模型开发工具

Caffe

使用文档

Version 4.0.1.0

目录

1 关于文档

本文档包含 Gyrfalcon Technology Inc.专有和机密的信息，仅供接收方用于评估或使用 Gyrfalcon Technology 的产品。此外接收方应明确表示不会将该文档的内容泄露给第三方或以其他方式盗用本文中的信息。

美国专利申请中。Gyrfalcon Technology logo 是 Gyrfalcon Technology Inc. 的注册商标。

联系 Gyrfalcon

硅谷总部

1900 麦卡锡大道

208 号

米尔皮塔斯，加利福尼亚州 95035

美国

中国

中国广东省深圳市南山区科技南 12 路，长虹科技大厦 2411 室

官网

www.gyrfalcontech.ai

2 概述

GTILightspeeur®芯片系列由利用模型压缩和量化技术的低功耗高性能 AI 加速器芯片组成。

本使用文档介绍了如何使用模型开发工具包（MDK），包括模型训练工具和模型转换工具。模型转换工具提取预训练模型的参数，然后将它们转换为可以部署到 GTI 芯片中的格式。

该 MDK 依赖于开源 Caffe 框架，并假设用户精通 Caffe 并熟悉典型的浮点模型训练和部署过程。此版本的 Caffe MDK 旨在与 GTI2801，GTI2803 和 GTI5801 芯片一起使用，并支持基于 Mobilenet，Resnet 和 VGG 的 CNN 架构。建议在训练过程中使用支持 Nvidia CUDA 的 GPU。

发行说明

版本号	芯片	支持的网络结构	更新信息	更新日期
2.0	2803	Mobilenet，Resnet	最初的 2803 版本	2019/1/25
2.1	2803	Mobilenet，Resnet，VGG	为中间层的 2803 输出增加了 VGG 支持	2019/3/20
3.0	2803	Mobilenet，Resnet，VGG	在自定义图层的量化设置中添加了 5801 支持更新	2019/4/6
	5801	Mobilenet，Resnet，VGG		
3.1.0.0	2801	VGG	添加了 2801 支持； 添加了上采样支持。	2019/4/29
	2803	Mobilenet，Resnet，VGG		
	5801	Mobilenet，Resnet，VGG		
3.2.0.0	2801	VGG	添加了级联模式的支持； 添加了自动 ping-pong 缓冲区地址计算，并删除了.json 中的 ping-pong 地址	2019/5/10
	2803	Mobilenet，Resnet，VGG		
	5801	Mobilenet，Resnet，VGG		
3.2.1.0	2801	VGG	为 GTI2803 和 GTI5801 添加了 10	2019/5/24
	2803	Mobilenet，Resnet，VGG		
	5801	Mobilenet，Resnet，VGG		
3.2.2.0	2801	VGG	增加了对 GTI2803 和 GTI5801 的 4 个输入通道的支持。	2019/6/5
	2803	Mobilenet，Resnet，VGG		
	5801	Mobilenet，Resnet，VGG		
3.2.3.0	2801	VGG	为 GTI2803 和 GTI5801 增加了 10 位激活（仅用于最后的卷积层）。	2019/6/14
	2803	Mobilenet，Resnet，VGG		
	5801	Mobilenet，Resnet，VGG		
3.2.4.0	2801	VGG	修复了 Mobilenet 偏差计算的	2019/6/25

	2803	Mobilenet, Resnet，VGG	错误； 添加了调试模式转储移位文件。	
	5801	Mobilenet, Resnet，VGG		
3.2.6.0	2801	VGG	支持深度卷积的加速，以加快模型训练的速度； 在样本池之前，为中间层输出添加了样本池层； 添加了量化数据层。	2019/7/15
	2803	Mobilenet, Resnet，VGG		
	5801	Mobilenet, Resnet，VGG		
3.2.8.0	2801	VGG	添加了对 Python3 的支持； 将 quant_enable 添加到 input data 层，并在整个训练过程中进行； 增加了对 GTI 2803 的 5 位量化的支持。	2019/7/22
	2803	Mobilenet, Resnet，VGG		
	5801	Mobilenet, Resnet，VGG		
3.2.10.0	2801	VGG	修复了有关偏差量化中裁剪阈值计算的错误。	2019/10/17
	2803	Mobilenet, Resnet，VGG		
	5801	Mobilenet, Resnet，VGG		
4.0.0.0	2801	VGG	●增加了对 GTI SDK v5.0 的支持； ●修复了转换工具中与卷积核求和和 1x1 卷积层有关的错误； ●修复了 CPU_ONLY 选项的 caffe 安装； ●添加了用于模型调试的新部分（请参见『7.10 调试模型』）。在转换脚本（“评估”）中添加了新的可选参数，用于 GPU / CPU 与芯片之间的位匹配。 ●当输出大小小于 GTI2801 的预期大小时，添加了一条警告消息。	2019/11/15
	2803	Mobilenet, Resnet，VGG		
	5801	Mobilenet, Resnet，VGG		
4.0.1.0	5801	VGG-SSD、Mobilenet-SSD	●增加对 Caffe SSD 的安装及示例；	2020/04/14

3 Linux 操作系统上的 GTI MDK 支持

Gyrfalcon Technology Inc.为针对 AI 和机器学习的应用程序开发提供硬件和软件。SDK 包括用于 AI 学习和推理的驱动程序，库和源代码。MDK 软件包提供工具和参考示例，以训练和验证针对 GTI Plai Plug 优化的模型。一旦成功生成模型，可将其用于 SDK 软件包中的参考应用程序中进行验证。

注：不需要 GTI Plai Plug 来执行训练和转换。支持的 SDK 版本是 v4.5 和更高版本。

3.1 所需硬件和软件的要求

所需硬件设备：

- Linux（Ubuntu 16.04,64 位版本）
- Intel i5 或更好的处理器（i7 首选）3.0 GHz 或更快
- 16GB 或更多 RAM
- 至少一个 USB 3.0 端口
- GPU 1070 TI 8G 或以上

第三方软件：

- Python 2.7 或 3.5
- OpenCV 3.2.0
- GNU 编译器（最低要求 g++版本 4.9，首选 5.4）
- Caffe 1.0
- Cuda 8.0 或 9.0 以及 CUDNN 6 或 7

本文档假定读者熟悉使用命令“caffe train”和“caffe test”以及“solver.prototxt”在 Caffe 中训练模型的标准方法。

一些可选的示例工作流程说明中可能还存在其他 Python 依赖项，只需使用 pip 安装缺少的 Python 依赖项。

3.2 安装 Caffe

有关安装 Caffe-1.0 的详细说明，请访问 *http://caffe.berkeleyvision.org/*。

- 下载原始的 Caffe 1.0 版并安装依赖项；有关详细说明，请参见：

  https://github.com/BVLC/caffe/wiki/Ubuntu-16.04-or-15.10-Installation-Guide

（也可以通过 "git clone https://github.com/BVLC/caffe” 下载）

- 对于其他操作系统，请参考：

  http://caffe.berkeleyvision.org/installation.html?utm_source=jiji.io

- 使用"-DUSE_CUDNN=1 -DUSE_NCCL=1" 或 "-DCPU_ONLY=1" 启用/禁用 GPU。

附：Caffe 1.0 安装教程参考：

链接：https://pan.baidu.com/s/110XWpXHK3D9BY6nuCIylfg
提取码：m8f0

3.3 安装 CUDA 和 cuDNN

可以在 NVIDIA 网站上找到安装 CUDA 和 cuDNN 的详细说明，网址：https://developer.nvidia.com/cuda-zone

简要说明如下：

1) 下载 CUDA 8 debian 文件：

- wget

  http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/cuda-repo-ubuntu1604_8.0.61-1_amd64.deb

- wget

  http://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1604/x86_64/libcudnn6_6.0.21-1%2Bcuda8.0_amd64.deb

- wget

  http://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1604/x86_64/libcudnn6-dev_6.0.21-1%2Bcuda8.0_amd64.deb

2) 安装 CUDA 8 和 cuDNN 6 所需的库，如下所示：

- `sudo dpkg -i cuda-repo-ubuntu1604_8.0.61-1_amd64.deb`
- `sudo dpkg -i libcudnn6_6.0.21-1+cuda8.0_amd64.deb`
- `sudo dpkg -i libcudnn6-dev_6.0.21-1+cuda8.0_amd64.deb`
- `sudo apt-get update`
- `sudo apt-get install cuda=8.0.61-1`
- `sudo apt-get install libcudnn6-dev`

3) 重新启动系统以加载 NVIDIA 驱动程序。

4) 通过修改 PATH 和 LD_LIBRARY_PATH 变量来设置开发环境，并将它们添加到 .bashrc 文件的末尾：

```
export PATH=/usr/local/cuda-8.0/bin${PATH:+:${PATH}}
export
LD_LIBRARY_PATH=/usr/local/cuda-8.0/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PAT
```

```
H}}
```

5) 运行 nvidia-smi 以验证安装。

注意：如果您的系统中没有 GPU，则安装 NVIDIA 驱动程序可能会导致一些问题，例如出现无限登录循环的情况。要解决此问题，请登录到文本模式，然后删除 NVIDIA 驱动程序，删除代码为：`sudo apt-get purge nvidia*`

