EMBEDEEP
壹 普 智 慧

EEP-TPU 编译器使用手册

eep-ug050 (v0.6.1)

2023-02-01

厦门壹普智慧科技有限公司

修订历史：

版本	日期	描述	作者
0.1.0	2019-03-18	初始版本	何 xx
0.2.0	2020-05-12	V0.2	何 xx
0.3.0	2021-02-01	V0.3	何 xx
0.4.0	2021-12-11	V0.4	何 xx
0.5.0	2022-04-01	V0.5	何 xx
0.6.0	2022-011-09	V0.6	何 xx
0.6.1	2023-02-01	Github release	何 xx

目　　录

1、前言

本文档是 EEP-TPU 编译器的使用说明。

编译器版本：**eeptpu_compiler v2.4.1 及以上。**

编译器编译和运行环境：X86 桌面版 Ubuntu 18.04 LTS（在其他 Linux 系统下运行若遇到问题，请安装 Ubuntu18.04 后重试）。

2、使用

在 X86 桌面版 Ubuntu 系统上的命令行终端执行。

2.1 命令行参数

```
./eeptpu_compiler -h
Usage:
    ../eeptpu_compiler [options]
options:
    --help(-h)              # Print this help message
    --version(-v)           # Print version
  For caffe:
    --prototxt   <path>     # Caffe prototxt file path
    --caffemodel <path>     # Caffe caffemodel file path
  For onnx:
    --onnx <path>           # Onnx file path
  For darknet:
    --darknet_cfg <path>    # Darknet cfg file path
    --darknet_weight <path> # Darknet weight file path
  For ncnn:
    --ncnn_param <path>     # Ncnn param file path
    --ncnn_bin   <path>     # Ncnn bin file path
  For keras:
    --keras_flag <n>        # Keras flag, set n=1 when the net model is from keras h5 file.
  Other params:
    --image <path>           # One typical input image for this neural network. (Support formats: jpg,pgm,bmp,png)
    --pixel_order <str>      # Input pixel order(support: RGB, BGR, RGBA, BGRA, GRAY, default). Default is BGR. Darknet will auto use RGB.
    --input_img <path>      # Same as "--image"
    --input_npy <path>      # One typical input data for this neural network. ( npy format )
    --input_shape <shape>   # Neural network input shape, string format, 4 dims. Such as '1,3,224,224'. (batch, channel, height, width)
    --input_list    <list>                  #    For    multi-inputs.    Format: 'name1;path1;[option1]#name2;path2;[option2]#...'. Use 'Netron' tool to find the input
```

```
names.
                    The 'name' and 'path' are necessary, optional paramaters can be: mean, norm, pack_type, pack_shape, del_last_channels.
                    Each input use '#' to seperate; each paramater in same input use ';' to seperate.
                    Example1:
'input1;./image1.jpg#input2;./data2.npy#input3;./data3.npy'
                    Example2:
'input1;./image1.jpg;mean:103.94,116.78,123.68;norm:0.017,0.017,0.017;pack_type:7;pack_shape:3,256,256#input2;./data2.npy;del_last_channels:1'
    --int8                  # Enable int8 quantization mode.
    --input_folder <path>   # For int8 quantize mode.
    --quant_method <n>      # 0(none),1(kl). Default is 1.
    --quantize <n>          # Quantization mode.
                              n=1: normal int8 quantization, same as '--int8';
                              n=2: improved quantization mode, use quantized model and quantize table;
                              n=3: EF8 quantization mode, use quantized model and quantize table.
    --trunc_mode <n>         # Truncation mode for 'quantize=2' mode. Default is n=0. If n=1, will do truncation before bias in 4D module.
    --qwt_mode <n>           # Quantize weight mode(for 'quantize=2'). Default is n=0, weight data use quantize range [-127,127]; if n=1, use [-128,127].
    --qtable <path>         # Quantize table for quantized model.(Each line format: "Layer_name,shift_value")
    --mean <values>          # Mean values. Format: float array string, such as "103.94,116.78,123.68"
    --norm <values>          # Normalize values. Format: float array string, such as "0.017,0.017,0.017"
    --hw_mean               # Use hareware to process mean&norm for input data
    --opt <n>                # Optimization options, default is 1; If set to 0, will not optimize network.
    --extinfo <ext>          # Extend info. Store customized format string to pass to your application.
    --output <folder>       # Output bin file to this folder
    --public_bin            # Generate 'public' mode bin file (xxx.pub.bin).
    --hybp                  # Use hybrid precision mode.
    --input_pad <l,r,t,b>   # Add padding to input data. Order: left, right, top, bottom.
    --del_last_softmax      # Auto remove last softmax layer.
    --del_last_swlayers      # Auto delete software layers that at the end of
```

```
network.
    --del_last_channels <n> # Auto remove input data's last n channels.
    --jump <n>              # Jump mode. When net1 done, will jump to net2 to run.
n=1: net1; n=2: net2.
    --tpu_threads <n>       # Use TPU multi-cores threads mode, n is TPU threads
count. (Need TPU support!)
    --pack_type <n>         # Set input raw data directly to TPU. Input data memory
layout: HWC order(default), CHW order.
                              Value n is input data format. 1:float32; 2:float16;
3:int8; 4:int16; 5:int32; 7:uint8
                              If set this param, will auto set '--hw_mean' too.

    --pack_shape <shape>    # Set pack output shape. Such as '3,256,256' (channel,
height, width).
```

