HiSVP 开发指南

文档版本　17

发布日期　2021-01-18

商标声明

 、HISILICON、海思和其他海思商标均为海思技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

上海海思技术有限公司

地址：　深圳市龙岗区坂田华为总部办公楼　邮编：518129

网址：　https://www.hisilicon.com/cn/

客户服务邮箱：　support@hisilicon.com

前言

概述

本文档旨在帮助用户了解SVP (Smart Vision Processing) 平台的硬件特性、工具链及开发流程，以期达到快速上手和开发出充分利用SVP特性的智能方案。

说明

- 未有特殊说明，Hi3559CV100/Hi3569V100与Hi3559AV100内容一致。
- 未有特殊说明，Hi3556AV100/Hi3568V100与Hi3519AV100内容一致。
- 未有特殊说明，Hi3516DV300/Hi3516AV300/Hi3559V200/Hi3562V100/Hi3566V100与Hi3516CV500内容一致。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3559A	V100
Hi3559C	V100
Hi3569	V100
Hi3519A	V100
Hi3556A	V100
Hi3568	V100
Hi3516D	V300
Hi3516C	V500
Hi3559	V200
Hi3516A	V300
Hi3531D	V200
Hi3535A	V100
Hi3562	V100

产品名称	产品版本
Hi3566	V100
Hi3521D	V200
Hi3520D	V500

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
危险	用于警示紧急的危险情形，若不避免，将会导致人员死亡或严重的人身伤害。
警告	用于警示潜在的危险情形，若不避免，可能会导致人员死亡或严重的人身伤害。
注意	用于警示潜在的危险情形，若不避免，可能会导致中度或轻微的人身伤害。
须知	用于传递设备或环境安全警示信息，若不避免，可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “注意”不涉及人身伤害。
说明	用于突出重要/关键信息、最佳实践和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害。

修改记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

日期	版本	修改描述
2021-01-18	17	第2章涉及修改 3.1、3.3、5.1.1和5.3.9小节涉及修改 3.7.3小节，新增图3-48 删除 "5.8 Runtime模型管理功能”和 "5.9 模型级联编辑功能" 小节。
2020-07-15	16	新增3.5.3和6.5小节。 5.5.2小节，涉及更新。
2020-06-08	15	添加Hi3568V100相关内容
2020-05-30	14	3.1.6.2小节，表3-2涉及更新。 新增3.6.1小节， 3.1.6.3、3.4.1、3.4.2、3.6.4、3.6.5、5.4.1小节，涉及更新。
2020-04-30	13	3.1.6小节，表3-1涉及更新。 3.1.6.3、3.2.5.2、3.2.5.4、3.5.3小节，涉及更新。 3.5.2小节，表3-11涉及更新。 第6章涉及更新。 新增第7章。
2020-03-31	12	添加Hi3569V100的相关内容
2020-02-29	11	3.1.6小节，涉及修改。
2019-12-20	10	新增Hi3521DV200和Hi3535AV100相关内容。
2019-11-30	09	新增Hi3562V100和Hi3566V100相关内容。
2019-10-15	08	新增Hi3535AV100相关内容。
2019-09-15	07	新增Hi3531DV200相关内容。 1.3~1.5小节，2.2小节涉及修改 新增3.2.11和5.5.11小节
2019-07-10	06	3.4.4、5.5.2、5.2.3、5.2.4和6.1.3小节涉及修改 新增5.4.2小节
2019-06-25	05	5.1.2.2小节涉及修改
2019-05-20	04	3.4.1.1~3.4.1.3、3.4.2 小节涉及修改 新增4.11和5.2.6.10小节
2019-04-15	03	新增“PROC开关”小节。

日期	版本	修改描述
2019-03-09	02	新增“4.2状态迁移”、“4.3.4插件库板端部署”、“4.3.5客户自定义插件”、“4.8.3 插件库路径可配置”、“4.8.4全局参数解析约束”、“4.10.2 模型输出BLOB名称”、“5.12.5如何在Ruyi工具中修改插件库名称”和5.12.6小节 新增第6章 5.8.2, 5.9.4, 5.10.3和5.10.4小节涉及修改
2018-12-10	01	新增5.3.4、5.5.9和5.5.10小节 3.5.2和5.13.2小节涉及修改
2018-11-13	00B14	4.5小节涉及修改 新增4.7和4.8小节
2018-10-26	00B13	3.1.6.2小节，表3-2涉及修改 新增5.9和5.10小节 5.13.2小节涉及修改
2018-10-15	00B12	1.3小节，表1-1涉及修改 3.1.1，3.6.2、3.6.3和4.1.2小节涉及修改
2018-09-04	00B11	新增第4章 修改第5章的相关内容 添加Hi3516CV500和Hi3516DV300内容
2018-08-08	00B10	新增2.2小节，2.6.2小节中的注意涉及修改 3.2.7~3.2.9，3.4.1~3.4.4小节涉及修改 3.5.1小节，修改表3-5 4.1.1、4.1.2、4.2.2小节涉及修改；新增4.4.1和4.7小节内容
2018-07-30	00B09	添加Hi3556AV100的相关内容
2018-07-06	00B08	3.4.2.1、4.1.2和4.7.2小节涉及修改 新增3.5.1和4.7.3小节
2018-06-20	00B07	4.1.2小节的步骤3和4.5.3小节的步骤2涉及修改 4.5.4小节涉及修改 新增4.6小节
2018-05-15	00B06	3.4~3.6小节涉及修改 新增4.1.2“Python3.5+caffe环境配置”小节 删除原来的4.1.4“NNIE编译器依赖库OpenCV安装与配置”小节 新增4.5.4~4.6小节
2018-04-10	00B05	第五次临时版本发布 添加Hi3519AV100相关内容

日期	版本	修改描述
2018-04-04	00B04	第四次临时版本发布 3.5.1小节，表3-5涉及修改 4.2.3、4.2.4、4.5.1和4.5.2小节涉及修改 新增4.5.3小节
2018-03-15	00B03	第三次临时版本发布 修改3.1.1、3.2和3.5小节 新增3.4小节和第4章
2018-01-30	00B02	第二次临时版本发布 3.3.1.3小节与3.3.2.6小节的lib64:$LD_LIBRARY_PATH改成lib:$LD_LIBRARY_PATH 3.5.4、3.8小节涉及刷新 新增3.9小节
2018-01-10	00B01	第一次临时版本发布。

目 录

插图目录

表格目录

1 概述

1.1 SVP 简介

SVP(Smart Vision Platform)是上海海思媒体处理芯片智能视觉异构加速平台。该平台包含了CPU、DSP、NNIE(Neural Network Inference Engine)等多个硬件处理单元和运行在这些硬件上SDK开发环境，以及配套的工具链开发环境。

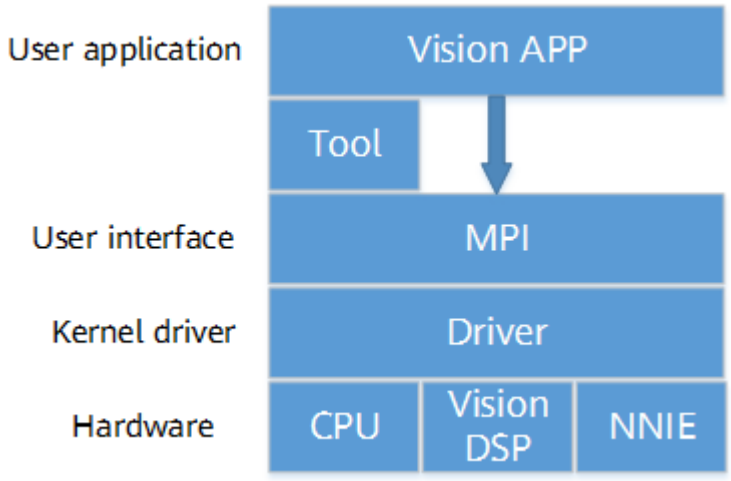
本文档主要介绍SVP的硬件特性、配套工具链及开发流程，旨在帮助用户快速入门以及开发出充分利用SVP硬件特性的智能应用。软件开发接口介绍请参考《HiSVP API参考》文档。

1.2 开发框架

SVP开发框架如图1-1所示。目前SVP中包含的硬件处理单元有CPU、vision DSP、NNIE，其中某些硬件可能有多核。

不同的硬件有不同的配套工具链，用户的应用程序需要结合这些工具的使用来开发。

图 1-1 SVP 开发框架



1.3 硬件资源

不同的芯片SVP会使用不同硬件资源，如表1-1所示。

表 1-1 不同芯片下的 SVP 硬件资源

芯片	CPU	DSP	NNIE
Hi3559AV100ES	双核A73+双核A53	2个	1个
Hi3559AV100	双核A73+双核A53	4个	2个
Hi3569V100	双核A73+双核A53	4个	2个
Hi3519AV100	单核A53+单核A53	1个	1个
Hi3556AV100	单核A53+单核A53	1个	1个
Hi3568V100	单核A53+单核A53	1个	1个
Hi3516DV300	单核A7+单核A7	0个	1个
Hi3516CV500	单核A7+单核A7	0个	1个
Hi3531DV200	四核A53	0个	1个
Hi3535AV100	四核A53	0个	1个
Hi3521DV200	四核A7	0个	1个
Hi3520DV500	四核A7	0个	1个

针对CPU的具体规格，请参考ARM官方文档。

DSP和NNIE的硬件规格，可参考对应的芯片手册。

须知

不同的芯片SVP可能会使用不同的硬件资源，即便是使用相同的硬件型号，硬件的配置也不一定相同。

1.4 软件开发

SVP是上海海思媒体处理芯片的智能加速平台，因此需要结合上海海思MPP平台一起来进行软件开发，可参考相关文档《 HiMPP 媒体处理软件 Vx.0 开发参考 》。用户可以根据SVP的软硬件特性开发出能最大化利用SVP硬件资源的视觉处理应用。

1.5 开发环境

不同的芯片SVP会在不同的环境上运行，如表1-2所示。

表 1-2 不同芯片下的 SVP 运行环境

芯片	系统架构
Hi3559AV100ES	big.LITTLE(Linux)
Hi3559AV100	big.LITTLE/Multi-Core (Linux)
Hi3519AV100	SMP (Linux)/AMP(Linux/Huawei LiteOS)
Hi3516DV300	SMP(Linux)/AMP(Linux/Huawei LiteOS)
Hi3516CV500	SMP(Linux)/AMP(Linux/Huawei LiteOS)
Hi3531DV200	SMP(Linux)
Hi3535AV100	SMP(Linux)
Hi3559V200	AMP(Linux/Huawei LiteOS)
Hi3562V100	AMP(Linux/Huawei LiteOS)
Hi3566V100	AMP(Linux/Huawei LiteOS)
Hi3521DV200	SMP(Linux)
Hi3520DV500	SMP(Linux)
Hi3569V100	big.LITTLE
Hi3568V100	AMP(Linux/Huawei LiteOS)

对于Hi3559AV100ES/Hi3559AV100：SVP布局在双核A73和双核A53组成的big.LITTLE上，芯片媒体业务布局在单核A53上，智能业务需要的图像数据是通过核间通信从单核A53上获取。具体多核业务开发可以参考《 Hi3559A／C V100 多核使用指南 》和SDK包提供的sample。

1.6 相关文档

《 Hi35xxVxxx xxx 用户指南 》

《 HiSVP API参考 》

《 HiMPP 媒体处理软件 Vx.0 开发参考 》

《 Hi3559A／C V100 多核使用指南 》

2 DSP 开发指南

对应的DSP侧代码从之前版本拷贝，文档也请参考之前版本，具体如下：

表 2-1 不同芯片 DSP 开发指南参考版本及文档

芯片	版本	文档	章节
Hi3559AV100 Hi3559CV100 Hi3569V100	Hi3559AV100R001C02SPC031	《HiSVP 开发指南.pdf》	第二章节
Hi3519AV100 Hi3556AV100 Hi3568V100	Hi3519AV100R001C02SPC020	《HiSVP 开发指南.pdf》	第二章节
Hi3516DV300	不支持	不支持	不支持
Hi3516CV500	不支持	不支持	不支持
Hi3531DV200	不支持	不支持	不支持
Hi3535AV100	不支持	不支持	不支持
Hi3521DV200	不支持	不支持	不支持
Hi3520DV500	不支持	不支持	不支持

3 NNIE 开发指南

3.1 NNIE 介绍

NNIE是Neural Network Inference Engine的简称，是上海海思媒体SoC中专门针对神经网络特别是深度学习卷积神经网络进行加速处理的硬件单元，支持现有大部分的公开网络，如Alexnet、VGG16、Googlenet、Resnet18、Resnet50、Mobilenet等分类网络，Faster R-CNN、YOLO(v1/v2/v3)、SSD、RFCN、MTCNN等检测网络，以及SegNet、FCN等场景分割网络。

目前NNIE配套软件及工具链仅支持Caffe框架，使用其他框架的网络模型需要转化为Caffe框架下的模型。网络中的不支持算子，可使用自定义算子方式实现，详见“**3.1.5 Non-support层处理方式**”。

3.1.1 工具链介绍

SVP NNIE在HiSVP_PC_Vx.x.x.x.rar组件包中，提供如下的工具链：

- nnie_mapper(tools\nnie\linux\mapper目录)：简称mapper，该工具将用户通过开源深度学习框架训练得到的模型转化成在Hi35xx芯片上或者在仿真库中可以加载的数据指令文件（文件后缀为wk，后文中提到的wk文件即由该工具生成）。
- 仿真库(software\x64目录)：模拟NNIE的硬件执行和软件接口调用，在“相同的输入”下仿真库与硬件得到相同的结果。仿真库可以使用户脱离硬件在PC环境下仿真，且便于调试，有助于用户提前快速开发算法原型。仿真库有基于Visual Studio和MinGW版本，后者集成在RuyiStudio中。
- 仿真Sample工程(software\sample_simulator目录)：包含仿真sample源代码供开发者学习参考，支持Visual Studio或RuyiStudio环境运行。
- 模型包(software\data)：包含若干sample中用到的网络的caffe模型文件及对应的NNIE mapper配置文件、wk文件、图像文件等。
- Windows版IDE(tools\nnie\windows目录)工具RuyiStudio，集成Windows版的NNIE mapper和仿真库，用户可以将仿真Sample工程导入运行、调试；IDE还集成了代码编辑、编译、调试、执行、画框、相似度比对等功能，具体参考“**RuyiStudio工具使用指南**”章节。

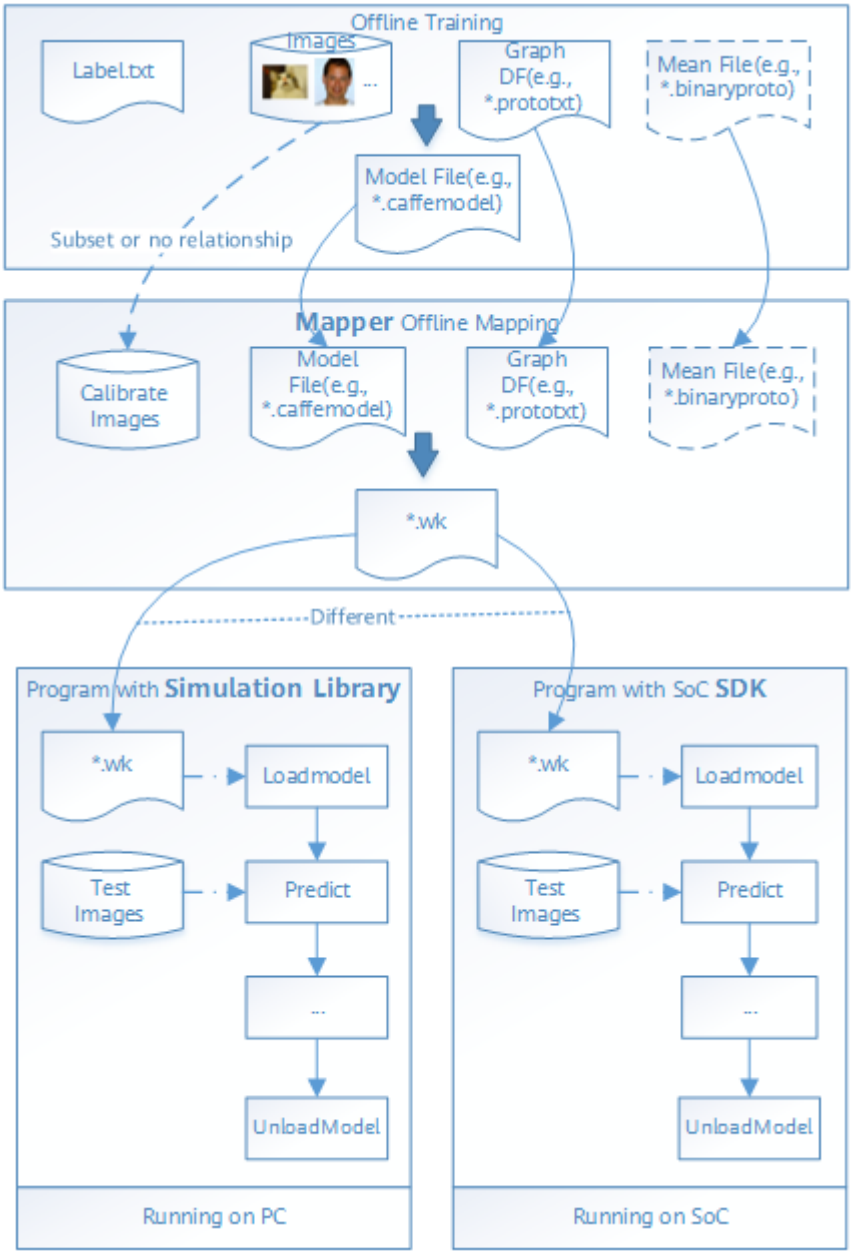
须知

- “相同的输入”是指相同的原始模型，相同的图像数据输入。由mapper转化得到的仿真库上的模型和在芯片上的是不相同的。
- NNIE工具链目前只支持Caffe框架，且以Caffe-1.0版本为基础。

3.1.2 开发流程

以Caffe框架上训练的模型为例，NNIE的开发流程如图3-1所示。在Caffe上训练、使用NNIE的mapper工具转化都是离线的。通过设置不同的模式，mapper将*.caffemodel转化成在仿真器、仿真库或板端上可加载执行的数据指令文件。一般在开发前期，用户可使用仿真器对训练出来的模型进行精度、性能、带宽初步评估，符合用户预期后再使用仿真库进行完整功能的仿真，最后将程序移植到板端。

图 3-1 NNIE 软件开发流程



3.1.3 网络层的分类

一个网络的层可分为如下的3类：

- 标准层：NNIE支持的Caffe标准层，比如Convolution，Pooling层等；
- 扩展层：NNIE支持的公开但非Caffe标准层，分为2种：
  - 一种是基于Caffe框架进行自定义扩展的层，比如Faster RCNN中的ROIPooling层、SSD中Normalize层、RFCN中的PSROIPooling层，SegNet中的UpSample层等；
  - 另外一种是来源于其他深度学习框架的自定义层，比如YOLOv2中Passthrough层等；

- Non-support层：NNIE不支持的层，比如Caffe中专用于Tranning的层、其他非Caffe框架中的一些层或者用户自定义的私有层等。

3.1.4 扩展层规则

Faster RCNN、SSD、RFCN和SegNet等网络都包含了一些原始Caffe中没有定义的层结构，如ROIPooling、Normalize、PSROI Pooling和Upsample等。NNIE的mapper目前仅支持Caffe框架，且以Caffe-1.0为基础。为了使mapper能支持这些网络，需要对原始的Caffe进行扩展。

须知

针对使用了扩展层的网络，用户可以二选一操作：

- 按照3.1.4节扩展Caffe的caffe.proto以及扩展层代码实现，并以此基础来进行训练，对应部署的deploy.prototxt满足mapper要求。
- 使用开源的扩展caffe框架来训练（比如rbgirshick/py-faster-rcnn等），不需要扩展caffe.proto以及扩展层代码实现，但是需检查对应的扩展层规格是满足要求，且最后输入给mapper的deploy.prototxt必须满足3.1.4.3中要求以便nnie_mapper识别。

3.1.4.1 扩展 Caffe proto 文件

扩展层可能涉及到一些相关的输入参数，因此需要扩展Caffe的proto文件，使得Caffe在解析prototxt文件时能够识别出对应的参数声明。

以ROIPooling层为例，需要修改Caffe源代码src/caffe/proto/caffe.proto文件：

1. 首先需要在LayerParameter这个message中添加自定义层参数的声明，下图定义了ROIPoolingParameter这个参数类型，其名称为 roi_pooling_param，等号右边的数字可以是一个任意的目前LayerParameter中没有被使用的字段。

   本文档中限定扩展层参数字段从100000开始。roi_pooling_param为声明的该parameter对应的名称，可以在源代码中使用layer_param_.roi_pooling_param()来获取该参数对应的数值。

   图 3-2 在 caffe.proto 文件中扩展 LayerParameter

   ```
   optional ROIPoolingParameter roi_pooling_param = 100000;
   optional NormalizeParameter norm_param = 100001;
   optional PSROIPoolingParameter psroi_pooling_param = 100002;
   optional UpsampleParameter upsample_param = 100003;
   ```

2. 完成ROIPoolingParamter的参数声明后，还需要在caffe.proto文件中添加一个新的message，内部定义该参数具体的字段名称、数据类型、默认数值等。如下图即定义了ROI Pooling层有3个可以配置的参数，分别为pooled_h、pooled_w和spatial_scale。

图 3-3 在 caffe.proto 文件中定义新 layer 的 message

```
// Message that stores parameters used by ROIPoolingLayer
message ROIPoolingParameter {
  // Pad, kernel size, and stride are all given as a single value for equal
  // dimensions in height and width or as Y, X pairs.
  optional uint32 pooled_h = 1 [default = 0]; // The pooled output height
  optional uint32 pooled_w = 2 [default = 0]; // The pooled output width
  // Multiplicative spatial scale factor to translate ROI coords from their
  // input scale to the scale used when pooling
  optional float spatial_scale = 3 [default = 1];
}
```

3.1.4.2 扩展层代码实现

经过上述caffe.proto文件的扩展之后，Caffe就能够识别prototxt中层参数相关的设定。但若需要支持扩展层的解析、training、inference，还需要根据扩展层的处理方式，完成对应底层代码实现，如果该层处理要在CPU上运行，就要实现对应的C++代码，如果该层处理要在GPU上运行，就要实现对应的CUDA C/C++代码。

在实现代码的过程中，层的参数可以使用Caffe提供的接口函数，结合proto文件中声明的参数名称来获取。以ROI Pooling为例：

1. Faster RCNN就在原始的caffe中添加了ROI Pooling层的CPU与GPU代码，分别为roi_pooling_layer.cpp和roi_pooling_layer.cu，并将这两个文件放在了Caffe源代码的src/caffe/layers目录下。在源代码中的参数获取方式如下图，其中roi_pooling_param为proto文件中定义的ROIPoolingParameter的名称，pooled_h、pooled_w和spatial_scale为该参数包含的3个具体参数的名称。获取了参数后，就可以在后续的代码实现中使用该参数。

   图 3-4 从 prototxt 中获取参数示例

```
template <typename Dtype>
void ROIPoolingLayer<Dtype>::LayerSetUp(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top) {
  ROIPoolingParameter roi_pool_param = this->layer_param_.roi_pooling_param();
  CHECK_GT(roi_pool_param.pooled_h(), 0)
      << "pooled_h must be > 0";
  CHECK_GT(roi_pool_param.pooled_w(), 0)
      << "pooled_w must be > 0";
  pooled_height_ = roi_pool_param.pooled_h();
  pooled_width_ = roi_pool_param.pooled_w();
  spatial_scale_ = roi_pool_param.spatial_scale();
  LOG(INFO) << "Spatial scale: " << spatial_scale_;
}
```

2. 在实现底层代码的过程中，主要需要实现该层Forward与Backward对应的代码，并在文件末尾调用Caffe内建的层声明函数，如下图roi_pooling_layer.cpp的末尾。如果某些函数没有实现，在代码中加入NOT_IMPLEMENTED宏，INSTANTIATE_CLASS和REGISTER_LAYER_CLASS即为调用Caffe内建的层声明函数。

图 3-5 声明自定义的层

```
template <typename Dtype>
void ROIPoolingLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
      const vector<bool>& propagate_down, const vector<Blob<Dtype>*>& bottom) {
  NOT_IMPLEMENTED;
}


#ifdef CPU_ONLY
STUB_GPU(ROIPoolingLayer);
#endif

INSTANTIATE_CLASS(ROIPoolingLayer);
REGISTER_LAYER_CLASS(ROIPooling);

}  // namespace caffe
```

3. 上述的操作完成了一个名称为ROIPooling的Layer的声明。当然，代码还有对应的头文件，头文件统一放在Caffe源代码的include/caffe目录下，ROIPooling层将头文件整个到了一个名称为fast_rcnn_layers.hpp的头文件中。如果有必要也可以将对应的头文件放在include/caffe/layers目录下，只要保证其路径的正确性使工程在编译的时候能够找到对应的文件即可。

图 3-6 在实现文件中添加头文件

```
#include "caffe/fast_rcnn_layers.hpp"
```

4. 完成了上述步骤之后，就可以重新编译整个Caffe工程。成功编译后的Caffe就可以解析prototxt中自定义的层了。如下图为Faster RCNN的prototxt中包含ROI Pooling层的示例，其layer的type字段为ROIPooling，并分别定义了roi_pooling_param参数中pooled_w、pooled_h和spatial_scale的具体数值。

图 3-7 ROIPooling 层在 prototxt 中的示例

```
layer {
  name: "roi_pool5"
  type: "ROIPooling"
  bottom: "conv5_3"
  bottom: "rois"
  top: "pool5"
  roi_pooling_param {
    pooled_w: 7
    pooled_h: 7
    spatial_scale: 0.0625 # 1/16
  }
}
```

3.1.4.3 扩展层在 prototxt 中的定义

扩展层的实现可以在Caffe或其他深度学习框架中实现，扩展层中的第一种是基于Caffe框架进行扩展实现的，如ROI Pooling、Normalize、PSROI Polling和Upsample层等，因此有公开的prototxt标准定义。而扩展层中的第二种并不是在Caffe框架下实现

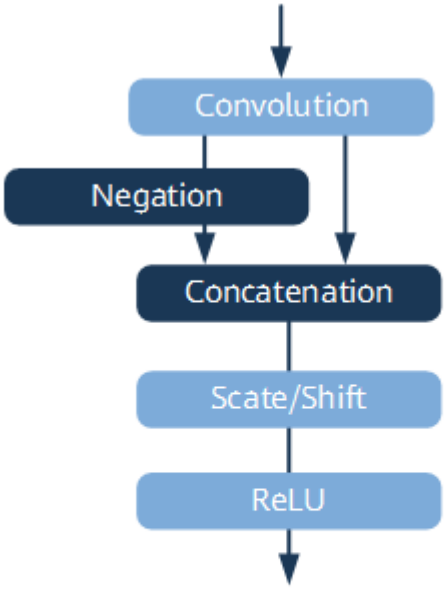
的，对于这类网络中的自定义层，也需要给出prototxt的标准定义，用于在prototxt中定义网络中相应的层。

CReLU 层

CReLU层为PVANet中特殊的层结构，其结构如下，在Caffe中并没有标准的CReLU层作为单独的一层。

注意：在CReLU最早提出的论文中《Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units》（http://cn.arxiv.org/abs/1603.05201），并没有如图3-8所述的Scale/Shift层，也即没有训练参数。不过，在NNIE mapper支持的单独CReLU层的实现中，是以PVANet网络中的方式来实现，Caffemodel模型文件中需要包含Scale层对应的参数。如果要使用不同于此方式实现的CReLU层，可以通过多个Caffe标准层组合实现。

图 3-8 CReLU 层实现示例



Note: Our CReLU building block **Negation** simply multiplies -1 to the output of Convolution. **Scale/Shift** applies trainable weight and bias to each channel, allowing activations in the negated part to be adaptive.