3.4 安装 Caffe-SSD

请从网盘下载 GTI Caffe-SSD 安装包，里面已经包含 GTI MDK 文件，无需进行进行第四步 MDK 安装。

链接：https://pan.baidu.com/s/1agW64mREYUiqomcLZOn7IQ

提取码：b6l0

解压到本地目录，先 make clean ;make

4 GTI MDK 安装

4.1 MDK 文件结构

GTI MDK 软件包以压缩包格式提供，您可以将其解压缩到本地文件系统。

解压后文件夹包括以下文件：

- \ caffe_installation：用于更新 Caffe-1.0 安装的文件。
- \ conversion_tool：转换工具的实用程序，以及示例配置 JSON 文件。
- \ convert.py：转换工具。
- \ train：用于训练和评估的示例脚本。
- \ README.txt：有关 MDK 用法的说明。

执行：

```
chmod 777 -R release/
```

4.2 安装

MDK 包括来自 Caffe 1.0 的升级版 Caffe，用于训练与 GTI 兼容的 Caffe 模型。模型训练在主处理器上运行，不需要 GTI 芯片或 PLAI 插件。

4.2.1 更新 Caffe 框架

根据下面列出的两个选项修改 Caffe 框架：

1) 根据表 1 中列出的说明更新现有的 Caffe 1.0 框架。所有必需的文件都位于“caffe-installation”目录中。

文件	复制到
caffe.proto	CAFFE_DIRECTORY/src/caffe/proto/ (如果使用 1.0 以外的 Caffe 版本，则必须手动**合并**而不是复制文件，以便添加新的参数定义。)
quant_conv_layer.cpp	CAFFE_DIRECTORY/src/caffe/layers/
quant_conv_layer.cu	CAFFE_DIRECTORY/src/caffe/layers/
quant_conv_layer.hpp	CAFFE_DIRECTORY/include/caffe/layers/
quant_relu_layer.cpp	CAFFE_DIRECTORY/src/caffe/layers/
quant_relu_layer.cu	CAFFE_DIRECTORY/src/caffe/layers/
quant_relu_layer.hpp	CAFFE_DIRECTORY/include/caffe/layers/
filler.hpp	CAFFE_DIRECTORY/include/caffe/
data_transformer.cpp	CAFFE_DIRECTORY/src/caffe/

pooling_layer.cpp	CAFFE_DIRECTORY/src/caffe/layers
pooling_layer.hpp	CAFFE_DIRECTORY/include/caffe/layers
pooling_layer.cu	CAFFE_DIRECTORY/src/caffe/layers

表 1

2) 如果您的 Caffe 版本不是 1.0，请手动合并 caffe.proto 文件（请参阅下面的完整内容）。然后按照表 1 中的其余说明进行操作。

```
     optional QuantConvolutionParameter quant_convolution_param = 301;
     optional QuantReLUParameter quant_relu_param = 302;
    }

// Message for layers with reduced word with arithmetic
// GTI parameter
message QuantConvolutionParameter{
    enum Precision {
       FLOATING_POINT = 0;
       DYNAMIC_FIXED_POINT = 1;
       ONE_BIT = 2;
       TWO_BITS = 3;
       THREE_BITS = 4;
       FIVE_BITS = 5;
    }
    optional Precision coef_precision = 1 [default = FLOATING_POINT];
    // Fixed-point bit width
    optional uint32 bw_params = 2 [default = 8];
    // Shift-based quantization: on or off
    optional bool shift_enable = 3 [default = false];
}

//GTI parameter
message QuantReLUParameter {
   // Initial value of a_i. Default is a_i=0.25 for all i.
   optional FillerParameter filler = 1;
   // Whether or not slope parameters are shared across channels.
   optional bool  channel_shared = 2 [default = true];
   optional int32 act_bits = 3 [default = 30];
   optional bool  quant_enable  = 4   [default = false];
   optional float negative_slope = 5 [default = 0];
}
message PoolingParameter {
   enum PoolMethod {
      MAX = 0;
      AVE = 1;
      STOCHASTIC = 2;
   }
   optional bool sample = 14 [default = false];  //GTI parameter
   optional uint32 pad = 4 [default = 0]; // The padding size (equal in Y, X)
   optional uint32 pad_h = 9 [default = 0]; // The padding height
   optional uint32 pad_w = 10 [default = 0]; // The padding width
   optional uint32 kernel_size = 2; // The kernel size (square)
   optional uint32 kernel_h = 5; // The kernel height
```

```
   optional uint32 kernel_w = 6; // The kernel width
   optional uint32 stride = 3 [default = 1]; // The stride (equal in Y, X)
   optional uint32 stride_h = 7; // The stride height
   optional uint32 stride_w = 8; // The stride width
   enum Engine {
      DEFAULT = 0;
      CAFFE = 1;
      CUDNN = 2;
   }
   optional Engine engine = 11 [default = DEFAULT];
   optional bool global_pooling = 12 [default = false];
   enum RoundMode {
      CEIL = 0; //GTI parameter if use caffe-1.0
      FLOOR = 1; //GTI parameter if use caffe-1.0
   }
   // add this for caffe-1.0
   optional RoundMode round_mode = 13 [default = CEIL]; //GTI parameter if use caffe-1.0
}

message TransformationParameter {
   optional float scale = 1 [default = 1];
   // Specify if we want to randomly mirror data.
   optional bool mirror = 2 [default = false];
   // Specify if we would like to randomly crop an image.
   optional uint32 crop_size = 3 [default = 0];
   // mean_file and mean_value cannot be specified at the same time
   optional string mean_file = 4;
   repeated float mean_value = 5;
   // Force the decoded image to have 3 color channels.
   optional bool force_color = 6 [default = false];
   // Force the decoded image to have 1 color channels.
   optional bool force_gray = 7 [default = false];
   // GTI chip quant: convert bw_in to bw_out if quant_enable true
   optional bool quant_enable = 8 [default = false];          // GTI parameter
   optional uint32 bw_in = 9 [default = 8];                   // GTI parameter
   optional uint32 bw_out = 10 [default = 5];                 // GTI parameter
}
```

4.2.2 构建框架

- 使用 make 或 cmake 命令重新编译 Caffe，首先转到 Caffe 目录下（cd CAFFE_DIRECTORY/），然后按照以下命令操作：
  - Make 命令（使用 Makefile.config）：

```
make clean
make all
make test (可选)
make runtest (可选)
make pycaffe (用于 python 绑定)
make distribute
```

  - Cmake 命令:

```
mkdir build
```

```
cd build
cmake ..
make all
make test
make runtest
make pycaffe （用于 python 绑定）
make distribute
```

● 将下面命令添加到 .bashrc 文件的末尾：

```
export PYTHONPATH=CAFFE_DIRECTORY/python:$PYTHONPATH
```

重新编译后，如果能够导入 caffe，则表明您已成功安装 MDK。

注意：如果培训仅在 CPU 上进行，则应使用相应的设置构建 Caffe 环境。例如，如果使用 cmake 构建/编译，请使用 CPU_ONLY 选项更新 CAFFE_DIRECTORY/build/CMakeCache.txt。

1）双击打开 CMakeCache.txt 文件，然后选中 CPU_ONLY 选项中的复选框。

2）单击“配置”，然后单击“生成”并关闭文件。

3）遵循安装步骤。

4.2.3 使用流行的容器平台安装 Caffe

您可以根据流行的 Docker，Anaconda，Miniconda 设置您的 caffe 环境。下面介绍使用 Miniconda 的示例：

```
wget
https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
source .bashrc
```

创建一个 caffe 环境：

```
conda create -n testcaffe python
source activate testcaffe
```

对于其他依赖项，请按照 Anaconda / Miniconda 网站中的说明进行操作。

5 模型设计细节

以下是 GTI 模型开发过程中可能有助于改进精度但易于遗漏的重要细节。

5.1 芯片输入范围规格

GTI 芯片支持[0,31]范围内的输入值。因此需要将此归一化范围添加到训练数据处理中。如果输入范围不是[0,31]，例如[-1,1]，[0,1]，[0,255]，则可能需要重新调整输入以符合此规范。MDK 提供了将[0,255]输入转换为[0,31]的示例。例如：clip（（（（x >> 2）+1）>> 1），0,31），其中>>表示按位右移（即右移 2 位，加 1，右移 1 位）。

5.2 权重值和偏置值的不均衡

当 max（|B|）>> max（|W|）（即，偏置值的最大绝对值远大于权重的最大绝对值）或 max（|B|）<< max（|W|）（即偏置值的最大绝对值远小于权重的绝对值）时，可能会出现一个问题，即量化模型在 PC 上实现高精度但在芯片上并不高。量化范围由 W 和 B 的值确定，因此如果其中一个比另一个大得多，则量化范围受到另一个量化范围的严重限制，导致精度降低。一种补救措施是对其中一个施加约束。MDK 提供了一个示例，将偏置值约束到[-max|W|，max|W|]，确保偏置值不会超出权重值。其他方法（例如，最大范数，L2 正则化）也可以起作用。

要查看模型是否存在此问题，请检查 conversion.log 文件在转换过程中为每个层生成的 max|W|和 max|B|。

在构建混合模型（芯片模型的一部分和 PC 上的 Caffe 中的另一部分）时，要将芯片的输出输入到 Caffe 层时，可能需要 transpose/reshape 芯片输出的张量以符合 Caffe 接受的形式。当把 caffe 张量输入芯片时，也可能需要进行 transpose 和 reshape 操作。

6 模型开发流程

GTI 芯片加速神经网络的一部分，例如 CNN 层，而 FC 层在主处理器上运行。CNN 层需要使用 GTI 自定义量化卷积层和量化激活层进行微调。

MDK 提供 GTI 自定义量化卷积层和量化激活层，可在构建模型（即 QuantConvolution 层和 QuantReLU 层）时替代常规 Caffe 的相关层。它包括以下步骤（这些步骤必须按顺序执行）：