2.2 参数说明

对于 caffe 平台：

--prototxt：prototxt 文件的路径。

--caffemodel：caffemodel 文件的路径。

对于 onnx 平台：

--onnx：模型文件的路径，目前仅支持 pytorch 框架导出的 onnx。在 pytorch 导出 onnx 时，建议使用：

torch.onnx.export(model, input_read, your_onnx_path, verbose=True, opset_version=11)

对于 darknet 平台：

--darknet_cfg：cfg 文件的路径。

--darknet_weight：weight 文件的路径。

对于 ncnn 平台：

--ncnn_param：param 文件的路径。

--ncnn_bin：bin 文件的路径。

对于 keras 平台：

--keras_flag：keras 标志位。目前版本编译器对 keras 的支持有限，需要进行以下转换：第一步，keras 需导出 h5 文件；第二步，通过 keras_convert 工具将 h5 文件转换为 ncnn 平台格式；第三步，将“--ncnn_param, --ncnn_bin, --keras_flag”这 3 个参数传入编译器。之后可以就生成 eeptpu.bin 文件。

量化模式：（需 EEP-TPU 硬件支持）

--int8：使能 int8 量化编译模式（等价于“--quantize 1”，推荐使用“--quantize 1”）。

--input_folder：int8 量化编译时需要指定该神经网络的多个典型输入数据用于量化，我们推荐该输入数据的数量大于 5000 个（若仅用于测试评估，此处数量可任意）。该参数用于指定输入数据的文件夹路径。

--quant_method：选择量化方法。0-none，使用原始阈值；1-，使用 KL 量化方法（默认选择）。

--quantize <n>：量化选项。目前支持 3 种模式：

n=1 时：等价于--int8，是传统的 int8 量化模式。跟“--int8”一样，使用时需配合--input_folder。

n=2 时：移位量化模式。需要传入已量化过的网络模型和量化表。

n=3 时：EF8 移位量化模式。需要传入已量化过的网络模型和量化表。

--qtable <path>：量化表路径。文件内容示例：

Conv_0,6

Conv_4,5

Conv_6,7

每行的格式为：层名称，移位值。（两者用英文逗号隔开）。例如上面第一行的“Conv_0”为该网络中一个层的名称，第一行的 6 表示该层计算出来的数据需右移 6 位。数据移位后，按四舍五入方式取整。

--trunc_mode <n>：用于量化模式 2（--quantize 2），选择饱和截断模式。（量化模式为 2 时（--quantize 2），饱和截断模式的选择。）默认值为 n=0，表示 4D 计算后面不

做饱和截断。n=1 时表示 4D 计算后面要做饱和截断。

--qwt_mode <n>：用于量化模式 2（--quantize 2），选择量化范围。（量化模式为 2 时（--quantize 2），量化范围的选择。）默认值为 n=0，表示量化范围是-127 到 127；n=1 时表示量化范围是-128 到 127。

其他参数：

--image（或--input_img）：适用于当前这个神经网络的一张典型图片的路径。支持 jpg、bmp、png、pgm 等格式的图片。

--pixel_order：用于设置输入图像的像素格式，以字符串形式传入，支持 RGB，BGR，RGBA，BGRA，GRAY 等像素格式。默认格式为 BGR；若使用 Darknet 网络，编译器会自动将其设置为 RGB（Darknet 网络测试程序中，用户需自行将图片读入为 RGB 格式）。

--input_npy：适用于当前神经网络的一个典型输入数据，npy 文件格式。获取 npy 文件的一种方式是可由 python numpy 导出。（可以通过 python numpy 导出 npy 文件。）Npy 文件支持的数据格式为：f4(float32)，f2(float16)，i4(int32)，i2(short)，u1(unsigned char)，i1(char)；Fortran order 为 False。

