics to W&B during training add the flag `--logger` to the previous command and use the prefix "wandb-" to specify arguments for initializing the wandb run.

```
python tools/train.py -n yolox-s -d 8 -b 64 --fp16 -o [--cache] --logger wandb wandb-
↪project <project name>
                      yolox-m
                      yolox-l
                      yolox-x
```

Multi Machine Training

We also support multi-nodes training. Just add the following args:

- –num_machines: num of your total training nodes
- –machine_rank: specify the rank of each node

When using -f, the above commands are equivalent to:

```
python tools/train.py -f exps/default/yolox-s.py -d 8 -b 64 --fp16 -o [--cache]
                      exps/default/yolox-m.py
                      exps/default/yolox-l.py
                      exps/default/yolox-x.py
```

1.4 4.Evaluation

We support batch testing for fast evaluation:

```
python tools/eval.py -n  yolox-s -c yolox_s.pth -b 64 -d 8 --conf 0.001 [--fp16] [--
↪fuse]
                         yolox-m
                         yolox-l
                         yolox-x
```

- –fuse: fuse conv and bn
- -d: number of GPUs used for evaluation. DEFAULT: All GPUs available will be used.
- -b: total batch size across on all GPUs

To reproduce speed test, we use the following command:

```
python tools/eval.py -n  yolox-s -c yolox_s.pth -b 1 -d 1 --conf 0.001 --fp16 --fuse
                         yolox-m
                         yolox-l
                         yolox-x
```

CHAPTER

TWO

MODEL ZOO

2.1 Standard Models.

2.2 Light Models.

CHAPTER

THREE

TRAIN CUSTOM DATA

This page explains how to train your own custom data with YOLOX.

We take an example of fine-tuning YOLOX-S model on VOC dataset to give a more clear guide.

3.1 0. Before you start

Clone this repo and follow the README to install YOLOX.

3.2 1. Create your own dataset

Step 1 Prepare your own dataset with images and labels first. For labeling images, you can use tools like Labelme or CVAT.

Step 2 Then, you should write the corresponding Dataset Class which can load images and labels through `__getitem__` method. We currently support COCO format and VOC format.

You can also write the Dataset by your own. Let's take the VOC Dataset file for example:

```
    @Dataset.resize_getitem
    def __getitem__(self, index):
        img, target, img_info, img_id = self.pull_item(index)

        if self.preproc is not None:
            img, target = self.preproc(img, target, self.input_dim)

        return img, target, img_info, img_id
```

One more thing worth noting is that you should also implement pull_item and load_anno method for the `Mosiac` and `MixUp` augmentations.

Step 3 Prepare the evaluator. We currently have COCO evaluator and VOC evaluator. If you have your own format data or evaluation metric, you can write your own evaluator.

Step 4 Put your dataset under `$YOLOX_DIR/datasets`, for VOC:

```
ln -s /path/to/your/VOCdevkit ./datasets/VOCdevkit
```

- The path "VOCdevkit" will be used in your exp file described in next section. Specifically, in `get_data_loader` and `get_eval_loader` function.

You can download the mini-coco128 dataset by the link, and then unzip it to the `datasets` directory. The dataset has been converted from YOLO format to COCO format, and can be used directly as a dataset for testing whether the train environment can be runned successfully.

3.3 2. Create your Exp file to control everything

We put everything involved in a model to one single Exp file, including model setting, training setting, and testing setting.

A complete Exp file is at yolox_base.py. It may be too long to write for every exp, but you can inherit the base Exp file and only overwrite the changed part.

Let's take the VOC Exp file as an example.

We select `YOLOX-S` model here, so we should change the network depth and width. VOC has only 20 classes, so we should also change the `num_classes`.

These configs are changed in the `init()` method:

```
class Exp(MyExp):
    def __init__(self):
        super(Exp, self).__init__()
        self.num_classes = 20
        self.depth = 0.33
        self.width = 0.50
        self.exp_name = os.path.split(os.path.realpath(__file__))[1].split(".")[0]
```

Besides, you should also overwrite the `dataset` and `evaluator`, prepared before training the model on your own data.