注：如希望直接进行实际操作，可先跳转至第 7 节 “实例”。

6.1 训练完整的浮点模型

第一步训练完整的浮点模型。浮点模型可以由常规的 convolution 和 relu,以及 batch normalization 定义。此外，也可以使用我们提供的 QuantConvolution 和 QuantRelu，若是浮点模型，注意每个 QuantConvolution 层的 coef_precision 应为 FLOATING_POINT 和 shift_enable 为 false, 以及每个 QuantRelu 层 quant_enable 为 false。

6.2 使用 GTI 自定义量化卷积层进行微调

获得较好的浮点模型后，通过量化卷积层的参数，并进一步微调模型。微调模型的精度尽量接近浮点模型精度。

样例：

1)如表 2 中所示，可以设置 coef_precision 来指定 QuantConvolution 层中的 quantization bits。另外请确保每个 QuantConvolution 层中的 shift_enable ： false 和每个 QuantReLU 层中的 quant_enable ： false。如下为 8-bit 定点量化例子：

```
layer {
    name: "conv1"
    type: "QuantConvolution"
    ...
    ...
    quant_convolution_param {
        coef_precision: DYNAMIC_FIXED_POINT
        bw_params: 8
        Shift_enable: false
    }
}
layer {
    name: "conv0/q_relu"
    type: "QuantReLU"
    ...
    ...
```

```
    quant_enable: false
    }
}
```

Chip	Quantization Bits	QuantConvolution Settings
2801	1-bit	coef_precision: ONE_BIT bw_params: 12
	3-bit	coef_precision: THREE_BITS bw_params: 12
2803	2-bit	coef_precision: TWO_BITS bw_params: 12
	5-bit	coef_precision: FIVE_BITS bw_params: 12
	12-bit	coef_precision: DYNAMIC_FIXED_POINT bw_params: 12
2801	1-bit	coef_precision: ONE_BIT bw_params: 8
	3-bit	coef_precision: THREE_BITS bw_params: 8
	8-bit	coef_precision: DYNAMIC_FIXED_POINT bw_params: 8

表 2 QuantConvolution 层的 quant_convolution_param 中的量化位设置

注意: GTI 2801 不支持 DYNAMIC_FIXED_POINT。

在 Caffe MDK 发行版本 3.2.10.0 中，修复了偏差量化中的裁剪阈值计算。clipping 阈值从“ max（bias）” 校正为“ max（abs（bias））” 。在极端情况下，尤其是对于 2803 的 2 位量化，以前的方法可能会导致 GPU 和芯片推理之间的位不匹配。

6.3 使用 GTI 自定义量化激活层进行微调

在量化、微调卷积层的参数之后，调整每个 ReLU 激活层的输出范围。train / scripts / CalibrateQuantReLU.py 为用于调整的脚本，即设置每个激活层的输出的上限，它将通过推断多个图像批次来估计激活层输出的最大值。该脚本默认的批次大小为 20，较大批次值可以获得较好 Relu 层的估计值。

在调整激活层的范围后，需要在启用量化激活层和量化卷积层的情况下微调模型。与常规 ReLU 层不同，每个 QuantReLU 层都具有可训练参数，该参数由层的校准输出范围值初始化（由 CalibrateQuantReLU.py 生成）。

GTI 设备（GTI2803 和 GTI5801）支持激活输出的 5 位和 10 位量化。对于 10 位激

活，必须将通道数量减少为等效 5 位激活模型数量的一半。例如，如果 5 位模型对该层最多使用 32 个通道，则如果用户希望对同一模型使用 10 位激活，则该层的通道数必须减少到 16 个。

样例：

确保新.prototxt 的**每个** QuantReLU 层，**quant_enable: true**；

```
    layer {
        name: "conv0/q_relu"
        type: "QuantReLU"
        ...
        ...
        quant_relu_param {
            filler {
                type: "constant"
                value: 151.121     # 自动生成
            }
                channel_shared: true
                act_bits: 5    #10
                quant_enable: true     # 打开 ReLU 量化
        }
}
```

6.4 模型融合

如果模型包含批量归一化或极端激活范围，则可能需要此融合步骤才能更准确地模拟 GTI 芯片操作。本质上，模型融合执行数学转换，将批量归一化和激活范围参数合并为权重和偏差。它还从模型中完全消除了批处理规范层，并使激活范围参数固定为芯片支持的范围（即，激活范围不再可训练）。合并后，新的权重和偏差可能会违反芯片的其他约束，因此必须再次对其进行微调。

6.4.1 融合批量归一化

批处理归一化参数在文献中有很好的定义。对于输入的 mini batch $B = \{\chi_{1,\dots,m}\}$和 $y_i = BN_{\gamma,\beta}(\chi_i)$的输出，其中：

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m}\chi_i$$

$$\sigma^2_B \leftarrow \frac{1}{m}\sum_{i=1}^{m}(\chi_i-\mu_B)^2$$

$$\chi_i \leftarrow \frac{\chi_i-\mu_B}{\sqrt{\sigma^2_B+\epsilon}}$$

$$y_i \leftarrow \gamma\chi_i+\beta \equiv BN_{\gamma,\beta}(\chi_i)$$

其中：

γ 和 β 表示 learned per layer，μ 表示 均值，σ 表示 方差

GTI 芯片不执行批量归一化，但是，我们可以将批量归一化参数与卷积层权重等效地融合在一起，并产生偏差，如下所示：

$$\omega' = \frac{\gamma}{\sigma}\omega$$

$$b' = \beta - \frac{\gamma\mu}{\sigma}$$

6.4.2 融合 ReLU Cap

假设我们有片上卷积层 **i**，接着另一个卷积层 **i + 1**：

$$y_i = ReLU(\omega_i * x_i + b_i)$$

$$y_{i+1} = ReLU(\omega_{i+1} * x_i + b_{i+1})$$

在量化激活层之前，我们为这些卷积层设置 ReLU Caps 和 α，并相应地训练权重和偏差，使得：

$$y_i \in [0, \alpha_i]$$

$$y_{i+1} \in [0, \alpha_{i+1}]$$

为了符合芯片激活范围（例如，以 5bits 或[0，31]量化），我们需要转换 $\mathbf{y_i}$ 和 $\mathbf{y_{i+1}}$，使得：

$$y'_i \in [0, 31]$$

$$y'_{i+1} \in [0, 31]$$

为此，我们可以将 $\mathbf{y_i}$ 和 $\mathbf{y_{i+1}}$ 乘以它们各自的因数：

$$y'_i = y_i \frac{31}{\alpha_i}$$

$$y'_{i+1} = y_{i+1} \frac{31}{\alpha_{i+1}}$$

我们无法将 α 参数存储在芯片上以直接修改 y; 但是，要获得相同的结果，我们可以将相同的乘积因子带入权重和偏差，然后将固定在 31 的 ReLU 上限设置为不可训练的参数：

$$\omega'_{i+1} = \omega_{i+1} \frac{31}{\alpha_{i+1}} \div \frac{31}{\alpha_i}$$

$$b'_{i+1} = b_{i+1} \frac{31}{\alpha_{i+1}}$$

注意 $\mathbf{w_{i+1}}$ 被除以一个额外的因子 $31/\alpha_i$，因为前一个卷积层的输出（即当前层的输入）$\mathbf{y_i}$ 已被乘以相同的因子 $31/\alpha_i$。原因是要消除此因子，以便我们不会对 $\mathbf{w_{i+1}}$ 造成不必要的干扰。如果我们位于卷积的第一层，假设输入（RGB 图像）数据范围已从[0，255]截断为[0，31]，那么我们不将 $\mathbf{w_0}$ 除以这个额外因子。

如果 $\mathbf{y_{i+1}}$ 连接到芯片外层（例如 fc 层），则在模型融合步骤中还必须转换芯片外层的 weight：

$$y_{fc} = \omega_{fc} \cdot \quad y_{i+1} + b_{fc}$$

$$\omega'_{fc} = \omega_{fc} \div \frac{31}{\alpha_{i+1}}$$

原因是 y_{i+1} 已乘以$31/\alpha_{i+1}$，为抵消芯片外层的这种影响，我们将权重除以相同的因子。注意，芯片外层的 bias 不需要改变，芯片外层也可以不立即连接到芯片输出而无需改变。

6.4.3 模型融合后的微调

根据批量归一化参数和 ReLU 上限的值，上述数学转换可能会扰乱权重和偏差，以至于它们不再符合芯片的其他约束。因此，在融合以模拟并满足所有芯片约束之后，应再次对网络进行微调。在最后一个微调步骤中，批量归一化层被完全删除，所有 ReLU 上限均固定为最大量化值（例如，对于 5 位激活为 31）。

这一步骤对从步骤 4.3 获得的模型权重执行线性变换，然后进一步微调模型以提高精度。这些线性变换包括批量归一化，输入图像的均值/比例和 gain editing（对于 5 位和 10 位激活，将网络输入范围均衡为[0，31]或[0，1023]）并将每个 QuantReLU 层的输出范围分别设置为[0，31]或[0，1023]）。请注意，如果仅最后一个卷积层使用 10 位激活，则相应 QuantReLU 层的输出范围必须为[0，31.96875]。文件夹/train/scripts/RefineNetwork.py 中提供了一个示例脚本。该脚本将采用步骤 5.3 中的.prototxt 和.caffemodel 作为输入，并自动生成一组新的.prototxt 和.caffemodel 文件，以用于后续的微调步骤。建议在较小的学习率（例如，<1e-4）下启用量化激活和量化卷积层进一步来微调模型。