--input_shape：输入数据的维度。通常编译器会自动识别输入维度；若编译器无法获取到神经网络的输入维度时，可使用这个选项来指定输入维度。

--input_list：用于多输入网络的输入参数设置。当一个网络只有一个输入时，可使用不同的输入参数（例如--input_img，--mean，--norm 等）来配置参数。当一个网络拥有多个输入时，则必须使用“--input_list”来配置，格式如下：

输入 1 名称;输入 1 数据文件路径;[可选参数 1]# 输入 2 名称;输入 2 数据文件路径;[可选参数 2]#.....# 输入 N 名称;输入 N 数据文件路径;[可选参数 N]

其中：

输入名称：输入 blob 的名称，可使用 Netron 工具来查看。

输入名称和输入路径是必须配置的。其他可选参数可根据实际需求来设置。可选参数包括：mean，norm，pack_type，pack_shape，del_last_channels。

每个输入配置之间用**英文**井号“#”来分隔；同一输入的不同参数之间用**英文**分号

“;”分隔；可选参数的参数名和参数值之间用**英文**冒号“:”分隔。以上符号**均为英文字符。注意输入文件路径里不要包含“#”和“;”。**

示例 1：

--input_list 'input1;./image1.jpg#input2;./data2.npy#input3;./data3.npy'

示例 2：

--input_list 'input1;./image1.jpg;mean:103.94,116.78,123.68;norm:0.017,0.017,0.017;pack_type:7;pack_shape:3,256,256#input2;./data2.npy;del_last_channels:1'

✧ **多输入模式下的量化编译**：与单输入一样，需要配置“--quantize 1 --input_folder ./inputs/”。主要区别在于输入文件夹里面的数据存放。在单输入模式下，只需要把所有的输入数据文件存放在 input_folder 指定的文件夹即可。而在多输入模式下，input_folder 指定的文件夹下有特定要求的存放格式（以./inputs/为例）：

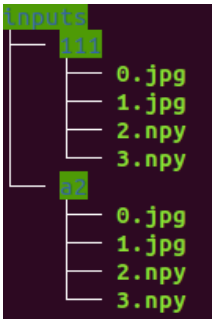
多输入模式下，将一次推理所需要的多个输入数据组成一组输入数据，每组输入数据包含多个输入文件。

Inputs 文件夹中可包含多个子文件夹，用于存放输入数据，子文件夹的名称可任意命名。各个子文件夹中需存放 N 个输入数据，这 N 个输入数据的命名有要求，文件名为“输入 ID 号+后缀”，例如 0.jpg, 1.npy。

获取输入 ID 号的方式：用编译器预先按非量化模式来编译网络，此时会在命令行打印出各个输入数据的信息，其中就有包括 Input ID。例如：

```
Neural network inputs:
    InputID[0] 1,3,224,224; name: image_left
    InputID[1] 1,3,224,224; name: image_right
    InputID[2] 1,3,224,224; name: image_face
    InputID[3] 1,625,1,1; name: facegrid
```

文件夹结构示例（网络包含 4 个输入）：



--mean, --norm：用于均值、归一化。

均值-归一化计算公式：X' = (X - mean) * norm。其中：X 为输入的值；X'为均值归一化计算后的值，作为神经网络的输入。均值-归一化的输入需与神经网络输入的通道数一致。例如：输入通道数是 3，则 mean、norm 也需要是一个 3 个数值的数组，每个数值对应一个通道。目前不支持文件形式。

如何获取均值-归一化的值：一般可以从该神经网络的训练文件（例如：train.prototxt）中获取。以 MobilenetV1 为例，在其训练文件中可以找到：

```
transform_param {
    scale: 0.017
    mirror: true
    crop_size: 224
    mean_value: [103.94, 116.78, 123.68]
    }
```

mean 从“mean_value”获取：103.94, 116.78, 123.68；

norm 从“scale”获取，并扩展到 3 通道：0.017, 0.017, 0.017；

注意：Pytorch 中的均值-归一化：一般以 mean、std 来命名，其中 std 对应上述的 norm。

均值-归一化相当于对输入数据的预处理。以 python 中用 opencv 对图像预处理为例：

```
img = img / 255
img -= 0.5
img /= 0.5
```

从上述处理中可得到：

input = ((img / 255) - 0.5) / 0.5

*= (img - 0.5*255) * 2 / 255*

*= (img - 127.5) * 0.007843137*

得到 mean=127.5，norm=0.007843137。扩展到 3 通道数据后，即为： *--mean "127.5,127.5,127.5" --norm "0.007843137,0.007843137,0.007843137"*。