虽然没有一个独立的CReLU层，不过该层所实现的操作可以使用Caffe中已有的标准层来进行实现，Negation使用标准的Power层，参数配置为power为1，scale为-1.0，shift为0；Concatenation使用标准的Concat层将原始的feature map和取反后的feature map进行拼接；Scale/Shift使用标准的Scale层实现；ReLU同样使用标准的层实现。

综上，一个CRelu层可以使用如下的prototxt定义，使用Caffe中标准层进行实现。

```
layer {
  name: "conv1_1/neg"
  type: "Power"
  bottom: "conv1_1/conv"
  top: "conv1_1/neg"
```



```
  power_param {
    power: 1
    scale: -1.0
    shift: 0
  }
}
layer {
  name: "conv1_1/concat"
  type: "Concat"
  bottom: "conv1_1/conv"
  bottom: "conv1_1/neg"
  top: "conv1_1"
}
```

```
layer {
  name: "conv1_1/scale"
  type: "Scale"
  bottom: "conv1_1"
  top: "conv1_1"
  scale_param {
    bias_term: true
  }
}
layer {
  name: "conv1_1/relu"
  type: "ReLU"
  bottom: "conv1_1"
  top: "conv1_1"
}
```

除了采用多个Caffe中标准层组合的方式实现外，NNIE mapper也支持解析一个独立定义的CReLU层，CReLU层在prototxt中进行定义的方式示例如下，layer的type定义为CReLU。

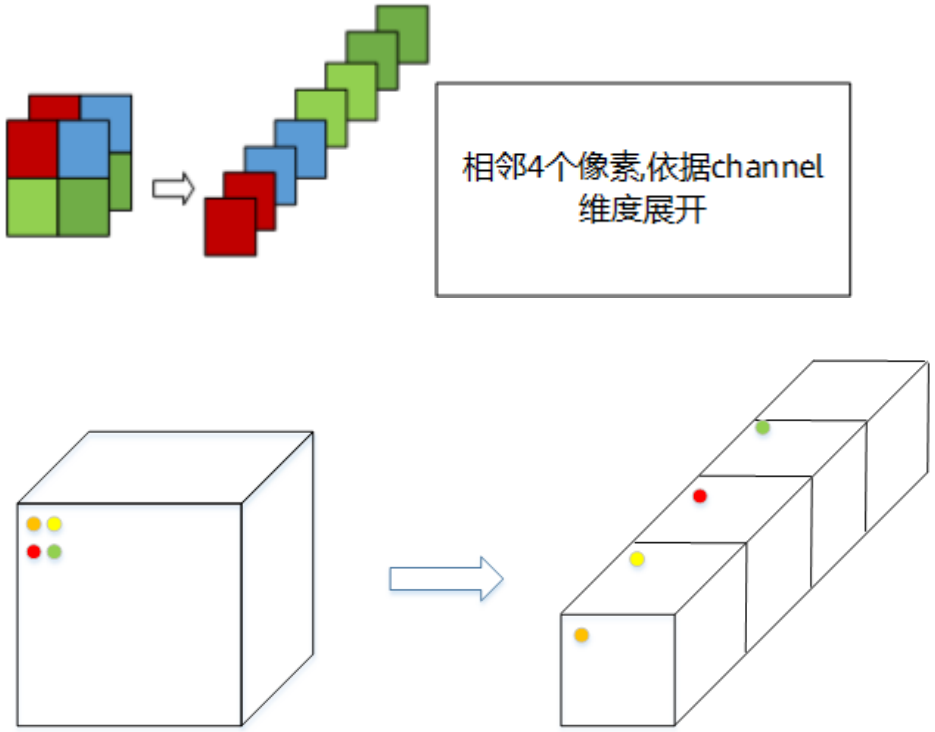
```
layer {
  name: "crelu_layer"
  type: "CReLU"
  bottom: "some_input"
  top: "some_output"
    scale_param {
      bias_term: true
  }
}
```

CReLU的操作为上述限定的操作，由于Power、Concat层的参数都是限定的，ReLU层不包含参数，因此仅仅需要配置scale_param，即Scale层对应的参数。scale_param可以复用标准Scale层的参数配置，因此不需要对caffe.proto文件进行扩展，只需要开发CReLU层对应的层实现代码即可。

PassThrough 层

PassThrough层为Yolo v2中的一个自定义层，由于Yolo v2并不是使用Caffe框架实现，因此对于该层没有标准的定义。该层实现的功能为将feature map在spatial维度上的数据展开到channel维度上，原始在channel维度上连续的元素在展开后的feature map中依然是连续的。如将26×26×512的feature变成13×13×2048的feature，做法为将相邻的像素展开到channel维度，示意图如图3-9。

图 3-9 PassThrough 层操作示意图



该层操作在实现过程中需要给定的参数包括在spatial维度上进行展开的窗口大小，以下称为block，需要定义该block窗口的高度height和宽度width，两者均为正整数，分别即为block_height和block_width，且需要保证block_height和block_width能够被输入feature map的height和width整除。若输入feature map的channel维度为num_in，则输出feature map的channel维度num_output=num_in*block_height*block_width。如上述的示例中，block_height=2，block_width=2，num_output=2048。

在定义该层的参数时，需要block_height、block_width和输出feature map的channel维度数num_output。虽然num_output可以通过num_in、block_height、block_width计算得出，但是在参数定义是还是需要进行设定，用于参数合法性的检查。

PassThrough层在prototxt中进行定义的方式示例如下，layer的type定义为PassThrough。

```
layer {
  name: "pass_through"
  type: "PassThrough"
  bottom: "some_input"
  top: "some_output"
  pass_through_param{
    num_output: xxx
    block_height: xxx
    block_width: xxx
  }
}
```

为了使caffe能够对其进行解析，需要对proto文件进行扩展。在caffe.proto文件的LayerParameter中加入名称为PassThroughParameter的定义（下图中的100004为一个任意的当前caffe.proto的LayerParameter中没有被占用的数值）；并在文件中定义一个名称为PassThroughParameter的message定义。

图 3-10 增加 PassThrough 层所需的对 caffe.proto 的修改

```
optional ROIPoolingParameter roi_pooling_param = 100000;
optional NormalizeParameter norm_param = 100001;
optional PSROIPoolingParameter psroi_pooling_param = 100002;
optional UpsampleParameter upsample_param = 100003;
optional PassThroughParameter pass_through_param = 100004;
```

```
// Message that stores parameters used by PassThroughLayer
message PassThroughParameter {
  //the number of channels in output feature map
  optional uint32 num_output = 1 [default = 0];
  //size of the spatial window block
  optional uint32 block_height = 2 [default = 0];
  optional uint32 block_width = 3 [default = 0];
}
```

NNIE对该层的支持规格为在保证参数合法有效的情形下，block_height可以为一个1到255之间的正整数，block_width可以为一个1到255之间的正整数，且block_height和block_width可以不相等。

DepthwiseConv 层

Depthwise Convolution层为Xception网络中的自定义层，在caffe框架中同样也是没有进行标准定义的。Depthwise Convolution实现的操作为针对输入的每一个channel，单独做K*K的卷积，假设输入的channel是M1，输出结果的channel依然是M1。因此，可以认为有M1个K*K*1的卷积kernel。

由于该层的计算是基于标准卷积的一种变种，需要定义的参数可以复用Caffe中卷积参数的定义，所以不需要对caffe.proto文件进行扩展，直接复用ConvolutionParameter即可。

Depthwise Convolution层在prototxt中进行定义的方式示例如下，layer的type定义为DepthwiseConv。

```
layer {
  name: "depthwise_convolution"
  type: "DepthwiseConv"
  bottom: "some_input"
  top: "some_output"
  convolution_param{
    stride: xxx
    pad: xxx
    kernel_size: xxx
    num_output: xxx
    bias_term: true
  }
}
```

与Caffe中convolution_param一样，在定义的时候同样可以使用kernel_h、kernel_w、stride_h、stride_w、pad_h、pad_w来定义height和width方向上的卷积kernel参数和padding。

对于Depthwise Convolution层的bias，同样是通过convolution_param中的bias_term来确定，由于Caffe的卷积层的bias的维度是通过读取prototxt的convolution_param的num_output来定义，因此DepthwiseConv层的定义中，num_output也是

convolution_param中必须要定义的字段，且由于DepthwiseConv层的计算方式，必须要求num_output与输入数据的channel维度大小一致，bias为每个输出channel对应一个bias值。

RReLU 层

RReLU层即Randomized Leaky ReLU，在https://arxiv.org/abs/1505.00853中有提到，并在文章中通过MXNet框架进行实现，在caffe框架中同样也是没有进行标准定义的。RReLU层类似Leaky ReLU，只不过在负数区间的斜率与Leaky ReLU不同。Leaky ReLU在负数区间的斜率是配置好固定的，RReLU在负数区间的斜率在训练中是从一个[l,u)区间的均匀分布随机采样确定的，在推理阶段，RReLU在负数区间的斜率为一个确定的数值l+u/2。

如图3-11所示为MXNet中使用RReLU的一个示例，当act_type设定为rrelu后，需要设定lower_bound和upper_bound，即对应以上描述的l和u。

图 3-11 MXNet 中 RReLU API 说明

mxnet.symbol. **LeakyReLU**(data=None, act_type='_Null', slope='_Null', lower_bound='_Null', upper_bound='_Null', name=None, attr=None, out=None, **kwargs)

Applies Leaky rectified linear unit activation element-wise to the input.

Leaky ReLUs attempt to fix the "dying ReLU" problem by allowing a small *slope* when the input is negative and has a slope of one when input is positive.

The following modified ReLU Activation functions are supported:

- *elu*: Exponential Linear Unit. *y = x > 0 ? x : slope * (exp(x)-1)*
- *leaky*: Leaky ReLU. *y = x > 0 ? x : slope * x*
- *prelu*: Parametric ReLU. This is same as *leaky* except that *slope* is learnt during training.
- *rrelu*: Randomized ReLU. same as *leaky* but the *slope* is uniformly and randomly chosen from *[lower_bound, upper_bound)* for training, while fixed to be *(lower_bound+upper_bound)/2* for inference.

由于该层的计算是基于Leaky ReLU的一种变种，需要定义的参数可以复用Caffe中ReLU参数的定义，所以不需要对caffe.proto文件进行扩展，直接复用ReLUParameter即可。

RReLU层在prototxt中进行定义的方式示例如下，layer的type定义为RReLU，negative_slope参数即根据训练时使用的upper_bound和lower_bound设定为upper_bound+lower_bound/2。

```
layer {
  name: "rrelu_layer"
  type: "RReLU"
  bottom: "some_input"
  top: "some_output"
  relu_param{
    negative_slope: xxx
  }
}
```

MatMul 层

MatMul即Matrix Multiplication，该层实现的功能即为两个输入矩阵的乘法操作。该层需要定义两个输入、一个输出。若第一个输入矩阵的维度为N*K，则第二个输入矩阵的行方向维度必须为K，列方向维度为M，那么该层实现N*K与K*M矩阵的乘法，输出的维度为N*M。两个输入矩阵的维度信息通过定义一个matmul_param传入，需要设定三个维度信息，其中dim_1为N、dim_2K、dim_3为M。

定义了该层的输入、输出以及相关的参数后，需要扩展caffe.proto文件，以使得在prototxt中定义的MatMul层和相关参数设定能够被解析。首先在caffe.proto的LayerParameter中添加MatMulParameter的参数声明。

图 3-12 在 LayerParameter 中添加 MatMulParameter

```
optional ROIPoolingParameter roi_pooling_param = 100000;
optional NormalizeParameter norm_param = 100001;
optional PSROIPoolingParameter psroi_pooling_param = 100002;
optional UpsampleParameter upsample_param = 100003;
optional PassThroughParameter pass_through_param = 100004;
optional MatMulParameter matmul_param = 100005;
```

之后在caffe.proto中的新添加一个名称为MatMulParameter的message，需要定义dim_1、dim_2、dim_3三个参数。

图 3-13 定义 MatMulParameter 参数

```
message MatMulParameter {
  optional uint32 dim_1 = 1; //row of input matrix one
  optional uint32 dim_2 = 2; //column of input matrix one and row of input matrix two
  optional uint32 dim_3 = 3; //column of input matrix two
}
```

对caffe.proto文件完成修改后，就可以解析prototxt中定义的MatMul层，示例如下。

```
layer {
  name: "some_input_one"
  type: "Input"
  top: "some_input_one"
  input_param {
    shape {
      dim: 1
      dim: 1
      dim: N
      dim: K
    }
  }
}
layer {
  name: "some_input_two"
  type: "Input"
  top: "some_input_two"
  input_param {
    shape {
      dim: 1
      dim: 1
      dim: M
      dim: K
    }
  }
}
layer {
  name: "matmul_layer"
  type: "MatMul"
  bottom: "some_input_one"
  bottom: "some_input_two"
```



```
  top: "some_output"
  matmul_param{
    dim_1: N
    dim_2: K
    dim_3: M
  }
}
```

Permute 层

Permute层实现的功能是将一个多维的Tensor的维度顺序进行切换，例如将一个NCHW顺序的Tensor转换为NHWC顺序的Tensor。NNIE的Permute层只支持对一个四维输入Tensor进行顺序转换，且转换的顺序只能够为0、2、3、1，即将原始0、1、2、3顺序的Tensor转换为新的0、2、3、1顺序的Tensor，也就是上述的NCHW顺序的Tensor转换为NHWC。

在SSD网络中有实现Permute层的操作，且SSD网络的Permute层定义了PermuteParameter，用于配置输出Tensor的维度顺序。所以需要对caffe.proto文件进行扩展，添加PermuteParameter相应的内容。

图 3-14 在 LayerParameter 中添加 PermuteParameter

```
optional ROIPoolingParameter roi_pooling_param = 100000;
optional NormalizeParameter norm_param = 100001;
optional PSROIPoolingParameter psroi_pooling_param = 100002;
optional UpsampleParameter upsample_param = 100003;
optional PassThroughParameter pass_through_param = 100004;
optional MatMulParameter matmul_param = 100005;
optional PermuteParameter permute_param = 100006;
```

图 3-15 定义 PermuteParameter 参数

```
message PermuteParameter {
  // The new orders of the axes of data. Notice it should be with
  // in the same range as the input data, and it starts from 0.
  // Do not provide repeated order.
  repeated uint32 order = 1;
}
```

具体如图3-15，Permute层需要配置的参数为order，用于配置输出Tensor的维度顺序，prototxt中配置的order参数的条目数必须与输入的维度数一致且互不相等，一个示例的Permute层定义如下。

```
layer {
  name: "permute_layer"
  type: "Permute"
  bottom: "some_input"
  top: "some_output"
  permute_param {
    order: 0
    order: 2
    order: 3
    order: 1
  }
}
```

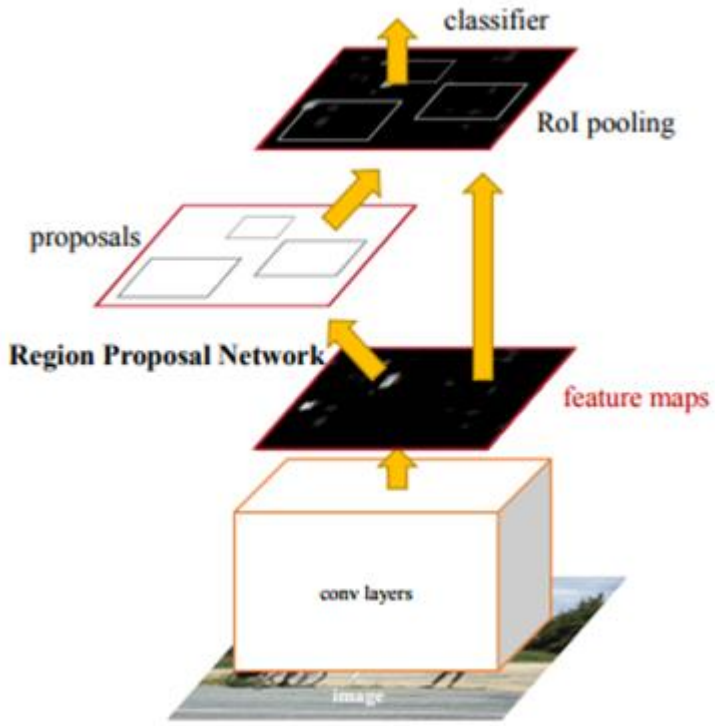

须知

目前仅支持如上示例的order配置，对一个四维输入Tensor进行0231的Permute变换。

ROIPooling 层

ROIPooling层为Faster RCNN网络中用于将不同图像经过卷积层后得到的feature map进行维度统一的操作，输入为一个feature map以及需要从中提取的ROI框坐标值，输出为一个维度归一化的feature map。如下图为一个ROIPooling在Faster RCNN网络中的应用方式，经过卷积层后，得到输入图片的feature map，feature map输入到RPN网络输出一系列ROI框的坐标数值，然后ROIPooling层根据框的坐标从feature map中选取对应的区域，然后对该区域进行一个输出维度统一的pooling操作，如7*7，即使用不同ROI框选出的不同尺寸的feature map经过ROIPooling后在spatial维度上会归一到7*7，具体做法是将选出的feature map在行、列方向上进行7等分，然后对划分的7*7=49个小区域进行Maxpooling操作，每个区域对应输出的一个数值。

图 3-16 Faster RCNN 示意图



ROIPooling层需要对caffe.proto文件进行扩展，定义ROIPoolingParameter，扩展方式示例如图3-17，定义的参数包括spatial scale，即输入feature map与原始图片的尺寸比例，用于转换ROI的坐标，因为ROI坐标是在原图尺寸上的；pooled_h、pooled_w，输出feature map在spatial维度上h和w的大小。

图 3-17 在 LayerParameter 中添加 ROIPoolingParameter

```
optional ROIPoolingParameter roi_pooling_param = 100000;
optional NormalizeParameter norm_param = 100001;
optional PSROIPoolingParameter psroi_pooling_param = 100002;
optional UpsampleParameter upsample_param = 100003;
optional PassThroughParameter pass_through_param = 100004;
optional MatMulParameter matmul_param = 100005;
optional PermuteParameter permute_param = 100006;
```

图 3-18 定义 ROIPoolingParameter 参数

```
// Message that stores parameters used by ROIPoolingLayer
message ROIPoolingParameter {
  // Pad, kernel size, and stride are all given as a single value for equal
  // dimensions in height and width or as Y, X pairs.
  optional uint32 pooled_h = 1 [default = 0]; // The pooled output height
  optional uint32 pooled_w = 2 [default = 0]; // The pooled output width
  // Multiplicative spatial scale factor to translate ROI coords from their
  // input scale to the scale used when pooling
  optional float spatial_scale = 3 [default = 1];
}
```

因此一个ROIPooling层在prototxt中定义的示例如下。

```
layer {
  name: "roipooling_layer"
  type: "ROIPooling"
  bottom: "some_input_feature_map"
  bottom: "some_input_rois"
  top: "some_output"
  roi_pooling_param {
    spatial_scale: 0.0625
    pooled_h: 7
    pooled_w: 7
  }
}
```

PSROIPooling 层

PSROIPooling层的操作与ROIPooling层类似，不同之处在于不同空间维度输出的图片特征来自不同的feature map channels，且对每个小区域进行的是Average Pooling，不同于ROIPooling的Max Pooling，如图3-19为一个PSROIPooing层的示意图。

图 3-19 PSROIPooing 示意图

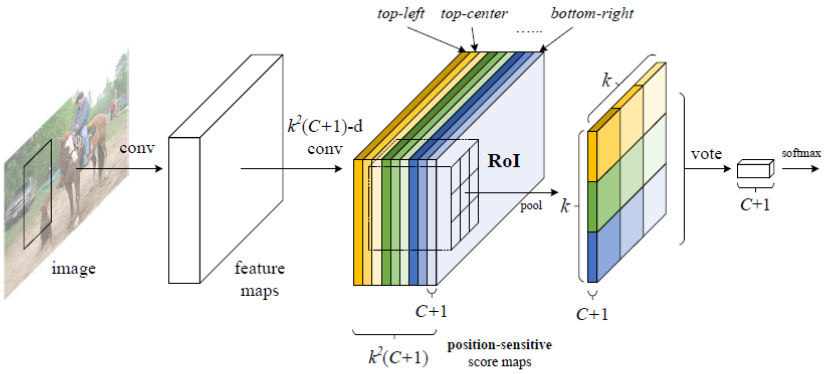


Figure 1: Key idea of **R-FCN** for object detection. In this illustration, there are $k \times k = 3 \times 3$ position-sensitive score maps generated by a fully convolutional network. For each of the $k \times k$ bins in an RoI, pooling is only performed on one of the k^2 maps (marked by different colors).

对于一个输出k*k的结果，不同空间维度的特征取自输入feature map中不同的组，即将输入的feature map均匀分为k*k组，每组的channel数与输出的channel一致，得到上述输出。

PSROIPooling层需要对caffe.proto文件进行扩展，定义PSROIPoolingParameter，扩展方式示例如图3-20，定义的参数包括spatial scale，即输入feature map与原始图片的尺寸比例；output_dim，输出feature map的channel数；group_size，输出的spatial维度，即上述的k。

图 3-20 在 LayerParameter 中添加 PSROIPoolingParameter

```
optional ROIPoolingParameter roi_pooling_param = 100000;
optional NormalizeParameter norm_param = 100001;
optional PSROIPoolingParameter psroi_pooling_param = 100002;
optional UpsampleParameter upsample_param = 100003;
optional PassThroughParameter pass_through_param = 100004;
optional MatMulParameter matmul_param = 100005;
optional PermuteParameter permute_param = 100006;
```

图 3-21 定义 PSROIPoolingParameter 参数

```
message PSROIPoolingParameter {
  required float spatial_scale = 1;
  required int32 output_dim = 2; // output channel number
  required int32 group_size = 3; // equal to pooled_size
}
```

因此一个PSROIPooling层在prototxt中定义的示例如下。

```
layer {
  name: "psroipooling_layer"
  type: "PSROIPooling"
  bottom: "some_input_feature_map"
  bottom: "some_input_rois"
  top: "some_output"
  psroi_pooling_param {
```

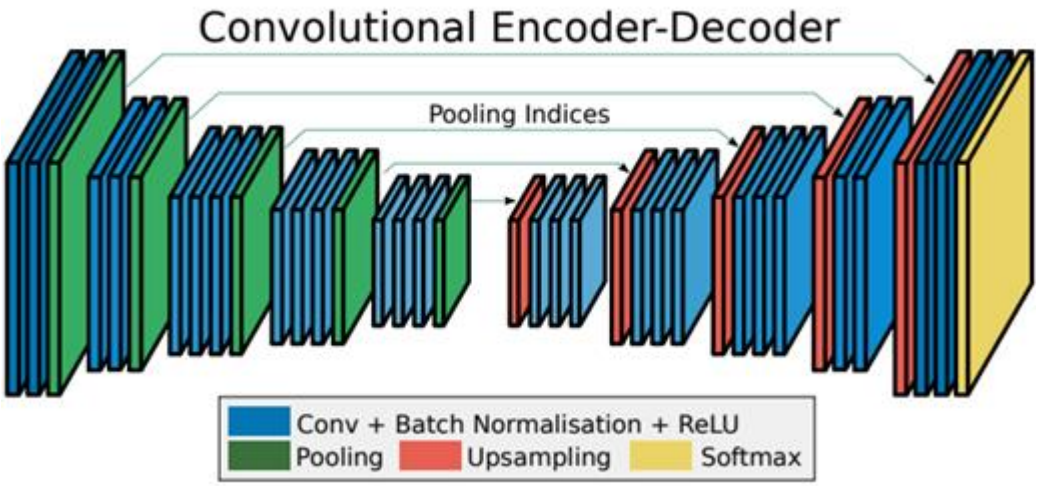


```
    spatial_scale: 0.0625
    output_dim: 21
    group_size: 7
  }
}
```

Upsample 层

Upsample层为Pooling层的逆操作，下图为一个示意图，其中每个Upsample层均与网络之前一个对应大小输入、输出Pooling层一一对应，完成feature map在spatial维度上的扩充。

图 3-22 Upsample 层示意图



Upsample层需要对caffe.proto文件进行扩展，定义UpsampleParameter，扩展方式示例如下图，定义的参数包括scale，即输出与输入的尺寸比例，如2；scale_h、scale_w，同scale，分别为h、w方向上的尺寸比例；pad_out_h、pad_out_w，仅在scale为2时有用，对输出进行额外padding在h、w方向上的数值；upsample_h、upsample_w，输出图像尺寸的数值。在Upsample的相关代码中，推荐仅仅使用upsample_h、upsample_w准确定义Upsample层的输出尺寸，其他所有的参数都不推荐继续使用。

图 3-23 在 LayerParameter 中添加 UpsampleParameter

```
optional ROIPoolingParameter roi_pooling_param = 100000;
optional NormalizeParameter norm_param = 100001;
optional PSROIPoolingParameter psroi_pooling_param = 100002;
optional UpsampleParameter upsample_param = 100003;
optional PassThroughParameter pass_through_param = 100004;
optional MatMulParameter matmul_param = 100005;
optional PermuteParameter permute_param = 100006;
```


图 3-24 定义 UpsampleParameter 参数

```
message UpsampleParameter {
  // DEPRECATED. No need to specify upsampling scale factors when
  // exact output shape is given by upsample_h, upsample_w parameters.
  optional uint32 scale = 1 [default = 2];
  // DEPRECATED. No need to specify upsampling scale factors when
  // exact output shape is given by upsample_h, upsample_w parameters.
  optional uint32 scale_h = 2;
  // DEPRECATED. No need to specify upsampling scale factors when
  // exact output shape is given by upsample_h, upsample_w parameters.
  optional uint32 scale_w = 3;
  // DEPRECATED. Specify exact output height using upsample_h. This
  // parameter only works when scale is 2
  optional bool pad_out_h = 4 [default = false];
  // DEPRECATED. Specify exact output width using upsample_w. This
  // parameter only works when scale is 2
  optional bool pad_out_w = 5 [default = false];
  optional uint32 upsample_h = 6;
  optional uint32 upsample_w = 7;
}
```

支持两种Upsample，通过bottom个数区分。第一种bottom为1，是Yolov3的Upsample层，在prototxt中定义的示例如下：

```
layer {
  name: "upsample_layer"
  type: "Upsample"
  bottom: "some_input_feature_map"
  top: "some_output"
  upsample_param {
    scale: 2
}
}
```

第二种bottom为2，是Segnet的Upsample层，在prototxt中定义的示例如下。

```
layer {
  name: "upsample_layer"
  type: "Upsample"
  bottom: "some_input_feature_map"
  bottom: "some_input_pool_index"
  top: "some_output"
  upsample_param {
    upsample_h: 224
    upsample_w: 224
}
}
```

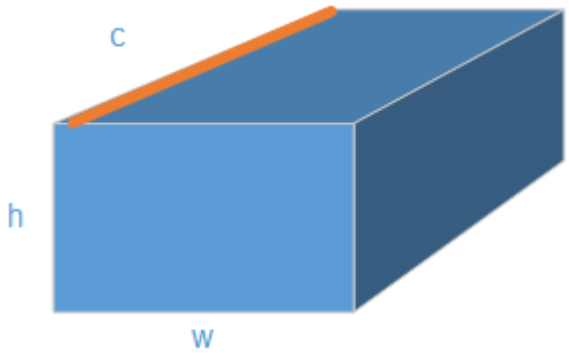
Normallize 层

Normalize层为SSD网络中一个层结构，其进行的操作为对于一个c*h*w的三维tensor，输出是同样大小的tensor，其中间计算为每个元素以channel方向的平方和的平方根求normalize，其具体计算公式为：

$$x_i' = \frac{x_i}{\sqrt{\sum_{i=1}^{c} x_i^2}}$$

其中分母位置的平方和的累加向量为同一h与w位置的所有c方向的向量，如图3-25中的橙色区域。

图 3-25 Normalize 层计算示意图



经过上述计算归一化后，再对每个feature map做scale， 每个channel对应一个scale值。

Normalize层需要对caffe.proto文件进行扩展，定义NormalizeParameter，扩展方式示例如下图，定义的参数包括across_spatial，是否在channel或者spatial维度上做normalization，目前仅支持False的情形，即只支持在channel维度做normalization；scale_filler，对feature map进行scale时scale参数的初始化方法；channel_shared，scale在channel方向是否共享，默认为True；eps，防止normalize过程中除以0的一个很小数值，默认为1e-10，且不支持配置；此外在NNIE支持的Normalize层中，层的参数相比开源实现新添加了sqrt_a参数，该参数是用于保证NNIE在定点处理过程中数据不会发生溢出。为了防止在上述计算normalize过程中上溢出，引入系数a，normalize的计算公式转换为：

$$x_i' = \frac{x_i}{\sqrt{\sum_{i=1}^{c} x_i^2}} = \frac{\sqrt{\alpha}\, x_i}{\sqrt{\alpha * \sum_{i=1}^{c} x_i^2}} = \frac{\sqrt{\alpha}\, x_i}{\sqrt{\sum_{i=1}^{c} (\sqrt{\alpha} * x_i) * (\sqrt{\alpha} * x_i)}}$$

其中，系数 α 可以通过离线数据统计得到，记 $sum = \sum_{i=1}^{c} x_i^2$ ，即保证 $\alpha * |sum|_{max} \le 2^{19} + 2^{-12}$ （数据以20.12定点方式储存），且保证 $\sqrt{a} \ge 2^{-12}$ （$\sqrt{a}$ 可以20.12定点方式储存），即保证系数 $\sqrt{a}$ 不会下溢出。$\sqrt{a}$ 即为sqrt_a，是normalize层中需要定义的参数。

sqrt_a的默认值为1，即如果用户没有在prototxt中进行配置的话，该值为1；如果配置了sqrt_a参数，则为实际配置的数值。

图 3-26 在 LayerParameter 中添加 NormalizeParameter

```
optional ROIPoolingParameter roi_pooling_param = 100000;
optional NormalizeParameter norm_param = 100001;
optional PSROIPoolingParameter psroi_pooling_param = 100002;
optional UpsampleParameter upsample_param = 100003;
optional PassThroughParameter pass_through_param = 100004;
optional MatMulParameter matmul_param = 100005;
optional PermuteParameter permute_param = 100006;
```


图 3-27 定义 NormalizeParameter 参数

```
// Message that stores parameters used by NormalizeLayer
message NormalizeParameter {
  optional bool across_spatial = 1 [default = true];
  // Initial value of scale. Default is 1.0 for all
  optional FillerParameter scale_filler = 2;
  // Whether or not scale parameters are shared across channels.
  optional bool channel_shared = 3 [default = true];
  // Epsilon for not dividing by zero while normalizing variance
  optional float eps = 4 [default = 1e-10];
  // square root of a value used in normalize
  optional float sqrt_a = 5 [default = 1];
}
```

一个示例的Normalize层定义如下。

```
layer {
  name: "normalize_layer"
  type: "Normalize"
  bottom: "some_input"
  top: "some_output"
  normalize_param {
  across_spatial: False
    channel_shared: True
    sqrt_a: 0.5
}
}
```

3.1.4.4 扩展层的参考设计

Caffe官方提供的开发新层的示例，地址https://github.com/BVLC/caffe/wiki/Development。

ROIPooling，参考设计为Faster RCNN网络，开源工程地址**https://github.com/rbgirshick/py-faster-rcnn**。

Normalize，参考设计为SSD网络，开源工程地址**https://github.com/weiliu89/caffe/tree/ssd**。

PSROIPooling，参考设计为RFCN网络，开源工程地址**https://github.com/daijifeng001/caffe-rfcn**。

Upsample，支持两种计算方式。

- 第一种参考设计为Yolov3网络，开源工程地址**https://github.com/pjreddie/darknet**。
- 第二种参考设计为SegNet网络，开源工程地址**https://github.com/alexgkendall/caffe-segnet**，
- 注意在使用Upsample层时，需要配合网络在Upsample层之前的Maxpooling层输出的max元素的最大值，Averagepooling由于无法输出对应的max元素的mask，所以无法配合Upsample层使用。

CReLU层，参考设计为PVANet网络，开源工程地址**https://github.com/sanghoon/pva-faster-rcnn**。

上述自定义层为基于Caffe框架实现的，下述的自定义层基于其他深度学习框架，用于参考基本实现原理。

PassThrough层，参考设计为YOLO v2网络，开源工程地址为**https://pjreddie.com/darknet/yolo/**。

Depthwise Convolution层，参考设计为Xeption网络，开源工程地址为**https://github.com/fchollet/keras/blob/master/keras/applications/xception.py**。

RReLU层，参考论文中的定义**https://arxiv.org/abs/1505.00853**以及MXNet API中的实现**http://mxnet.io/api/python/symbol.html**。

Permute层，参考设计为SSD网络，开源工程地址**https://github.com/weiliu89/caffe/tree/ssd**。

3.1.5 Non-support 层处理方式

当网络中存在Non-support层时，需要将网络进行切分，不支持的部分由用户使用CPU或者DSP等方式实现，统称为非NNIE方式。由此整个网络会出现NNIE->非NNIE->NNIE… 的分段执行方式。

nnie_mapper将NNIE的Non-support层分为两种，“Proposal”层和“Custom”层：

- Proposal层输出的是矩形信息（Bbox，即Bounding box）；Proposal自定义层格式如下，包含name\type\bottom\top等关键字段参数，type必须为“Proposal”，支持任意多个的bottom，但是top只有一个。

```
layer {
  name: "proposal"
  type: "Proposal"
  bottom: "rpn_cls_prob_reshape"
  bottom: "rpn_bbox_pred"
  top: "rois"
}
```

Proposal层还支持以下参数的解析，但不产生实际的效果。

Proposal	uint32	feat_stride	默认值16
	uint32	base_size	默认值16
	uint32	min_size	默认值16
	float	ratio	NA
	float	scale	NA
	uint32	pre_nms_topn	默认值6000
	uint32	post_nms_topn	默认值300
	float	nms_thresh	默认值0.7
	float	remove_thresh	默认值0

- Custom层输出的是tensor（feature map、vector）；Custom自定义层格式如下，仅包含name\type\bottom\top\ custom_param\shape\dim关键字段，type必须为“Custom”，支持任意多个top\bottom，custom_param中的shape信息是描述top输出的形状，如果有多个输出，则按top顺序书写多个shape。用户原有其他的参数字段需删除。

```
layer {
 name: "custom1"
 type: "Custom"
 bottom: "conv1"
 bottom: "conv2"
 bottom: "pooling1"
```