表 3 提供了 QuantConvolution 和 QuantReLU 层中四个训练步骤设置的摘要（以 12 位动态量化为例）。

Training Steps	QuantConvolution	QuantReLU
Step1: Training floating-point model	coef_precision:FLOATING_POINT shift_enable:false	quant_enable:false
Step2: Fine-tune with quantized convolution	coef_precision:DYNAMIC_FIXED_POINT bw_params: 8 shift_enable:false	quant_enable:false
Step3: Fine-tune with	coef_precision:DYNAMIC_FIXED_POINT bw_params: 8 shift_enable:false	quant_enable:true

quantized ativation		
Step4: Further Fine-tune after linear transformation	coef_precision:DYNAMIC_FIXED_POINT bw_params: 8 shift_enable:true	quant_enable:true

表 3

6.5 转换模型并评估片内性能

模型转换过程需要从 Caffe 模型（.caffemodel,.prototxt）中提取参数，并将它们转换为.model 文件以部署到 GTI 芯片中。

6.5.1 说明

转换脚本是一个 Linux 命令行，它将芯片类型，caffe.prototxt 和 caffe.caffemodel，network.json，fullmodel.json 和 labels.txt 作为输入，并创建一个带文件的输出目录（coef.dat，net.json，fc.bin，out.model）和日志文件（log.txt）。

切换到“conversion_tool”目录中的转换工具：

```
python convert.py \
*.prototxt \
*.caffemodel \
network*.json \
fullmodel_def*.json \
-o
[--label=]{path} \
[--shift_max=]{INT} \
[--ouput_dir=]{path} \
[--evaluate=]{path} \
[--debug=]{true|false} \
[--input_scale=]{FLOAT} \
[--edit_gain=]{true|false}
```

参数：

- ***_.prototxt:**网络结构的卷积层数应与 network*.json 中定义的子层数相同，以及与 fullmodel_def*.json 相同数量的其他层（例如，FC 层）。执行第 5 节中描述的步骤后，卷积层类型应为“QuantConvolution”，激活层类型应为“QuantReLu”。
- ***.caffemodel** ：已训练的 caffemodel，对应于.prototxt。

可选参数：

- **label:** 类别标签。对于图像分类模型，标签的数量应与最后一个 FC 层的 num_output 相匹配。默认情况下，在转换过程中使用 Imagenet 1000 类的标签。

  有两个选项可以使用您自己的标签文件（labels.txt）：

  ■ 将您自己的标签文件重命名为“labels.txt”，并替换 release/conversion_tool/中的默认 labels.txt。

■ 提供您自己的标签文件的完整路径作为转换的可选参数。

如果缺少标签文件，请从网络定义文件（“fullmodel_def*.json”）中完全删除标签用法。

- **shift_max** 控制相对权重，推荐为 13，以便在稳定性和转换精度之间取得良好平衡。
- **input_scale** 是用于在训练期间相对于 255 缩放图像像素值的浮点值。默认值为 1.0，表示输入图像像素范围为[0,255]。
- **output_dir** 是存储转换工具生成的所有文件的路径。默认是 output/
- **debug** 设置为 true 时将在转换过程中会在 output/debug 文件夹中生成所有中间文件，以进行调试。默认值为 false。
- **edit_gain** 设置为 true 时将在转换期间执行 gain editing，将在输出文件夹中生成 conversion.log。gain editing 是指网络权重和偏差参数的线性变换，使得输入像素范围和所得网络的每个量化层的 α 都被调整为 31（5 位表示）。关于 10 位激活，请注意，如果所有卷积层都使用 10 位激活，则将 α 调整为 1023。如果仅最后一个卷积层使用 10 位激活，则将 α 调整为 31.96875（即 1023/32）。默认值为 false。
- **evaluate** 在芯片输出和 caffemodel 之间执行分层位匹配，以确保与在 CPU/GPU 上已训练的 caffemodel 相比，芯片推理结果始终相同。应用此选项进行模型调试，有关更多详细信息，请参见“6.10 调试模型”部分。

转换生成的 out.model 文件可以与 SDK 包中提供的参考应用程序一起使用 / 验证（见 Apps / Demo）。了解有关 GTI SDK 的更多信息，请参阅 GTI SDK 用户手册。

conversion_tool 目录包含示例 network*.json 和 fullmodel_def*.json 文件以及参考配置。

以下是模型转换过程：

1）从 Caffemodel 中提取卷积层的参数并将它们转换为.dat 文件以便在 GTI 芯片上运行；

2）提取并将全连接层的参数存储到.bin 文件中，以便在主机处理器上运行；

3）（可选）定义一个 LABELS 文本文件，其中包含：LABEL_ID LABEL_NAME 条目。请参阅 data / [EXAMPLE] _labels.txt 中的示例。条目必须与您的训练标签匹配。

4）将.dat 和.bin 文件打包成.model 文件，部署到 GTI 芯片中进行推理。

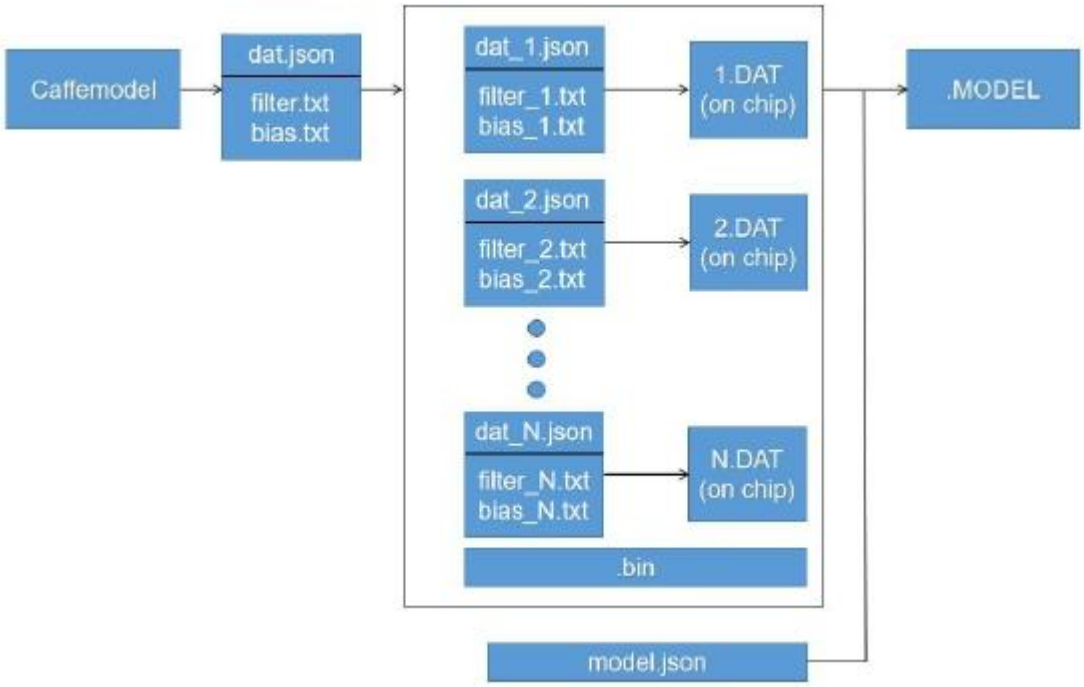


图 1 将.DAT 和.BIN 文件合并到一个.MODEL 文件中

两个 JSON 配置文件：

- 芯片配置定义（例如 conversion_tool/network2803_vgg-12bit_template.json）
  - 包括定义在 GTI 芯片上运行的 CNN 层的网络配置。网络定义和其他模型参数组合成.dat 芯片支持的文件格式。
- 完整模型定义（例如 conversion_too/fullmodel_def_vgg-12bit_template.json）
  - 完整的.model 文件包含.dat 和.bin 文件，其分别表示卷积层和全连接层。每次执行转换时，都会自动生成并覆盖配置文件中的“data file”路径（.dat 和.bin），无需手动修改。

.dat 文件仅使用了卷积层，如图 2 所示。然后使用.model 文件在 GTI SDK 中的 API 函数进行推理，将从芯片的最后一层产生输出，若在 network*.json 中设置“learning”为 true，则从中间层输出，可以自定义串行连接。最后，用户可以提取所需层的输出。