Please see get_data_loader, get_eval_loader, and get_evaluator for more details.

You can also see the `exps/example/custom` directory for more details.

3.4 3. Train

Except special cases, we always recommend to use our COCO pretrained weights for initializing the model.

Once you get the Exp file and the COCO pretrained weights we provided, you can train your own model by the following below command:

```
python tools/train.py -f /path/to/your/Exp/file -d 8 -b 64 --fp16 -o -c /path/to/the/
↪pretrained/weights [--cache]
```

- –cache: we now support RAM caching to speed up training! Make sure you have enough system RAM when adopting it.

or take the `YOLOX-S` VOC training for example:

```
python tools/train.py -f exps/example/yolox_voc/yolox_voc_s.py -d 8 -b 64 --fp16 -o -
↪c /path/to/yolox_s.pth [--cache]
```

For example:

- If you download the mini-coco128 and unzip it to the `datasets`, you can direct run the following training code.

```
python tools/train.py -f exps/example/custom/yolox_s.py -d 8 -b 64 --fp16 -o -c /
↪path/to/yolox_s.pth
```

(Don't worry for the different shape of detection head between the pretrained weights and your own model, we will handle it)

3.5 4. Tips for Best Training Results

As **YOLOX** is an anchor-free detector with only several hyper-parameters, most of the time good results can be obtained with no changes to the models or training settings. We thus always recommend you first train with all default training settings.

If at first you don't get good results, there are steps you could consider to improve the model.

Model Selection We provide `YOLOX-Nano`, `YOLOX-Tiny`, and `YOLOX-S` for mobile deployments, while `YOLOX-M/L/X` for cloud or high performance GPU deployments.

If your deployment meets any compatibility issues. we recommend `YOLOX-DarkNet53`.

Training Configs If your training overfits early, then you can reduce max_epochs or decrease the base_lr and min_lr_ratio in your Exp file:

```
# ---------------  training config ---------------------- #
    self.warmup_epochs = 5
    self.max_epoch = 300
    self.warmup_lr = 0
    self.basic_lr_per_img = 0.01 / 64.0
    self.scheduler = "yoloxwarmcos"
    self.no_aug_epochs = 15
    self.min_lr_ratio = 0.05
    self.ema = True

    self.weight_decay = 5e-4
    self.momentum = 0.9
```

Aug Configs You may also change the degree of the augmentations.

Generally, for small models, you should weak the aug, while for large models or small size of dataset, you may enchance the aug in your Exp file:

```
# --------------- transform config ----------------- #
    self.degrees = 10.0
    self.translate = 0.1
    self.scale = (0.1, 2)
    self.mosaic_scale = (0.8, 1.6)
    self.shear = 2.0
    self.perspective = 0.0
    self.enable_mixup = True
```

Design your own detector You may refer to our Arxiv paper for details and suggestions for designing your own detector.

CHAPTER

FOUR

YOLOX-TENSORRT IN PYTHON

This tutorial includes a Python demo for TensorRT.

4.1 Install TensorRT Toolkit

Please follow the TensorRT Installation Guide and torch2trt gitrepo to install TensorRT and torch2trt.

4.2 Convert model

YOLOX models can be easily conveted to TensorRT models using torch2trt

If you want to convert our model, use the flag -n to specify a model name:

```
python tools/trt.py -n <YOLOX_MODEL_NAME> -c <YOLOX_CHECKPOINT>
```

For example:

```
python tools/trt.py -n yolox-s -c your_ckpt.pth
```

<YOLOX_MODEL_NAME> can be: yolox-nano, yolox-tiny. yolox-s, yolox-m, yolox-l, yolox-x.