--hw_mean：使用 EEP-TPU 硬件计算均值归一化。

例如传入参数"--mean 值列表 --norm 值列表 --hw_mean"，编译器即可将该均值归一化配置为硬件计算。

若配置了该选项，均值归一化由 TPU 来计算，CPU 相关前处理程序则不需要再另行计算均值归一化。

--opt：神经网络优化选项。默认开启（n=1），可优化某些网络结构。设置为 0 则关闭优化。

--extinfo：自定义的扩展配置字符串。用户可自定义写入的格式，可在用户程序中通过 API 函数读出并由用户自行解析。用户可以从编译阶段保存一些特定数据到 bin 文件中，并在应用程序中使用。例如，用户可以将目标检测网络的目标名称列表保存在"--extinfo"指定的字符串中，然后在测试程序中通过 API 函数将其取出。

--output：保存编译生成 bin 文件的文件夹名（不是文件名）。

--public_bin：用于编译生成 public 版本的 bin 文件。默认后缀为.pub.bin。这个 public 版本 bin 文件，需配合 *libeeptpu_pub.so* 库文件来使用。当前的 public 版本编译器，尚不提供软件层和硬件层交替使用的支持。

用户在编译 public 版本 bin 文件时，下列两种情况可以成功：

（1）全部算子可硬件支持，即全硬件推理；

（2）神经网络可分为前后两大部分，前面部分的算子全部硬件支持，后面部分的算子由软件支持（库文件软件计算），即先硬件推理，然后再由软件推理并得出整体网络推理结果。例如，yolo 网络最后面的 detection_output 层可由软件计算，mobilenet 等分类网络最后面的 softmax 层可由软件计算。

当神经网络中出现不支持硬件计算的层时，编译器会进行错误提示。

--hybp：使用混合精度模式进行编译。如果不加这个参数，则默认是使用单精度模式。

EEP-TPU 有单一精度和混合精度两种不同模式，混合精度模式下可以使用 INT8 量化。对于 EEP-TPU 来说，单精度和混合精度的 bin 文件不能兼容使用，即 EEP-TPU 编译器编译出来的单精度 bin 文件只能在支持单精度的 EEP-TPU 硬件上使用；编译出来的混合精度 bin 文件只能在支持混合精度的 EEP-TPU 硬件上使用。

在 EEP-TPU 编译器编译结束时，会输出该 bin 文件适用的平台：

Generated 'eeptpu.bin' for platform: EEPTPU C8S1

其中，“S1”代表的是单精度；“S2”代表的是混合精度。

在 EEP-TPU SDK 提供的示例程序中，可读取 EEP-TPU 硬件的配置，例如：

EEPTPU hardware info: EEP-TPU;M1024;N1;C8;LS2;S1;

配置信息最后的“S1”代表的即为单精度；如果这个信息是“S2”，则为混合精度。

在应用程序中，如果 bin 文件和 EEP-TPU 的精度模式不匹配，则会报错。这时需要重新编译 bin 文件以适配 EEP-TPU 的精度模式：

Error: EEP-TPU use S2, but bin file use S1.

--input_pad <l,r,t,b>：对输入数据加 padding 操作。仅限于使用在网络第一个层是卷积层的情况。使用该参数后，原网络模型的输入维度将发生变化，新的输入维度加上 padding 值才等于原有的输入维度。例如，原始网络输入维度是 3×210×210，编译时，通过“--input_pad ‘5,5,5,5’”在新的输入数据的左右上下各设置 padding=5，那么新的输入维度变成 3×200×200。

--del_last_softmax：常用于分类网络。若神经网络以 softmax 层结束，则在编译阶段自动把最后一层的 softmax 去掉，实际上不影响分类结果。

--del_last_swlayers：在编译阶段将网络最后面不支持 TPU 硬件计算的层移除。（某些情况下可代替上述 del_last_softmax）。

--del_last_channels：将输入数据最后的 N 个 channel 删除。例如，某个网络对应的

输入为 3 通道的图片（BGR），而输入图片是 4 通道（BGRA），使用这个参数就可以自动将输入图片的最后一个通道删除，使其适配该网络的输入维度。

--jump：双算法跳转模式，支持从算法 1 跳转到算法 2 执行。算法 1 和算法 2 编译时都需要加这个参数。算法 1 使用“--jump 1”；算法 2 使用“--jump 2”。算法 1 的输出作为算法 2 的输入，要求算法 1 的输出维度与算法 2 的输入维度一致。若算法 2 有做均值归一化处理，则算法 2 在编译时需要加参数“--hw_mean”将其硬件化处理。

--tpu_threads <n>：多线程配置，用多线程联合对网络进行推理。参数 n 为线程数量。一个线程对应一个 TPU 核。一般情况下，当 EEP-TPU 硬件包含多个核时，使用多线程方式，可以提高网络的推理速度。