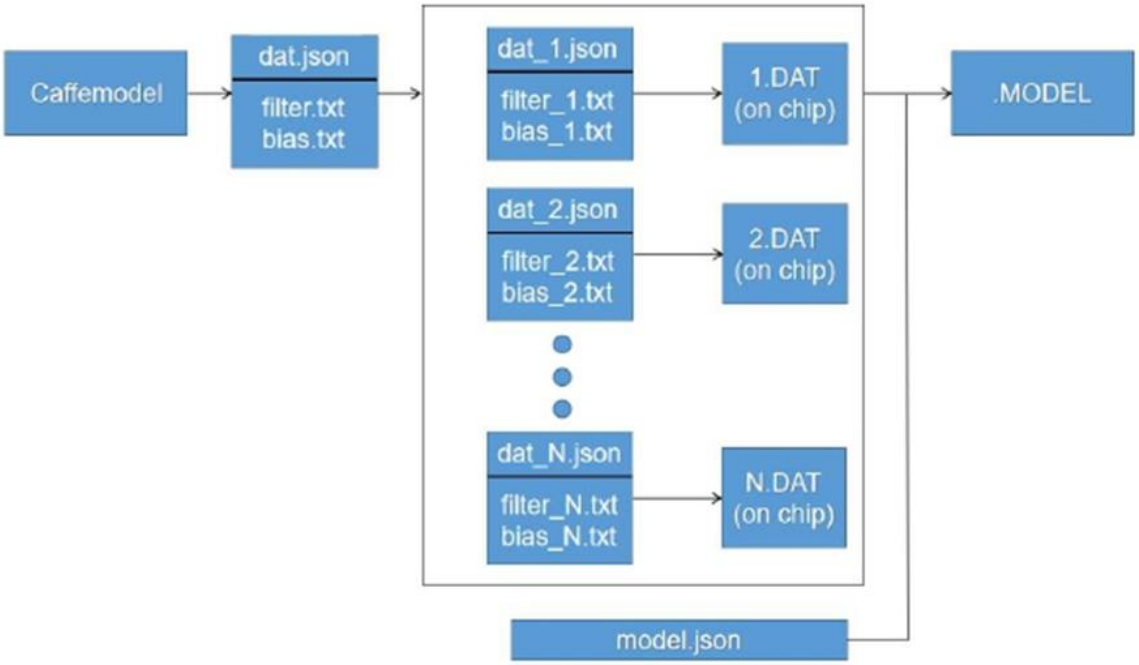


图 2 将.DAT 文件合并到一个.MODEL 文件中

6.6 改进转换效果

如果上芯片结果不满意，则可能存在问题并需要改进。请参阅 “第 5 节 模型设计细节” 部分，以避开模型开发周期中可能遗漏的常见陷阱。

6.7 启用中间层输出

模型的输出仅来自最后一个主要层的最后一个子层。从 MDK Caffe v3.0 开始，MDK 可以通过配置 network*.json 文件来启用或禁用中间层的输出。中间层是指最后一个主要层的最后一个子层之前的层。MDK 每个主要层支持 2 种不同的模式:

- learning 模式：启用/禁用此主要图层的所有子图层的输出。
- Last_layer_out 模式：启用/禁用此主要图层的最后一个子图层的输出。

这两种模式不能一起使用。如果启用了 learning 模式，则无法在任何其他主要层中启用 Last_layer_out 模式。此外，如果启用 Last_layer_out 模式，则无法在任何其他主要层中启用 learning 模式。

作为一个例子，请参考以下内容：

- 如果在其中一个主要图层中启用了 learning 模式，则所有主要图层中的 Last_layer_out 模式必须为 false。但是，您可以在其他主要图层中启用 learning 模式。
- 如果在其中一个主要图层中启用 Last_layer_out 模式，则所有主要图层中的 learning 模式必须为 false。但是，您可以在其他主要图层中启用 Last_layer_out 模式。

```
* _dat.json 的片段
“learning”：true，＃enable/disable 此主要层中所有子层的输出
“last_layer_out”：false，＃enable/disable 此主要层的最后一个子层的输出
```

6.8 级联模式

如果模型需要级联模式，且将模型拆分为多个芯片，则用户需要在 network*.json 文件中给 convert.py 中指定芯片数量，以便将单个 filter.txt / bias.txt / net.json 拆分为多个 filter.txt / bias.txt /net. json 组。convert.py 还为每个芯片创建.dat 文件，并创建.model 文件。用户在 network*.json 文件中定义芯片数量，如下所示。

```
"model": [
    {
        "ChipType": 2803,
        "ModelType" : "resnet50",
        "ResNetMode": 0,
        "ChipNums": 2
    }
```

在训练和部署期间，每个芯片模型可以具有不同的位宽精度。通常，建议前几个芯片使用尽可能高的位宽（只要权重参数大小可以适合芯片存储器），以便于训练收敛和得到更好的性能。

6.9 上采样

GTI 芯片支持重复或零模式下的 2 倍上采样。以下模型结构可用于 GTI 芯片。

1.重复填充模式和卷积的 2 倍上采样（卷积核 3x3）

2.零填充模式和卷积的 2 倍上采样（卷积核 3x3）

1) 重复填充模式下的上采样：

MDK 支持反卷积特性，在重复模式下执行 2 倍上采样(或最近邻插值)，然后执行 3x3 卷积。在这种模式下，GTI 设备上唯一可用的池化功能是 max pooling。

2) 零填充模式下的上采样：

类似于重复模式，但是，如果在零填充模式下使用反卷积层，则整个模型唯一可用的池化功能将是 sample pooling。

3) 对于以上两种模式：

- 使用深度反卷积来模拟 2 倍上采样（卷积核 = 2，步长 = 2）
- 通过训练冻结 weights，它不被视为子层
- Num_output 和 group 应该相同
- 如果使用 max pooling，则上采样必须处于重复填充模式

- 如果使用 sample pooling，则上采样必须处于零填充模式

6.9.1 重复填充模式下的上采样

- 固定卷积核权重[1,1,1,1]以执行数据重复

```
layer {
	name: "upsampling"
	type: "Deconvolution"
	bottom: "input"
	top: "upsampling"
	convolution_param {
		num_output: 16
		group: 16
		pad: 0
		kernel_size: 2
		stride: 2
		bias_term: false
		weight filler {
			type: "constant"
			value: 1
		}
	}
	param {
		lr_mult: 0
	}
}
layer {
	name: "conv"
	type: "Convolution"
	bottom:"upsampling"
	top: "conv"
	convolution_param {
		num_output: 32
		pad: 1
		kernel: 3
		stride: 1
		bias_term: true
	}
}
layer {
	name: "relu"
	type: "ReLU"
	bottom: "conv"
	top: "relu"
}
```

6.9.2 零填充模式下的升采样

- 固定的卷积核权重[1,0,0,0]可以填充零。
- 填充类型“constant_array”在 Caffe MDK 中实现，标准 Caffe 不支持。

```
layer {
	name: "upsampling"
	type: "Deconvolution"
	bottom: "input"
	top: "upsampling"
	convolution_param {
		num_output: 16
		group: 16
		pad: 0
		kernel_size: 2
		stride: 2
		bias_term: false
		weight filler {
			type:"constant_array",
			value_array: 1
			value_array: 0
			value_array: 0
			value_array: 0
		}
	}
	param {
		lr_mult: 0
	}
}
layer {
	name: "conv"
	type: "Convolution"
	bottom:"upsampling"
	top: "conv"
	convolution_param {
		num_output: 32
		pad: 1
		kernel_size: 3
		stride: 1
		bias_term: true
	}
}
```

```
layer {
       name: "relu"
       type: "ReLU"
       bottom: "conv"
       top: "relu"
}
```

6.10 调试模型

在运行模型转换脚本之前，请对 checkpoint 进行质量检查。在转换并部署到芯片后，checkpoint 可能会出现片上累积溢出。使用逐层位匹配来确定在芯片上运行的 checkpoint 是否会生成与在 Caffe 环境中运行时相同的输出。

通常，如果严格遵循 MDK 开发工作流程，则芯片输出应该和 Caffe 推理输出相匹配。但是，如果发现在芯片上推理的性能下降，则可以使用 MDK 软件包中的调试工具对大量验证图像进行芯片推理输出与 Caffe 推理输出之间的位匹配。