If you want to convert your customized model, use the flag -f to specify you exp file:

```
python tools/trt.py -f <YOLOX_EXP_FILE> -c <YOLOX_CHECKPOINT>
```

For example:

```
python tools/trt.py -f /path/to/your/yolox/exps/yolox_s.py -c your_ckpt.pth
```

yolox_s.py can be any exp file modified by you.

The converted model and the serialized engine file (for C++ demo) will be saved on your experiment output dir.

4.3 Demo

The TensorRT python demo is merged on our pytorch demo file, so you can run the pytorch demo command with `--trt`.

```
python tools/demo.py image -n yolox-s --trt --save_result
```

or

```
python tools/demo.py image -f exps/default/yolox_s.py --trt --save_result
```

CHAPTER

FIVE

YOLOX-TENSORRT IN C++

As YOLOX models are easy to convert to tensorrt using torch2trt gitrepo, our C++ demo does not include the model converting or constructing like other tenorrt demos.

5.1 Step 1: Prepare serialized engine file

Follow the trt python demo README to convert and save the serialized engine file.

Check the 'model_trt.engine' file generated from Step 1, which will be automatically saved at the current demo dir.

5.2 Step 2: build the demo

Please follow the TensorRT Installation Guide to install TensorRT.

And you should set the TensorRT path and CUDA path in CMakeLists.txt.

If you train your custom dataset, you may need to modify the value of `num_class`.

```
const int num_class = 80;
```

Install opencv with `sudo apt-get install libopencv-dev` (we don't need a higher version of opencv like v3.3+).

build the demo:

```
mkdir build
cd build
cmake ..
make
```

Then run the demo:

```
./yolox ../model_trt.engine -i ../../../../assets/dog.jpg
```

or

```
./yolox <path/to/your/engine_file> -i <path/to/image>
```

CHAPTER

SIX

YOLOX-CPP-MEGENGINE

Cpp file compile of YOLOX object detection base on MegEngine.

6.1 Tutorial

6.1.1 Step1: install toolchain

```
* host: sudo apt install gcc/g++ (gcc/g++, which version >= 6) build-essential git␣
↪git-lfs gfortran libgfortran-6-dev autoconf gnupg flex bison gperf curl zlib1g-dev␣
↪gcc-multilib g++-multilib cmake
```

- cross build android: download NDK
  - after unzip download NDK, then export NDK_ROOT="path of NDK"

6.1.2 Step2: build MegEngine

```
git clone https://github.com/MegEngine/MegEngine.git

# then init third_party

export megengine_root="path of MegEngine"
cd $megengine_root && ./third_party/prepare.sh && ./third_party/install-mkl.sh

# build example:
# build host without cuda:
./scripts/cmake-build/host_build.sh
# or build host with cuda:
./scripts/cmake-build/host_build.sh -c
# or cross build for android aarch64:
./scripts/cmake-build/cross_build_android_arm_inference.sh
# or cross build for android aarch64(with V8.2+fp16):
./scripts/cmake-build/cross_build_android_arm_inference.sh -f

# after build MegEngine, you need export the `MGE_INSTALL_PATH`
# host without cuda:
export MGE_INSTALL_PATH=${megengine_root}/build_dir/host/MGE_WITH_CUDA_OFF/MGE_
↪INFERENCE_ONLY_ON/Release/install
# or host with cuda:
export MGE_INSTALL_PATH=${megengine_root}/build_dir/host/MGE_WITH_CUDA_ON/MGE_
↪INFERENCE_ONLY_ON/Release/install
```

(continues on next page)

(continued from previous page)

```
# or cross build for android aarch64:
export MGE_INSTALL_PATH=${megengine_root}/build_dir/android/arm64-v8a/Release/install
```

- you can refs build tutorial of MegEngine to build other platform, eg, windows/macos/ etc!