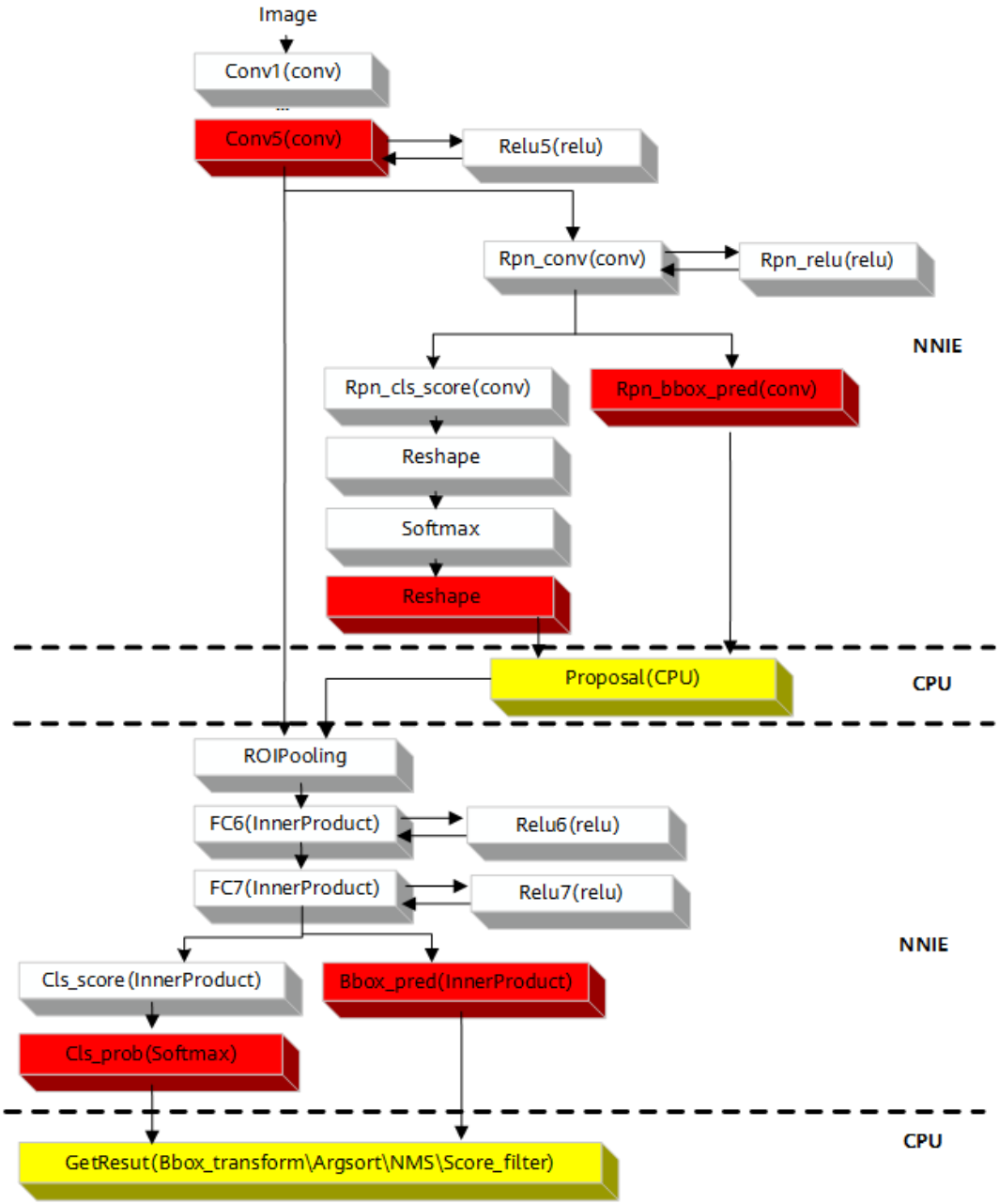
```
top: "custom1_1"
top: "custom1_2"
custom_param
{
 shape {
 dim: 1
 dim: 256
 dim: 64
 dim: 64
 }
 shape {
 dim: 1
 dim: 128
 dim: 128
 dim: 128
 }
}
}
```

在nnie_mapper对网络模型进行转化之前，用户需根据上述特性将Non-support层修改为“Proposal”层或者“Custom”层，具体修改方式参考“**3.2 Prototxt要求**”。

图3-28以Faster RCNN为例，Faster RCNN中RPN部分的Proposal层(Bbox_transform、Argsort、NMS等操作)输出的是Bounding Box信息。图示的Faster RCNN分为如下的几段来执行（非NNIE图示均由CPU来执行）：

a. 第1段（NNIE执行），卷积部分，包含RPN前面的部分；
b. 第2段（非NNIE执行），Proposal部分，生成矩形框信息；
c. 第3段（NNIE执行），FC部分，包含ROIPooling，得到矩形框调整值和置信度；
d. 第4段（非NNIE执行），结果获取部分，由FC部分的矩形框调整值和置信度经过bbox_transfomr、argsort、NMS、阈值过滤等步骤获取最终的矩形框和置信度信息，CPU执行。

图 3-28 Faster RCNN 分段执行示意图



3.1.6 NNIE 规格

3.1.6.1 层级联关系

网络层的级联关系如表3-1所示；请注意表格中包含有NNIE的Non-support的“Proposal”层和“Custom”层的级联关系。

表 3-1 SVP NNIE 网络层级联关系表

Layer type	Convolution Deconvolution	Pooling DepthwiseConv	InnerProduct	LRN	BatchNorm	Scale Bias	Eltwise	ReLU PreLU AbsVal TanH Sigmoid BNLL ELU
Convolution Deconvolution	●	●	●	●	●	●	●	●
Pooling Depthwise Conv	●	●	●	●	●	●	●	●
InnerProduct	╳	╳	●	╳	●	●	●	●
LRN	●	●	●	●	●	●	●	●
BatchNorm	●	●	●	●	●	●	●	●
Scale Bias	●	●	●	●	●	●	●	●
Eltwise	●	●	●	●	●	●	●	●
ReLU PReLU AbsVal TanH Sigmoid BNLL ELU	●	●	●	●	●	●	●	●
CReLU	●	●	●	●	●	●	●	●
LSTM RNN	╳	╳	●	╳	╳	╳	╳	●
Softmax	●	●	●	●	●	●	●	●
Exp Log	●	●	●	●	●	●	●	●

Layer type	Convolution Deconvolution	Pooling DepthwiseConv	InnerProduct	LRN	BatchNorm	Scale Bias	Eltwise	ReLU PreLU AbsVal TanH Sigmoid BNLL ELU
Reshape Flatten	●	●	●	●	●	●	●	●
Split	●	●	●	●	●	●	●	●
Slice	●	●	●	●	●	●	●	●
Concat	●	●	●	●	●	●	●	●
SPP	╳	╳	●	╳	●	●	●	●
Power	●	●	●	●	●	●	●	●
Threshold	●	●	●	●	●	●	●	●
MVN	●	●	●	●	●	●	●	●
Reduction	●	●	●	●	●	●	●	●
Proposal	╳	╳	╳	╳	╳	╳	╳	╳
ROIPooling	●	●	●	●	●	●	●	●
PSROIPooling	●	●	●	●	●	●	●	●
Normalize	●	●	●	●	●	●	●	●
PassThrough	●	●	●	●	●	●	●	●
Upsample	●	●	●	●	●	●	●	●
Permute	●	●	●	●	●	●	●	●
MatMul	╳	╳	╳	╳	╳	╳	╳	╳
Custom	●	●	●	●	●	●	●	●

Layer type	CReLU	LSTM RNN	Softmax	Exp Log	Reshape Flatten	Split	Slice	Concat
Convolution Deconvolution	●	╳	●	●	●	●	●	●

Layer type	CReLU	LSTM RNN	Softmax	Exp Log	Reshape Flatten	Split	Slice	Concat
Pooling DepthwiseConv	●	╳	●	●	●	●	●	●
InnerProduct	╳	╳	●	●	●	●	●	●
LRN	●	╳	●	●	●	●	●	●
BatchNorm	●	╳	●	●	●	●	●	●
Scale Bias	●	╳	●	●	●	●	●	●
Eltwise	●	╳	●	●	●	●	●	●
ReLU PreLU AbsVal TanH Sigmoid BNLL ELU	●	╳	●	●	●	●	●	●
CReLU	●	╳	●	●	●	●	●	●
LSTM RNN	╳	╳	╳	╳	╳	╳	╳	╳
Softmax	●	╳	●	●	●	●	●	●
Exp Log	●	╳	●	●	●	●	●	●
Reshape Flatten	●	╳	●	●	●	●	●	●
Split	●	╳	●	●	●	●	●	●
Slice	●	╳	●	●	●	●	●	●
Concat	●	╳	●	●	●	●	●	●
SPP	╳	╳	●	●	●	●	●	●
Power	●	╳	●	●	●	●	●	●
Threshold	●	╳	●	●	●	●	●	●
MVN	●	╳	●	●	●	●	●	●
Reduction	●	╳	●	●	●	●	●	●
Proposal	╳	╳	╳	╳	╳	╳	╳	╳
ROIPooling	●	╳	●	●	●	●	●	●

Layer type	CReLU	LSTM RNN	Softmax	Exp Log	Reshape Flatten	Split	Slice	Concat
PSROIPooling	●	╳	●	●	●	●	●	●
Normalize	●	╳	●	●	●	●	●	●
PassThrough	●	╳	●	●	●	●	●	●
Upsample	●	╳	●	●	●	●	●	●
Permute	●	╳	●	●	●	●	●	●
MatMul	╳	╳	╳	╳	╳	╳	╳	╳
Custom	●	╳	●	●	●	●	●	●

Layer type	SPP	Power	Threshold	MVN	Reduction	Proposal	ROIPooling
Convolution Deconvolution	●	●	●	●	●	●	●
Pooling DepthwiseConv	●	●	●	●	●	●	●
InnerProduct	╳	●	●	╳	╳	●	╳
LRN	●	●	●	●	●	●	●
BatchNorm	●	●	●	●	●	●	●
Scale Bias	●	●	●	●	●	●	●
Eltwise	●	●	●	●	●	●	●
ReLU PreLU AbsVal TanH Sigmoid BNLL ELU	●	●	●	●	●	●	●
CReLU	●	●	●	●	●	●	●
LSTM RNN	╳	╳	╳	╳	╳	╳	╳
Softmax	●	●	●	●	●	●	●

Layer type	SPP	Power	Threshold	MVN	Reduction	Proposal	ROIPooling
Exp Log	●	●	●	●	●	●	●
Reshape Flatten	●	●	●	●	●	●	●
Split	●	●	●	●	●	●	●
Slice	●	●	●	●	●	●	●
Concat	●	●	●	●	●	●	●
SPP	╳	●	●	╳	╳	●	╳
Power	●	●	●	●	●	●	●
Threshold	●	●	●	●	●	●	●
MVN	●	●	●	●	●	●	●
Reduction	●	●	●	●	●	●	●
Proposal	╳	╳	╳	╳	╳	╳	●
ROIPooling	●	●	●	●	●	╳	╳
PSROIPooling	●	●	●	●	●	╳	╳
Normalize	●	●	●	●	●	●	●
PassThrough	●	●	●	●	●	●	●
Upsample	●	●	●	●	●	●	●
Permute	●	●	●	●	●	●	●
MatMul	╳	╳	╳	╳	╳	╳	╳
Custom	●	●	●	●	●	╳	●

Layer type	PSROIPooling	Normalize	PassThrough	Upsample	Permute	MatMul	Custom	as input	as output
Convolution Deconvolution	●	●	●	●	●	╳	●	●	●
Pooling Depthwise Conv	●	●	●	●	●	╳	●	●	●

Layer type	PSROIPooling	Normalize	PassThrough	Upsample	Permute	MatMul	Custom	as input	as output
InnerProduct	✗	✗	✗	✗	✗	✗	●	●	●
LRN	●	●	●	●	●	✗	●	●	●
BatchNorm	●	●	●	●	●	✗	●	●	●
Scale Bias	●	●	●	●	●	✗	●	●	●
Eltwise	●	●	●	●	●	✗	●	●	●
ReLU PreLU AbsVal TanH Sigmoid BNLL ELU	●	●	●	●	●	✗	●	●	●
CReLU	●	●	●	●	●	✗	●	●	●
LSTM RNN	✗	✗	✗	✗	✗	✗	✗	✗	●
Softmax	●	●	●	●	●	✗	●	●	●
Exp Log	●	●	●	●	●	✗	●	●	●
Reshape Flatten	●	●	●	●	●	✗	●	●	●
Split	●	●	●	●	●	✗	●	●	●
Slice	●	●	●	●	●	✗	●	●	●
Concat	●	●	●	●	●	✗	●	●	●
SPP	✗	✗	✗	✗	✗	✗	●	●	●
Power	●	●	●	●	●	✗	●	●	●
Threshold	●	●	●	●	●	✗	●	●	●
MVN	●	●	●	●	●	✗	●	●	●
Reduction	●	●	●	●	●	✗	●	●	●
Proposal	●	✗	✗	✗	✗	✗	●	✗	●

Layer type	PSROIPooling	Normalize	PassThrough	Upsample	Permute	MatMul	Custom	as input	as output
ROIPooling	╳	●	●	●	●	╳	●	●	●
PSROIPooling	╳	●	●	●	●	╳	●	●	●
Normalize	●	●	●	●	●	╳	●	●	●
PassThrough	●	●	●	●	●	╳	●	●	●
Upsample	●	●	●	●	●	╳	●	●	●
Permute	●	●	●	●	●	╳	●	●	●
MatMul	╳	╳	╳	╳	╳	╳	╳	●	●
Custom	●	●	●	╳	●	╳	╳	╳	╳

3.1.6.2 层规格描述

网络层的具体规格如表3-2~表3-4所示。

表 3-2 NNIE 支持的标准层规格表

layer	参数类型	参数名	参数含义	描述	规格	
AbsVal	无	无	无	求绝对值	支持	w<=4096
BatchNorm	bool	use_global_stats	false表示使用滑动平均；true表示使用全局的，不每次进行计算	-	支持true	w<=4096
	float	moving_average_fraction	默认为.999，每次迭代滑动平均衰减系数		不支持配置	
	float	eps	默认为1e-5，用来防止被0除		不支持配置	

layer	参数类型	参数名	参数含义	描述	规格	
Bias	int32	axis	默认是1，不同的值对应的相加维度不同 For example, if bottom[0] is 4D with shape 100x3x40x60, the output top[0] will have the same shape, and bottom[1] may have any of the following shapes (for the given value of axis): (axis == 0 == -4) 100; 100x3; 100x3x40; 100x3x40x60 (axis == 1 == -3) 3; 3x40; 3x40x60 (axis == 2 == -2) 40; 40x60 (axis == 3 == -1) 60	计算两个输入的elementwise和	支持两种配置(1,-3)，默认值1，遵循caffe规则	w<=4096
	int32	num_axes	默认为1，当只有一个bottom时有效，指定训练的bias参数维度大小		支持配置1或者-1，与axis配套使用，在张量与C维度向量相乘时配置为-1，其余情况为1，遵循caffe规则；	
BNLL	无	无	无	按照一固定公式计算，激活函数	支持	w<=4096
Concat	int32	axis	默认为1，表示哪个维度concat，可以为负数	至少两个输入，将它们在某个维度上拼接起来	支持配置，-4~3，不支持N维度的concat	axis 1: w<=UINT_MAX axis 2: w<=16384 axis 3: w<=4096 vecotr: w<=UINT_MAX bottom num<=32
	uint32	concat_dim	DEPRECATED；跟axis含义相同，不支持负值		支持配置0~3，与axis配套使用，遵循caffe规则；	

layer	参数类型	参数名	参数含义	描述	规格	
Convolution	uint32	num_output	输出的channel大小	对输入做卷积	支持配置，最大32768；	w<=4096 kernel_h <=2048/(w/(16*stride_w)*stride_w)； kernel_h * kernel_w < 640
	bool	bias_term	默认为1，表示是否加bias		支持配置0或1；	
	uint32	pad	默认为0，表示补边的大小		支持，配置范围0~255；	
	uint32	kernel_size	kernel的大小		支持1~255	
	uint32	stride	默认为1，stride的大小		支持1~255	
	uint32	dilation	默认为1，dilation的大小		支持1~255，配置后(kernel-1)*dilation +1<256;	
	uint32	pad_h	默认为0，表示height方向的padding		支持，配置范围0~255；	
	uint32	pad_w	默认为0，表示width方向的padding		支持，配置范围0~255；	
	uint32	kernel_h	kernel的height大小		支持，配置范围1~255；	
	uint32	kernel_w	kernel的width大小		支持，配置范围1~255；	
	uint32	stride_h	stride的height大小		支持1~255	
	uint32	stride_w	stride的width大小		支持1~255	
	uint32	group	默认为1，group的大小		支持最大2048	
	int32	axis	默认为1，axis值之前的维度认为是独立输入，axis之后的维度认为是空间维度，即在做卷积时会求和		不支持配置	

layer	参数类型	参数名	参数含义	描述	规格	
Deconvolution	uint32	num_output	输出的channel大小	卷积的反过程，将卷积的前向传播和后向传播置换	支持配置，最大32768；	w<=4096 kernel_h <= (32768/w-1)*stride_h + 1; kernel_h * kernel_w < 640
	bool	bias_term	默认为1，表示是否加bias		支持配置0或1；	
	uint32	pad	默认为0，表示补边的大小		支持，配置范围0~255；	
	uint32	kernel_size	kernel的大小		支持1~255	
	uint32	stride	默认为1，stride的大小		支持1~255	
	uint32	dilation	默认为1，dilation的大小		支持1~255，配置后(kernel-1)*dilation +1<256;	
	uint32	pad_h	默认为0，表示height方向的padding		支持，配置范围0~255；	
	uint32	pad_w	默认为0，表示width方向的padding		支持，配置范围0~255；	
	uint32	kernel_h	kernel的height大小		支持，配置范围1~255；	
	uint32	kernel_w	kernel的width大小		支持，配置范围1~255；	
	uint32	stride_h	stride的height大小		支持1~255	
	uint32	stride_w	stride的width大小		支持1~255	
	uint32	group	默认为1，group的大小		支持最大2048	
	int32	axis	默认为1，axis值之前的维度认为是独立输入，axis之后的维度认为是空间维度，即在做卷积时会求和		不支持配置	

layer	参数类型	参数名	参数含义	描述	规格	
Eltwise	EltwiseOp	operation	默认为SUM，element-wise的操作，加、乘、求最大	进行elementwise的计算操作，包括相加、相乘、求最大	支持,相加、相乘、求最大；	w<=4096 bottom num<=32
	float	coeff	blob_wise系数		支持	
	bool	stable_prod_grad	对于乘操作，大于两输入的情况，选择是否使用更慢但更稳定的方法求梯度		不支持配置	
ELU	float	alpha	默认为1，公式中的alpha系数	一种固定公式的激活函数	支持	w<=4096
Exp	float	base	默认为-1.0	计算exp	支持，-1或是正数	w<=4096
	float	scale	默认为1.0		支持	
	float	shift	默认为0.0		支持	
Flatten	int32	axis	flatten开始的维度	将输入的blob转成向量	axis取值范围[-4,3]，默认值为1，-4与0含义相同。不支持N维度。	w<=65536
	int32	end_axis	flatten结束的维度		end_axis取值范围[-4,3]，默认值为-1，表示最后一条轴。 End_axis应大于axis。	
InnerProduct	uint32	num_output	输出的维度大小	全连接层，计算内积	支持，最大支持32768；	w<=4096
	bool	bias_term	是否加bias		支持配置0或1；	
	int32	axis	进行内积计算的维度		不支持配置 当LSTM后面接FC时，支持且仅支持axis=2	
	bool	transpose	是否转置权值矩阵		支持配置0或1；	

layer	参数类型	参数名	参数含义	描述	规格	
Input	BlobShape	shape	定义blob的大小，可以每个分别定义，可以统一定为同样大小，可以不定义	-	支持最大input节点16个	w<=4096
Log	float	base	默认为-1.0	计算log	支持-1和除了1的任意正数	w<=4096
	float	scale	默认为1.0		支持任意值	
	float	shift	默认为0.0		支持任意值	
LRN	uint32	local_size	默认为5	一种normalization的方法	支持3、5、7	w<=4096
	float	alpha	默认为1.0		支持	
	float	beta	默认为0.75		支持	
	NormRegion	norm_region	默认为ACROSS_CHANNELS，可选WITHIN_CHANNEL		只支持ACROSS_CHANNEL	
	float	k	默认为1.0		支持	
LSTM	uint32	num_output	默认为0，输出的维度大小	LSTM网络	支持，最大支持5456	num_output<=5456
	bool	debug_info	默认为false,是否输出debug_info		不支持配置	
	bool	expose_hidden	默认为false,表示是否增加额外输入输出层，对于LSTM是同时增加2层		支持配置0或1；	
MVN	bool	normalize_variance	默认为true, 设为false时只减均值	x'=(x-mean(x))/std(x)	支持配置true或false	w<=4096
	bool	across_channels	默认为flase, 设为true时就是CHW视为一个向量		支持配置true或false	
	float	eps	默认为1e-9，用来防止除0		不支持配置	
Pooling	PoolMethod	pool	默认为MAX，pooling的类型，另一种是AVE	对输入做pooling	支持	w<=4096 kernel_h<=2048/(w/(16*stride_

layer	参数类型	参数名	参数含义	描述	规格	
	uint32	pad	默认为0，表示补边的大小		支持，配置范围0~255；	w)*stride_w)
	uint32	pad_h	默认为0，表示height方向的padding		支持，配置范围0~255；	
	uint32	pad_w	默认为0，表示width方向的padding		支持，配置范围0~255；	
	uint32	kernel_size	kernel的大小		支持，配置范围1~255；	
	uint32	kernel_h	kernel的height大小		支持，配置范围1~255；	
	uint32	kernel_w	kernel的width大小		支持，配置范围1~255；	
	uint32	stride	默认为1，stride的大小		支持，配置范围1~255；	
	uint32	stride_h	stride的height大小		支持，配置范围1~255；	
	uint32	stride_w	stride的width大小		支持，配置范围1~255；	
	RoundMode	round_mode	默认为CEIL，计算输出形状的取整方式。		支持配置CEIL或FLOOR；	
	bool	global_pooling	默认为false，是否全平面做pooling		支持配置0或1；	
Power	float	power	默认为1.0	f(x)=(shift+scale∗x)^power	支持部分配置；-3、-2、-1、-1/2、1/2、1、2、3；	w<=4096
	float	scale	默认为1.0		支持	
	float	shift	默认为0.0		支持	

layer	参数类型	参数名	参数含义	描述	规格	
PReLU	bool	channel_shared	默认为false，表示负向斜率是否channel共享	带参数的ReLU激活函数	支持	w<=4096
Reduction	ReductionOp	operation	默认为SUM，可选ASUM、SUMSQ、MEAN	将输入的blob reduction成一个scalar	支持SUM、ASUM、SUMSQ、MEAN	w<=4096
	int32	axis	默认为0，表示从第几个维度开始后面的维度全部reduction		支持[2,3]	
	float	coeff	默认为1.0，输出的系数		支持	
ReLU	float	negative_slope	默认为0，负值的斜率	ReLU激活函数	支持	w<=4096
Reshape	BlobShape	shape	指示输出的维度大小，0表示跟bottom一致，-1表示该维度由输入的blob及输出的其他维度决定	维度变换	支持[1,4]个shape_dim配置	w<=65536
	int32	axis	默认为0，表示shape中第一数值与输出的第几个起始维度对应		支持[-4,3]范围内可配，不支持N维度，默认值0，遵循caffe规则	
	int32	num_axes	默认为-1，表示从axis开始做几个维度的reshape		支持[-1,3]范围内可配，默认值-1表示对axis起始的所有轴进行变换	
RNN	uint32	num_output	默认为0，输出的维度大小	RNN网络	支持，最大支持16384	num_output<=16384
	bool	debug_info	默认为false,是否输出debug_info		不支持配置	
	bool	expose_hidden	默认为false,表示是否增加额外输入输出层，对于RNN是增加1层		支持配置0或1；	

layer	参数类型	参数名	参数含义	描述	规格	
Scale	int32	axis	默认为1，输入的bottom[0]做scale的起始维度，axis不同则bottom[1]的shape不同	计算两个输入的elementwise乘，支持两个输入维度大小不一致	支持两种配置(1,-3)，默认值1，遵循caffe规则	w<=4096
	int32	num_axes	默认是1，当只有一个bottom的时候有效，表示做scale的维度，-1表示axis开始全部，0表示是参数只有一个scalar		支持配置1或者-1，与axis配套使用，在张量与C维度向量相乘时配置为-1，其余情况为1，遵循caffe规则；	
	bool	bias_term	默认为false，表示是否加bias		支持配置0或1；	
Sigmoid	NA	NA	NA	simoid激活函数	支持	w<=4096
Slice	int32	axis	默认为1，表示切块的维度	沿某个维度切割输入的blob，输出多个切割的blob	支持[-4,3]范围内可配，不支持N维度，默认值1，遵循caffe规则	top num<=32
	uint32	slice_point	表示输出的各个blob的切割点，不定义的则均分		支持，切割点数小于32；	
	uint32	slice_dim	默认为1，跟axis含义相同		支持配置[1,3]，与axis配套使用，遵循caffe规则；	
Softmax	int32	axis	默认为1，计算softmax的维度	计算softmax函数	支持配置[1,2,3]；	axis 1: vector c<=65536 tensor c<=4096 axis 3: w<=4096
Split	NA	NA	NA	复制输入的blob到多个节点	支持	top num<=32

layer	参数类型	参数名	参数含义	描述	规格	
SPP	uint32	pyramid_height	spp的高，即做2^n次pooling	对不同的输入做金字塔的pooling，输出一样的大小	支持，2^n次，n<7； pooling kernel小于等于255	等同pool+reshape+concat的规格
	Pool Method	pool	默认为MAX，pooling的类型，另一种是AVE		支持	
TanH	NA	NA	NA	tanh激活函数	支持	w<=4096
Threshold	float	threshold	默认为0	输入与一个阈值相比较，大于则输出1，小于等于则输出0	支持	w<=4096

表 3-3 NNIE 支持的扩展层规格表

layer	参数类型	参数名	参数含义	描述	规格	
Normalize	bool	across_spatial	是否在channel方向做L2 norm	L2 norm	支持:false：在channel维度做normalize；不支持:true：在HW维度做normalize；（SSD采用在channel维度做）	w<=4096
	FillerParameter	scale_filler	scale参数初始化方式		支持，如果没有这个参数则从caffemodel去读取	
	bool	channel_shared	scale参数在channel间是否共享		不支持true：只用scale_filler[0]或者caffemodel中参数[0]； 支持false：全用；	
	float	eps	默认为1e-10，用来防止被0除		不支持配置	
	float	sqrt_a	数据缩放比例，防止定点处理溢出		支持，sqrt_a>0，默认值为1	
PSROIPooling	float	spatial_scale	进行ROIPooling特征输入的尺寸相比原始输入的比例	类似ROIPooling，不同位置的输出来自不同	支持，从原图坐标生成feature map坐标，下到指令	w<=4096 2048/(w*(output_dim/16))/

layer	参数类型	参数名	参数含义	描述	规格	
	int32	output_dim	输出channel数量	channel的输入	支持，输出ROI的channel维度大小	(group_size*group_size/16) > 2 w*h <= 32768/(output_dim/16)
	int32	group_size	输出的size，h和w维度相同		支持，输出ROI的H和W大小，H=W	
ROIPooling	uint32	pooled_h	输出的height	从feature map中根据bounding box的坐标crop出对应部分后再根据输出size的配置进行max pooling	支持	w<=4096 2048/w/(c/16/16) >= 2 w*h <= 32768
	uint32	pooled_w	输出的width		支持	
	float	spatial_scale	进行ROIPooling特征输入的尺寸相比原始输入的比例		支持，从原图坐标生成feature map坐标，下到指令	
Upsample	uint32	scale	DEPRECATED；尺寸放大的比例，w和h方向上一致，默认为2	Pooling操作的逆操作	支持，尺寸放大的比例，w和h方向上一致，默认为2，但不能配置，只能2，要求与前面max pooling一致；	pooling mask: w<=4096 upsampling: w<=4096
	uint32	scale_h	DEPRECATED；h方向尺寸放大的比例		不支持	
	uint32	scale_w	DEPRECATED；w方向尺寸放大的比例		不支持	
	bool	pad_out_h	DEPRECATED；是否需要在输出H维度-1；		不支持	
	bool	pad_out_w	DEPRECATED；是否需要在输出W维度-1；		不支持	
	uint32	upsample_h	输出在h方向上的尺寸大小		支持，输出H大小，要求与实际算出来一致，否则告警；与上面的scale参数不会同时用，同时出现采用本条参数；	

layer	参数类型	参数名	参数含义	描述	规格	
	uint32	upsample_w	输出在w方向上的尺寸大小		支持，输出W大小，要求与实际算出来一致，否则告警；与上面的scale参数不会同时用，同时出现采用本条参数；	
Permute	uint32	order	转置的顺序，	支持faster rcnn中定义的permute旋转，	只支持order=0,order=2,order=3,order=1模式	w<=4096
MatMul	uint32	dim_1	第一个矩阵N	矩阵乘法的操作,第二个矩阵必须是转置的矩阵（两个矩阵的width相同）	支持	8 bits: dim_2<=10208 16 bits: dim_2<=5104
	uint32	dim_2	第一个矩阵的K		支持	
	uint32	dim_3	第二个矩阵的M		支持	
RReLU	float	negative_slope	负值的斜率	随机化的leaky ReLU，在训练的时候negative_slope为给定均匀分布区间内的随机值，inference的时候，negative_slope固定为随机值的期望	支持	w<=4096
DepthwiseConv	uint32	num_output	输出的channel大小	对输入做depthwise的卷积	支持配置，必须配置为与input channel值一样的大小；	w<=4096 kernel_h <= 2048/(w/(16*stride_w)*stride_w)； kernel_h * kernel_w < 640
	bool	bias_term	默认为1，表示是否加bias		支持配置0或1；	
	uint32	pad	默认为0，表示补边的大小		支持，配置范围0~255；	
	uint32	kernel_size	kernel的大小		支持1~255	
	uint32	stride	默认为1，stride的大小		支持1~255	
	uint32	dilation	默认为1，dilation的大小		不支持配置	

layer	参数类型	参数名	参数含义	描述	规格	
	uint32	pad_h	默认为0，表示height方向的padding		支持，配置范围0~255；	
	uint32	pad_w	默认为0，表示width方向的padding		支持，配置范围0~255；	
	uint32	kernel_h	kernel的height大小		支持，配置范围1~255；	
	uint32	kernel_w	kernel的width大小		支持，配置范围1~255；	
	uint32	stride_h	stride的height大小		支持1~255	
	uint32	stride_w	stride的width大小		支持1~255	
	uint32	group	默认为1，group的大小		不支持配置	
	int32	axis	默认为1，axis值之前的维度认为是独立输入，axis之后的维度认为是空间维度，即在做卷积时会求和		不支持配置	
PassThrough	uint32	num_output	第一个矩阵的行方向维度	Yolo v2网络中的pass through层	支持，其数值必须等于输入channel值*block_height*block_width	w<=4096 block_height <= 2048/(w/(16*block_width)*block_width * (c/16))
	uint32	block_height	第一个矩阵的列方向维度/第二个矩阵的行方向维度		支持,配置范围1~255，且必须能够被输入H方向大小整除	
	uint32	block_width	第二个矩阵的列方向维度		支持,配置范围1~255，且必须能够被输入W方向大小整除	
CReLU	int32	axis	默认为1，输入的bottom[0]做scale的起始维度	PVA网络中的CReLU层	只支持配置为1	同power+concat+scale+relu规格限制

layer	参数类型	参数名	参数含义	描述	规格	
	int32	num_axes	默认是1，当只有一个bottom的时候有效，表示做scale的维度		只支持配置为1	
	bool	bias_term	默认为false，表示是否加bias		只支持配置为true	

表 3-4 NNIE non-support 层替换为 Custom\Proposal layer 规格表

layer	参数类型	参数名	参数含义	描述	规格	
Custom	repeated BlobShape	shape	输出的blob维度	用户自定义层，输出feature map	支持[1,32]个输出配置	无限制
Proposal	uint32	feat_stride	默认值16	-	只支持解析，不产生实际效果	-
	uint32	base_size	默认值16	-		
	uint32	min_size	默认值16	-		
	float	ratio	-	-		
	float	scale	-	-		
	uint32	pre_nms_topn	默认值6000	-		
	uint32	post_nms_topn	默认值300	-		
	float	nms_thresh	默认值0.7	-		
	float	remove_thresh	默认值0	-		

3.1.6.3 其他特性

- 支持任意中间层上报：除了网络的输出层，用户若还关心网络的某个中间层输出时可指定该层上报输出，中间层输出不影响后续隐藏层继续运算；
- 支持任意层高精度（16bit）计算模式：某些网络层对精度要求高时可以指定这些层为高精度模式，但是注意高精度模式的计算能力会比低精度要弱；
- 支持任意切分：用户有自定义私有层或者NNIE不支持的层时，需要对网络进行切分，同时也支持用户对支持的层进行切分，但是建议用户尽量减少分段段数，减少硬件执行过程中的软件交互开销，以期获得更高的性能；
- 支持binary/ternary/sparse参数的高压缩比定制压缩，用户提供的参数只有三种值且正负对称时，mapper会自动进入Ternary模式；用户提供的参数只有两种值且包含0时，mapper会自动进入Binary模式；用户提供的参数稀疏比在80%以上时