工作流程如下所述：

阶段 1：扫描验证文件夹，比较芯片上和 Caffe 上的最后一层输出

(1) 在仅启用最后一层输出的情况下，将 checkpoint 转换为“.model”文件。

(2) 从验证文件夹中随机获取一个图像。

(3) 将每个图像发送到芯片和 Caffe 环境中。

(4) 以二进制格式提取最后一个卷积层的芯片输出。

(5) 以二进制格式提取最后一个卷积层的 Caffe 推理输出。

(6) 检查（4）和（5）的输出二进制文件是否按位匹配。

(7) 重复（1）-（6），直到完成所有图像并计算匹配率。

(8) 如果有任何不匹配，请找出是哪个图像导致的，然后转到步骤 2。

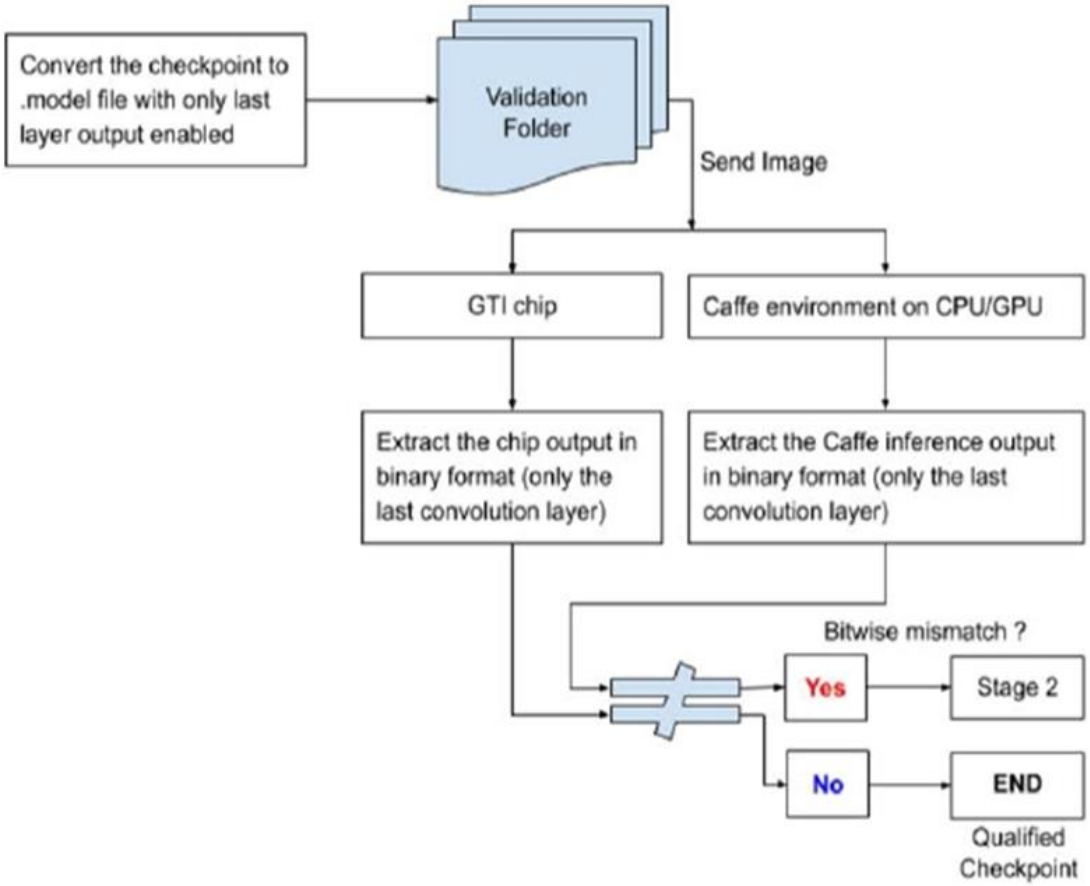


图 3 扫描验证文件夹，以识别片上和 Caffe 推理输出之间的按位不匹配

这可以通过启用转换期间的评估模式来实现：

```
python convert.py \
    deploy.prototxt \
    deploy.caffemodel \
    network2803.json \
    fullmodel_template.json \
    -o --evaluate=JPEGImages/
```

转换工具将在转换模型之前编辑 net.json 文件以将所有 learn 模式设置为“false”，并通过扫描目录中的所有图像在 caffe 输出和最后一层的芯片输出之间执行位匹配。

图 3 提供了阶段 1 的所有步骤，但是类似的过程对阶段 2 有效，但有以下区别：

1）处理单个图像而不是图像目录。

2）在阶段 1 中仅考虑最后一层的输出进行位匹配，在阶段 2 中比较所有层的输出。

阶段 2：通过检查每一层输出来检查导致不匹配的单个图像

(1) 将 checkpoint 转换为启用了 learn 模式的所有层输出的“.model”文件。

(2) 将此图像发送到芯片和 Caffe 环境中。

(3) 以二进制格式提取所有卷积层的芯片输出。

(4) 提取二进制格式输出的所有卷积层的 Caffe 推理。

(5) 检查芯片和 Caffe 的输出二进制文件是否与每个对应的卷积层匹配。

(6) 找出第一个不匹配的卷积层。

(7) 解析两个二进制文件并找到第一个不匹配位的位置。

(8) 通过提取权重/偏差和图层输入数据，进一步分析 Caffe 和芯片上计算。

执行阶段 2 步骤 1-6 的示例命令如下：

```
python convert.py \
    deploy.prototxt \
    deploy.caffemodel \
    network2803.json \
    fullmodel_template.json \
     -o --evaluate=test_image.jpg
```

转换工具将在转换模型之前编辑 net.json 文件以将所有学习模式设置为 true，并使用图像作为网络输入在所有中间层的 caffe 输出和芯片输出之间执行位匹配。

7 示例

7.1 mobilenet 224 示例

数据集 imagenet6_224：

链接：https://pan.baidu.com/s/1aKuqZSty8393d4RWx5ZqIA

提取码：zbcd

下载解压后将 lmdb 文件夹复制到 train 目录下。

该 MDK 默认使用 lmdb 格式的数据训练。原始数据压缩成 lmdb 格式参考 caffe-1.0/examples/imagenet 下的 create_imagenet.sh 脚本。

7.1.1 训练完整的浮点模型

训练浮点模型使用 train 文件夹下的 2803_mobilenet_224_train.prototxt

7.1.1.1 不带权重训练（建议带权重训练，容易收敛，见 7.1.1.2）

1）修改两个 Data 层的 data_param 中 source 分别为 lmdb 文件夹中的 train_lmdb_6_224 和 val_lmdb_6_224 路径；

2）确认 fc7 层的 num_output 为 6（类别数）；

3）确认 solver.prototxt 的 net 为 2803_mobilenet_224_train.prototxt，其他参数可自定义，详细说明见附录；

在 train/ 终端执行以下命令：

```
/home/xxx/caffe-1.0/build/tools/caffe \
train \
--solver=solver.prototxt \
-gpu 0 2>&1 | tee log_float_mobilenet.log
```

注：若有 nvidia GPU，并且编译 caffe 时，打开了 USE_CUDNN，则须有：-gpu 0

若是只有 CPU，并且是编译 cpu 的 caffe，则去掉：-gpu 0

7.1.1.2 带权重训练

我们提供了一个 mobilenet224.caffemodel 预训练权重，从以下链接下载：

链接：https://pan.baidu.com/s/1B2YGPKlI_oyyEhSBBJ7zUQ

提取码：p83w

1）修改两个 Data 层的 data_param 中 source 分别为 lmdb 文件夹中的 train_lmdb_6_224 和 val_lmdb_6_224 路径；

2）修改 fc7 层的 num_output 为 6（类别数）；

3）确认 solver.prototxt 的 net 为 2803_mobilenet_224_train.prototxt，其他参数可自定义，详细说明见附录，推荐 base_lr 设置为 0.0001；

在 train/ 终端执行以下命令：

```
/home/xxx/caffe-1.0/build/tools/caffe \
train \
--solver=solver.prototxt \
--weights=mobilenet224.caffemodel \
-gpu 0 2>&1 | tee log_float_mobilenet.log
```

注：--weights 为上述下载到当前文件夹下的 mobilenet224 预训练权重。如果训练收敛不了，建议尝试再调低学习率。

7.1.2 使用 GTI 自定义量化卷积层进行微调

（假设 snapshot_iter_20000.caffemodel 为步骤 7.1.1 生成的最好精度的 caffemodel）

1）修改 2803_mobilenet_224_train.prototxt 每个 Quantconvolution 层中的 coef_precision 为 DYNAMIC_FIXED_POINT，并重命名为 2803_mobilenet_224_quantConv_train.prototxt

2）修改 solver.prototxt 的 net 为 2803_mobilenet_224_quantConv_train.prototxt，推荐 base_lr 设置为 0.001；

在 train/ 终端执行以下命令：

```
/home/xxx/caffe-1.0/build/tools/caffe \
train \
--solver=solver.prototxt \
--weights=./snapshot/snapshot_iter_20000.caffemodel \
-gpu 0 | tee log_quantConv_mobilenet.log
```

7.1.3 使用 GTI 自定义量化激活层进行微调

（假设 snapshot_iter_10000.caffemodel 为步骤 7.1.2 生成的最好精度的 caffemodel）

若使用的是 cpu 的 caffe，则需要将 CalibrateQuantReLU.py 中的 caffe.set_mode_gpu()改为 caffe.set_mode_cpu()

1）在 train/scripts/ 终端下执行以下命令：

```
python CalibrateQuantReLU.py \
--prototxt=../2803_mobilenet_224_quantConv_train.prototxt \
--caffemodel=../snapshot/snapshot_iter_10000.caffemodel \
```

当前目录下会生成 QuantRelu 文件夹，以及 2803_mobilenet_224_quantConv_train_quantRelu.prototxt 和 snapshot_iter_10000_quantRelu.caffemodel。

2）确保 2803_mobilenet_224_quantConv_train_quantRelu.prototxt 每个 QuantReLU 层中的 quant_enable 为 true；

3）修改 solver.prototxt 的 net 为上述生成的 2803_mobilenet_224_quantConv_train_quantRelu.prototxt，推荐 base_lr 设置为 0.0001；

在 train/ 终端执行以下命令：

```
/home/xxx/caffe-1.0/build/tools/caffe \
train \
--solver=solver.prototxt \
--weights=./scripts/QuantRelu/snapshot_iter_10000_quantRelu.caffemodel \
-gpu 0 | tee log_quantRelu_mobilenet.log
```

7.1.4 对步骤 7.1.3 执行线性变换后的模型进一步微调

（假设 snapshot_iter_15000.caffemodel 为步骤 7.1.3 生成的最好精度的 caffemodel）

若使用的是 cpu 的 caffe，则需要将 RefineNetwork.py 中的 caffe.set_mode_gpu()改为 caffe.set_mode_cpu()

1）在 train/scripts/ 终端下执行以下命令：

```
python RefineNetwork.py \
--prototxt=../2803_mobilenet_224_quantConv_train_quantRelu.prototxt \
--caffemodel=../snapshot/snapshot_iter_15000.caffemodel \
```

当前目录下会生成 Refined 文件夹，以及 2803_mobilenet_224_quantConv_train_quantRelu_refined.prototxt 和 snapshot_iter_15000_refined.caffemodel

2）确保 2803_mobilenet_224_quantConv_train_quantRelu_refined.prototxt 每个 Quantconvolution 层中的 shift_enable 为 true;

注：如果芯片出来的结果需要继续传到后面非芯片计算层进行计算，需要在芯片出结果的层和下一层中间添加一个 scale 层，其值恒定为第三步训练出来的 model 中芯片出结果那一层对应 relu 层的值除以 31。

```
layer {
  name: "scale_conv12/dw"
  type: "Scale"
  bottom: "conv12/dw"
  top: "scale_conv12/dw"
  param {
    lr_mult: 0
    decay_mult: 0.0
  }
  scale_param {
    filler {
```

```
            value: 1.0   # 1.0 is example, please use QuantReLU_Value_O
    f_conv12_dw/31.0
          }
          bias_term: false
        }
      }
```