6.1.3 Step3: build OpenCV

```
git clone https://github.com/opencv/opencv.git

git checkout 3.4.15 (we test at 3.4.15, if test other version, may need modify some␣
↪build)
```

- patch diff for android:

```
# ```
#     diff --git a/CMakeLists.txt b/CMakeLists.txt
#     index f6a2da5310..10354312c9 100644
#     --- a/CMakeLists.txt
#     +++ b/CMakeLists.txt
#     @@ -643,7 +643,7 @@ if(UNIX)
#        if(NOT APPLE)
#          CHECK_INCLUDE_FILE(pthread.h HAVE_PTHREAD)
#          if(ANDROID)
#     -      set(OPENCV_LINKER_LIBS ${OPENCV_LINKER_LIBS} dl m log)
#     +      set(OPENCV_LINKER_LIBS ${OPENCV_LINKER_LIBS} dl m log z)
#          elseif(CMAKE_SYSTEM_NAME MATCHES "FreeBSD|NetBSD|DragonFly|OpenBSD|Haiku")
#            set(OPENCV_LINKER_LIBS ${OPENCV_LINKER_LIBS} m pthread)
#          elseif(EMSCRIPTEN)

# ```
```

- build for host

```
cd root_dir_of_opencv
mkdir -p build/install
cd build
cmake -DBUILD_JAVA=OFF -DBUILD_SHARED_LIBS=ON -DCMAKE_INSTALL_PREFIX=$PWD/install
make install -j32
```

- build for android-aarch64

```
cd root_dir_of_opencv
mkdir -p build_android/install
cd build_android

cmake -DCMAKE_TOOLCHAIN_FILE="$NDK_ROOT/build/cmake/android.toolchain.cmake" -
↪DANDROID_NDK="$NDK_ROOT"  -DANDROID_ABI=arm64-v8a -DANDROID_NATIVE_API_LEVEL=21 -
↪DBUILD_JAVA=OFF -DBUILD_ANDROID_PROJECTS=OFF -DBUILD_ANDROID_EXAMPLES=OFF -DBUILD_
↪SHARED_LIBS=ON -DCMAKE_INSTALL_PREFIX=$PWD/install ..

make install -j32
```

- after build OpenCV, you need export `OPENCV_INSTALL_INCLUDE_PATH` and `OPENCV_INSTALL_LIB_PATH`

```
# host build:
export OPENCV_INSTALL_INCLUDE_PATH=${path of opencv}/build/install/include
export OPENCV_INSTALL_LIB_PATH=${path of opencv}/build/install/lib
# or cross build for android aarch64:
export OPENCV_INSTALL_INCLUDE_PATH=${path of opencv}/build_android/install/sdk/native/
↪jni/include
export OPENCV_INSTALL_LIB_PATH=${path of opencv}/build_android/install/sdk/native/
↪libs/arm64-v8a
```

6.1.4 Step4: build test demo

```
run build.sh

# if host:
export CXX=g++
./build.sh
# or cross android aarch64
export CXX=aarch64-linux-android21-clang++
./build.sh
```

6.1.5 Step5: run demo

Note: two ways to get `yolox_s.mge` model file

- reference to python demo's `dump.py` script.
- For users with code before 0.1.0 version, wget yolox-s weights here.
- For users with code after 0.1.0 version, use python code in megengine to generate mge file.

```
# if host:
LD_LIBRARY_PATH=$MGE_INSTALL_PATH/lib/:$OPENCV_INSTALL_LIB_PATH ./yolox yolox_s.mge ..
↪/../../assets/dog.jpg cuda/cpu/multithread <warmup_count> <thread_number>

# or cross android
adb push/scp $MGE_INSTALL_PATH/lib/libmegengine.so android_phone
adb push/scp $OPENCV_INSTALL_LIB_PATH/*.so android_phone
adb push/scp ./yolox yolox_s.mge android_phone
adb push/scp ../../../assets/dog.jpg android_phone

# login in android_phone by adb or ssh
# then run:
LD_LIBRARY_PATH=. ./yolox yolox_s.mge dog.jpg cpu/multithread <warmup_count> <thread_
↪number> <use_fast_run> <use_weight_preprocess>  <run_with_fp16>

# * <warmup_count> means warmup count, valid number >=0
# * <thread_number> means thread number, valid number >=1, only take effect␣
↪`multithread` device
# * <use_fast_run> if >=1 , will use fastrun to choose best algo
# * <use_weight_preprocess> if >=1, will handle weight preprocess before exe
# * <run_with_fp16> if >=1, will run with fp16 mode
```