自动进入sparse模式；这三种模式相对normal模式能提供更高的压缩比，以达到更高的带宽提升；

- 硬件预处理支持YUV转换成RGB：从上海海思媒体SOC的其他模块中，比如VI、VPSS、VGS（参考《HiMPP 媒体处理软件 Vx.0 开发参考》）等模块中获取的均是YUV数据，而用户训练网络模型使用的一般是RGB数据。NNIE硬件支持YUV输入转到RGB，因此用户在实际使用场景中可以把YUV转为RGB交给NNIE硬件实现；
- 支持高效模式完成in-place激活运算：类似Conv+ReLu计算，使用3.6章节中的in-place描述方案，硬件执行效率更高；
- 支持多输入、多输出：支持多输入、多输出（包含任意上报的输出）的网络配置；
- 支持网络结构的自动优化：例如
  - ROIPooling/PSROIPooling/Upsample/层前后会插入Permute层；
  - Normalize层后会插入Permute层；
  - Softmax层输入为张量，且axis为2时，前后会插入Permute层；
  - Softmax层输入为张量，且axis为1时，若输出Channel大于8192，则前后插入Permute层，否则前后插入Reshape层；
  - 同轴Concat级联时将被优化成一层以提升处理效率。
- 支持自动识别片上Cache以提升处理性能：数据在隐藏层内传递时，如果数据足够小，可以完整的存入片内RAM，mapper将实现片上数据传递以提升处理性能；

3.1.7 NNIE 硬件资源利用率

在现有硬件规格下，用户可以参考以下准则来提升硬件利用率：

- 总体原则：
  - 一般情况下，相同计算量场景，尽量增加单层的计算量，减少网络层数，以减少层之间切换造成的利用率损失，提高端到端的利用率。
  - 尽量使用卷积、反卷积、Pooling、FC层，提高硬件资源利用率，减少LRN、MVN、Normalize、Softmax层的使用。
  - 建议使用RELU、Sigmod、Tanh、PRELU、RRELU激活函数并使用in-place的的配置方式（与前一层共享blob），以便硬件高效完成计算；
- 特别地针对卷积、反卷积、Pooling、DepthwiseConv层运算：
  - 输出特征数据尺寸在16*16时，读参数的DDR带宽与卷积计算基本匹配；特征数据尺寸更小时，计算单元受限于DDR带宽会无法全部利用。
  - 硬件进行卷积计算时，数据按16组并行计算，输入的C维度是16倍数，输出的维度C是4的倍数，输出的W维度是16的倍数时，可以充分利用MAC的计算性能。

3.2 Prototxt 要求

NNIE mapper对prototxt的输入层格式、layer格式、激活层、Scale、Bias层、RNN、LSTM层及特殊的中间层上报、高精度配置、指定支持层有CPU执行等有特定的规范约束。

3.2.1 deploy.prototxt 输入层格式

deploy.prototxt输入层支持如下两种格式，n维度的dim值建议写1，mapper会根据参考图片路径中的图片张数自动生成n值:

格式一：

```
input: "data"
input_shape{
  dim:1
  dim:3
  dim:224
  dim:224
}
```

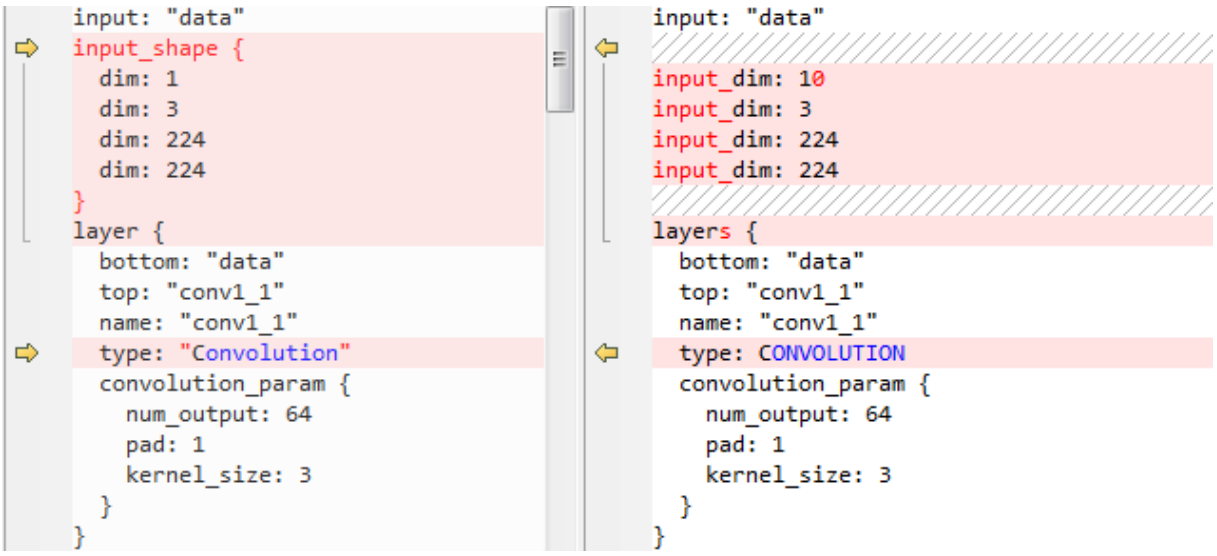
格式二：

```
layer {
name: "data"
type: "Input"
top: "data"
input_param {
    shape: {
      dim: 1
      dim: 3
      dim: 227
      dim: 227
    }
  }
}
```

3.2.2 prototxt layer 描述格式

prototxt格式上只支持图3-29左边的格式；层以layer开头，type以带双引号字符串表示。

图 3-29 NNIE 支持 prototxt 格式说明示意图（左边支持）



3.2.3 prototxt 激活层描述方式

1. batchnorm，scale，bias，relu，sigmoid，tanh、prelu、absval激活层bottom与top名称相同时，即in-place方式，该层激活在NNIE中执行时会与被优化处理，性能更高。

```
layer {
    bottom: "res5c_branch2a"
    top: "res5c_branch2b"
    name: "res5c_branch2b"
    type: "Convolution"
    convolution_param {
        num_output: 512
        kernel_size: 3
        pad: 1
        stride: 1
        bias_term: false
    }
}
layer {
    bottom: "res5c_branch2b"
    top: "res5c_branch2b"
    name: "bn5c_branch2b"
    type: "BatchNorm"
    batch_norm_param {
        use_global_stats: true
    }
}
layer {
    bottom: "res5c_branch2b"
    top: "res5c_branch2b"
    name: "scale5c_branch2b"
    type: "Scale"
    scale_param {
        bias_term: true
    }
}
```

2. mapper也支持非in-place配置激活函数，即bottom和top 名称不同；此时NNIE上运行该激活层与前一层独立，但支持的激活类型更丰富。格式如下所示：

```
layer {
    bottom: "res5c_branch2a"
    top: "res5c_branch2b"
    name: "res5c_branch2b"
    type: "Convolution"
    convolution_param {
        num_output: 512
        kernel_size: 3
        pad: 1
        stride: 1
        bias_term: false
    }
}
layer {
    bottom: "res5c_branch2b"
    top: " bn5c_branch2b"
    name: "bn5c_branch2b"
    type: "BatchNorm"
    batch_norm_param {
        use_global_stats: true
    }
```



```
}
layer {
    bottom: "bn5c_branch2b"
    top: " scale5c_branch2b"
    name: "scale5c_branch2b"
    type: "Scale"
    scale_param {
        bias_term: true
    }
}
```

3. batchnorm 层必须有 batch_norm_param 参数，并且 use_global_stats 必须为 true，格式如下所示：

```
layer {
    bottom: "res5c_branch2b"
    top: "res5c_branch2b"
    name: "bn5c_branch2b"
    type: "BatchNorm"
    batch_norm_param {
        use_global_stats: true
    }
}
```

3.2.4 prototxt 针对 Scale/Bias 层书写规范

对于Scale/Bias层，NNIE当前仅支持如下三种使用场景：

1. c*h*w的张量数据输入，每一个点对应一个scale值或bias值，即scale、bias参数也是c*h*w的张量；
2. c*h*w的张量数据输入，每一个channel对应一个scale值或bias值，或者同时对应一个scale和一个bias值，即scale、bias参数是一个c维的向量；
3. c维向量数据输入，每一个点对应一个scale值或bias值，或者同时对应一个scale和一个bias值，即scale、bias参数也是一个c维的向量。

Scale/Bias层的参数来源分两种场景：

1. 在线场景：Scale\Bias参数是网络某一层的输出，即在线计算得到；
2. 离线场景：Scale\Bias参数从caffemodel中解析，即离线训练得到；

针对Scale/Bias层，prototxt书写需符合如下规范要求：

- 在线场景要求bottom数目为2，且顺序固定，bottom[0]为当前层的Feature map输入，bottom[1]为当前层的参数输入；

```
layer {
    name: "scale1"
    type: "Scale"
    bottom: "mvn1"
    bottom: "data2"
    top: "scale1"
    scale_param {
     axis: 1
     num_axes: 3
     bias_term: false
  }
}
```

```
layer {
    name: "bias1"
    type: "Bias"
    bottom: "mvn1"
```



```
    bottom: "data2"
    top: "bias1"
    bias_param {
     axis: 1
     num_axes: 3
  }
}
```

- 离线场景bottom数目为1，当Scale/Bias的输入为向量时，num_axes支持配置为1或-1，两者意义相同；

```
layer {
    name: "scale1"
    type: "Scale"
    bottom: "fc1"
    top: "scale1"
    scale_param {
     axis: 1
     num_axes: 1
     bias_term: false
  }
}
```

```
layer {
    name: "bias1"
    type: "Bias"
    bottom: "fc1"
    top: "bias1"
    bias_param {
     axis: 1
     num_axes: 1
  }
}
```

- 不论在线还是离线场景，Scale层数据输入和参数输入都是c*h*w的张量时，只支持bias_term为false；

```
layer {
    name: "data2"
    type: "Input"
    top: "data2"
    input_param {
     shape {
            dim: 3
            dim: 27
            dim: 27
       }
    }
}
layer {
    name: "mvn1"
    type: "MVN"
    bottom: "data1"
    top: "mvn1"
    mvn_param {
     normalize_variance: false
     across_channels: true
    }
}
layer {
    name: "scale1"
    type: "Scale"
    bottom: "mvn1"
    bottom: "data2"
    top: "scale1"
```



```
  scale_param {
   axis: 1
   num_axes: 3
   bias_term: false
  }
}
```

- 不论在线还是离线场景，Scale/Bias层的axis参数只能配置1或-3，两者意义相同。

```
layer {
  name: "scale1"
  type: "Scale"
  bottom: "mvn1"
  bottom: "data2"
  top: "scale1"
  scale_param {
   axis: 1
   num_axes: 3
   bias_term: false
  }
}
```

```
layer {
  name: "bias1"
  type: "Bias"
  bottom: "mvn1"
  bottom: "data2"
  top: "bias1"
  bias_param {
   axis: 1
   num_axes: 3
  }
}
```

NNIE当前支持的3种使用场景可以细分为如下16种具体场景：

表 3-5 Scale 层 10 种场景配置

序号	数据输入	参数输入	axis	num_axes	bias_term
1	c*h*w张量	在线输入，c*h*w张量	1或-3	-	false
2	c维向量	在线输入，c维向量	1或-3	-	true
3	c维向量	在线输入，c维向量	1或-3	-	false
4	c*h*w张量	在线输入，c维向量	1或-3	-	true
5	c*h*w张量	在线输入，c维向量	1或-3	-	false
6	c*h*w张量	离线输入，c*h*w张量	1或-3	-1	false
7	c维向量	离线输入，c维向量	1或-3	1或-1	true
8	c维向量	离线输入，c维张量	1或-3	1或-1	false
9	c*h*w张量	离线输入，c维向量	1或-3	1	true

序号	数据输入	参数输入	axis	num_axes	bias_term
10	c*h*w张量	离线输入，c维向量	1或-3	1	false

表 3-6 Bias 层 6 种使用场景配置

序号	数据输入	参数输入	axis	num_axes
1	c*h*w张量	在线输入，c*h*w张量	1或-3	-
2	c维向量	在线输入，c维向量	1或-3	-
3	c*h*w张量	在线输入，c维向量	1或-3	-
4	c*h*w张量	离线输入，c*h*w张量	1或-3	-1
5	c维向量	离线输入，c维向量	1或-3	1或-1
6	c*h*w张量	离线输入，c维张量	1或-3	1

3.2.5 prototxt 针对 RNN 层书写规范

RNN的基本计算公式如下 (与Caffe一致)：

输入: x_t, h_{t-1}

输出：O_t,h_t

隐藏层：$h_t=\tanh(w_{xh}x_t+w_{hh}h_{t-1}+b_h)$

输出层：$o_t=\tanh(w_{ho}h_t+b_0)$

若有额外固定输入向量 x_{static}，计算公式变为：

输入: x_t, h_{t-1}

输出：O_t,h_t

隐藏层：$h_t=\tanh(w_{xh}x_t+w_{hh}h_{t-1}+w_{sh}x_{static}+b_h)$

输出层：$o_t=\tanh(w_{ho}h_t+b_0)$

基于以上计算公式，RNN的prototxt有以下4种场景：

3.2.5.1 通常场景（expose_hidden=false，1 输入，1 输出）

Input: $x_t(1\sim T)$

Output: $o_t(1\sim T)$

Prototxt示例如下，有1个bottom和1个top，分别对应$x_t(1\sim T)$和$o_t(1\sim T)$：

- 若recurrent_param的expose_hidden 没配，默认值为false；
- 不支持recurrent_param 的debug_info，配置不会生效。

```
layer {
  name: "rnn1"
  type: "RNN"
  bottom: "data0_xt"
  top: "rnn1_ht "
  recurrent_param {
    num_output: 1000
    weight_filler {
      type: "uniform"
      min: -0.08
      max: 0.08
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

3.2.5.2 隐藏参数显式输入输出（expose_hidden=true，2 输入，2 输出）

Input: $x_t(1\sim T),h_0$

Output: $o_t(1\sim T),h_T$

Prototxt示例如下，有2个bottom和2个top，其中bottom[0]对应$x_t(1\sim T)$，bottom[1]对应h_0，top[0]对应$o_t(1\sim T)$，top[1]对应h_T;

- h_0的向量维度须与recurrent_param的num_output一致；
- 只有第一个bottom与第一个top有时间的维度。
- expose_hidden=true配置下，不支持后接InnerProduct层。

```
layer {
  name: "rnn1"
  type: "RNN"
  bottom: "data0_xt"
  bottom: "data1_h0"
  top: "rnn1_ht "
  top: "rnn1_hT"
  recurrent_param {
    num_output: 1000
    expose_hidden=1
    weight_filler {
      type: "uniform"
      min: -0.08
      max: 0.08
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

3.2.5.3 具有额外的固定输入（expose_hidden=false，2 输入，1 输出）

Input: $x_t(1\sim T),x_{static}$

Output: $o_t(1\sim T)$

Prototxt示例如下，有2个bottom和1个top，其中bottom[0]对应$x_t(1\sim T)$，bottom[1]对应x_{static}，top对应$o_t(1\sim T)$；

- $x_t(1\sim T)$的向量维度，必须与x_{static}的向量维度一致；
- 如果recurrent_param 的expose_hidden 没配，默认值为false；
- 不支持recurrent_param 的debug_info，配置不会生效。

```
layer {
  name: "rnn1"
  type: "RNN"
  bottom: "data0_xt"
  bottom: "data0_xstatic"
  top: "rnn1_ht "
  recurrent_param {
    num_output: 1000
    weight_filler {
      type: "uniform"
      min: -0.08
      max: 0.08
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

3.2.5.4 既有固定输入，又有显式的隐藏参数（expose_hidden=true，3 输入，2 输出）

Input: $x_t(1\sim T)$,x_{static},h_0

Output: $o_t(1\sim T)$,h_T

Prototxt示例如下，有3个bottom和2个top，其中bottom[0]对应$x_t(1\sim T)$，bottom[1]对应x_{static}，bottom[2]对应h_0，top[0]对应$o_t(1\sim T)$，top[1]对应h_T；

- h_0的向量维度必须与recurrent_param的num_output一致；
- $x_t(1\sim T)$的向量维度，必须与x_{static}的向量维度一致；
- 只有第一个bottom与第一个top有时间的纬度。
- expose_hidden=true配置下，不支持后接InnerProduct层。

```
layer {
  name: "rnn1"
  type: "RNN"
  bottom: "data0_xt"
  bottom: "data1_xstatic"
  bottom: "data1_h0"
  top: "rnn1_ht"
  top: "rnn1_hT"
  recurrent_param {
  num_output: 1000
  expose_hidden=1
    weight_filler {
      type: "uniform"
      min: -0.08
```



```
      max: 0.08
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

3.2.6 prototxt 针对 LSTM 层书写规范

RNN的基本计算公式如下 (与Caffe一致)：

输入门：$i_t=sigmoid(w_{xi}x_t+w_{hi}h_{t-1}+b_i)$

遗忘门：$f_t=sigmoid(w_{xf}x_t+w_{hf}h_{t-1}+b_f)$

输入值：$g_t=tanh(w_{xg}x_t+w_{hg}h_{t-1}+b_g)$

当前cell：$c_t=i_t*g_t+f_t*c_{t-1}$

输出门：$o_t=sigmoid(w_{xo}x_t+w_{ho}h_{t-1}+b_0)$

若有额外固定输入向量x_{static}，计算公式变为：

输入门：$i_t=sigmoid(w_{xi}x_t+w_{hi}h_{t-1}+w_{si}x_{static}+b_i)$

遗忘门：$f_t=sigmoid(w_{xf}x_t+w_{hf}h_{t-1}+w_{sf}x_{static}+b_f)$

输入值：$g_t=tanh(w_{xg}x_t+w_{hg}h_{t-1}+w_{sg}x_{static}+b_g)$

当前cell：$c_t=i_t*g_t+f_t*c_{t-1}$

输出门：$o_t=sigmoid(w_{xo}x_t+w_{ho}h_{t-1}+w_{so}x_{static}+b_0)$

基于以上计算公式，LSTM的prototxt有以下4种场景：

3.2.6.1 通常场景（expose_hidden=false，1 输入，1 输出）

Input: $x_t(1\sim T)$

Output: $h_t(1\sim T)$

Prototxt示例如下，有1个bottom和1个top，分别对应$x_t(1\sim T)$和$h_t(1\sim T)$：

- 若recurrent_param的expose_hidden 没配，默认值为false；
- 不支持recurrent_param 的debug_info，配置不会生效。

```
layer {
  name: "lstm1"
  type: "LSTM"
  bottom: "data0_xt"
  top: "lstm1_ht"
  recurrent_param {
    num_output: 1000
    weight_filler {
      type: "uniform"
      min: -0.08
      max: 0.08
```



```
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

3.2.6.2 隐藏参数显式输入输出（expose_hidden=true，3 输入，3 输出）

Input: $x_t(1\sim T)$,h_0,c_0

Output: $h_t(1\sim T)$,h_T,c_T

Prototxt示例如下，有3个bottom和3个top，其中bottom[0]对应$x_t(1\sim T)$，bottom[1]对应h_0，bottom[2]对应c_0，top[0]对应$h_t(1\sim T)$，top[1]对应h_T，top[2]对应c_T；

- h_0,c_0的向量维度须与recurrent_param的num_output一致；
- 只有第一个bottom与第一个top有时间的维度；
- 不支持recurrent_param 的debug_info，配置不会生效。
- expose_hidden=true配置下，不支持后接InnerProduct层。

```
layer {
  name: "lstm1"
  type: "LSTM"
  bottom: "data0_xt"
  bottom: "data1_h0"
  bottom: "data2_c0"
  top: "lstm1_ht"
  top: "lstm1_hT"
  top: "lstm1_cT"
  recurrent_param {
  num_output: 1000
  expose_hidden = 1
    weight_filler {
      type: "uniform"
      min: -0.08
      max: 0.08
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

3.2.6.3 具有额外的固定输入（expose_hidden=false，2 输入，1 输出）

Input: $x_t(1\sim T)$, x_{static}

Output: $h_t(1\sim T)$

Prototxt示例如下，有2个bottom和1个top，其中bottom[0]对应$x_t(1\sim T)$，bottom[1]对应x_{static}，top对应$h_t(1\sim T)$；

- $x_t(1\sim T)$的向量维度，必须与x_{static}的向量维度一致；
- 只有第一个bottom与第一个top有时间的纬度；

- 如果recurrent_param 的expose_hidden 没配，默认值为false；
- 不支持recurrent_param 的debug_info，配置不会生效；

```
layer {
  name: "lstm1"
  type: "LSTM"
  bottom: "data0_xt"
  bottom: "data1_xstatic"
  top: "lstm1_ht"
  recurrent_param {
  num_output: 1000
    weight_filler {
      type: "uniform"
      min: -0.08
      max: 0.08
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

3.2.6.4 既有固定输入，又有显式的隐藏参数（expose_hidden=true，4 输入，3 输出）

Input: $x_t(1\sim T)$,x_{static},h_0,c_0

Output: $h_t(1\sim T)$,h_T, c_T

Prototxt示例如下，有4个bottom和3个top，其中bottom[0]对应$x_t(1\sim T)$，bottom[1]对应x_{static}，bottom[2]对应h_0，bottom[3]对应c_0，top[0]对应$h_t(1\sim T)$，top[1]对应h_T，top[2]对应c_T；

- h_0,c_0的向量维度必须与recurrent_param的num_output一致；
- $x_t(1\sim T)$的向量维度，必须与x_{static}的向量维度一致；
- 只有第一个bottom与第一个top有时间的纬度；
- 不支持recurrent_param 的debug_info，配置不会生效。
- expose_hidden=true配置下，不支持后接InnerProduct层。

```
layer {
  name: "lstm1"
  type: "LSTM"
  bottom: "data0_xt"
  bottom: "data1_xstatic"
  bottom: "data2_h0"
  bottom: "data3_c0"
  top: "lstm1_ht"
  top: "lstm1_hT"
  top: "lstm1_cT"
  recurrent_param {
  num_output: 1000
  expose_hidden = 1
    weight_filler {
      type: "uniform"
      min: -0.08
      max: 0.08
    }
```



```
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

3.2.7 prototxt 指定任意中间层上报

中间层是指不在网络段结尾处的层。

用户需要中间层结果输出时，需要对应层的“top”域中添加“_report”标识符进行标注。

如果某一中间层有多个top都需要输出，用户可以为每一个top添加上报标注。

```
layer {
    name: "conv5 "
    type: "Convolution"
    bottom: "conv4"
    top: "conv5_report"
    convolution_param {
        num_output: 256
        kernel_size: 3
        pad: 1
        stride: 1
    }
}
```

mapper解析_report规则如下：

- top后续无节点，自然上报，_report不增加上报点；
- top对应的后续节点有多个bottom，且其中一个bottom是cpu层，则该top上报；
- top对应的后续节点是cpu层（其中，cpu层指proposal、custom、_cpu层）；
- top有后续节点，_report增加上报点；
- custom有top加_report，报错；
- proposal有top加_report，不报错，也不增加上报点；
- _cpu有top加_report，不报错，也不增加上报点；
- data层加_report，不会报错，也不会增加上报点；
- inplace激活，_report应加在conv层上，原因是多个激活共享了conv的blob，因此这些层只输出一个blob，加在激活层上不会报错，也不会增加上报点，其他inplace层加_report的情况，按此规则同样处理；
- conv加激活，如果用户想在conv层上报，必须把两个节点拆开（激活写成non-inplace方式，即激活的top、bottom不同名）；

3.2.8 prototxt 指定任意层配置高精度

用户指定自定义计算精度（compile_mode = 2）时，在对应层的层名后加上高精度“_hp”（16比特）标记，可实现指定任意层为高精度输入，格式如下所示。

```
layer {
    name: "conv5_hp"
    type: "Convolution"
    bottom: "conv4"
    top: "conv5"
    convolution_param {
```



```
    num_output: 256
    kernel_size: 3
    pad: 1
    stride: 1
  }
}
```

只有在compile_mode = 2的情况下，才可以由用户通过 “_hp”标记指定某些层为高精度处理，其他情况下（compile_mode配置为0或1），在层名后加高精度“_hp”标记，编译器会报错。

3.2.9 prototxt 指定某些层由 CPU 执行

对于mapper支持层，可以通过在name字段增加_cpu标记来指定该层切换为cpu执行（包含CPU、DSP等非NNIE执行的，均使用_cpu标志），格式如下所示。

```
layer {
  bottom: "rpn_cls_score"
  top: "rpn_cls_score_reshape"
  name: "rpn_cls_score_reshape_cpu"
  type: "Reshape"
  reshape_param {
    shape {
      dim: 0
      dim: 2
      dim: -1
      dim: 0
    }
  }
}
```

注意如下的限制：

- _cpu不能添加在data层；
- data、custom、proposal层后面不能接_cpu层；
- _cpu不能添加在inplace的激活层；
- _cpu层后面inplace连接的激活层也是_cpu层；
- _cpu层后面不能接upsample层。

3.2.10 分段执行网络 prototxt 示例

针对FasterRCNN、RFCN、SSD等存在Non-support层的网络，NNIE采用软硬件协同的方法来完成这类网络的运算。下面给出经典网络的分段以及一个构造网络的示例。

3.2.10.1 Faster RCNN 网络

参考“3.1.5 Non-support层处理方式”描述，Faster RCNN分为“NNIE→非NNIE→NNIE→非NNIE”4段来执行：

1. 自动切分方式：用户需要将Proposal层按约定的格式书写，以辅助mapper自动完成网络切割，切分后的网络见图3-28；
2. 自定义切分方式一：如果用户因某些原因（比如从整体性能考虑）希望手动切割网络，将某些NNIE支持的层转由CPU处理（或者DSP等非NNIE硬件），可通过在layer name字段上增加_cpu标记来完成，格式如下；

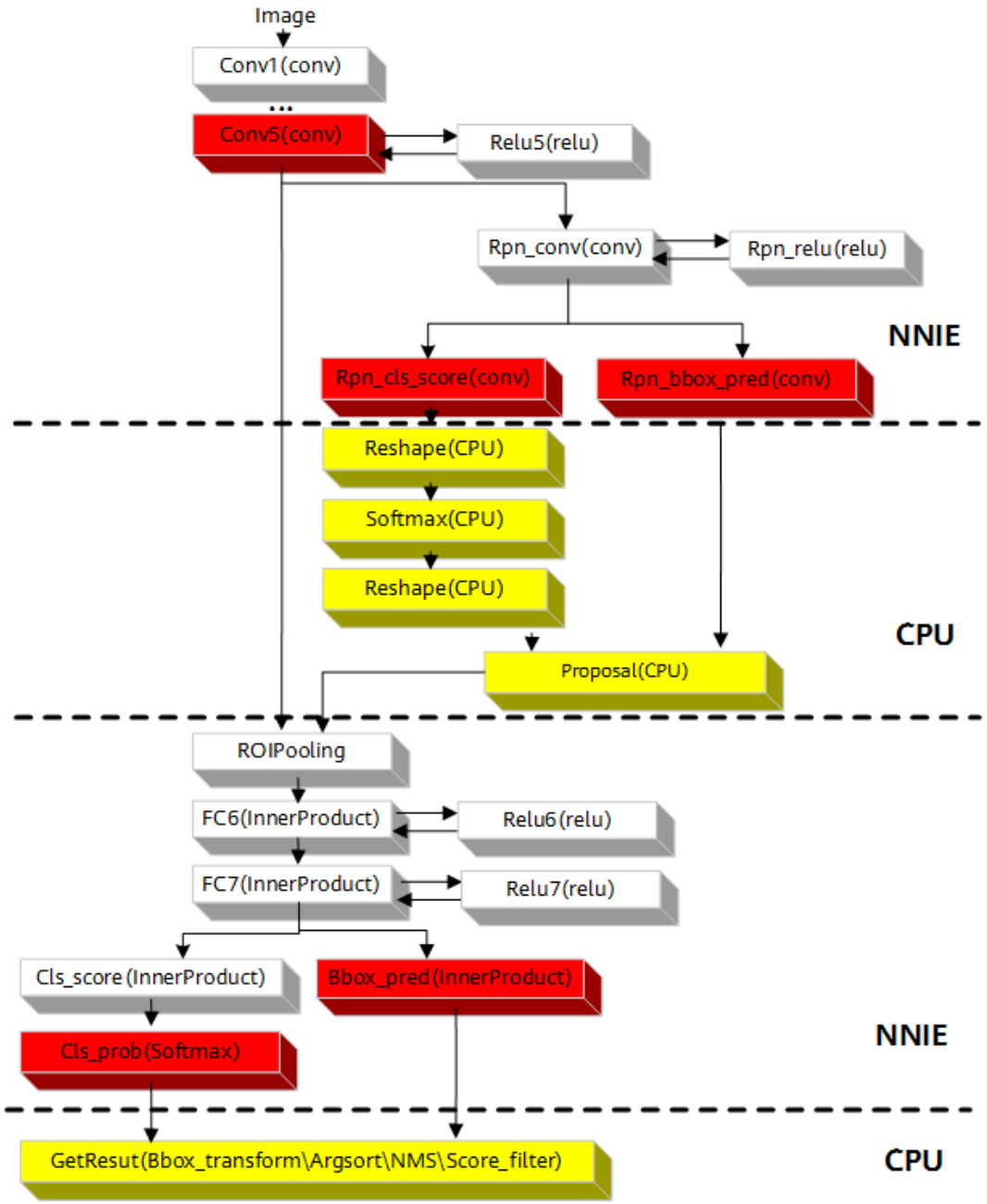
```
layer {
  bottom: "rpn_cls_score"
```



```
  top: "rpn_cls_score_reshape"
  name: "rpn_cls_score_reshape_cpu"
  type: "Reshape"
  reshape_param { shape { dim: 0 dim: 2 dim: -1 dim: 0 } }
}
```

将图3-28的Faster RCNN Rpn_cls_score层后面的Reshape->Softmax->Reshape跟Proposal层切分在一起且由CPU执行的网络，如图3-30：

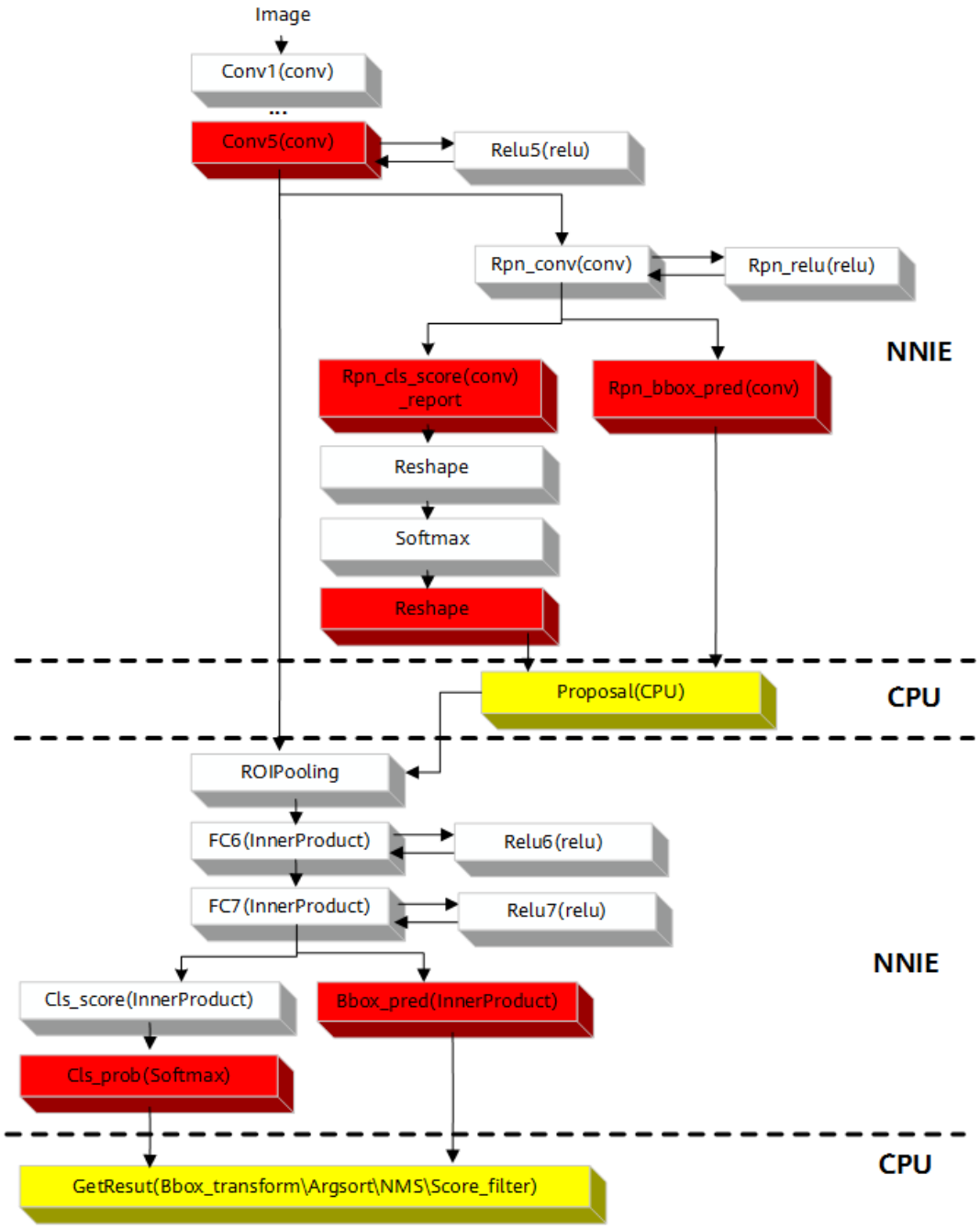
图 3-30 Faster RCNN 网络自定义分段示意图一



3. 自定义切分方式二：如果用户因某些原因需要Rpn_cls_sore的输出，而且跟Proposal相连接的Reshape->Softmax->Reshape希望NNIE执行，那么只需按照