3）修改 solver.prototxt 的 net 为上述生成的 2803_mobilenet_224_quantConv_train_quantRelu_refined.prototxt，推荐 base_lr 设置为 0.00001；

在 train/ 终端执行以下命令：

```
/home/xxx/caffe-1.0/build/tools/caffe \
train \
--solver=solver.prototxt \
--weights=../scripts/Refined/snapshot_iter_15000_refined.caffemodel \
-gpu 0 | tee log_refined_mobilenet.log
```

7.1.5 模型转换

（假设 snapshot_iter_25000.caffemodel 为步骤 7.1.4 生成的最好精度 caffemodel）

1）将 2803_mobilenet_224_quantConv_train_quantRelu_refined.prototxt 头部的两个 Data 层删除，以及删除尾部的 SoftmaxWithLoss 层和 Accuracy 层。再将 train 目录下 mobilenet224_float_deploy.prototxt 的 Input 层以及 Softmax 层分别添加到 mobilenet224_quantConv_train_quantRelu_refined.prototxt 的头部和尾部，并另存为 2803_mobilenet_224_quantConv_quantRelu_refined_deploy.prototxt。

注：详细可参考 mobilenet224_float_deploy.prototxt 结构

2）修改 conversion_tool/network_examples/fullmodel_def2803_mnet_224_template.json 的”name”为 fc 和 softmax 的 output channels 为 6(类别数)

3）在 conversion_tool/目录下新建 mobilenet224 文件夹，并将 1）中保存的 2803_mobilenet224_quantConv_quantRelu_refined_deploy.prototxt，snapshot_iter_25000.caffemodel，以及提供数据集中的 labels.txt 复制到 mobilenet224 文件夹中。

4）预先对 conversion_tool 文件夹赋权限，执行：

```
sudo chmod 777 -R conversion_tool/
```

在 conversion_tool/下终端执行以下命令：

```
python convert.py \
mobilenet224/2803_mobilenet224_quantConv_quantRelu_refined_deploy.prototxt \
mobilenet224/snapshot_iter_25000.caffemodel \
network_examples/network2803_mobilenet_224_template.json \
network_examples/fullmodel_def2803_mobilenet_224_template.json \
```

mobilenet/labels.txt

注：labels.txt 为上述提供数据集对应的 labels.txt，不同数据集 labels.txt 不同。

最终在 conversion_tool/下生成 output 文件夹，以及相关文件。如有错误请查看 log.txt 文件。

7.2 mobilenet448 示例

数据集 imagenet6_448：

链接：https://pan.baidu.com/s/1ZvKxgRF9sgF-qd0fFk61ZA

提取码：m8yf

训练和推理过程同 mobilenet224。

7.3 ResNet 示例

7.3.1 ResNet18

训练和推理过程同 mobilenet224。有一点不同的是，Resnet18 量化卷积时，每个 Quantconvolution 层中的 coef_precision 使用 TWO_BITS。

7.3.2 ResNet 级联模式

Resnet50 训练量化卷积时，每个 Quantconvolution 层中的 coef_precision 使用 TWO_BITS。

GTI ResNet 可以通过一组 module-0，module-1 和 module-2 网络进行分组，其中

- module-0 网络：前几个主要层
- module-1 网络：中间重复层
- module-2 网络：最后几个主要层

因此，任何需要 N 个芯片的 GTI ResNet 都是以下组合计算的：

表 3 采用级联模式的 GTI ResNet 实现

例如，ResNet50 按 1 个 Module-0 网络，2 个 Module-1 网络和 1 个 Module-2 网络分组，如表 2 所示，其中芯片 1 包括 Module 0，芯片 2 包括 Module 1-1，芯片 3 包括 Module 1 -2，芯片 4 包括 Module 2。

	Module 0		Dimension	Feature Size	Filter number	Comp Ratio	Effective Flt Num
Chip1	Layer 1	Conv 1_1	16*64	224*224	1024	1	1024
		Conv 1_2	64*64	224*224	4096	1	4096
		Max Pooling					
	Layer 2	Conv 2_1	64*64	112*112	4096	1	4096
		Conv 2_2	64*64	112*112	4096	1	4096
		Max Pooling					
	Layer 3	Conv 3_1	64*64	56*56	4096	1	4096
		Conv 3_2	64*64	56*56	4096	1	4096
		with shortcuts					
		Conv 3_3	64*64	56*56	4096	1	4096
		Conv 3_4	64*64	56*56	4096	1	4096
		with shortcuts					
		Conv 3_5	64*64	56*56	4096	1	4096
		Conv 3_6	64*64	56*56	4096	1	4096
		with shortcuts					
		Max Pooling					
	Layer 4	Conv 4_1	64*128	28*28	8192	1	8192
		Conv 4_2	128*128	28*28	16384	1	16384
		Conv 4_3	128*128	28*28	16384	1	16384
		Conv 4_4	128*128	28*28	16384	1	16384
		with shortcuts					
		Conv 4_5	128*128	28*28	16384	1	16384
		Conv 4_6	128*128	28*28	16384	1	16384
		with shortcuts					
		Conv 4_7	128*128	28*28	16384	2	16384
		Conv 4_8	128*128	28*28	16384	2	16384
		with shortcuts					
		Max Pooling					

	Layer 5	Conv 5_1	128*256	14*14	32768	2	16384
		Conv 5_2	256*256	14*14	65536	2	32768
		Conv 5_3	256*256	14*14	65536	2	32768
		Conv 5_4	256*256	14*14	65536	2	32768
		with shortcuts					
	Module 1-1		**Dimension**	**Feature Size**	**Filter number**	**Comp Ratio**	**Effective Flt Num**
Chip2	**Layer 5**	Conv 5_5	256*256	14*14	65536	4	16384
		Conv 5_6	256*256	14*14	65536	4	16384
		with shortcuts					
		Conv 5_7	256*256	14*14	65536	4	16384
		Conv 5_8	256*256	14*14	65536	4	16384
		with shortcuts					
		Conv 5_9	256*256	14*14	65536	4	16384
		Conv 5_10	256*256	14*14	65536	4	16384
		with shortcuts					
	Module 1-2		**Dimension**	**Feature Size**	**Filter number**	**Comp Ratio**	**Effective Flt Num**
Chip3	**Layer 5**	Conv 5_11	256*256	14*14	65536	4	16384
		Conv 5_12	256*256	14*14	65536	4	16384
		with shortcuts					
		Conv 5_13	256*256	14*14	65536	4	16384
		Conv 5_14	256*256	14*14	65536	4	16384
		with shortcuts					
		Conv 5_15	256*256	14*14	65536	4	16384
		Conv 5_16	256*256	14*14	65536	4	16384
		with shortcuts					
	Module 2		**Dimension**	**Feature Size**	**Filter number**	**Comp Ratio**	**Effective Flt Num**
Chip4	**Layer 6**	Conv 6_1	256*512	14*14	131072	4	32768
		Conv 6_2	512*512	14*14	262144	4	65536
		with shortcuts					
		Conv 6_3	512*512	7*7	262144	4	65536

		Conv 6_4	512*512	7*7	262144	4	65536
		with shortcuts					
		Conv 6_5	512*512	7*7	262144	4	65536
		Conv 6_6	512*512	7*7	262144	4	65536
		with shortcuts					

GTI 修改的 ResNet50 模型的网络结构参考：

`/conversion_tool/network_examples/2803(5801)/fullmodel_def2803_resnet50_4chips_template.json`

对于 ResNet-152 实现，第一个芯片有四个池化层，最后一个芯片将有另一个池化层。中间的所有芯片都没有任何池化层。

7.4 VGG-16 示例

训练和推理过程同 mobilenet224。有一点不同的是，VGG16 量化卷积时，每个 Quantconvolution 层中的 coef_precision 使用 TWO_BITS，则可配置 13 层卷积（2，2，3，3，3）。若使用 DYNAMIC_FIXED_POINT，则配置 10 层卷积（2，2，2，2，2）。

7.5 VGG-SSD 人脸目标检测示例

数据集及 VGG_SSD_224_5801 prototxt :

链接：https://pan.baidu.com/s/1CM5Sc5ZSyNXlnbAvHfSDtw

提取码：d9xj

其中，face.tar.gz 是数据集，SSD_face_iter_77681.caffemodel 是预训练模型；

VGG_SSD_224_5801_test.prototxt 是测试的 prototxt；

VGG_SSD_224_5801_train_quant.prototxt 是训练的 prototxt；

上面的两个文件在 solver_face.prototxt 里面指定。

注：对于 SPR5801 芯片来说，每一层输入输出之和不能大于 4816896。

1. 浮点：

在 data 层里面把

data_param{

source:xxx_path 改为 数据集路径

}，46 行

把前五层卷积里面的 quant_convolution_param 里面的 coef_precision 的值改为 FLOAT_POINTING, prototxt 里面的分类结果根据自己的分类情况修改。

/home/xxx/caffe-ssd/build/tools/caffe train --solver=solver.prototxt --weights=vgg16.caffemodel --gpu 0 2>&1 | tee log.log

这一步会生成 caffemodel 模型文件,作为下一步的预训练模型.