6.2 Bechmark

- model info: yolox-s @ input(1,3,640,640)
- test devices

```
  * x86_64  -- Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz
  * aarch64 -- xiamo phone mi9
  * cuda    -- 1080TI @ cuda-10.1-cudnn-v7.6.3-TensorRT-6.0.1.5.sh @ Intel(R) Xeon(R)␣
↪CPU E5-2620 v4 @ 2.10GHz
```

6.3 Acknowledgement

CHAPTER

SEVEN

YOLOX-PYTHON-MEGENGINE

Python version of YOLOX object detection base on MegEngine.

7.1 Tutorial

7.1.1 Step1: install requirements

```
python3 -m pip install megengine -f https://megengine.org.cn/whl/mge.html
```

7.1.2 Step2: convert checkpoint weights from torch's path file

```
python3 convert_weights.py -w yolox_s.pth -o yolox_s_mge.pkl
```

7.1.3 Step3: run demo

This part is the same as torch's python demo, but no need to specify device.

```
python3 demo.py image -n yolox-s -c yolox_s_mge.pkl --path ../../../assets/dog.jpg --
↪conf 0.25 --nms 0.45 --tsize 640 --save_result
```

7.1.4 [Optional]Step4: dump model for cpp inference

> **Note**: result model is dumped with `optimize_for_inference` and `enable_fuse_conv_bias_nonlinearity`.

```
python3 dump.py -n yolox-s -c yolox_s_mge.pkl --dump_path yolox_s.mge
```

CHAPTER

EIGHT

YOLOX-ANDROID-NCNN

Andoird app of YOLOX object detection base on ncnn

8.1 Tutorial

8.1.1 Step1

Download ncnn-android-vulkan.zip from releases of ncnn. This repo uses 20210525 release for building.

8.1.2 Step2

After downloading, please extract your zip file. Then, there are two ways to finish this step:

- put your extracted directory into **app/src/main/jni**
- change the **ncnn_DIR** path in **app/src/main/jni/CMakeLists.txt** to your extracted directory

8.1.3 Step3

Download example param and bin file from onedrive or github. Unzip the file to **app/src/main/assets**.

8.1.4 Step4

Open this project with Android Studio, build it and enjoy!

8.2 Reference

CHAPTER

NINE

YOLOX-CPP-NCNN

Cpp file compile of YOLOX object detection base on ncnn.YOLOX is included in ncnn now, you could also try building from ncnn, it's better.

9.1 Tutorial

9.1.1 Step1

Clone ncnn first, then please following build tutorial of ncnn to build on your own device.

9.1.2 Step2

Use provided tools to generate onnx file. For example, if you want to generate onnx file of yolox-s, please run the following command:

```
cd <path of yolox>
python3 tools/export_onnx.py -n yolox-s
```

Then, a yolox.onnx file is generated.

9.1.3 Step3

Generate ncnn param and bin file.

```
cd <path of ncnn>
cd build/tools/ncnn
./onnx2ncnn yolox.onnx model.param model.bin
```

Since Focus module is not supported in ncnn. Warnings like:

```
Unsupported slice step !
```

will be printed. However, don't worry! C++ version of Focus layer is already implemented in yolox.cpp.