3.2.7 prototxt指定任意中间层上报中描述，指定Rpn_cls_sore中间层输出，如图3-31：

图 3-31 Faster RCNN 网络自定义分段示意图二



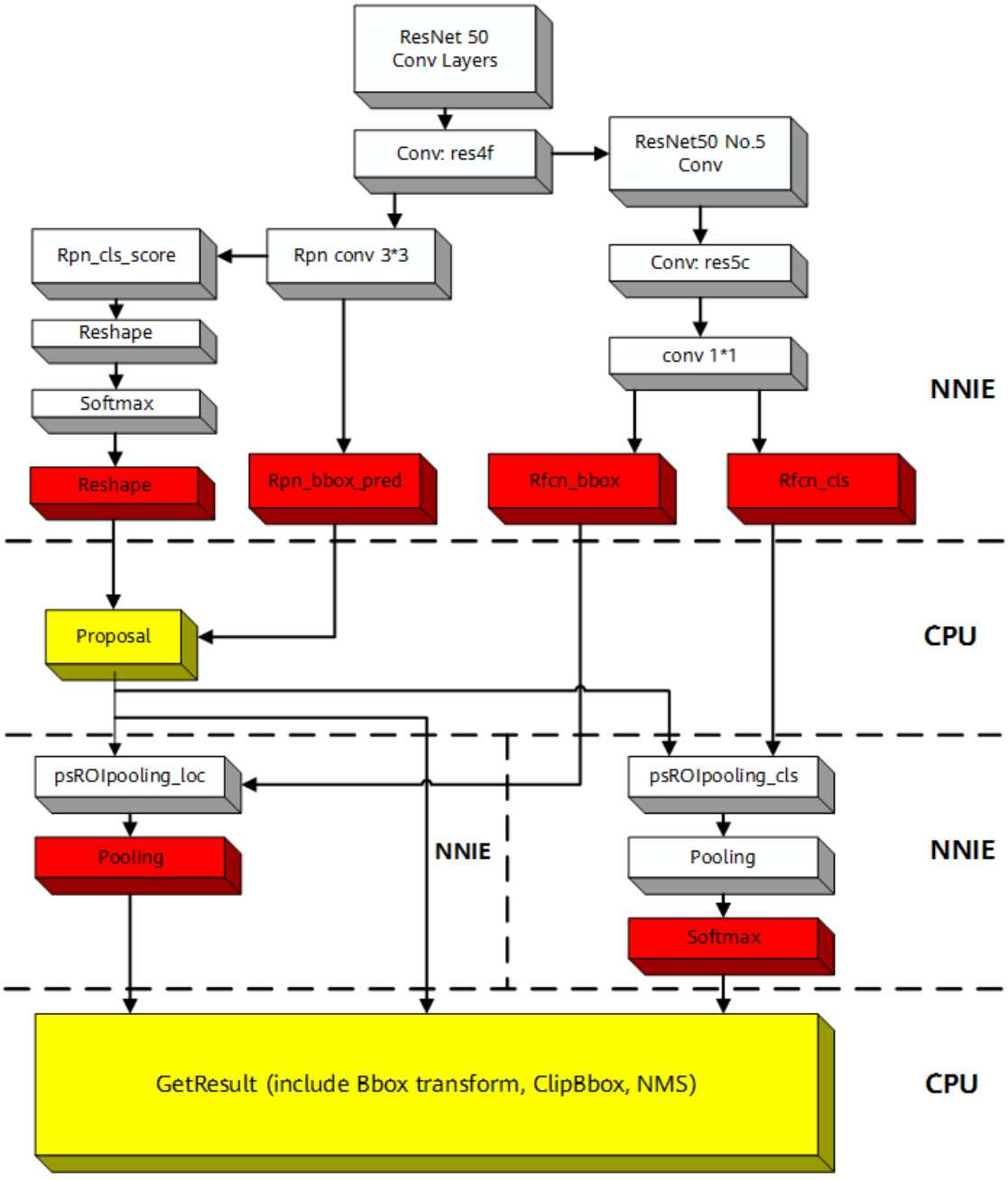
3.2.10.2 RFCN 网络

RFCN跟Faster RCNN类似，RPN部分的Proposal层(Bbox_transform、Argsort、NMS等操作)输出的是Bounding Box信息，由此网络被分为5段NNIE：

1. 第1段（NNIE执行），卷积部分，包含RPN前面的部分；

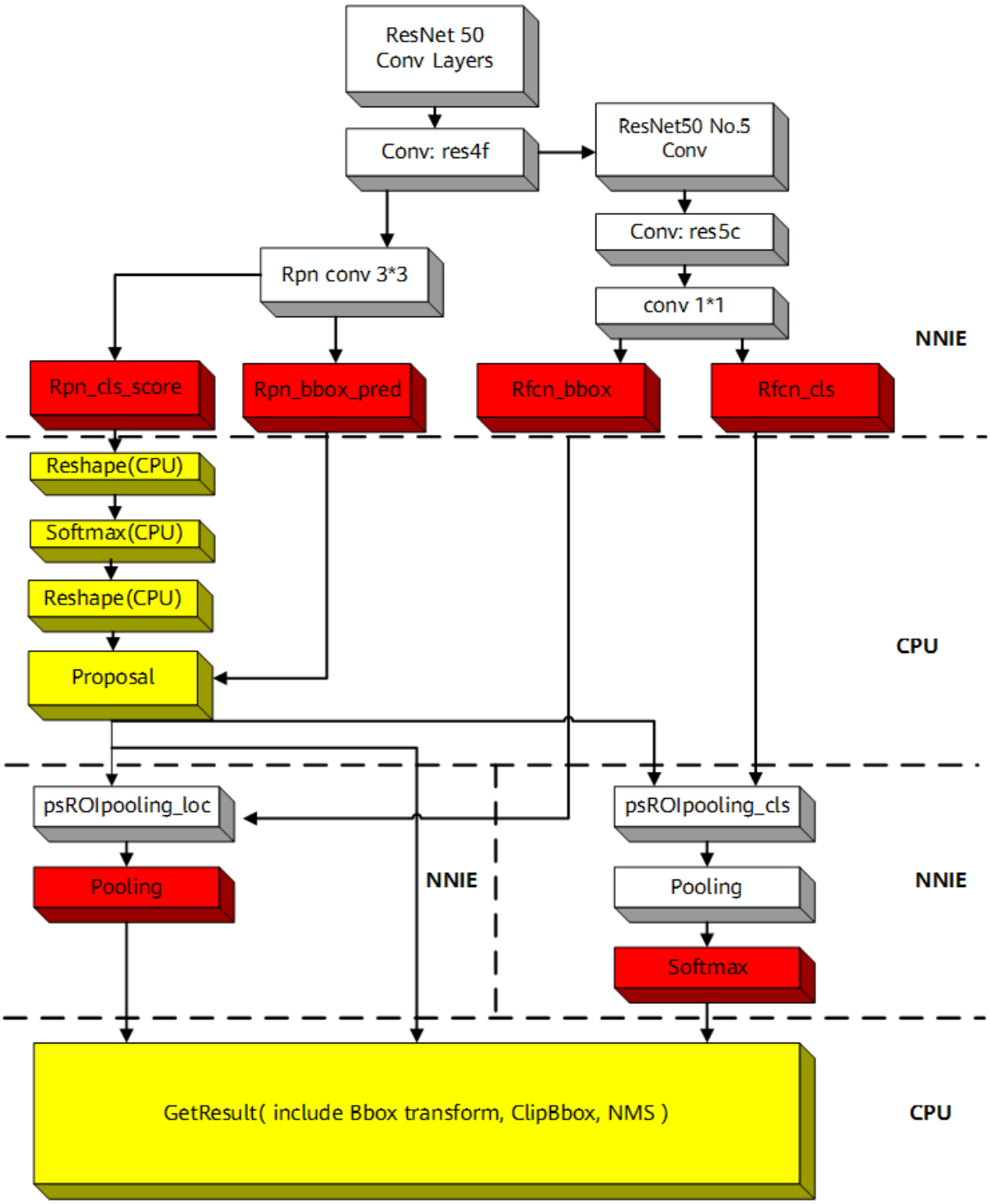
2. 第2段（非NNIE执行），Proposal部分，生成矩形框信息；
3. 第3段（NNIE执行），左边的PsRoiPooling部分以及后续的Pooling等，得到矩形框调整值；
4. 第4段（NNIE执行），右边的PsRoiPooling部分以及后续的Pooling等，得到矩形框置信度；
5. 第5段（非NNIE执行），结果获取部分，由3、4段得到的矩形框调整值和置信度经过bbox_transfom、argsort、NMS、ClipBox等步骤获取最终的矩形框和置信度信息。

图 3-32 RFCN 网络 mapper 自动分段示意图



类似Faster RCNN，考虑将Reshape->Softmax->Reshape->Proposal一起又CPU执行，则变为如图3-33：

图 3-33 RFCN 自定义分段示意图

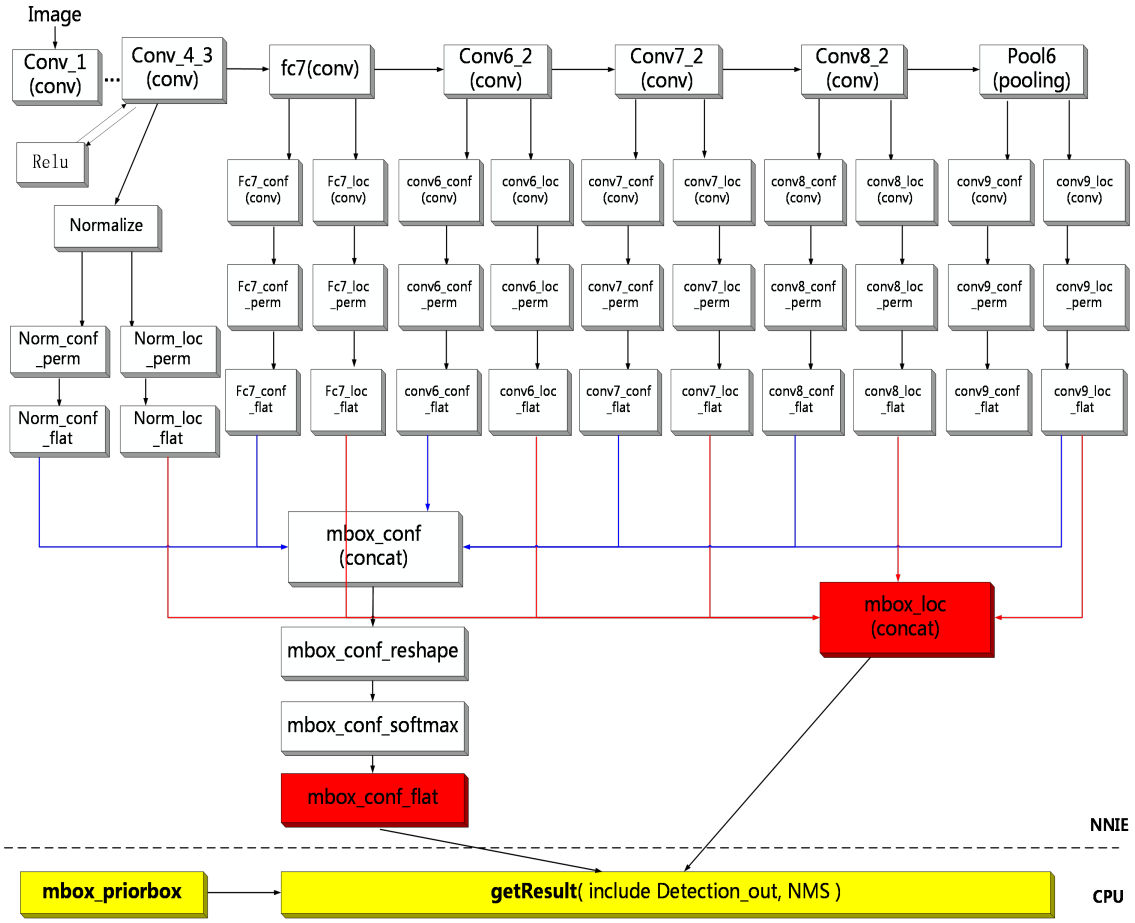


3.2.10.3 SSD 网络

SSD分为2段，前面的部分由NNIE执行，后面的一段由CPU执行。在SSD的prototxt中，除跟prior_box相关的层不支持，其他的层均支持；建议用户采取删除prototxt中跟prior_box相关的层的方法来修改prototxt。

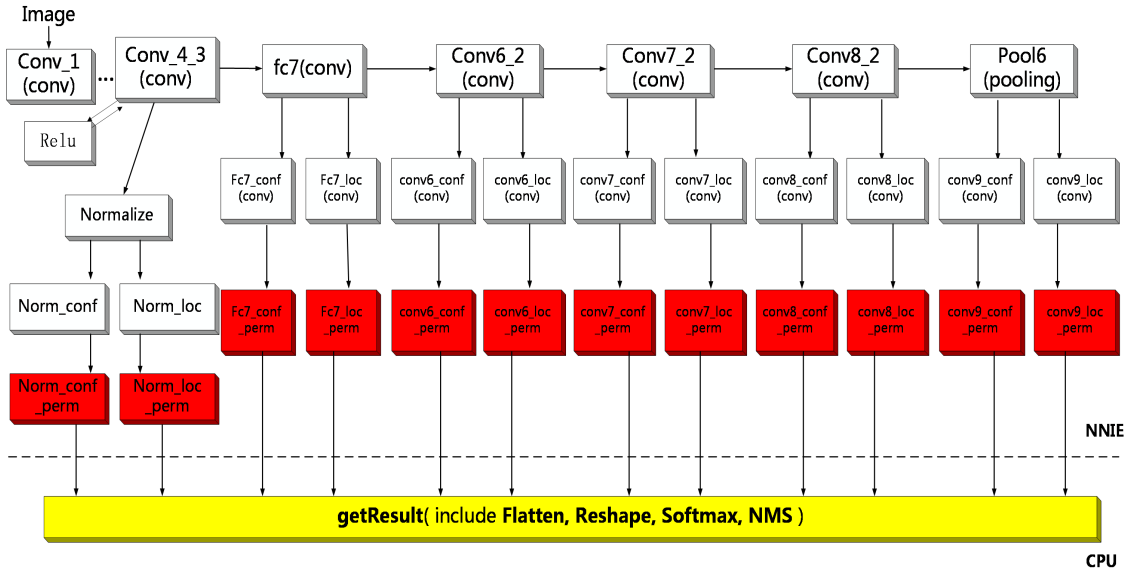
1. 自定义切分方式一：NNIE执行尽可能多的层，删除跟prior_box相关的层即可；

图 3-34 SSD 自定义分割方式一



2. 自定义切分方式二：NNIE执行到permute层，prototxt删除其后的层；

图 3-35 SSD 自定义分割方式二



3.2.11 FasterRCNN、RFCN、SSD、YOLOV1、YOLOV2、YOLOV3 检测网硬化加速 prototxt 示例

针对FasterRCNN、RFCN、SSD、YOLOV1、YOLOV2、YOLOV3等检测网络，nnie1.3（nnie版本和芯片的对应关系请参考3.5.1 描述）支持对Proposal及DetectionOutput等层进行硬化，以替代CPU算法，达到加速目的。本节给出了prototxt示例。

注意：Proposal、DetectionOutput、DecBox、Sort、Nms、Filter等硬化层不能作为首层；输出的得分是S12Q20格式。

3.2.11.1 硬化层参数配置

须知

FilterBox层在Sort层之前时不能配置top_k参数，FilterBox层在Sort层之后时可以配置top_k参数。

表 3-7 硬化层参数配置限制

参数	参数意义	DecBBox	FilterBox	Sort	Nms	Proposal	DetectionOutput
top_k	输出框数配置	NA	[1 ~ 524288]	[1 ~ 10000]	[1 ~ 10000]	NA	NA
num_classes	分类数目配置	NA	NA	NA	NA	[1~256]	[1~256]
multi_class_sorting	多类sort配置	true/false	NA	NA	NA	NA	NA

参数	参数意义	DecBBox	FilterBox	Sort	Nms	Proposal	DetectionOutput
calc_mode	网络类型配置，必须和网络类型匹配，参考表3-8和表3-9	[0~4]	NA	NA	NA	NA	NA

表 3-8 Faster RCNN RFCN SSD 硬化层参数配置

层/参数	参数	参数意义	FasterRCNN/RFCN	SSD
Proposal/DetectionOutput	num_anchors	锚点数目，必须配置	√	√
	num_classes	分类数目，必须配置	1	√
	num_coords	坐标数目，必须配置为4	4	4
DecBBox	share_location	是否是共享位置预测	false	true
	share_variance	是否共享方差，SSD网络必须配置为false	NA	false
	clip_bbox	是否使能框裁剪	false/ture	false/ture
	code_type	必须配置为CENTER_SIZE	CENTER_SIZE	CENTER_SIZE
	bias	YOLOV2 YOLOV3网络必须配置,值域[0,1024.0]	NA	NA
	variance	方差配置，SSD网络必须配置，且个数必须为4，值域[0,1.0]	NA	示例：variance: 0.1 variance: 0.1 variance: 0.2 variance: 0.2
	calc_mode	网络类型, 必须配置为表格中网络对应数值	0	4
FilterBox	top_k	输出框数	[1~524288]	[1~524288]

层/参数	参数	参数意义	FasterRCNN/RFCN	SSD
Sort	multi_class_sorting	是否使用多类sort	false	true/false 具体配置请参考prototxt示例
	top_k	输出框数	[1~10000]	[1~10000]
nms	top_k	输出框数	[1~10000]	[1~10000]

表 3-9 YOLOV1 YOLOV2 YOLOV3 硬化层参数配置

层/参数	参数	参数意义	YOLO_V1	YOLO_V2	YOLO_V3
DetectionOutput	num_anchors	锚点数目，必须配置	√	√	√
	num_classes	分类数目，必须配置	√	√	√
	num_coords	坐标数目，必须配置为4	4	4	4
	num_grids_width	YOLO网络栅格Width，YOLO网络必须配置	√	√	√
	num_grids_height	YOLO网络栅格Height，YOLO网络必须配置	√	√	√
DecBBox	share_location	是否是共享位置预测，必须配置为表格中网络对应数值	true	true	true
	share_variance	是否共享方差，SSD网络必须配置为false	NA	NA	NA
	clip_bbox	是否使能框裁剪	false/ture	false/ture	false/ture
	code_type	必须配置为CENTER_SIZE	CENTER_SIZE	CENTER_SIZE	CENTER_SIZE

层/参数	参数	参数意义	YOLO_V1	YOLO_V2	YOLO_V3
	bias	YOLOV2 YOLOV3网络必须配置,值域[0~1024.0]	NA	示例 bias: 1.08 bias: 1.19 bias: 3.42 bias: 4.41 bias: 6.63 bias: 11.3 bias: 9.42 bias: 5.11 bias: 16.6 bias: 10.5	示例: bias: 116 bias: 90 bias: 156 bias: 198 bias: 373 bias: 326
	variance	方差配置，SSD网络必须配置，且个数必须为4，值域[0,1.0]	NA	NA	NA
	calc_mode	网络类型, 必须配置为表格中网络对应数值	1	2	3
FilterBox	top_k	输出框数	[1~524288]	[1~524288]	[1~524288]
Sort	multi_class_sorting	是否使用多类sort，必须配置为表格中网络对应数值	false	false	false
	top_k	输出框数	[1~10000]	[1~10000]	[1~10000]
nms	top_k	输出框数	[1~10000]	[1~10000]	[1~10000]

3.2.11.2 Faster RCNN 网络

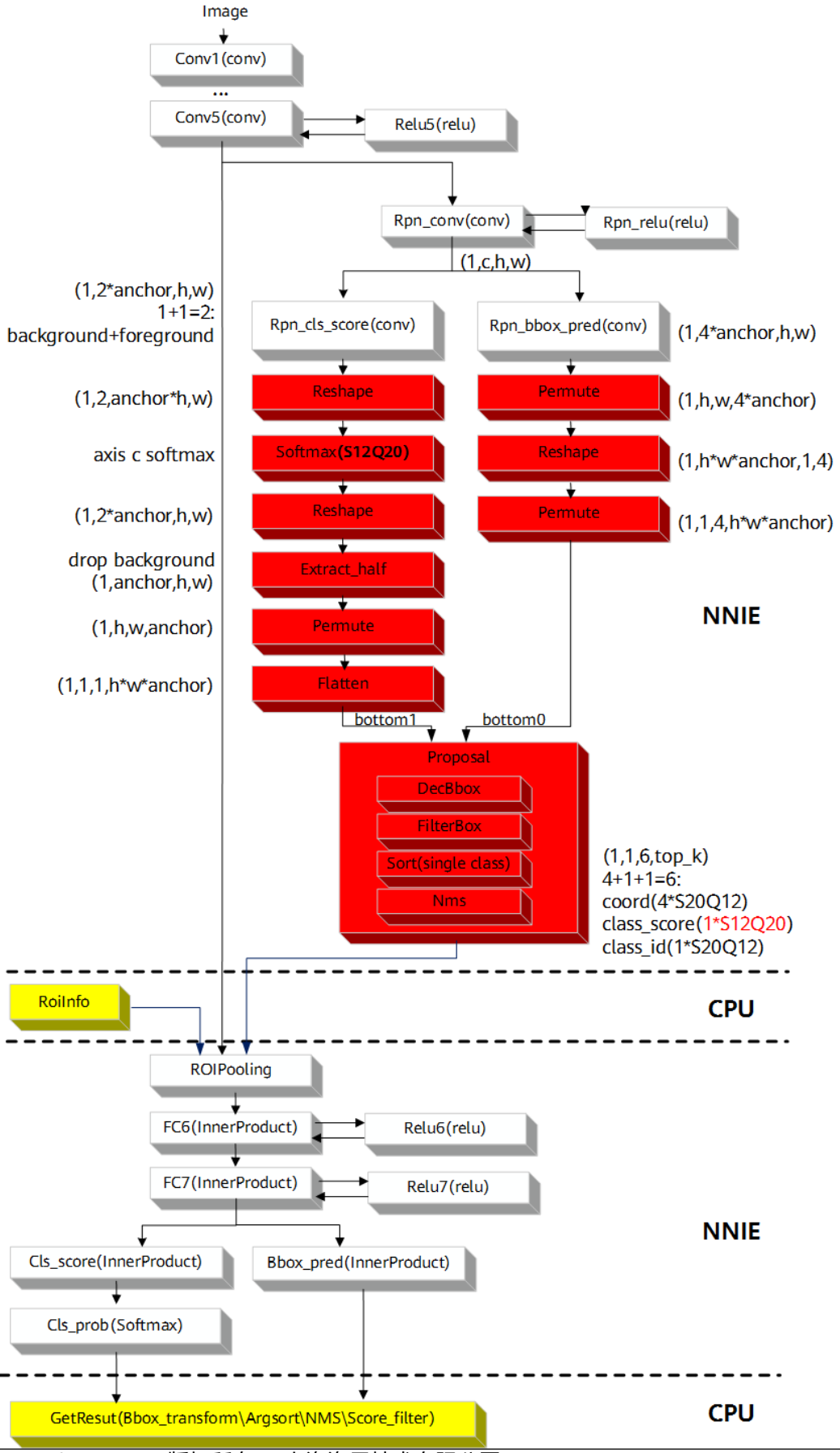
Faster RCNN Proposal层硬化结构如图3-36所示，SVP发布包中提供了基于alexnet的sample：software\data\detection\fasterRcnn\alexnet\fasterrcnn_alexnet_inst_V2.cfg，请参考此配置文件进行适配。

输出给Proposal硬化层的得分必须是20bit定点化分数，即S12Q20格式，因此输出前景、背景得分的Softmax层中name字段必须带**_s12q20后缀，**并且此Softmax层和Proposal硬化层之间不能有任何计算层，参考如下所示：

```
layer {
  name: "rpn_cls_prob_s12q20"
  type: "Softmax"
  bottom: "rpn_cls_score_reshape"
  top: "rpn_cls_prob"
}
```


Proposal硬化层描述请参考fasterrcnn_alexnet_inst_V2.cfg中指定的prototxt。

图 3-36 Faster RCNN Proposal 层硬化结构图



3.2.11.3 RFCN 网络

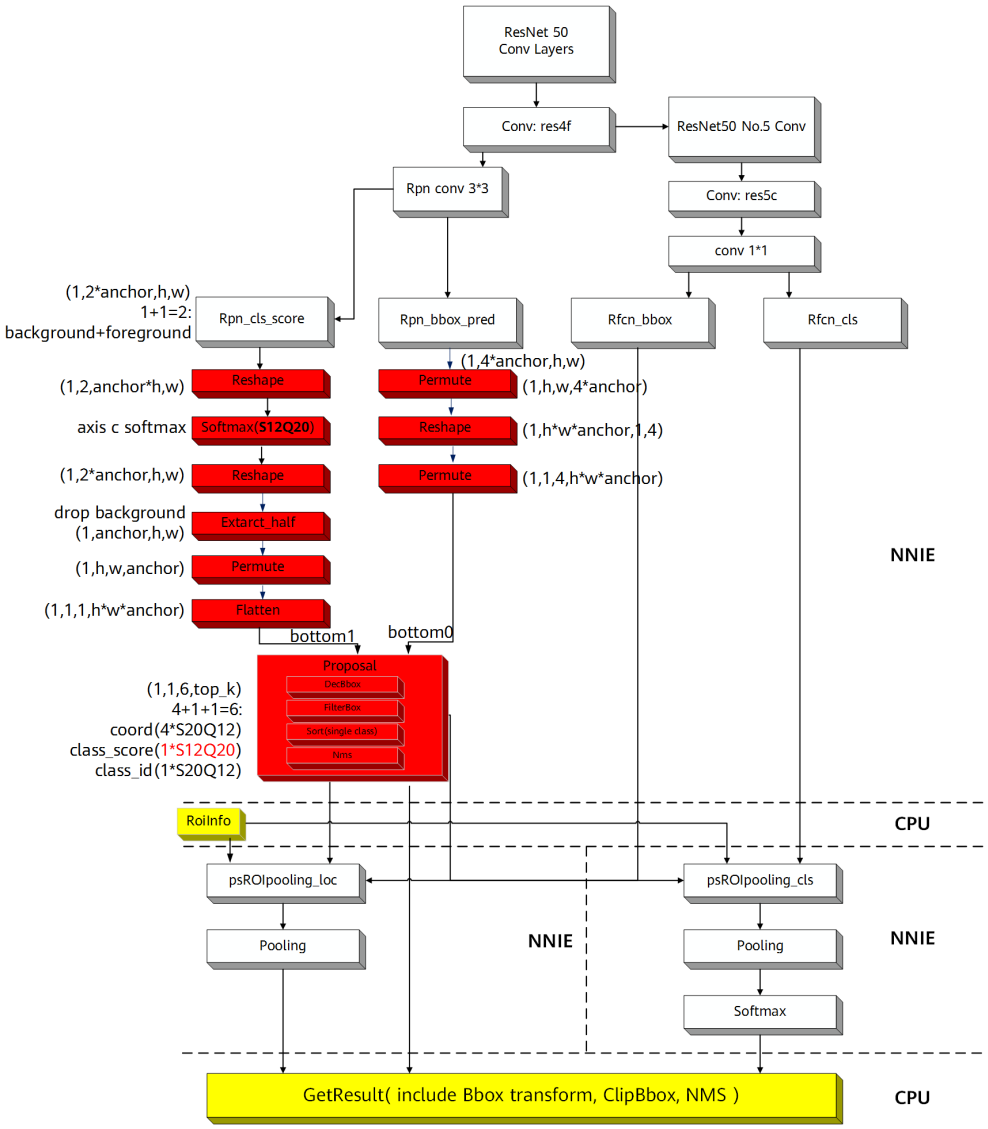
RFCN Proposal层硬化结构如图3-37所示，SVP发布包中提供了基于resnet50的sample：software\data\detection\rfcn\resnet50\rfcn_resnet50_inst_V2.cfg，请参考此配置文件进行适配。

输出给Proposal硬化层的得分必须是20bit定点化分数，即S12Q20格式，因此输出前景、背景得分的Softmax层中name字段必须带**_s12q20后缀**，并且此Softmax层和Proposal硬化层之间不能有任何计算层，如下所示：

```
layer {
  name: "rpn_cls_prob_s12q20"
  type: "Softmax"
  bottom: "rpn_cls_score_reshape"
  top: "rpn_cls_prob"
}
```

Proposal硬化层描述请参考rfcn_resnet50_inst_V2.cfg中指定的prototxt。

图 3-37 RFCN Proposal 层硬化结构图



3.2.11.4 SSD 网络

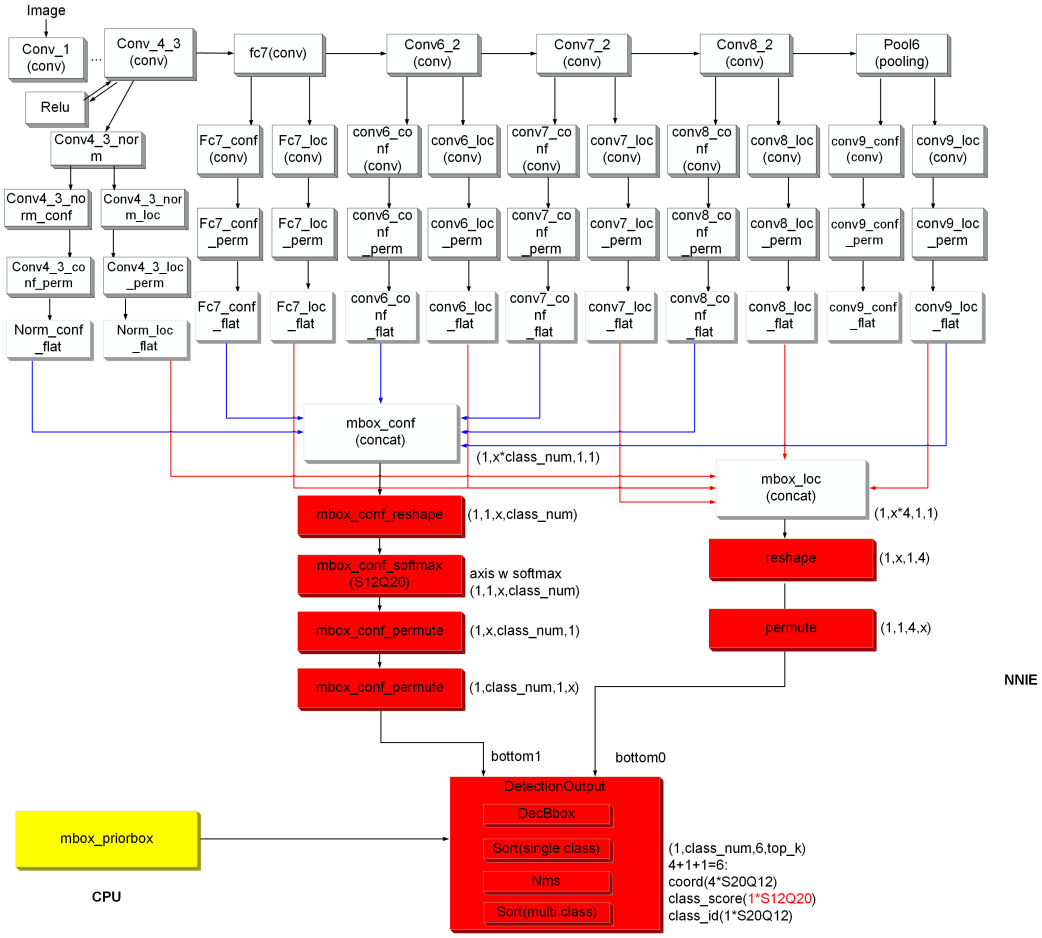
SSD DetectionOutput层硬化结构如图3-38所示，SVP发布包中提供了sample：software\data\detection\ssd\ssd_inst_V2.cfg，请参考此配置文件进行适配。