--pack_type <n> --pack_shape <c,h,w>：设置“pack 模式”的参数。“Pack 模式”下，不同精度的输入数据可被转换成 TPU 所需的格式。输入数据以 HWC 的 layout 格式存在，与 opencv 的数据格式相同。

编译器支持 FP32/FP16/INT8/INT16/INT32/UINT8 精度转换，可使用 TPU 将 pack 模式输入数据转换成 TPU 所需的 EF16 数据格式。

使用 PACK 时，输入数据维度必须固定。特别是对于分类网络，需要事先将输入图片转换成一个统一的维度，与编译器编译时“--pack_type <n> --pack_shape ‘c,h,w’”所指定的维度一致。

使用 pack 模式时，输入数据是一维数据（例如 1,1,196608）。Pack_shape 指定的是 pack 算子的输出维度，数值顺序是“C,H,W”（例如 3,256,256；3×256×256=196608）。R 如果神经网络输入维度与 pack 算子输出维度不一致，编译器会自动在 pack 算子后加入一个 resize 操作，将数据 resize 到网络输入大小。因此，pack_shape 维度不一定要与神经网络输入维度一致。

Pack_type 取值：

1：FP32； 2：FP16； 3：INT8；4：INT16； 5：INT32； 7：UINT8。

当该神经网络有均值归一化时，关于均值归一化的处理如下：

（1）可由应用程序处理，传入给编译器的 mean 选项值为全 0，norm 选项值为全 1，但此模式仅适用于浮点类型的输入数据。

（2）或者由编译器处理。

使用精度转换模式时，resize 和均值归一化都可由硬件处理。以 mobilenet 为例，该神经网络输入是 3×224×224。Pack_shape 设置为 3×256×256，编译时需要输入一维的 196608×N 字节的数据（N 的取值：fp32/int32 时 N=4，fp16/int16 时 N=2，int8 时 N=1）。TPU 会自动将该输入数据 resize 到 3×224×224，并做均值归一化处理。

精度转换模式下的输入数据格式：一维数据。以图像为例，数据格式为 HWC 排列方式。通过 opencv 读出来的图像数据排列方式就是 HWC（bgrbgrbgr....)。数据大小需要与编译阶段“pack_shape”指定的大小一致。例如 pack_shape 指定了“3,256,256”，那么输入的数据大小是 3×256×256=196608。

输入数据为图像且使用 pack 模式时，pack_type 设置为 7（uint8 类型，取值范围是 0 到 255)。

2.3 使用示例

- **Caffe Mobilenet:**

eeptpu_compiler --prototxt /path/to/prototxt --caffemodel /path/to/caffemodel --image /path/to/typical/image --mean '103.94,116.78,123.68' --norm '0.017,0.017,0.017' --output ./

- **Onnx:**

eeptpu_compiler --onnx /path/to/onnx_file --input_npy /path/to/typical/input_npy_file --output ./

- **DarkNet:**

eeptpu_compiler --darknet_cfg /path/to/darknet_cfg_file --darknet_weight /path/to/darknet_weight_file --input_img /path/to/typical/input_image_file --mean '0.0,0.0,0.0' --norm '0.003921569,0.003921569,0.003921569' --output ./

- **Ncnn Mobilenet:**

eeptpu_compiler --ncnn_param /path/to/ncnn_param --ncnn_bin /path/to/ncnn_bin --image /path/to/typical/image --mean '103.94,116.78,123.68' --norm '0.017,0.017,0.017' --output ./

- **Keras:**

eeptpu_compiler --ncnn_param /path/to/ncnn_param --ncnn_bin /path/to/ncnn_bin --image /path/to/typical/image --mean '103.94,116.78,123.68' --norm '0.017,0.017,0.017' --output ./ --keras_flag 1

2.4 单精度与混合精度

EEP-TPU 处理器拥有单精度与混合精度两种架构，这两种架构不可相互兼容。编译器通过“--hybp”选项来指定架构，请确保 EEP-TPU 硬件与编译器拥有相同的架构选择。

单精度模式

单精度模式是指数据格式为 FP16 的模式，该模式下 EEP-TPU 仅支持普通算子。有关普通算子请参考《eep-ug004 EEP-TPU 算子列表》。在该模式下，用户需要准备一张与算法对应的典型数据（如图片），通过内置于编译器的 EEP-TPU 行为模型进行一次推理计算，从而初步验证 EEP-TPU 是否支持该算法。相关编译器选项和使用说明，请参考 2.2 节和 2.3 节。

混合精度模式

混合精度模式是指数据同时为 FP16 和 INT8 的模式。该模式下，EEP-TPU 可同时支持普通算子、Quantize 算子和 FQuantize 算子，具体可参考《eep-ug004 EEP-TPU 算子列表》。