9.1.4 Step4

Open **model.param**, and modify it. Before (just an example):

```
295 328
Input            images                   0 1 images
Split            splitncnn_input0         1 4 images images_splitncnn_0 images_
↪splitncnn_1 images_splitncnn_2 images_splitncnn_3
Crop             Slice_4                  1 1 images_splitncnn_3 647 -23309=1,0 -
↪23310=1,2147483647 -23311=1,1
Crop             Slice_9                  1 1 647 652 -23309=1,0 -23310=1,2147483647 -
↪23311=1,2
Crop             Slice_14                 1 1 images_splitncnn_2 657 -23309=1,0 -
↪23310=1,2147483647 -23311=1,1
Crop             Slice_19                 1 1 657 662 -23309=1,1 -23310=1,2147483647 -
↪23311=1,2
Crop             Slice_24                 1 1 images_splitncnn_1 667 -23309=1,1 -
↪23310=1,2147483647 -23311=1,1
Crop             Slice_29                 1 1 667 672 -23309=1,0 -23310=1,2147483647 -
↪23311=1,2
Crop             Slice_34                 1 1 images_splitncnn_0 677 -23309=1,1 -
↪23310=1,2147483647 -23311=1,1
Crop             Slice_39                 1 1 677 682 -23309=1,1 -23310=1,2147483647 -
↪23311=1,2
Concat           Concat_40                4 1 652 672 662 682 683 0=0
...
```

- Change first number for 295 to 295 - 9 = 286(since we will remove 10 layers and add 1 layers, total layers number should minus 9).
- Then remove 10 lines of code from Split to Concat, but remember the last but 2nd number: 683.
- Add YoloV5Focus layer After Input (using previous number 683):

```
YoloV5Focus      focus                    1 1 images 683
```

After(just an example):

```
286 328
Input            images                   0 1 images
YoloV5Focus      focus                    1 1 images 683
...
```

9.1.5 Step5

Use ncnn_optimize to generate new param and bin:

```
# suppose you are still under ncnn/build/tools/ncnn dir.
../ncnnoptimize model.param model.bin yolox.param yolox.bin 65536
```

9.1.6 Step6

Copy or Move yolox.cpp file into ncnn/examples, modify the CMakeList.txt, then build yolox

9.1.7 Step7

Inference image with executable file yolox, enjoy the detect result:

```
./yolox demo.jpg
```

9.2 Acknowledgement

CHAPTER

TEN

YOLOX-ONNXRUNTIME IN PYTHON

This doc introduces how to convert your pytorch model into onnx, and how to run an onnxruntime demo to verify your convertion.

10.1 Download ONNX models.

10.2 Convert Your Model to ONNX

First, you should move to <YOLOX_HOME> by:

```
cd <YOLOX_HOME>
```

Then, you can:

1. Convert a standard YOLOX model by -n:

```
python3 tools/export_onnx.py --output-name yolox_s.onnx -n yolox-s -c yolox_s.pth
```

Notes:

- -n: specify a model name. The model name must be one of the [yolox-s,m,l,x and yolox-nane, yolox-tiny, yolov3]
- -c: the model you have trained
- -o: opset version, default 11. **However, if you will further convert your onnx model to OpenVINO, please specify the opset version to 10.**
- –no-onnxsim: disable onnxsim
- To customize an input shape for onnx model, modify the following code in tools/export.py:

```
dummy_input = torch.randn(1, 3, exp.test_size[0], exp.test_size[1])
```

1. Convert a standard YOLOX model by -f. When using -f, the above command is equivalent to:

```
python3 tools/export_onnx.py --output-name yolox_s.onnx -f exps/default/yolox_s.py -c␣
↪yolox_s.pth
```

1. To convert your customized model, please use -f:

```
python3 tools/export_onnx.py --output-name your_yolox.onnx -f exps/your_dir/your_
↪yolox.py -c your_yolox.pth
```

10.3 ONNXRuntime Demo

Step1.

```
cd <YOLOX_HOME>/demo/ONNXRuntime
```

Step2.

```
python3 onnx_inference.py -m <ONNX_MODEL_PATH> -i <IMAGE_PATH> -o <OUTPUT_DIR> -s 0.3␣
↪--input_shape 640,640
```

Notes:

- -m: your converted onnx model
- -i: input_image
- -s: score threshold for visualization.
- –input_shape: should be consistent with the shape you used for onnx convertion.