输出给DetectionOutput硬化层的得分必须是20bit定点化分数，即S12Q20格式，因此输出分类得分的Softmax层中name字段必须带**_s12q20后缀**，并且此Softmax层和DetectionOutput硬化层之间不能有任何计算层，如下所示：

```
layer {
  name: "mbox_conf_softmax_s12q20"
  type: "Softmax"
  bottom: "mbox_conf_reshape"
  top: "mbox_conf_softmax"
  softmax_param {
    axis: 3
  }
}
```

DetectionOutput硬化层描述请参考ssd_inst_V2.cfg 中指定的prototxt。

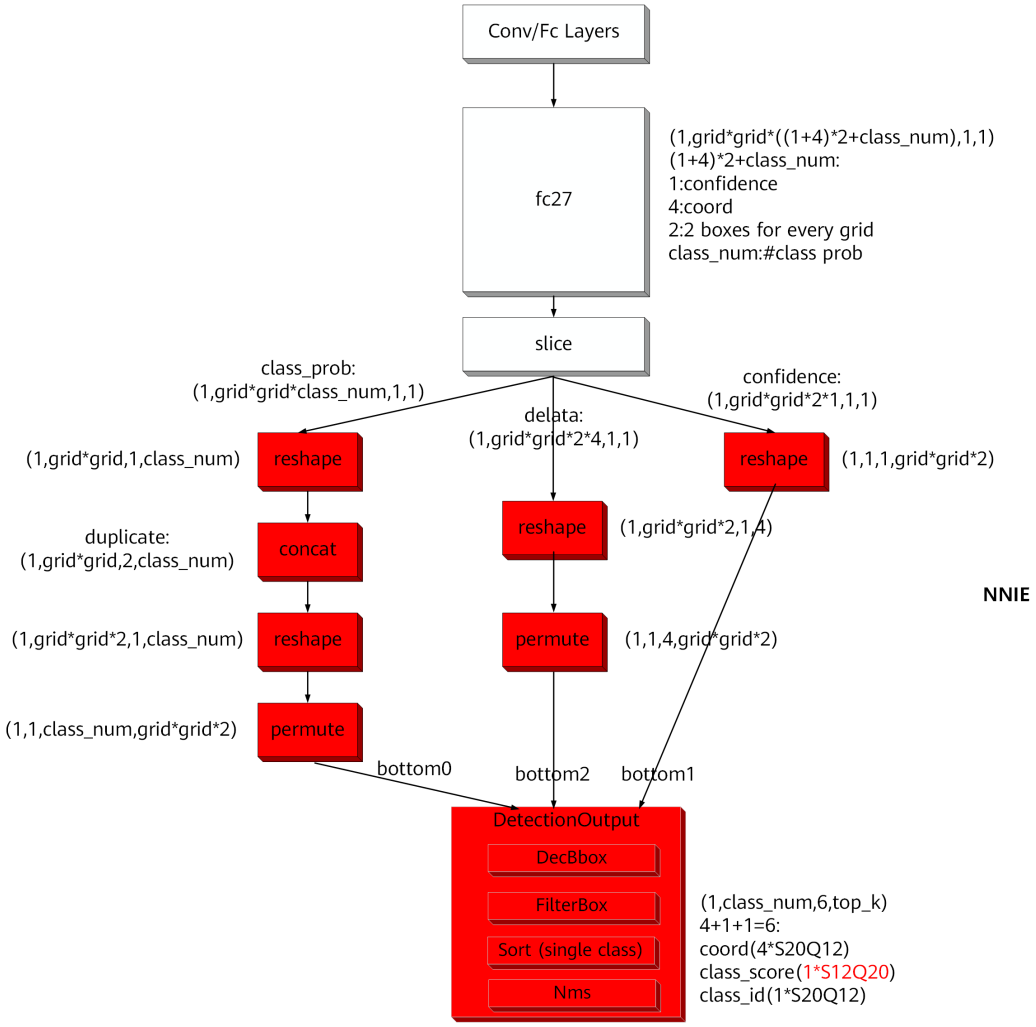
图 3-38 SSD DetectionOutput 层硬化结构图



3.2.11.5 YOLOV1 网络

YOLOV1 DetectionOutput层硬化结构如图3-39所示，SVP发布包中提供了sample：software\data\detection\yolov1\yolov1_inst_V2.cfg，请参考此配置文件进行适配。

图 3-39 YOLOV1 DetectionOutput 层硬化结构图



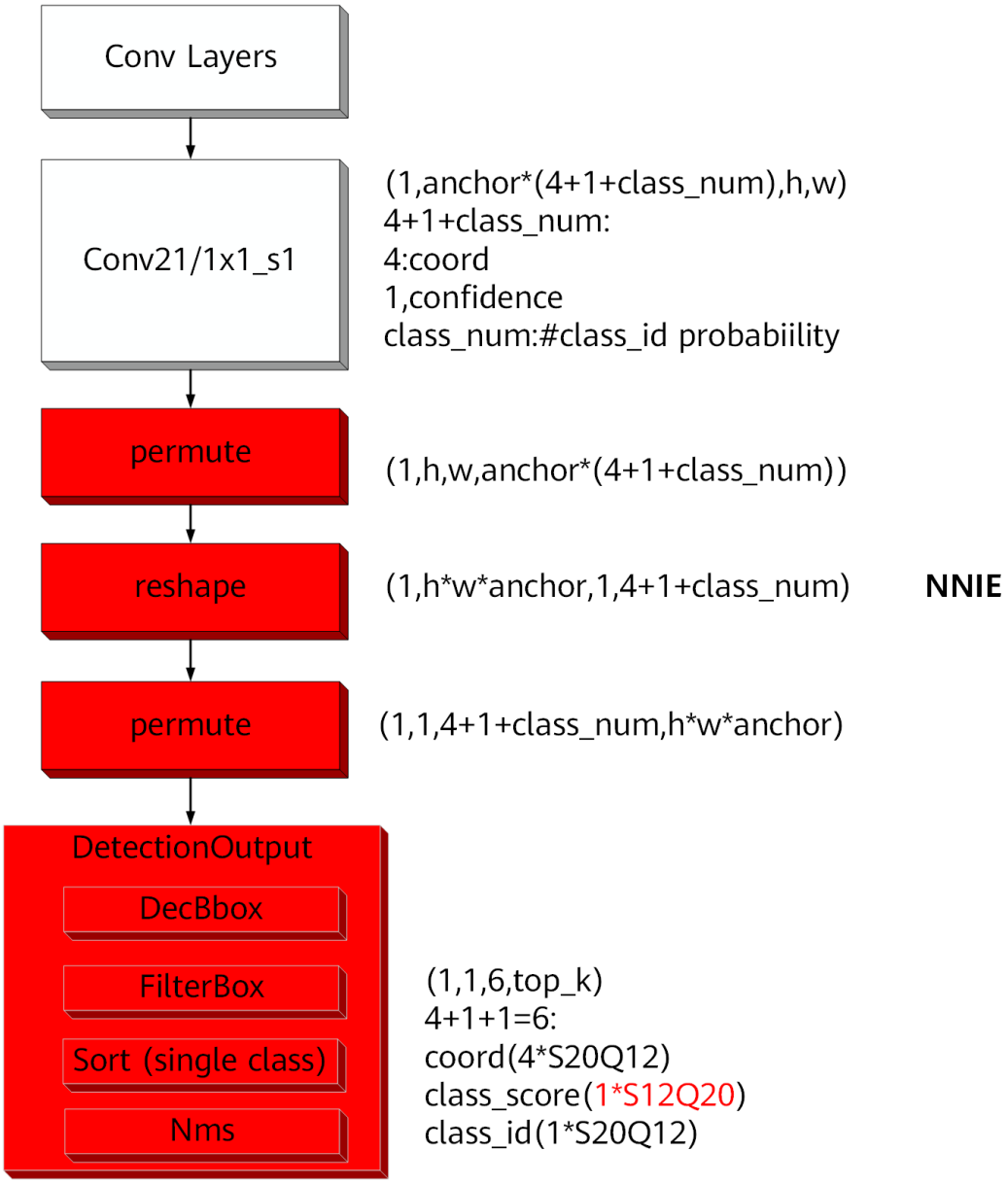
3.2.11.6 YOLOV2 网络

YOLOV2 DetectionOutput层硬化结构如图3-40所示，SVP发布包中提供了sample：software\data\detection\yolov2\yolov2_inst_V2.cfg，请参考此配置文件进行适配。

须知

DetectionOutput输出的是1*1*6*top_k的格式，类别中存放得分最高的分类序号，同YOLOV2/3中的1*class_num*6*top_k不同。

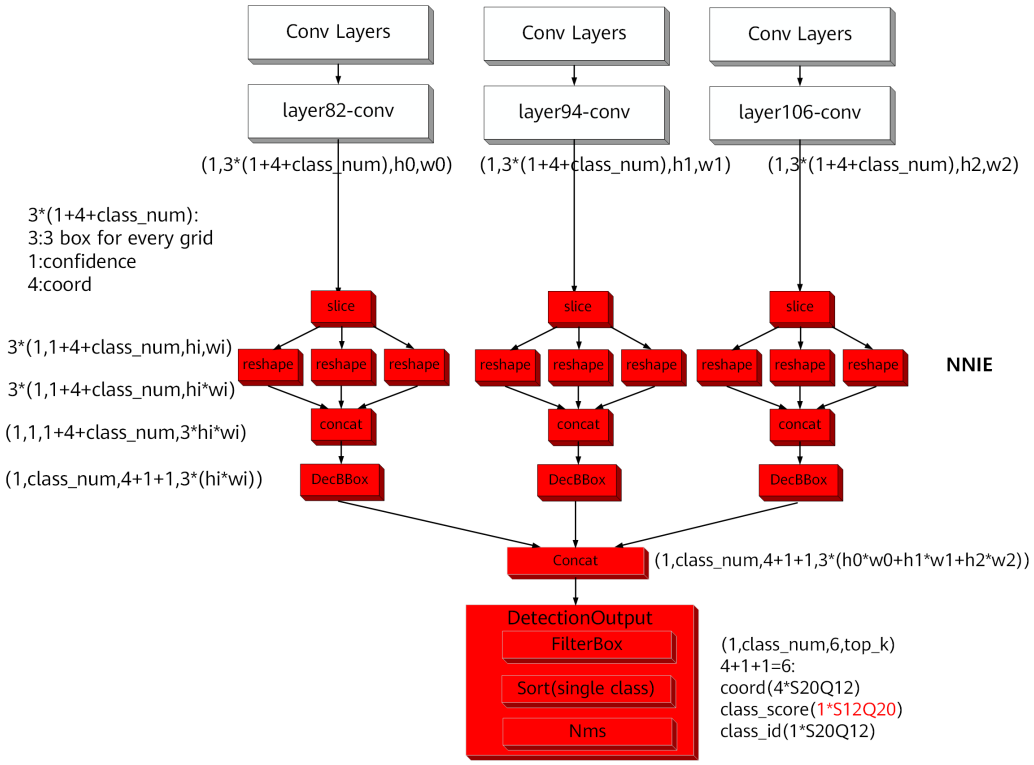
图 3-40 YOLOV2 DetectionOutput 层硬化结构图



3.2.11.7 YOLOV3 网络

YOLOV3 Decbbox/DetectionOutput层硬化结构如图3-41所示，SVP发布包中提供了sample：software\data\detection\yolov3\yolov3_inst_V2.cfg，请参考此配置文件进行适配。

图 3-41 YOLOV3 Decbbox/DetectionOutput 层硬化结构图

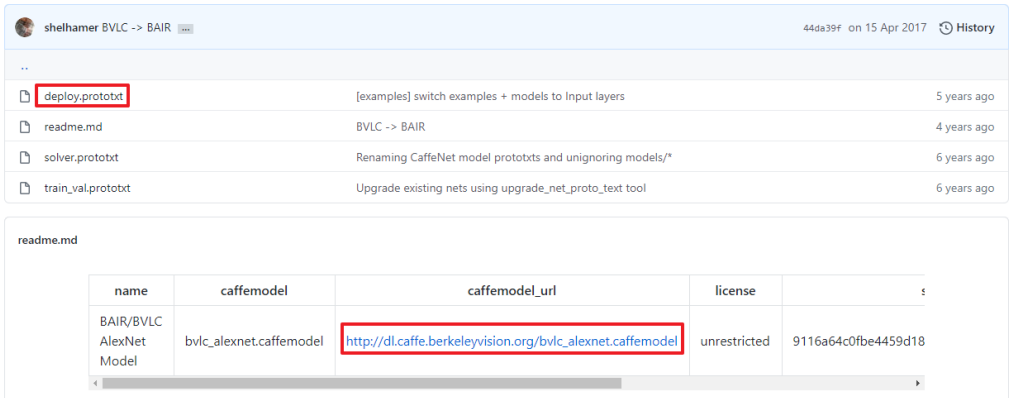


3.3 公开模型下载

3.3.1 Alexnet

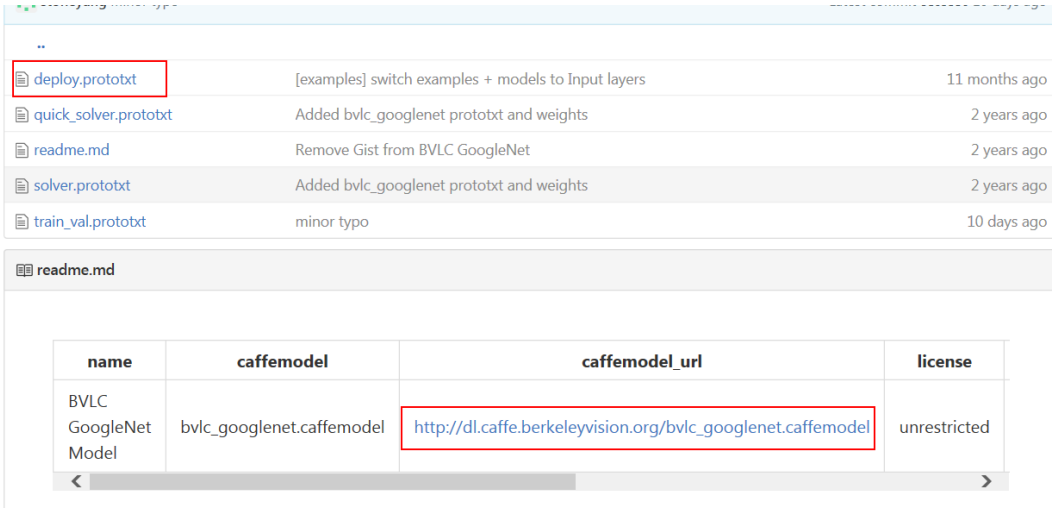
Github：**https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet**

Prototxt及Caffemodel：



3.3.2 Googlenet

Github：**https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet**



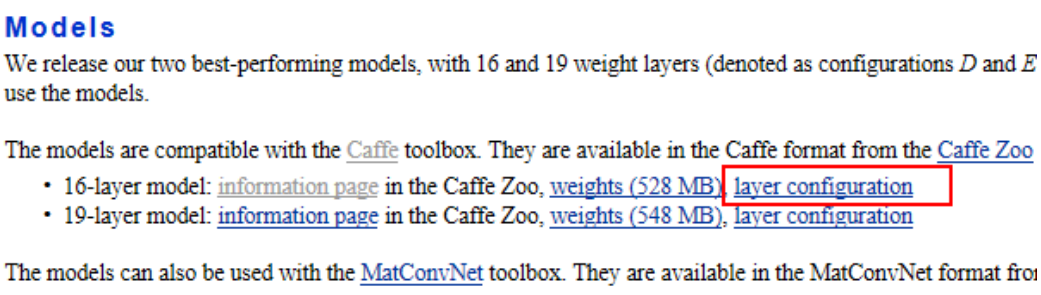
deploy.prototxt	[examples] switch examples + models to Input layers	11 months ago
quick_solver.prototxt	Added bvlc_googlenet prototxt and weights	2 years ago
readme.md	Remove Gist from BVLC GoogleNet	2 years ago
solver.prototxt	Added bvlc_googlenet prototxt and weights	2 years ago
train_val.prototxt	minor typo	10 days ago

readme.md

name	caffemodel	caffemodel_url	license
BVLC GoogleNet Model	bvlc_googlenet.caffemodel	http://dl.caffe.berkeleyvision.org/bvlc_googlenet.caffemodel	unrestricted

3.3.3 VGG16

Prototxt（注：prototxt格式需按**3.2 Prototxt要求**修改）：**http://www.robots.ox.ac.uk/~vgg/research/very_deep/**



Models

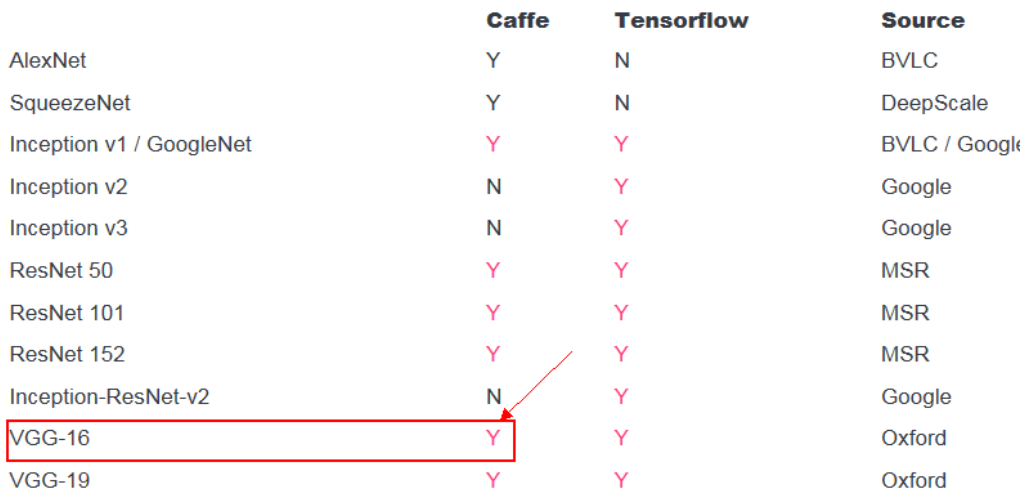
We release our two best-performing models, with 16 and 19 weight layers (denoted as configurations *D* and *E* use the models.

The models are compatible with the Caffe toolbox. They are available in the Caffe format from the Caffe Zoo

- 16-layer model: information page in the Caffe Zoo, weights (528 MB), layer configuration
- 19-layer model: information page in the Caffe Zoo, weights (548 MB), layer configuration

The models can also be used with the MatConvNet toolbox. They are available in the MatConvNet format fro

Caffemodel：**https://deepdetect.com/applications/list_models/**

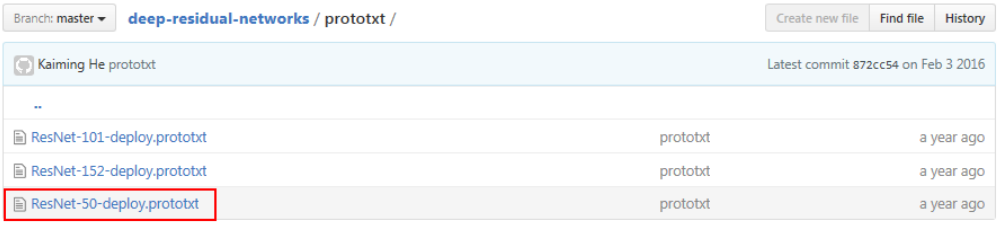


	Caffe	Tensorflow	Source
AlexNet	Y	N	BVLC
SqueezeNet	Y	N	DeepScale
Inception v1 / GoogleNet	Y	Y	BVLC / Google
Inception v2	N	Y	Google
Inception v3	N	Y	Google
ResNet 50	Y	Y	MSR
ResNet 101	Y	Y	MSR
ResNet 152	Y	Y	MSR
Inception-ResNet-v2	N	Y	Google
VGG-16	Y	Y	Oxford
VGG-19	Y	Y	Oxford

3.3.4 Resnet-50、resnet-101、resnet-152

Github：**https://github.com/KaimingHe/deep-residual-networks/tree/master/**

Prototxt：**https://github.com/KaimingHe/deep-residual-networks/tree/master/prototxt**



Caffemodel：**https://deepdetect.com/applications/list_models/**

Models are provided for image and text classification

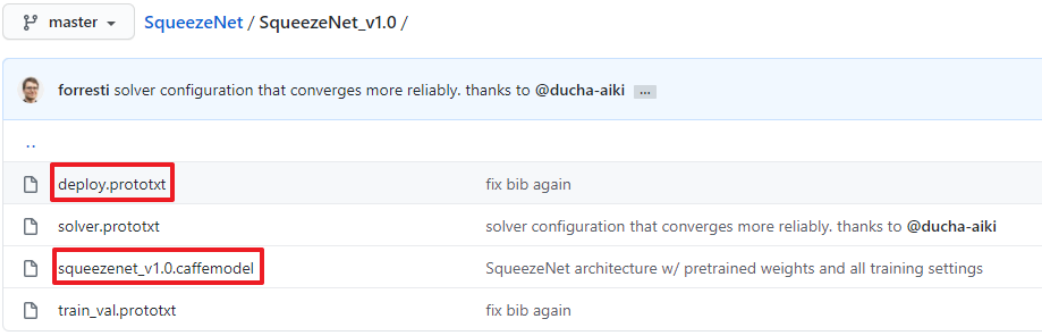
Generic image models

These models are pre-trained on the 1000 classes of the ImageNet ILSVRC Challenge. They can be used a is or finetuned to more targeted applications.

	Caffe	Tensorflow	Source	Top-1 Accuracy (ImageNet)
AlexNet	Y	N	BVLC	57.1%
SqueezeNet	Y	N	DeepScale	59.5%
Inception v1 / GoogleNet	Y	Y	BVLC / Google	67.9%
Inception v2	N	Y	Google	72.2%
Inception v3	N	Y	Google	76.9%
ResNet 50	Y	Y	MSR	75.3%
ResNet 101	Y	Y	MSR	76.4%
ResNet 152	Y	Y	MSR	77%
Inception-ResNet-v2	N	Y	Google	79.79%
VGG-16	Y	Y	Oxford	70.5%
VGG-19	Y	Y	Oxford	71.3%

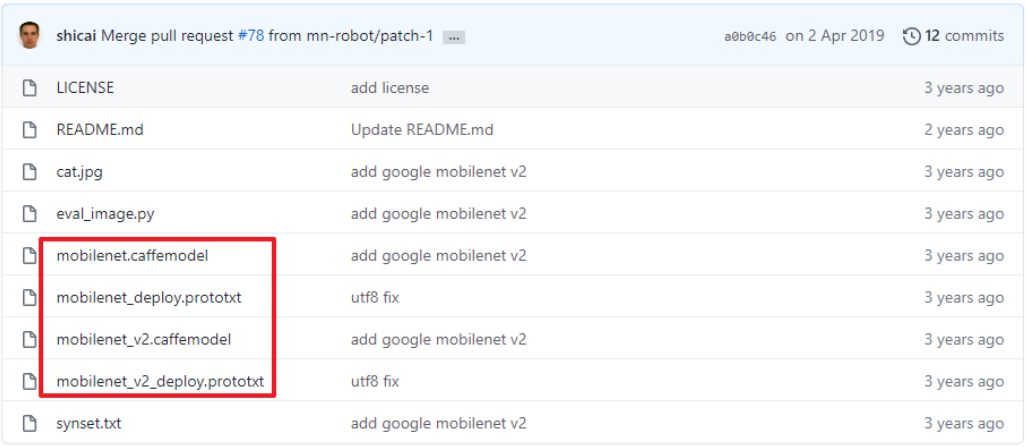
3.3.5 Squeezenet

Github：**https://github.com/forresti/SqueezeNet/tree/master/SqueezeNet_v1.0**



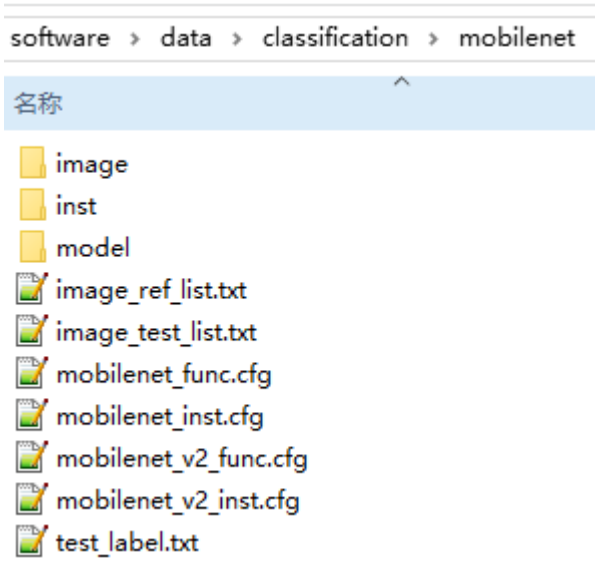
3.3.6 Mobilenet

Github：**https://github.com/shicai/MobileNet-Caffe**



可修改prototxt提高执行效率和推理精度。

- 对于Mobilenet v1，使用DepthwiseConv替换group Conv；
- 对于Mobilenet v2，使用DepthwiseConv替换group Conv，修改部分layer为inplace重训。详见“HiSVP_PC_Vx.x.x.x/software/data/classification/mobilenet”。



3.3.7 Faster-rcnn VGG16

Github：**https://github.com/rbgirshick/fast-rcnn**

Prototxt：**https://github.com/rbgirshick/fast-rcnn/tree/master/models**

Caffemodel：**https://dl.dropboxusercontent.com/s/e3ugqq3lca4z8q6/fast_rcnn_models.tgz**

详见**https://github.com/rbgirshick/fast-rcnn/blob/master/data/scripts/fetch_fast_rcnn_models.sh**

3.3.8 Faster-rcnn PVANet

Github：**https://github.com/sanghoon/pva-faster-rcnn/**

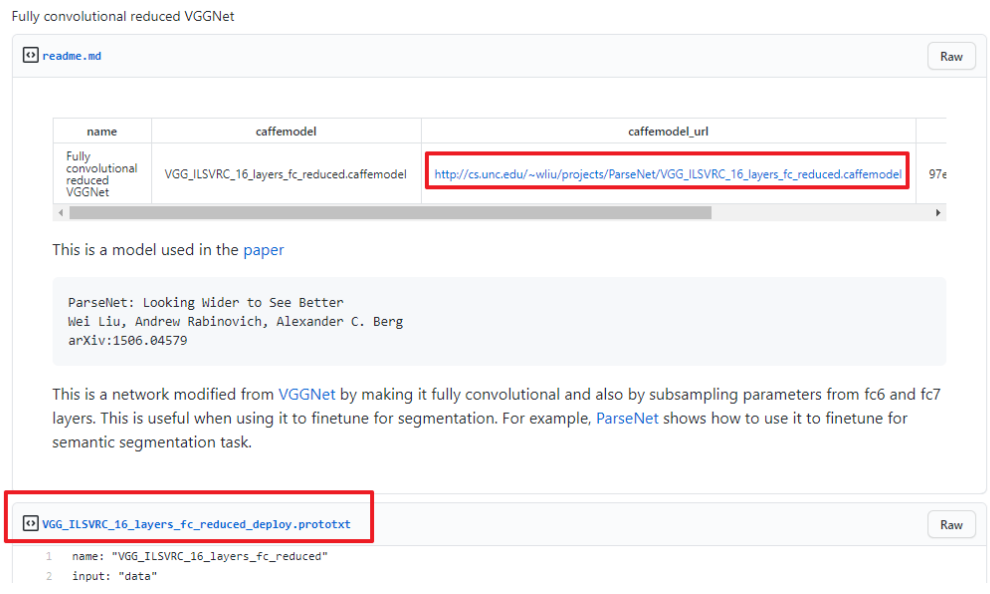
Prototxt：**https://github.com/sanghoon/pva-faster-rcnn/blob/master/models/pvanet/pva9.1/faster_rcnn_train_test_ft_rcnn_only_plus_comp.pt**

Caffemodel：**https://www.dropbox.com/s/76q7pdym70ji986/PVA9.1_ImgNet_COCO_VOC0712plus_compressed.caffemodel?dl=1**

3.3.9 SSD

Github：**https://github.com/stoneyang-detection/caffe_ssd**

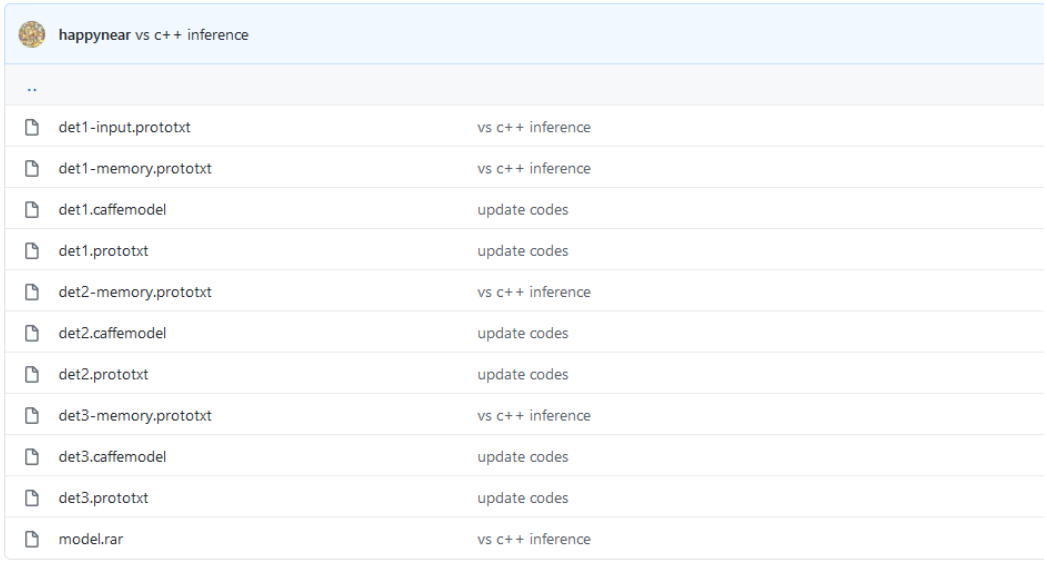
Prototxt及Caffemodel：**https://gist.github.com/weiliu89/2ed6e13bfd5b57cf81d6**



3.3.10 MTCNN

Github：**https://github.com/happynear/MTCNN_face_detection_alignment**

Prototxt及caffemodel：**https://github.com/happynear/MTCNN_face_detection_alignment/tree/master/code/codes/MTCNNv1/model**