混合精度模式下：

（1）可以仅使用 FP16 计算精度，其使用方法与单精度一致，相关编译器选项和使用说明，请参考 2.2 节和 2.3 节。

（2）也可以使用 FP16+INT8 的混合精度计算，此时编译器会优先使用 Quantize 算子与 FQuantize 算子。要使用混合精度计算，用户需要准备多个输入数据用于编译器对相应参数进行 INT8 量化，输入数据的数量建议大于 5000 个。混合精度编译时，编译器选项需要增加如下选项：--hybp --int8 --input_folder。例如：

> INT8:

eeptpu_compiler --int8 --input_folder /path/to/images_folder/ --output ./ --mean '103.94,116.78,123.68' --norm '0.017,0.017,0.017' --prototxt /path/to/prototxt --caffemodel /path/to/caffemodel --image /path/to/typical/image

3、兼容性

非 INT8 版本：编译器所生成的 eeptpu bin 文件，需要搭配使用的 EEPTPU libeeptpu.so API 版本为：V2.0 及以上（public bin 版本使用的 libeeptpu_pub.so API 版本为 V0.6.4 及以上）；需要搭配使用的 EEP-TPU 硬件版本为：0.7.2-R2 及以上。

INT8 量化版本：编译器所生成的 eeptpu bin 文件，需要搭配使用的 EEPTPU libeeptpu.so API 版本为：V2.0 及以上（public bin 版本使用的 libeeptpu_pub.so API 版本为 V0.6.4 及以后）；需要搭配使用支持 INT8 的 EEP-TPU 硬件。

EEP-TPU 编译器支持 Caffe/Darknet/PyTorch(onnx)/Keras/NCNN 框架的神经网络模型。其他平台（例如 Mxnet）生成的模型文件，可通过一些对应的模型转换工具转换成上述格式。

EEP-TPU 编译器支持软硬件协同计算，对于暂不能通过 EEP-TPU 硬件进行加速的层类型，可由软件进行计算。

EEP-TPU 编译器支持多个输出层。通过 EEPTPU API 函数，可将多个层的推理结果数据一起读取出来。

4、不同平台的神经网络如何使用 EEP-TPU 编译器

4.1 Caffe

EEP-TPU 编译器支持 caffe 平台的神经网络模型。

以 mobilenet-v1 模型为例：

eeptpu_compiler --prototxt /path/to/prototxt --caffemodel /path/to/caffemodel --input_img /path/to/typical/image --mean '103.94,116.78,123.68' --norm '0.017,0.017,0.017' --output ./

通过“--prototxt”指定模型的 prototxt 文件的路径；

通过“--caffemodel”指定模型的 caffemodel 文件的路径；

通过“--input_img”指定该神经网络的一张典型的输入图片的路径；

通过“--mean”和“--norm”指定该神经网络的均值、归一化值。

Caffe 平台下，均值一般可以从该神经网络的训练文件（例如：train.prototxt）中获取。以 MobilenetV1 为例：

```
transform_param {
    scale: 0.017
    mirror: true
    crop_size: 224
    mean_value: [103.94, 116.78, 123.68]
    }
```

mean 从“mean_value”获取：103.94, 116.78, 123.68；

norm 从“scale”获取，并扩展到 3 通道：0.017, 0.017, 0.017。

4.2 Darknet

EEP-TPU 编译器支持 darknet 平台的神经网络模型。

以 darknet yolov3 模型为例：

eeptpu_compiler --darknet_cfg /path/to/darknet_cfg_file --darknet_weight /path/to/darknet_weight_file --input_img /path/to/typical/input_image_file --mean '0.0,0.0,0.0' --norm '0.003921569,0.003921569,0.003921569' --output ./

--darknet_cfg：指定模型的 cfg 文件的路径；

--darknet_weight：指定模型的 weights 文件的路径；

--input_img：指定该神经网络的一张典型的输入图片的路径（与“--image”具有相同功能）。

均值、归一化：darknet 平台神经网络的归一化值为 1/255.0（即 0.003921569）。所以 mean 为 0，norm 为 0.003921569。输入图像为 3 通道的话，则将 mean 和 norm 扩展到 3 个通道的值：

--mean '0.0,0.0,0.0' --norm '0.003921569,0.003921569,0.003921569'

Darknet 的输入图像的顺序是 RGB 格式，在编译器编译 darknet 网络时会自动将输入图像处理成 RGB 顺序。例如，使用 opencv 读图时，图像的顺序是 BGR，用户在程序编写时需要注意这一点。用户在测试程序中，也需要自行将图像调整为 RGB 顺序。