执行每一步训练都需要修改 solver_face.prototxt 里面指定的网络文件(vgg.prototxt)

2. GTI 自定义的方式量化 convolution.

在浮点的基础上修改 test 和 train 的 prototxt,将前三层的把前五层卷积里面的 quant_convolution_param 里面的 coef_precision 的值改为 THREE_BITS，后两层的 coef_precision 的值改为 ONE_BIT。

查看芯片使用哪一个 bits 数目最合适，参照下面的计算：

每一层的 input_channel x output_channel / 压缩比，将每一层计算的结果相加之和小于 489216

压缩比：1bit 压缩比是 4，3bits 压缩比是 2，8bits 压缩比为 1 (DYNAMIC_FIXED_POINT 代表芯片支持的最高的 bits 数目).

/home/xxx/caffe-ssd/build/tools/caffe train --solver=solver.prototxt --weights=vgg16_float.caffemodel --gpu 0 2>&1 | tee log.log

3. GTI 自定义的方式量化 ReLU.

在 train/script/ 里面执行 python CalibrateQuantReLU.py 将脚本里面的 123 行的 prototxt 路径改为第二步使用的 prototxt,124 行的 model 改为第二步训练生成的 caffemodel,执行之后生成新的 prototxt 和 caffemodel.新生成的文件作为第三步训练的 prototxt 和与训练模型.

**注：在执行 CalibrateQuantReLU.py 之前需要把 prototxt 里面 data 层里面的 include 里面

include{

phase:TRAIN # TRAIN 改为 TEST

}8 行，

检查一下 新的 prototxt 里面每一个 QuantReLU 层中 quant_enable 为 true 以及将注 里面的 TEST 改为 TRAIN 以及将 VGG_SSD_224_5801_test.prototxt 里面的前五层一定要和 train 文件夹完全一致

/home/xxx/caffe-ssd/build/tools/caffe train --solver=solver.prototxt --weights=vgg16_quant_relu.caffemodel --gpu 0 2>&1 | tee log.log

4. fusion 模型

在 train/script/ 里面执行 python RefineNetWork.py 将 251 行的 prototxt 改为第三步训练的 prototxt,252 行的 caffemodel 改为第三步新生成的 caffemodel 文件. 执行之后生成新的 prototxt 和 caffemodel.新生成的文件作为第四步训练的 prototxt 和与训练模型.

**注：RefineNetWork.py 之前需要把 prototxt 里面 data 层里面的 include 里面

include{

phase:TRAIN # TRAIN 改为 TEST

} 8 行，

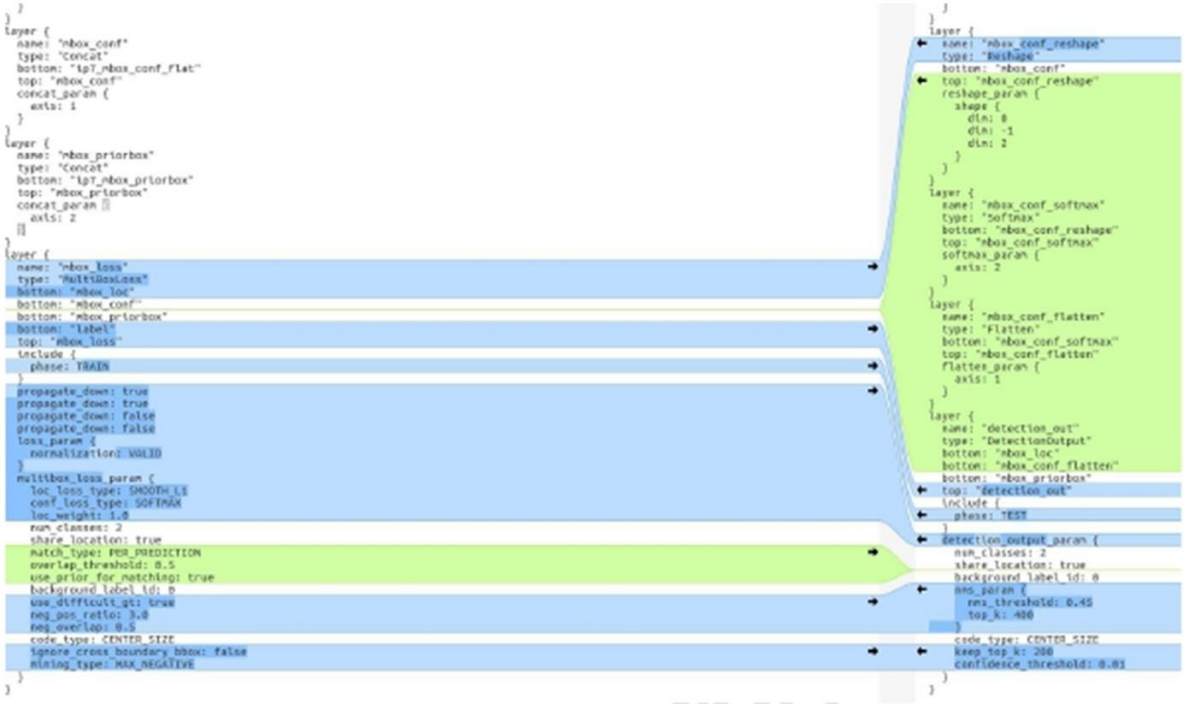
检查一下 新的 prototxt 里面每一个 QuantConvolution 层中 shift_enable 为 true 以及将 注 里面的 TEST 改为 TRAIN 以及将 VGG_SSD_224_5801_test.prototxt 里面的前五层一定要和 train 文件夹完全一致.

/home/xxx/caffe-ssd/build/tools/caffe train --solver=solver.prototxt --weights=net_refined.caffemodel --gpu 0 2>&1 | tee log.log

5. 模型转换

假设第四步训练做好的模型为 vgg_20000.caffemodel

在转换前需要讲 net_refine.prototxt 改为 net_refine_deploy.prototxt 文件，详细见图 1，在文件夹里面也提供了 net_refine_deploy.prototxt 作为参考。只需要修改 data 层以及 mbox_loss 层



1) 在 conversion_tools 文件夹里面进行转换.network_examples/5801 文件夹下找到 network5801_vgg-16_template.json(下面简称 network) 文件和 fullmodel_def5801_vgg-16_template.json(下面简称 fullmodel)

2) 我们的 SSD 网络里面的卷积使用的是标准的 vgg16 网络,没有从中间层 down 出来的部分,所以 network 文件不需要修改,在 fullmodel 文件里面把 fc 层去掉,softmax 层以及 label 层去掉.

3) 如果自己修改的 vgg-ssd 网络,从中间卷积层添加的有分支,需要在转化上芯片的时候 down 出来数据,要在 network 里面把需要 down 出来数据的那一层的 last_layer_out 设置为 true,这样出来的数据是不经过 pooling 的.

下面的 json 文件是介绍 network 里面各个参数的作用.改完 json 文件之后执行:

labels.txt 是打标的标签

python convert.py -h 查看需要的参数,比如

python convert.py ssd/*.prototxt ssd/vgg_20000.caffemodel ssd/network5801_vgg-16_template.json ssd/fullmodel_def5801_vgg-16_template.json -o label=labels.txt

这样生成新的.model 文件,这个文件就是上芯片的 model 模型.

6. 剩下的 fc 部分通过 ncnn 转成.param 文件和.bin 文件,具体可参考文件 VGG_SSD_5801.prototxt 文件

以及网址 https://blog.csdn.net/soralaro/article/details/81131615?utm_source=blogxgwz0

只需要生成新的.param 和.bin 文件。

7. 推理验证，可以参考 GTI_EXAMPLES。

How To Use

Build

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

Run

Use VGG_SSD_224_5801 as an example.

```bash
cd deploy/example/VGG_SSD_224_5801
./VGG_SSD_224_5801 ./../../data/test.mp4
```

具体使用请参看 ReadMe 文件。

7.6 MobileNet SSD 示例

Mobilenet448_ssd_test、train

链接：https://pan.baidu.com/s/1QxUvPR2J5vwvQI1_ICyjNA

提取码：ndnq

训练推理过程参考 7.5 VGG 的训练过程。

8.0 目标检测数据准备

8.1 目标检测 数据集—标注工具 labelImg/labelme

https://blog.csdn.net/wsp_1138886114/article/details/85017498?depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromBaidu-2&utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromBaidu-2

8.2 VOC COCO KITTI Caltech 数据集百度网盘下载 及提取自己想要的 目标类型

https://blog.csdn.net/weixin_40245131/article/details/102672734

附录 1

net: "mobilenet224_float_train.prototxt" #深度学习模型的网络结构文件，根据每个步骤生成的.prototxt 进行更新

test_iter: 1000 #1000 指的是测试的批次，测试样本较多时，一次性执行全部数据，效率较低，因此分几个批次进行执行

test_interval: 1000 #测试间隔，即每训练 1000 次，进行一次测试

base_lr: 0.001 #学习率

lr_policy: "step" # lr_policy 设置参数为 "step"

gamma: 0.95 #学习率变化的比率

stepsize: 1000 #每迭代 1000 次，调整一次学习率

display: 100 #每 100 次迭代，显示一次

max_iter: 400000 #最大迭代次数。

momentum: 0.9 #学习的参数

weight_decay: 0.0002 #权重衰减项，防止过拟合的一个参数

snapshot: 1000 #每迭代 1000 次，保存一次训练权值

snapshot_prefix: "./snapshot/snapshot" #设置保存训练权值的路径，不同步骤建议设置不同文件夹

solver_mode: GPU #选择使用 CPU 还是 GPU 运行