3.3.11 Segnet

Github：**https://github.com/alexgkendall/SegNet-Tutorial**

Prototxt：**https://github.com/alexgkendall/SegNet-Tutorial/tree/master/Models**

Caffemodel：**https://github.com/alexgkendall/SegNet-Tutorial/blob/master/Example_Models/segnet_model_zoo.md**

3.4 Linux 版 NNIE mapper 安装

linux版本的NNIE mapper编译环境：ubuntu 14.04，gcc version 4.8.5

下文以ubuntu14.04为例安装linux版本的NNIE mapper。

3.4.1 mapper 依赖库 Protobuf 安装与配置

3.4.1.1 安装包下载

请至https://github.com/protocolbuffers/protobuf/releases/tag/v3.9.0，进入下载页面点击protobuf-all-3.9.0.tar.gz：

图 3-42 Protobuf 下载页面

3.4.1.2 编译与安装

步骤1 将protobuf-all-3.9.0.tar.gz移至目标目录（假定为/home/test），终端输入tar xvzf protobuf-all-3.9.0.tar.gz进行解压。

步骤2 进入解压后目录，执行mkdir build_tmp && cd build_tmp 创建并进入build_tmp目录，终端输入cmake -Dprotobuf_BUILD_TESTS=OFF -Dprotobuf_BUILD_SHARED_LIBS=ON -DCMAKE_INSTALL_PREFIX=/home/test/protobuflib/protobuf-3.9.0 -D CMAKE_CXX_FLAGS=-D_GLIBCXX_USE_CXX11_ABI=0 ../cmake，指定安装目录为home/test/protobuflib/protobuf-3.9.0。

步骤3 继续在当前build_tmp目录进行编译，终端输入make命令完成编译。

步骤4 在终端输入 make install 完成protobuf的安装，最终安装目录即步骤2所指定目录。

----结束

须知

如果发生Permission denied导致无法新建、复制文件(夹)的情况，需要给目标文件夹*开通权限。*

请编译protobuf时cmake使用选项-D CMAKE_CXX_FLAGS=-D_GLIBCXX_USE_CXX11_ABI=0，以兼容GCC4.8ABI，否则运行nnie_mapper时报错undefined symbol: _ZN6google8protobuf8internal26fixed_address_empty_stringE。

3.4.1.3 环境变量设置

把Protobuf的头文件和lib文件加入相应的环境变量中，注意替换“/home/test”为用户自己的目标目录。

步骤1 在终端中输入 vi ~/.bashrc

步骤2 进入*vi编辑*bashrc文件，在文件末尾添加：

```
export PATH=/home/test/protobuflib/protobuf-3.9.0/bin:$PATH
export LD_LIBRARY_PATH=/home/test/protobuflib/protobuf-3.9.0/lib:$LD_LIBRARY_PATH
export PKG_CONFIG_PATH=/home/test/protobuflib/protobuf-3.9.0/lib/pkgconfig
```

步骤3 退出vi，在终端输入source ~/.bashrc

----结束

说明

用户也可以在当前终端中以命令形式依次输入步骤2追加内容，该方法只会对当前终端生效。

3.4.1.4 测试 Protobuf 是否成功安装

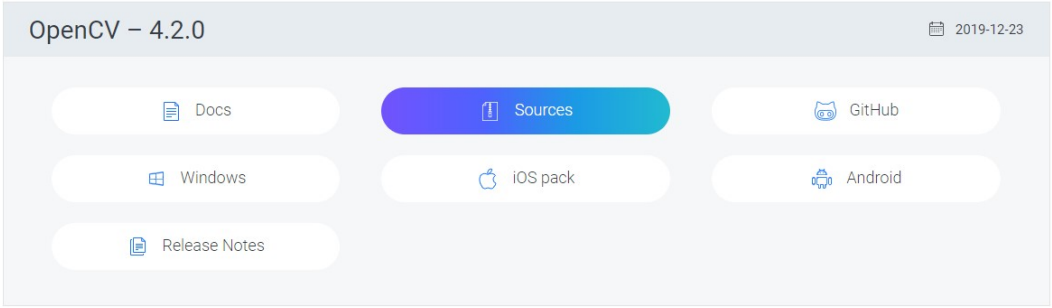
在终端输入protoc --version，如果执行通过，并显示版本号，表明环境正确。

3.4.2 mapper 依赖库 OpenCV 安装与配置

3.4.2.1 OpenCV 安装包下载

请到OpenCV的官方网站：**http://opencv.org/releases.html** 下载Opencv4.2.0。

图 3-43 下载 Opencv4.2.0



3.4.2.2 OpenCV 依赖库安装

根据需要安装OpenCV依赖库，其中cmake是必需的。

步骤1 安装cmake，输入：

sudo apt-get install cmake

步骤2 安装GTK+ 2.0或更高版本（可选），输入：

sudo apt-get install libgtk2.0-dev

步骤3 安装libav开发包：libavcodec-dev, libavformat-dev, libswscale-dev，输入：

sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev

----结束

3.4.2.3 OpenCV 安装包解压

步骤1 新建install目录，将下载好的opencv-4.2.0.zip复制到install。进入install目录，并在终端输入unzip opencv-4.2.0.zip，得到opencv-4.2.0文件夹。

步骤2 在opencv-4.2.0内，建一个build文件夹，用来存放编译文件。

----结束

3.4.2.4 OpenCV 的安装前配置—使用 cmake 方式

步骤1 进入build目录，cd build

步骤2 在终端输入cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/home/test/opencvlib/opencv4.2.0 -D WITH_GPHOTO2=OFF –D WITH_GTK=OFF -D WITH_CUDA=OFF -D WITH_FFMPEG=OFF -D WITH_LAPACK=OFF -D CMAKE_CXX_FLAGS=-D_GLIBCXX_USE_CXX11_ABI=0 ../

说明

WITH_GPHOTO2=OFF和WITH_GTK=OFF WITH_CUDA=OFF -D WITH_FFMPEG=OFF -D WITH_LAPACK=OFF用于关闭编译依赖项，建议关闭编译依赖项，降低编译复杂度。

-D CMAKE_CXX_FLAGS=-D_GLIBCXX_USE_CXX11_ABI=0 用于兼容GCC4.8 ABI。

----结束

3.4.2.5 OpenCV 的编译与安装

步骤1 将工作目录移动到opencv的build目录(本文是/install/opencv-4.2.0/build)，输入make

该过程将运行较长时间，可以使用“make –j ?”代替“make”命令进行并行编译，此处的“?”为要并行使用的CPU核心数（job数），可跟电脑的具体CPU核数修改该数字。

步骤2 Make完毕后，在终端中输入：make install完成OpenCV的安装。

----结束

3.4.2.6 OpenCV 环境变量设置

步骤1 在终端中输入vi ~/.bashrc 进入*vi编辑*.bashrc文件

步骤2 在文件末尾添加：

```
export PATH=/home/test/opencvlib/opencv4.2.0/bin:$PATH
export LD_LIBRARY_PATH=/home/test/opencvlib/opencv4.2.0/lib:$LD_LIBRARY_PATH
export PKG_CONFIG_PATH=/home/test/opencvlib/opencv4.2.0/lib/pkgconfig
```

步骤3 退出vi，在终端输入source ~/.bashrc

----结束

说明

如果按照本文所述，OpenCV会安装在/home/test/opecvlib/opencv4.2.0目录中，如果变更了安装目录，环境变量设置时应该同步变更。

3.4.2.7 测试 OpenCV 是否安装成功

步骤1 将工作目录转到OpenCV解压目录下的/samples/cpp/处。

步骤2 然后尝试编译其中人脸识别的示例程序，输入gcc `pkg-config --cflags opencv` -o facedetect facedetect.cpp `pkg-config --libs opencv` -lstdc++

如果顺利编译，说明OpenCV安装成功，并且会在该目录下生成facedetect的可执行程序。

说明

如果不是复制该命令的话，请注意 符号`不是单引号，而是反引号(大键盘数字1左边)。

步骤3 尝试运行该程序，输入 ./facedetect ../data/lena.jpg，该程序会用椭圆框出检测到的人脸。如果成功运行，说明OpenCV安装成功。

----结束

须知

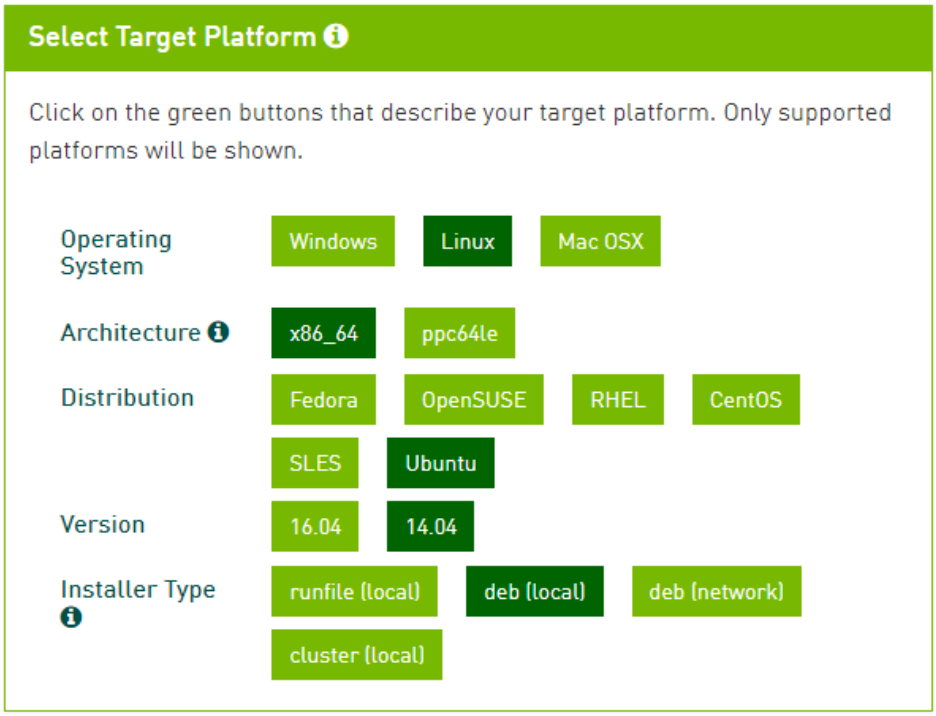
如果编译通过，但运行的时候提示error while loading shared libraries: libopencv_objdetect.so.3.4 错误。请执行sudo ldconfig命令，再用sudo ldconfig –v | grep –i opencv 查看动态链接库是否已刷新。可以手动更新一下索引库，输入sudo updatedb。再运行该程序，若还出现问题，请重启系统。

3.4.3 mapper 依赖库 CUDA 安装与配置

若使用CPU版本的mapper（参考“3.5.1 nnie_mapper 版本说明”），可以跳过该节。

3.4.3.1 CUDA 安装包下载

请到CUDA的官方网站：https://developer.nvidia.com/cuda-80-download-archive 下载CUDA 8.0。



说明

安全起见请在shell输入lspci | grep -i nvidia确认是否支持CUDA。

3.4.3.2 CUDA 安装

请参考官网安装指南https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#ubuntu-installation

3.4.4 mapper 本体安装

步骤1 在目标目录中新建bin目录，将HiSVP_PC_Vx.y.z.w \tools\nnie\linux下的mapper拷贝到该目录，下文以Hi3559AV100 nnie_mapper_11为例，gpu加速及其他芯片版本类似。

```
[redacted]@svp162:~/bin$ ls
nnie mapper 11
```

步骤2 在终端中输入vi ~/.bashrc 进入*vi编辑*.bashrc文件

步骤3 在文件末尾添加（注意替换为用户自己的目标路径，即nnie_mapper_11所在目录）：

```
export PATH=/home/bin/:$PATH
```

步骤4 退出vi，在终端输入source ~/.bashrc

步骤5 以HiSVP_PC_Vx.x.x.x/software/data/classification为例，进入alexnet目录

```
[redacted]/sample/data/classification/alexnet$ ls
alexnet_func.cfg  alexnet_inst.cfg  alexnet_no_group_func.cfg  alexnet_no_group_inst.cfg  inst  model
```

目录中的*.cfg文件说明参考“3.5.1 nnie_mapper 版本说明”；

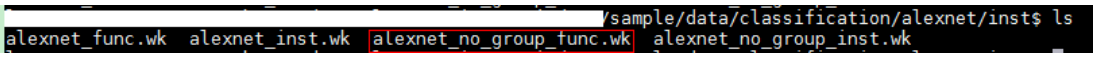
model目录中prototxt要求详见“3.2 Prototxt要求”章节，caffemodel严格遵循caffe规则，mean.txt或者*.binaryproto为预处理减均值文件。

步骤6 返回data上一级目录，命令行输入nnie_mapper_11 ./data/classification/alexnet/alexnet_no_group_func.cfg即可运行nnie_mapper_11，如下打印表示nnie_mapper正常执行；

```
[redacted]sample$ nnie_mapper_11 ./data/classification/alexnet/alexnet_no_group_func.cfg
Mapper Version 1.1.2.0_B010 (NNIE_1.1) 1807281514145495
begin net parsing....
end net parsing
begin prev optimizing....
end prev optimizing....
begin net quantalizing....
end quantalizing
begin POST optimizing....
end POST optimizing
begin NNIE[0] mem allocation....
end NNIE[0] memory allocating
begin NNIE[0] instruction generating....
end NNIE[0] instruction generating
begin lbs binary code generating....
end lbs binary code generating
```

步骤7 在cfg指定生成一个后缀为wk的文件，即为仿真器上的可执行文件；

```
alexnet_no_group_func.cfg
 1 [prototxt_file]  ./data/classification/alexnet/model/bvlc_alexnet_no_group_deploy.prototxt
 2 [caffemodel_file] ./data/classification/alexnet/model/bvlc_alexnet_no_group.caffemodel
 3 [batch_num] 256
 4 [net_type] 0
 5 [sparse_rate] 0
 6 [compile_mode] 0
 7 [is_simulation] 1
 8 [log_level] 2
 9 [instruction_name] ./data/classification/alexnet/inst/alexnet_no_group_func
10 [RGB_order] BGR
11 [data_scale] 0.0039062
12 [internal_stride] 16
13 [image_list] ./data/classification/imagenet/image_ref_list.txt
14 [image_type] 1
15 [mean_file] ./data/classification/alexnet/model/imagenet_mean.binaryproto
16 [norm_type] 1
```

----结束

3.5 Linux 版 nnie_mapper 使用说明

3.5.1 nnie_mapper 版本说明

表 3-10 芯片和 nnie mapper 版本对应关系表

芯片	NNIE版本	Mapper版本
Hi3559AV100	nnie1.1	nnie_mapper_11 nnie_mapper_gpu_11
Hi3559CV100	nnie1.1	nnie_mapper_11 nnie_mapper_gpu_11
Hi3569V100	nnie1.1	nnie_mapper_11 nnie_mapper_gpu_11
Hi3519AV100	nnie1.2	nnie_mapper_12 nnie_mapper_gpu_12
Hi3556AV100	不支持	不支持
Hi3568V100	nnie1.2	nnie_mapper_12 nnie_mapper_gpu_12
Hi3516DV300	nnie1.2	nnie_mapper_12 nnie_mapper_gpu_12
Hi3516CV500	nnie1.2	nnie_mapper_12 nnie_mapper_gpu_12
Hi3531DV200	nnie1.3	nnie_mapper_13 nnie_mapper_gpu_13
Hi3535AV100	nnie1.3	nnie_mapper_13 nnie_mapper_gpu_13
Hi3521DV200	nnie1.3	nnie_mapper_13 nnie_mapper_gpu_13
Hi3520DV500	nnie1.3	nnie_mapper_13 nnie_mapper_gpu_13

nnie_mapper_xy是纯CPU的mapper版本，nnie_mapper_gpu_xy是GPU加速的mapper版本。

须知

- NNIE板端运行、仿真，必须按上表的芯片型号和mapper版本匹配，否则可能出现意想不到的错误。
- GPU版本的nnie_mapper都需要安装CUDA8.0。
- GPU版本mapper和CPU版本的mapper编译出来的wk会有不同的地方，原因是cuda版本和CPU版本浮点的表示引起，caffe的CPU和GPU版本也会有些许差异。

3.5.2 配置文件说明

nnie_mapper配置选项说明如表3-11所示。

表 3-11 nnie_mapper 配置选项说明

配置选项	取值范围	描述
prototxt_file	-	网络描述文件，详细要求见“3.2 Prototxt要求”，描述外支持情况与caffe相同。
net_type	{0, 1, 2}	网络的类型。 0：CNN（不包含LSTM/RNN/ROIPooling/PSROIPooling的任意网络）； 1：ROI/PSROI（包含ROI Pooling和PSROI Pooling的网络）； 2：Recurrent（包含LSTM、RNN的网络）；
caffemodel_file	-	网络模型数据文件。

配置选项	取值范围	描述
image_list	-	NNIE mapper用于数据量化的参考图像list文件或feature map文件。该配置跟image_type相关。 NNIE mapper量化时需要的图片是典型场景图片，建议从网络模型的测试场景随机选择20~50张作为参考图片进行量化，选择的图像要尽量覆盖模型的各个场景（图像要包含分类或检测的目标，如分类网的目标是苹果、梨、桃子，则参考图像至少要包含苹果、梨、桃子。比如检测人、车的模型，参考图像中必须由人、车，不能仅使用人或者无人无车的图像进行量化）。图片影响量化系数，选择典型场景的图片计算出来的量化系数对典型场景的量化误差越小。所以请不要选择偏僻场景、过度曝光、纯黑、纯白的图片，请选择识别率高，色彩均匀的典型场景图片。 网络中如果存在多个输入层，则需要配置多个image_list顶，顺序、个数与prototxt完全对应。 如果网络的数据输入是灰度或者RGB图像输入，即image_type配置不为0，image_list配置为所有参考图片的list，内容示意如下图图示，图片的格式支持以下几种： ".bmp", ".dib", ".jpeg", ".jpg", ".jpe", ".jp2", ".png", ".webp", ".pbm", ".pgm", ".ppm", ".sr", ".ras", ".tiff", ".tif", ".BMP", ".DIB", ".JPEG", ".JPG", ".JPE", ".JP2", ".PNG", ".WEBP", ".PBM", ".PGM", ".PPM", ".SR", ".RAS", ".TIFF", ".TIF" image_ref_list.txt 1 ../data/classification/imagenet/ref/ILSVRC2012_val_00000001.JPEG 2 ../data/classification/imagenet/ref/ILSVRC2012_val_00000002.JPEG 3 ../data/classification/imagenet/ref/ILSVRC2012_val_00000003.JPEG 4 ../data/classification/imagenet/ref/ILSVRC2012_val_00000004.JPEG 如果网络的输入是feature map或者FC向量输入，即image_type配置为0，将c*h*w个点（即一个完整的张量）以浮点文本的形式输出在一行内，点与点之间以空格或逗号分隔。如果是多帧输入，则每一行输出一个完整的张量。图示如下： sigle_custom_multi_top_2_custom_1_f 1 0.852727574554 2.22428260776 7.28222365077 3.63192138208 4.44889580721 9.30500569598 7.51946751475 8.61053507313 5.11505773185 0.120343927741 1.55175787525 2.31107244679 7.32313007852 1.67330342699 9.80961173192 6.58306217981 8.58678546437 0.375591352241 2.96989362764 5.70649920558 5.16233427138 9.16785497049 3.61658342832 5.07637698021 5.11637303464 5.80114673548 9.99059787443 4.91098978583 8.48376954078 6.55287913647 7.33294496285 4.70459812637 6.85133955872 0.136886336092 1.42374282923 8.30464445037 9.1317536825 8.29587184813 0.294779163483 9.83021344124 3.69606456616 1.06406590556 7.67306035696 0.897114288304 4.88711021009 6.59023263591 7.48311238822 2.725971777 3.46687892092 2.99688353124 1.57899233521 2.83895671381 5.23338373429 2.66663094513 7.56908910552 6.84177125319 5.16009163347 6.81419291752 3.82894977639 8.12135448168 6.48772744135 7.2850829301 8.49627904474 1.11722630204 Recurrent输入时，格式等同于feature map输入，每行一个向量，一句话写成连续的多行，多句量化时需要将每一句的帧数都补齐为最大帧数。

配置选项	取值范围	描述
image_type	{0,1,3,5}	表示网络实际执行时输入给网络的数据类型，该配置跟image_list相关。 0：表示网络数据输入为 SVP_BLOB_TYPE_S32（参考《HiSVP API参考》）或者向量的类型（VEC_S32和SEQ_S32）；此时要求image_list配置为feature map文件； 1：表示网络数据输入为SVP_BLOB_TYPE_U8（普通的灰度图和RGB图）类型； 此时要求image_list配置是RGB图或者灰度图片的list文件； 3：网络数据输入为SVP_BLOB_TYPE_YUV420SP类型； 5：网络数据输入为SVP_BLOB_TYPE_YUV422SP类型； 当配置为3或者5时，image_list配置为RGB图片的list文件。
norm_type	{0, 1, 2, 3, 4, 5}	表示对网络数据输入的预处理方法。注意image_type配置为0时，norm_type只能配置为0；image_type配置为3或者5时，网络输入数据为YUV图像，但是NNIE硬件会根据RGB_order配置项自动转为RGB或者BGR图像，此时norm_type配置方法跟image_type为1时一致。 0：不做任何预处理； 1：mean file，减图像均值； 2：channel mean_value，减通道均值； 3：data_scale，对图像像素值乘以data_scale； 4：mean filewith data_scale，减图像均值后再乘以data_scale； 5：channel mean_value with data_scale，减通道均值后再乘以data_scale。
data_scale	(1/4096, FLT_MAX) default: 0.00390625	数据预处理缩放比例，配置为浮点数，配合norm_type使用 本参数可省略，默认为0.00390625=1/256。FLT_MAX等于3.402823466e+38。
mean_file	-	norm_type为1、4时，表示均值文件xxx.binaryproto； norm_type为2、5时，表示通道均值文件； norm_type为0、3时，用户也需要配置mean_file项，但具体内容可以是一个无效路径，比如null；通道均值文件mean.txt中每一行的浮点数表示对应的通道均值，如单通道只有一个值。

配置选项	取值范围	描述
batch_num	[0, 256] default: 256	0/1：single（单张）模式； >1：batch（多张）模式。 采用single模式mapper一个任务只能处理一张图片，内部存储全部为一张图片分配，减少数据调度次数。 采用batch模式，在计算FC时batch_num张图片同时计算，计算资源利用率高。
sparse_rate	[0, 1] default: 0	NNIE引擎采用了参数压缩技术以减少带宽占用，为了提高压缩率，可通对FC参数进稀疏处理。 用户通过sparse_rate数值指定多少比例的FC参数稀疏为0，例如配0.5，则FC参数有50%将被稀疏为0，由于数据变的稀疏，压缩模块会获得更好的压缩率。稀疏值越高，计算FC时所需参数带宽越低，但精度会有所下降。
compile_mode	{0, 1, 2} default: 0	0：Low-bandwidth(低带宽模式，默认)：通过量化算法使参数与数据位宽最少，使系统所需带宽达到最小，但会有精度损失； 1：High-precision(高精度模式): 结果精度最好，但是性能会下降； 2：User-specify(用户配置模式): 需要用户在prototxt中标明所有使用高精度计算的层，标注规则请见prototxt_file说明；注意：nnie_mapper_13仅支持compile_mode为0;
compress_mode	{0, 1} default: 0	配置压缩模式。 0：Normal模式（包含Normal、Ternary、Binary、Sparse四种压缩模式的自动切换）； 1：Bypass模式，关闭压缩功能。 要求： 可不填，默认为Normal模式；用户提供的参数只有三种值且正负对称时，nnie_mapper会自动进入Ternary模式；用户提供的参数只有两种值且包含0时，nnie_mapper会自动进入Binary模式；
max_roi_frame_cnt	[1, 5000] default: 300	包含ROI/PSROI网络的RPN阶段输出的候选框最大数目。 默认值：300。

配置选项	取值范围	描述
roi_coordinate_file	-	Mapper在网络模型转化过程中输入给ROI Pooling或PSROI Pooling层的配置参数，用于指定ROI框的坐标信息，每一行五个值，分别代表batch_index(int)、left_x（float）、top_y（float）、right_x（float）、bottom_y（float），不同的框以换行符分隔。 框坐标是在caffe中使用输入给mapper的image_list的相同图片运行到RPN层的输出结果，如Faster RCNN网络中Proposal 层的top为rois，在caffe forward结束后，通过np.savetxt('rois.txt', net.blobs['rois'].data[...], fmt="%.6f") 保存框坐标为文件。需要保证两者图片输入顺序相同，同时要保证caffe运行时输入给网络的分辨率跟配置给mapper的prototxt中的分辨率相同。 For example: 0 734.01 147.02 806.03 294.04 0 723.05 157.06 818.07 306.08 1 749.09 170.10 817.11 310.12 1 678.13 220.14 855.15 374.16 如果一个网络中有多个ROI Pooling或PSROI Pooling层，则需要配置多行坐标文件，个数与ROI Pooling或PSROI Pooling层个数对应，配置的顺序也需要与prototxt内对应层顺序相同；
is_simulation	{0, 1} default: 0	网络模型转化类型。 0：Chip，芯片模式，网络模型转化成在芯片上加载的wk文件，指令仿真也使用此模式； 1：Simulation，仿真模式，网络模型转化成在PC端仿真上加载的wk文件，功能仿真使用此模式；
instructions_name	default: inst	nnie_mapper生成的NNIE模型文件名称。 默认生成如下格式的NNIE模型文件名：inst.wk；用户也可以自行修改生成的NNIE模型文件名字。
internal_stride	{16, 32} default: 16	用户根据DDR颗粒对应的最佳读写效率配置中间结果的对齐方式。 要求： DDR3对应16，DDR4对应32，可不填，默认为16；
is_check_prototxt	{0, 1} default: 0	检查网络描述文件标志。 0：mapper模式，对prototxt、caffemodel等进行转化。 1：网络过滤器模式，对prototxt文件是否符合支持规格进行检查。

配置选项	取值范围	描述
log_level	{0, 1, 2, 3} default: 0	设置是否开启日志文件，以及配置打印的等级，本参数可省略，当省略时，为不打印日志文件。 0：打印main函数流程，cfg文件等信息； 1：打印nnie_mapper解析到的文件信息，包含image_list、prototxt、内存分配过程； 2：打印中间表示信息； 3：打印详细信息，有大量文件输出，转化耗时较长，请谨慎使用；
recurrent_tmax	[1, 1024] default: 1024	Recurrent网络（包含LSTM/RNN层）每一句话的最大桢数，支持[1, 1024]范围内的配置，减小配置值可以减小临时缓存大小。
RGB_order	{RGB, BGR} default: BGR	image_type设置为0时，该参数无效； image_type设置为1时，不管该参数配置何值，要求用户板端输入必须为BGR_Planar格式图像； image_type设置为3、5时，表示YUV图像数据转成RGB Planar或者BGR Planar图像输入给网络。 本参数可省略。
image_width	[8, 4096]	当image_type=0且网络中包含RPN硬化层时，必须配置原图的宽高。
image_height	[8, 4096]	当image_type=0且网络中包含RPN硬化层时，必须配置原图的宽高。

配置选项	取值范围	描述

gfpq_param_file	-	量化参数配置文件。
		1) 文件内容为层名和对应的量化参数。nnie_mapper按层名匹配，使用配置的gfpq_param作为量化参数。没有配置的层，则由nnie_mapper编译时生成。
		2) cfg 配置log_level 大于等于2，nnie_mapper输出gfpq_param.txt，内容为最终的量化参数，用于确认量化参数是否为用户的期望值，也可以修改此文件的gfpq_param.buf，用作[gfpq_param_file]。gfpq_param.txt 只包含需要使用量化参数的层，而[gfpq_param_file] 也只需配置用到量化参数的层。
		3) gfpq_param.buf 为16进制字符串，可通过调用量化库接口获取GFPQ_PARAM_ST, 把GFPQ_PARAM_ST.buf 输出为16进制字符串。
		4) 示例:
		layer {
		name: "conv1"
		gfpq_param {
		buf: "F8FBE3FF80FFFFFF5D5E465AA2A1B9A5"
		}
		}
		见HiSVP_PC发布包tools/gfpq_lib/gfpq_param_sample.tgz。

【注意】

Custom层需要为nnie_mapper配置输入层配置项，以下例子展示包含一个data层和一个双输出Custom层的配置文件。

```
1  [prototxt_file] ./sigle_custom_multi_top_2.prototxt
2  [caffemodel_file] ./sigle_custom_multi_top_2.caffemodel
3  [net_type] 0
4  [batch_num] 1
5  [norm_type] 3
6  [mean_file] ignore
7  [image_type] 1
8  [image_list] ./images
9  [norm_type] 3
10 [mean_file] ignore
11 [image_type] 0
12 [image_list] ./sigle_custom_multi_top_2_custom_1_1
13 [norm_type] 3
14 [mean_file] ignore
15 [image_type] 0
16 [image_list] ./sigle_custom_multi_top_2_custom_1_2
17
```


即当前用例有一个data层，为其配置①号框内的配置项，Custom有两个top，为其配置②③两个框内的配置项，image_list为一个文件名，文件内容是Custom对应top输出的caffe结果，格式见image_list配置项说明，文本格式的浮点数，以空格或逗号分隔，一个tensor(c*h*w)一行；

3.5.3 nnie_mapper 库说明

libnnie_mapper_*.so 提供转换模型的接口，用于网络训练结束时，直接使用内存中的prototxt和caffemodel转换模型。目的是避免把prototxt和caffemodel保存为文件使网络模型泄漏。头文件和库见发布包HiSVP_PC_Vx.y.z.w\tools\nnie\linux\mapper\libnnie_mapper。

接口定义：

```
bool GenerateModelBinary(const char *cfgFile, const char *protoArray, const int32_t protoArraySize,
const char *paramArray, const int32_t paramArraySize)
```

- 参数描述：
  - cfgFile：nnie_mapper的配置文件。不用配置prototxt_file和caffemodel_file。
  - protoArray：prototxt使用NetParameter->SerializeToArray()序列化后的的内存指针。
  - protoArraySize：prototxt的内存大小。
  - paramArray：caffemodel使用NetParameter->SerializeToArray()序列化后的的内存指针。
  - paramArraySize：caffemodel 的内存大小。
- 返回值：
  - false：转换模型失败。
  - true：转换模型成功。

3.5.4 nnie_mapper 输出

nnie_mapper默认输出如下文件：

- *.wk：mapper运行成功时生成的NNIE模型文件；用户可通过instruction_name配置选项更改保存路径和文件名。
- cnn_net_tree.dot：mapper运行成功时生成的网络结构dot描述；保存在工程所在目录中，使用graphviz可显示为dot视图。
- mapper_error.log：mapper运行出错时生成的错误日志。
- mapper_debug.log：mapper运行日志。

3.6 仿真库使用说明

NNIE PC端仿真库分为功能仿真(nniefc*.lib)、指令仿真(nnieit*.lib)和性能仿真：

- 功能仿真仅从功能一致性的角度去模拟硬件，有CUDA加速功能，速度较快；
- 指令仿真从指令一致性的角度去模拟硬件，速度较慢；
- 性能仿真输出各层的带宽、cycle数仿真结果；性能仿真默认配置下跟指令仿一并执行。