4.3 Pytorch

EEP-TPU 编译器需要通过 ONNX 来支持 Pytorch 平台的模型。用户需要将 pytorch 模型文件转换成 onnx 文件，并经“onnx_post”工具处理，然后通过 EEP-TPU 编译器进行编译。

（1）将 Pytorch 网络模型转换成 ONNX 格式

第一步，使用 pytorch 的 onnx export 功能 （torch.onnx.export），将 pytorch 网络模型转换成 onnx 格式文件。**需要特别注意的是，使用 pytorch export 功能时切记区分主**

干网络和“前处理&后处理”函数。通常情况下，EEP-TPU 仅支持主干网络的计算，前处理&后处理（如 YOLO 的后处理函数 Detectionout）函数不能通过 onnx 输出，必须通过 C 函数来完成。Onnx export 示例：

torch.onnx.export(model, input_read, your_onnx_path, verbose=True, opset_version=11)

第二步，onnx 通常拥有众多无用的胶水算子，这些算子在计算中无任何作用，需要被优化。可以通过 onnx_post 工具对 onnx 文件进行模型优化，得到新的 onnx 文件。之后，就可以使用 eeptpu 编译器进行编译。

（2）Onnx_post 工具

Onnx_post 工具由本公司提供。

使用环境：X86 PC，Ubuntu 18.04。

使用方式：打开命令行，执行如下命令：

./onnx_post input_model output_model [--input-shape 模型输入大小]

例如：*./onnx_post model.onnx new.onnx --input-shape 1,2,128,233*

其中：

model.onnx 为 pytorch 导出的 onnx 文件；

new.onnx 为 onnx_post 工具生成的新的 onnx 文件；

input-shape 为该神经网络模型的输入尺寸（可选项）。

（3）eeptpu_compiler 编译 onnx 文件

通过 onnx_post 工具生成的 new.onnx 模型文件可使用 eeptpu_compiler 进行编译。

命令示例：

./eeptpu_compiler --output ./ --onnx /path/to/onnx_file --input_img /path/to/typical/image

其中：

--onnx：经过 onnx_post 工具处理后的新的 onnx 文件的路径。

--input_img：该神经网络的一张典型的输入图片的路径。

4.4 Keras （TensorFlow Based）

Keras 网络模型需要先转换为 ncnn 模型，然后再通过 EEP-TPU 编译器进行编译。

（1）Keras_convert 工具

Keras_convert 工具由本公司提供，可将 keras 网络导出的 H5 格式的模型文件转换为 ncnn 模型格式。

使用环境：X86 PC, Ubuntu 18.04。

使用说明：keras_convert src_h5_path dst_folder

例如：*./keras_convert ./src.h5 ./*

执行完毕后，会在目标文件夹下生成 output.param 和 output.bin 文件。

（2）使用 eeptpu 编译器编译

Keras_convert 工具生成的 output.param 和 output.bin 是 ncnn 模型格式。通过编译器“--ncnn_param”和“--ncnn_bin”指定模型文件，且需要通过“--keras_flag 1”指示该模型文件来自 keras 框架。

编译脚本示例：

./eeptpu_compiler --output ./ --ncnn_param ./output.param --ncnn_bin ./output.bin --input_npy ./input_data.npy --keras_flag 1

其中：

--ncnn_param：指定由 keras_convert 工具生成的 output.param 文件路径；

--ncnn_bin：指定由 keras_convert 工具生成的 output.bin 文件路径；

--input_npy：指定该神经网络模型的一个典型输入文件路径。以 npy 为例: npy 文件可通过 python 生成，若为该文件为图片，可以把参数改为“--input_img”。

--keras_flag：需要设置为 1，标识当前网络是来自 keras 框架。

4.5 Tensorflow2

需要先将 Tensorflow2 模型转为 mlir 文件，然后再通过 EEP-TPU 编译器进行编译。

当前版本的 EEP-TPU 编译器还无法支持 mlir 文件，更高版本的编译器正在开发中。

5、编译器预处理辅助工具

EEP-TPU 编译器的预处理辅助工具，主要包含 Keras 和 ONNX（Pytorch）的转换工具，用于辅助完成 Keras 和 Pytorch 框架下 AI 算法的编译工作。

5.1 Keras_convert

详见本文档 4.4 节。

5.2 Onnx_post

适用于 pytorch 导出的 onnx 模型文件的优化处理，详见本文档 4.3 节。

6、常见使用问题

6.1 “Hardware on-chip memory not enough”问题

神经网络里面某层的数据过大，导致 EEP-TPU 内部资源不足。

解决方法：尽量减少神经网络中数据过大的层的输出或者参数量。例如，输出维度大时，可尽量修改网络使输出维度小一点；参数量太大时，尽量修改网络使该层的参数量变小。

或者可以联系本公司技术人员进行协助：herh@embedeep.com。

6.2 Caffe

（1）若编译器编译 caffe 模型时，出现下述错误：

Analysing caffe files...