CHAPTER

ELEVEN

YOLOX-OPENVINO IN PYTHON

This tutorial includes a Python demo for OpenVINO, as well as some converted models.

11.1 Download OpenVINO models.

11.2 Install OpenVINO Toolkit

Please visit Openvino Homepage for more details.

11.3 Set up the Environment

11.3.1 For Linux

Option1. Set up the environment tempororally. You need to run this command everytime you start a new shell window.

```
source /opt/intel/openvino_2021/bin/setupvars.sh
```

Option2. Set up the environment permenantly.

Step1. For Linux:

```
vim ~/.bashrc
```

Step2. Add the following line into your file:

```
source /opt/intel/openvino_2021/bin/setupvars.sh
```

Step3. Save and exit the file, then run:

```
source ~/.bashrc
```

11.4 Convert model

1. Export ONNX model

   Please refer to the ONNX toturial. **Note that you should set –opset to 10, otherwise your next step will fail.**

2. Convert ONNX to OpenVINO

```
cd <INSTSLL_DIR>/openvino_2021/deployment_tools/model_optimizer
```

Install requirements for convert tool

```
sudo ./install_prerequisites/install_prerequisites_onnx.sh
```

Then convert model.

```
python3 mo.py --input_model <ONNX_MODEL> --input_shape <INPUT_SHAPE> [--data_type␣
↪FP16]
```

For example:

```
python3 mo.py --input_model yolox.onnx --input_shape [1,3,640,640] --data_type␣
↪FP16 --output_dir converted_output
```

11.5 Demo

11.5.1 python

```
python openvino_inference.py -m <XML_MODEL_PATH> -i <IMAGE_PATH>
```

or

```
python openvino_inference.py -m <XML_MODEL_PATH> -i <IMAGE_PATH> -o <OUTPUT_DIR> -s
↪<SCORE_THR> -d <DEVICE>
```

CHAPTER

TWELVE

YOLOX-OPENVINO IN C++

This tutorial includes a C++ demo for OpenVINO, as well as some converted models.

12.1 Download OpenVINO models.

12.2 Install OpenVINO Toolkit

Please visit Openvino Homepage for more details.

12.3 Set up the Environment

12.3.1 For Linux

Option1. Set up the environment tempororally. You need to run this command everytime you start a new shell window.

```
source /opt/intel/openvino_2021/bin/setupvars.sh
```

Option2. Set up the environment permenantly.

Step1. For Linux:

```
vim ~/.bashrc
```

Step2. Add the following line into your file:

```
source /opt/intel/openvino_2021/bin/setupvars.sh
```

Step3. Save and exit the file, then run:

```
source ~/.bashrc
```

12.4 Convert model

1. Export ONNX model

   Please refer to the ONNX toturial. **Note that you should set –opset to 10, otherwise your next step will fail.**

2. Convert ONNX to OpenVINO

   ```
   cd <INSTSLL_DIR>/openvino_2021/deployment_tools/model_optimizer
   ```

   Install requirements for convert tool

   ```
   sudo ./install_prerequisites/install_prerequisites_onnx.sh
   ```

   Then convert model.

   ```
   python3 mo.py --input_model <ONNX_MODEL> --input_shape <INPUT_SHAPE> [--data_type␣
   ↪FP16]
   ```

   For example:

   ```
   python3 mo.py --input_model yolox_tiny.onnx --input_shape [1,3,416,416] --data_
   ↪type FP16
   ```

   Make sure the input shape is consistent with those in cpp file.

12.5 Build

12.5.1 Linux

```
source /opt/intel/openvino_2021/bin/setupvars.sh
mkdir build
cd build
cmake ..
make
```

12.6 Demo

12.6.1 c++

```
./yolox_openvino <XML_MODEL_PATH> <IMAGE_PATH> <DEVICE>
```