[libprotobuf FATAL ……/protobuf_2.6.1/include/google/protobuf/repeated_field.h:886] CHECK failed: (index) < (size()):

terminate called after throwing an instance of 'google::protobuf::FatalException'

what(): CHECK failed: (index) < (size()):

解决方法：检查这个 caffe 模型文件是否是错误格式，如果是，可通过 caffe 工具 upgrade_net_proto_text 和 upgrade_net_proto_binary 对 caffe 模型进行转换，例如：

upgrade_net_proto_text ./deploy.prototxt new.prototxt

upgrade_net_proto_binary ./deploy.caffemodel new.caffemodel

上述两个工具可在 caffe 源码里进行编译得到。

6.3 Darknet

（1）编译成 eeptpu.bin 后，硬件计算所得目标框数量与 darknet 软件不一致的问题

需确认 cfg 文件里的 ignore_thresh 值是否一致。darknet 的 ignore_thresh 默认是 0.25，

nms_thresh 默认是 0.45。

eeptpu 编译器在编译 darknet 时，会输出当前使用的 ignore_thresh 和 nms_thresh 值。

ignore_thresh 使用的是 cfg 文件最后一个[yolo]层的 ignore_thresh 值。

可手动修改 cfg 文件最后一个[yolo]层的 ignore_thresh 值，或者在[yolo]里添加 nms_thresh 参数（若不添加，则默认为 0.45，与 darknet 默认值一致）。

（2）输入图像的格式

Darknet 的输入图像的格式是 RGB 顺序。

编译器对 darknet 网络进行编译时，会默认将输入图像转换成 RGB 顺序格式。

如果测试程序中用 opencv 读图，opencv 读出来的图像是 BGR 格式，因此在进行推理之前需要将 BGR 格式转换成 RGB 格式。

（3）均值/归一化

Darknet 官方的归一化值是 1/255，因此编译器需要设置均值：（以 3 通道输入为例）

```
--mean '0.0,0.0,0.0' --norm '0.003921569,0.003921569,0.003921569'
```

6.4 ONNX

目前仅支持 pytorch 框架导出的 onnx 模型。通过 Pytorch 导出 onnx 模型文件后，需要再使用 EEP 提供的 onnx_post 工具对模型文件进行处理。生成新的 onnx 文件后，再进行编译。

（1）包含 interpolate 或 Upsample 算子时

插值模式需要为双线性插值，需将 align_corners 配置为 True。

通过 pytorch 导出 onnx 模型文件时，需将 opset_version 配置为 11，否则可能出现兼容性问题。

```
torch.onnx.export(model, input_read, your_onnx_path, verbose=True, opset_version=11)
```

（2）onnx_post 工具

使用 onnx_post 工具时，若发现对一些算子不支持；或者经过该工具处理后，编译时发现对一些算子不支持。可能时当前的编译器版本对该模型文件尚不支持，可以联系我们（herh@embedeep.com）进行协助处理。

6.5 输入编译器的数据为用户自行处理的数据

用户可自行处理输入数据（例如自定义的预处理、减均值或归一化等）。

可将数据保存为 npy 格式，数据类型支持：f4（float32），f2（float16），i4（int32），i2（short），u1（unsigned char），i1（char）；Fortran order 为 False。

用户给的输入数据若已经做了预处理，则编译器编译时无需使用 ‘—mean’和‘—norm’的均值选项（即默认 mean 为 0，norm 为 1）。

但是在应用程序中，用户需对输入数据进行同样的预处理，然后调用 eeptpu 库文件中的 eeptpu_set_input 函数：

```
int eeptpu_set_input(void* input_data, int dim1, int dim2, int dim3, int mode = 0);
```

这里的 mode 要设置为 1，表示 input_data 为 float 型数组指针。input_data 经过了用户的预处理，可直接作为该神经网络的输入数据。

npy 文件：可在 python 中用 numpy 保存。

npy 文件里，保存的数据为单个输入数据。

若 npy 的 Fortran order 为 True，可通过如下 python 脚本将其转换为 False：

```
import numpy as np
data = np.load("input_fortran.npy")      # input is: 'fortran_order': True
print(np.isfortran(data))
data = data.copy(order='C')          # convert to: 'fortran_order': False
print(np.isfortran(data))
np.save("input_new.npy", data)
```