功能仿真、指令仿真、硬件三者的结果保持完全一致。

3.6.1 环境说明

NNIE PC端仿真库基于Windows10环境，vc14编译，静态编译，运行库/MD。

提供Visual Studio样例工程nnie1.x/windows/sample_simulator/simulator_sample_vc14.sln。

使用RuyiStudio运行仿真库详见“5.4 **仿真NNIE功能**”。

3.6.2 使用限制

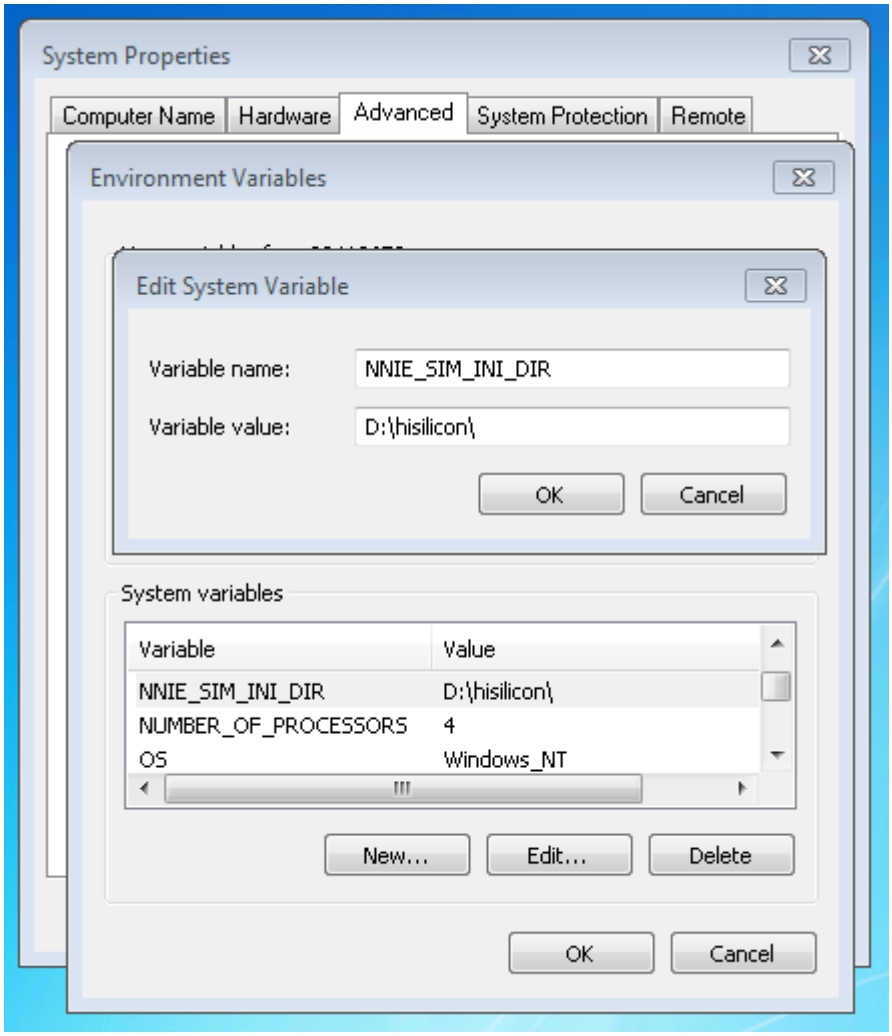
仿真库仅支持单线程执行，不支持多线程仿真。

3.6.3 配置文件设置

仿真库配置文件nnie_sim.ini可以控制仿真库的中间层结果输出、CUDA加速配置、性能仿真和指令仿真开关等选项。

- 仿真库首先尝试从可执行程序执行同级文件夹载入配置文件nnie_sim.ini；
- 用户也可通过配置系统环境变量NNIE_SIM_INI_DIR指定配置文件nnie_sim.ini所在文件夹路径；如**图3-44**表示ini配置文件在D:\hisilicon\目录中。
- 此外，仿真库还会尝试从以下固定系统路径载入配置文件nnie_sim.ini

图 3-44 配置文件示意图



Windows环境：C:/hisilicon/nnie_sim.ini

用户若需要修改仿真库配置项，可以根据上述不同方式，将HiSVP_PC_Vx.x.x.x/software/hisilicon中的配置文件nnie_sim.ini模板拷贝到可执行程序执行同级文件夹、环境变量NNIE_SIM_INI_DIR指定路径或固定系统路径（windows的C:/hisilicon/），并修改对应配置项。

配置文件nnie_sim.ini加载优先级：可执行程序执行同级文件夹 > 环境变量NNIE_SIM_INI_DIR指定路径 > 固定系统路径

若上述三种方式仿真库读取配置文件nnie_sim.ini均失败，则使用默认配置。

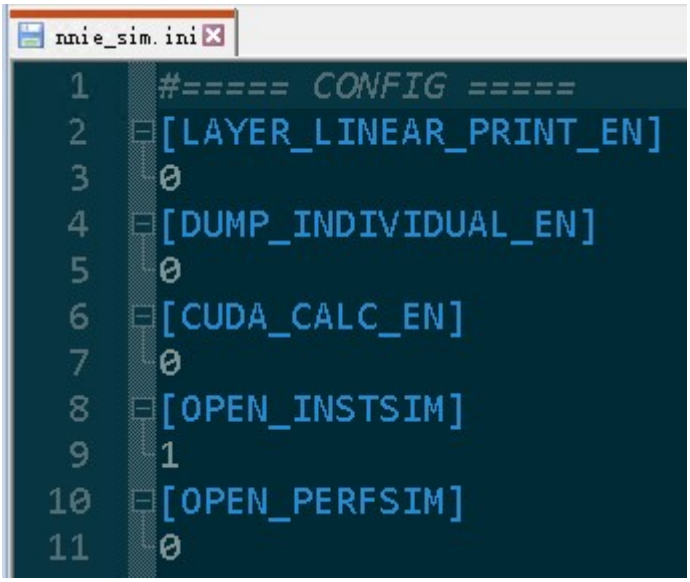
须知

- 在仿真器sample的Visual Studio解决方案(HiSVP_PC_Vx.x.x.x/software/sample_simulator/simulator_sample_vc14.sln)中执行sample项目程序时(以项目sample_func_vc14为例)，程序执行同级文件夹为项目名文件夹(如HiSVP_PC_Vx.x.x.x/software/sample_simulator/sample_func_vc14/)，欲使用程序执行同级文件夹的方法载入配置文件，需把配置文件nnie_sim.ini拷贝到该目录下。
- 在RuyiStudio中运行仿真sample程序的配置方法见“5.4.1 仿真配置文件编辑”。

3.6.4 配置文件说明

中括号[]内指定配置选项，第二行以0/1控制配置选项开关，1表示使能。

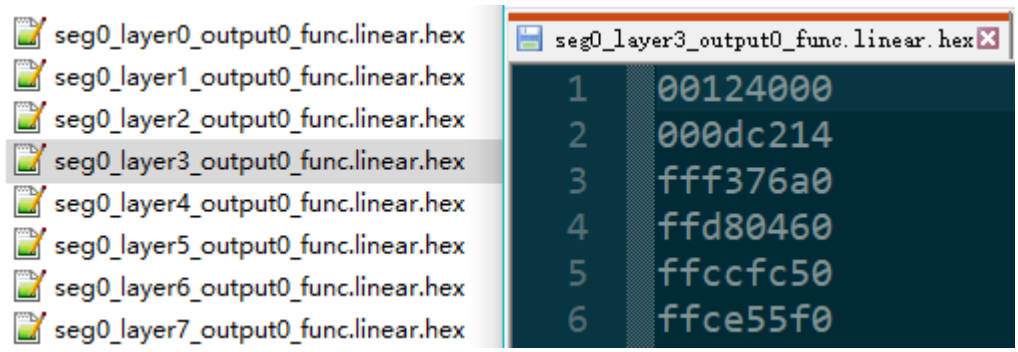
图 3-45 配置文件说明



[LAYER_LINEAR_PRINT_EN] 控制仿真中间层结果保存使能。该项对功能仿真库和指令仿真库均生效。如

功能仿真：sample_simulator\sample_func_vc14\func_layer_output_linear\seg0_layer0_output0_func.linear.hex

指令仿真：sample_simulator\sample_inst_vc14\inst_layer_output_linear\seg0_layer0_output0_inst.linear.hex

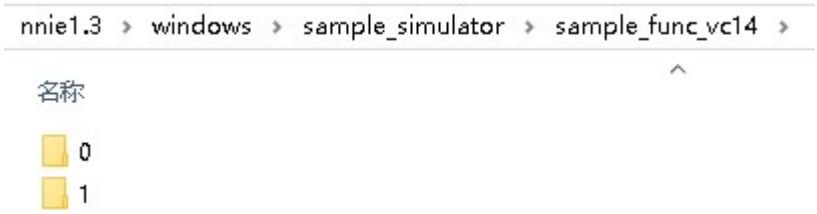


中间层结果以32位定点数20.12格式保存。

[DUMP_INDIVIDUAL_EN] 控制仿真中间结果独立保存使能。该项使能时，当一个程序加载了多个模型并执行时，网络执行中间结果inst_layer_output_linear文件夹以进程内模型加载顺序在0,1,2…文件夹依次独立存放，不彼此覆盖。该项对功能仿真库和指令仿真库均生效。

如顺序执行lenet和alexnet时，lenet的中间结果func_layer_output_linear在0文件夹内，alexnet的中间结果在1文件夹内。

图 3-46 中间结果独立存放效果举例



[CUDA_CALC_EN] 控制CUDA计算使能。该项仅对功能仿真库生效。

仿真CUDA配置方法详见“**3.6.5 功能仿真CUDA加速配置**”。

[OPEN_INSTSIM] 控制指令仿真使能。该项仅对指令仿真库生效。

[OPEN_PERFSIM] 控制性能仿真使能。该项仅对指令仿真库生效。

须知

- 配置文件中以#表示注释，行首、行尾勿添加空格、Tab等内容。
- 在跑多段网络时，如果将[OPEN_PERFSIM]置1，则必须将[OPEN_INSTSIM]置1.

3.6.5 功能仿真 CUDA 加速配置

3.6.5.1 环境要求

- Windows 10
- CUDA Toolkit 版本 >= 8.0
- GPU驱动版本 >= 418.96
- 建议显存 >= 6GB

3.6.5.2 架构支持

SM_30, SM_35, SM_37

SM_50, SM_52, SM_53

SM_60, SM_61, SM_62

SM_70, SM_75

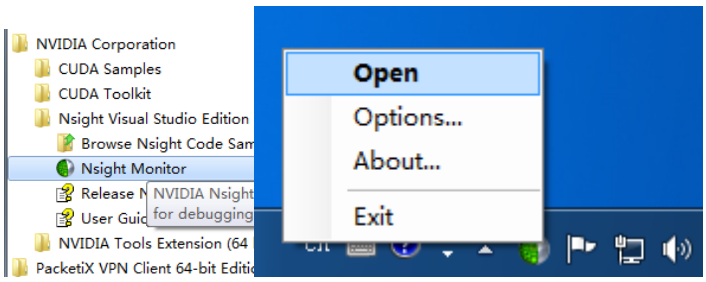
CUDA Toolkit版本、GPU驱动版本、NVIDIA架构支持说明详见:

https://docs.nvidia.com/deploy/cuda-compatibility/index.html

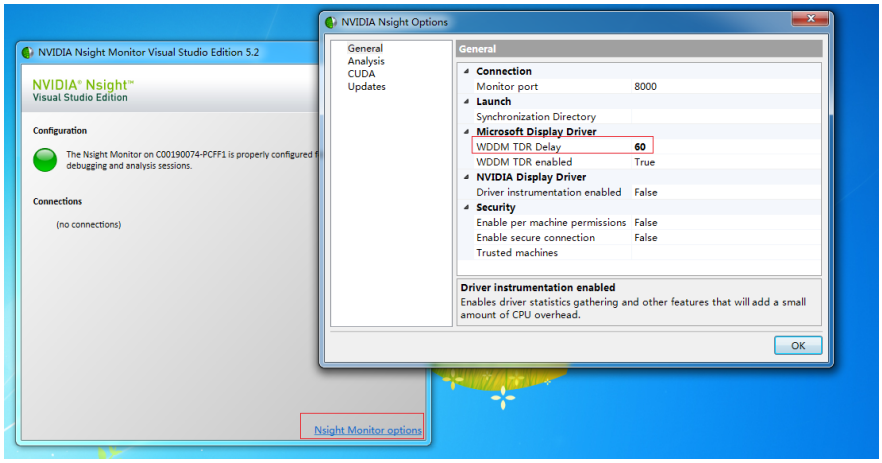
3.6.5.3 配置步骤

步骤1 确认已安装NVIDIA CUDA，要求CUDA版本至少8.0；

步骤2 以**管理员权限**在 开始菜单->NVIDIA Corporation->Nsight Visual Studio Edition 打开Nsight Monitor。若点击无反应则可能Nsight Monitor已运行，可在桌面右下角右键点击Nsight Monitor图标，选择Open打开。

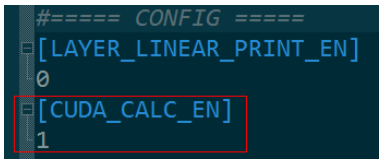


步骤3 点击Nsight Monitor窗口右下角的Nsight Monitor options，打开配置窗口。



步骤4 修改Nsight Monitor options中General标签页的WDDM TDR Delay，默认为2，调整至适当值如60，点击确认后重启计算机。此项表示GPU占用图形界面无响应时的重启时间，当CUDA连续计算超过该设定值时，会重启显示驱动，终止计算，恢复图形界面响应。

步骤5 在NNIE仿真器配置文件C:\hisilicon\nnie_sim.ini中，修改CUDA使能配置项[CUDA_CALC_EN]。修改为“1”使能仿真器CUDA加速。



步骤6 运行NNIE功能仿真器程序时，提示如下图红框所示的设备信息，即NNIE仿真器CUDA加速调用成功。


```
SVP Version: HiSVP_PC_V1.3.0.1_nightly_test
build sha1: b52d63b6

wk Version Info
      sdk_arch_ver:     1.1
      nnie_arch_ver:    1.3
      nnie_mapper_ver:  1.3.0.9
      test_ver:         B010
      patch_ver:        P000
      build_ver:        20042014388870
Tue May 12 20:29:16 2020 I >>>>>>>>> createNet >>>>>>>>> virAddr = 0x561fb0

Detected 1 CUDA Capable device(s)

********************************************************************************

                        CUDA Device(0) Properties


Identify:                                       Quadro M5000
CUDA Driver Version / Runtime Version           10.2 / 8.0
CUDA Capability Major/Minor version number:     5.2
Total amount of global memory:                  8192 MBytes (8589934592 bytes)
Current Free Memory(For reference only):        7949 MBytes (8335323136 bytes)
Clock Rate:                                     1038000 kHz
Max Grid Size:                                  2147483647 * 65535 * 65535
Max Threads Dim:                                1024 * 1024 * 64
Max Threads per Block:                          1024
Number of Multiprocessors:                      16
32bit Registers Available per Block:            65536
Warp Size:                                      32 threads
********************************************************************************

Set CUDA Device(0)

Tue May 12 20:29:16 2020 I created segment(0): CNN
Tue May 12 20:29:16 2020 I created layer(0): preprocess
Tue May 12 20:29:16 2020 I created layer(1): convolution
Tue May 12 20:29:16 2020 I created layer(2): poolingmax
Tue May 12 20:29:16 2020 I created layer(3): convolution
Tue May 12 20:29:16 2020 I created layer(4): poolingmax
Tue May 12 20:29:16 2020 I created layer(5): innerproduct
Tue May 12 20:29:16 2020 I created layer(6): innerproduct
Tue May 12 20:29:16 2020 I created layer(7): softmax
```

----结束

3.7 NNIE 调试

NNIE mapper运行无错误提示且在上板或者仿真库中能正常运行，但是结果不对或精度不正常时，可以参考下述步骤进行问题定位。

3.7.1 预处理一致性问题

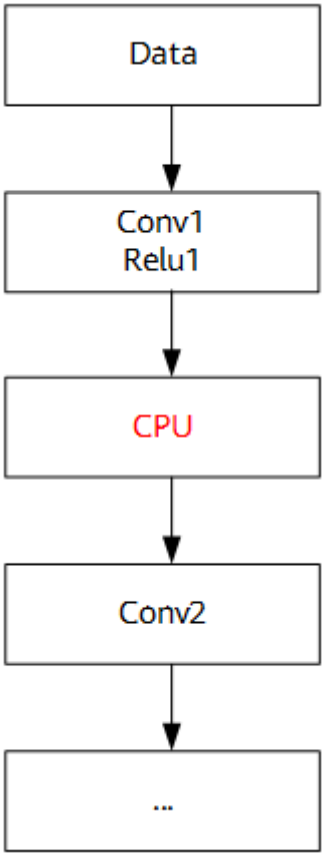
nnie_mapper支持的预处理方式有6种，参考3.5.1节中**表3-10**的norm_type说明。首先要保证NNIE mapper的预处理方式跟训练时候的一致。

3.7.2 多段网络时输入问题

网络有Propsal或Custom层时，网络被切分为多段。以下图示为例，NNIE mapper要求conv2层需要一个输入，该输入是该网络在Caffe中使用输入给mapper的参考图像在

caffe中运行到CPU层的结果。输入给mapper的Custom层输出格式参考**表3-10**的image_list说明，Proposal层输出格式参考**表3-10**的roi_coordinate_file说明。

图 3-47 多段网络示意图



3.7.3 精度下降问题

将nnie_mapper配置选项log_level设置为3，nnie_mapper会输出各层量化推断结果，每层结果保存在文件名为“layername_output$id_$C_$H_$W._quant.linear.hex”，记为结果A；

上板打印网络对应各层结果保存名为segid_layerid_output$id_board.linear.hex，记为结果$B_0$；

指令仿打印结果保存名为segid_layerid_output$id_inst.linear.hex，记为结果$B_1$；

功能仿真打印结果保存名为segid_layerid_output$id_func.linear.hex，记为结果$B_2$；

使用NNIE工具包中tools/nnie/RuyiStudio-x.x.x.zipRuyiStudio-x.x.x.zip中Resources/pythonScrip的脚本CNN_convert_bin_and_print_featuremap.py打印出在Caffe上运行的每层结果保存名为layername_output$id_$C_$H_$W._caffe.linear.float，记为结果C；

使用UI的比较工具分析结果A\$B_0$\$B_1$\$B_2$\C，出现相似度异常的层则表示该层出现了问题，在使用UI工具或者手动把相关层的数据给到上海海思进行问题复现和分析定位。

精度下降问题定位步骤见**图3-48**。

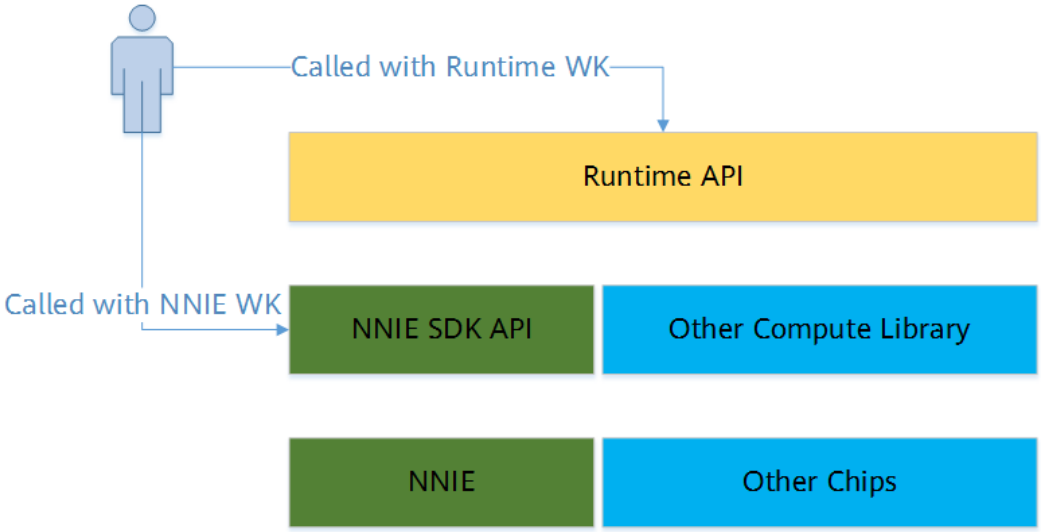
图 3-48 精度下降问题定位步骤示意图

4 Runtime 开发指南

4.1 Runtime 介绍

4.1.1 层次关系

图 4-1 Runtime API 交付层次关系图



4.1.2 工具链介绍

SVP Runtime在HiSVP_PC_Vx.x.x.x.rar组件包中，提供如下的工具链：

- Runtime Sample工程（software\sample_runtime目录），包含完整的Runtime Sample源代码供开发者学习参考，支持Visual Studio或RuyiStudio环境运行。
- 模型包(software\data)：包含若干sample中用到的网络的caffe模型文件及对应的Runtime mapper配置文件、wk文件、图像文件等。
- Windows版IDE(tools\nnie\windows目录)工具 Ruyi Studio，集成Windows版的Runtime mapper和仿真库，用户可以将Runtime Sample工程导入；IDE还集成了

客户自定义插件、runtime dot图等功能，具体参考“5 RuyiStudio工具使用指南”章节。

- 当前runtime_mapper支持在RuyiStudio中使用，支持脱离RuyiStudio单独命令行调用。

4.1.3 开发流程

以Caffe框架上训练的模型为例，NNIE的开发流程如图3-1所示。在Caffe上训练、使用NNIE的mapper工具转化都是离线的。通过设置不同的模式，mapper将*.caffemodel转化成在仿真器、仿真库或板端上可加载执行的数据指令文件。一般在开发前期，用户可使用仿真器对训练出来的模型进行精度、性能、带宽进行初步评估，符合用户预期后再使用仿真库进行完整功能的仿真，最后将程序移植到板端。

4.1.4 网络层分类

对于Runtime而言，一个网络的层可分为如下的3类：

- NNIE-Layer：NNIE支持层，可见3.1.3 网络层的分类小节，其中标准层、扩展层都属于NNIE支持的层。
- Runtime-Layer：由Runtime默认提供的运行在非NNIE芯片上的层，如Faster RCNN/RFCN网络中的Proposal层等。
- Custom-Layer：NNIE和Runtime均不支持的层，需要由用户自行编码实现。

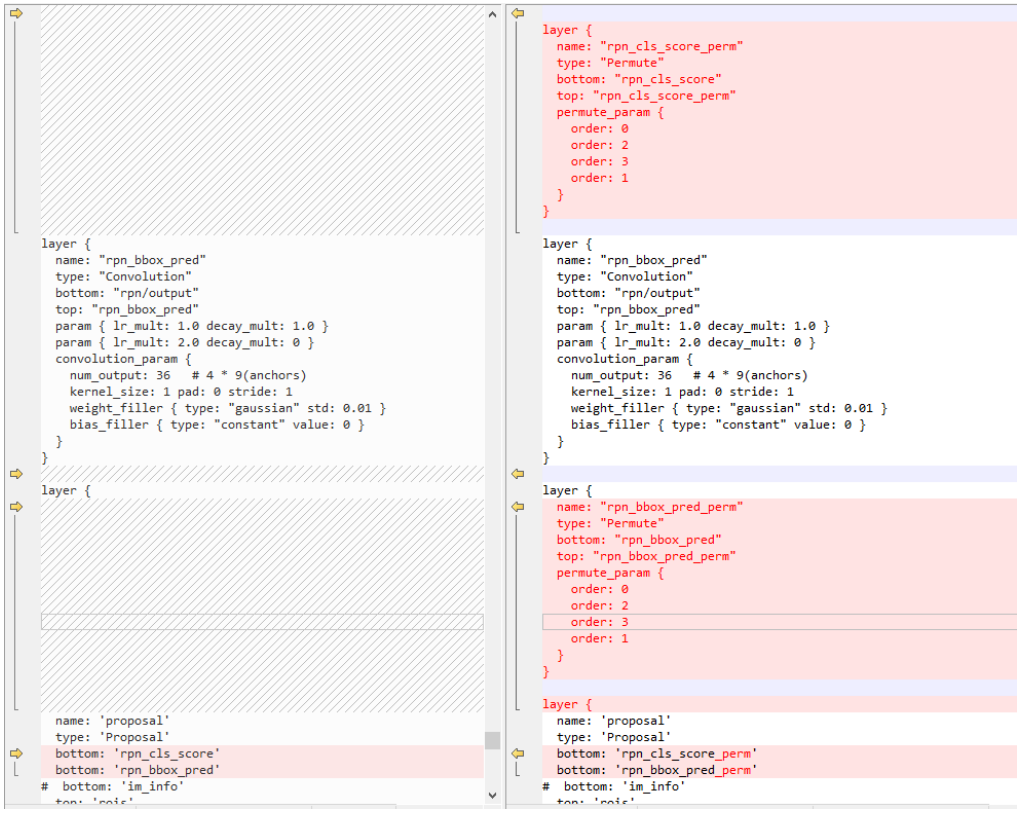
4.1.5 Runtime Layer 规格

Runtime-Layer默认支持层：

Faster RCNN/RFCN网络模型内使用的Proposal层，CPU实现，代码开源。

Layer Type	Chip Type	Support Net	Restrict
Proposal (with Permute) (without Permute)	CPU	Faster RCNN RFCN	1. 支持图3-30FRCNN网络（不支持图3-31） 2. 支持图3-33 RFCN网络（不支持图3-32）

With Permute和Without Permute区别为Permute由CPU/NNIE实现，对应的prototxt差异为（左边with，右边without）：



4.2 状态迁移

为了方便管理及客户使用，Runtime提供了统一的状态管理。

4.2.1 各状态描述

- INITING：用户调用Runtime初始化接口后的状态，表示Runtime正在初始化中；
- INIT：Runtime初始化结束后的状态；
- LOADING：用户调用Runtime加载模型接口后的状态，表示Runtime正在加载模型组；
- LOADED：表示Runtime加载模型组结束；
- UNLOADING：用户调用Runtime卸载模型接口的状态，表示Runtime正在卸载模型组；
- DEINITING：用户调用Runtime去初始化接口后的状态，表示Runtime正在去初始化；
- DEINIT：表示Runtime去初始化结束。

4.2.2 各状态迁移

图 4-2 Runtime 状态迁移图

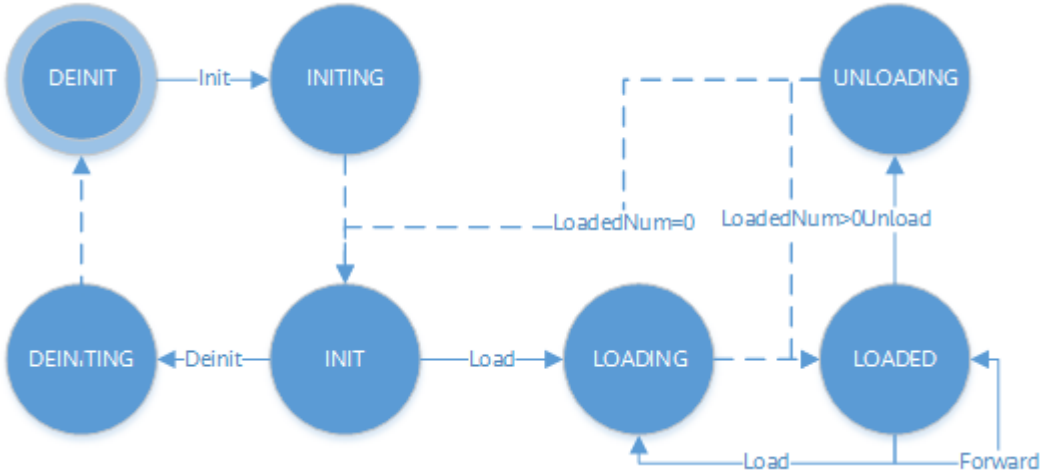


图4-2中实线箭头上的文字表示调用相关API接口，虚线箭头表示源状态是一个临时状态，由Runtime内部完成状态迁移，从图中可以看出各状态的迁移关系。

需要注意如下几点：

- 所有状态为全局状态；
- 只有在INIT和LOADED状态才可以Load模型，所以不支持多个模型组同时Load，如果需要Load多个模型组，需要错开Load操作的时间；
- 只有在LOADED状态才能进行Forward，所以一个模型组正在Forward的时候需要Unload另外一个模型组，则需要等待当前模型组的forward操作结束。

4.3 插件库管理

Runtime-Layer和Custom-Layer均以动态库的方式呈现，在网络运行时实时加载。

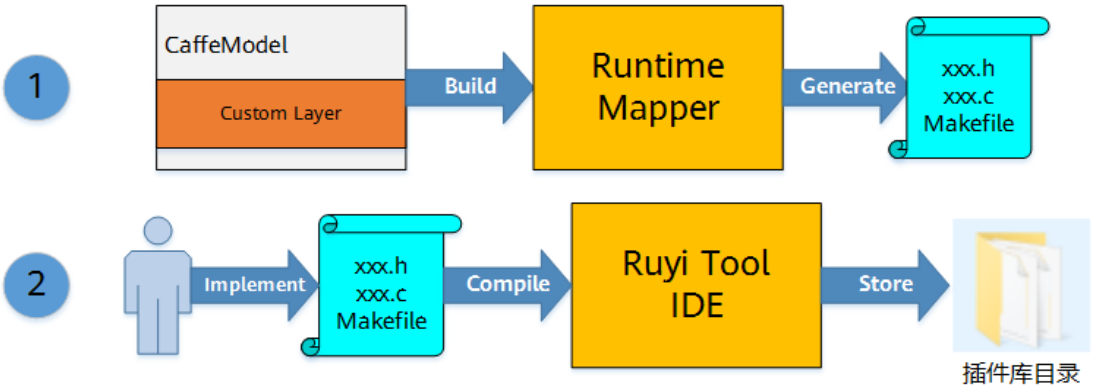
在Ruyi工具生成WK流程中的标记界面即需要进行Runtime-Layer和Custom-Layer动态库的注册。

4.3.1 提供 Custom-Layer 代码实现模板

Ruyi工具在识别到客户网络模型中存在Custom Layer时，会自动生成函数实现模板，方便客户进行算法实现；

客户实现具体算法后，通过Ruyi工具编译成插件库，并存放至插件库目录中。

图 4-3 插件库管理步骤图



4.3.2 提供 Custom-Layer 插件库复用功能

Runtime支持相同Layer Type的Custom-Layer动态插件库复用：

- 提供指定插件库路径功能，客户可选择本机PC中含有之前已实现Custom Layer插件库的目录作为后续复用库查询目录；或者客户可以将之前已实现插件库放至默认插件库查询目录中。
- Ruyi工具识别到Prototxt中含有Custom-Layer时，通过Custom Layer Type自动查询插件库目录下是否含有相同Type的库：
  - 存在：自动关联其中某一个库，且客户依旧可以选择其他库或新建实现。
  - 不存在：提示客户新建实现。

4.3.3 插件库限制

当前暂不支持带有训练参数的层作为Runtime-Layer和Custom-Layer。

如果Custom-Layer带有训练参数，需要客户自编码获取训练参数值做处理。

4.3.4 插件库板端部署

将RuyiStudio提供的自定义算子工程中的src文件夹拷贝到板端mpp/sample/svp/中的hirt/plugins/目录下，请根据自行需要修改文件名。

根据SDK相应版本设置Makefile中的CC和LD，设置完执行make编译即可。

将编译生成在当前目录下的so文件拷贝到hirt目录，或者参考**4.9.1 模型组全局配置项**，设置custom_plugin_lib_path即可。

4.3.5 客户自定义插件

请到mpp/sample/svp/目录中的hirt/plugins/目录，拷贝proposal到客户需要的算子编译路径下面，custom_plugin相关设置请参考目录下的readme，设置完执行make即可编译。

插件库的实现在hi_plugin_proposal.c中的Node Plugin Compute函数。

4.4 网络拓扑管理

Runtime支持用户使用prototxt方式灵活组网。

例如Rfcn → Connector → alexnet的网络拓扑结构prototxt可以表示为：

图 4-4 网络拓扑示例图

```
Name: "rfcn_alexnet"
priority: 1

input {
    name: "data"                                    //global input data
}

model {
    name: "rfcn"                                    //rfcn model name
    bottom: { name: "data" }                        //input data of rfcn
    top: { name: "proposal" }                       //report 0: proposal
    top: { name: "cls_prob_reshape" }               //report 1: cls_prob_reshape
    top: { name: "bbox_pred_reshape" }              //report_2: bbox_pred_reshape
    cops {
        name: "proposal"                            //custom op name
    }
}

connector {
    name: "rfcn_conn_alexnet"                       //connector name
    bottom: { name: "data" }                        //input 0: data
    bottom: { name: "rfcn.proposal" }               //input 1: proposal of model rfcn
    bottom: { name: "rfcn.cls_prob_reshape" }       //input 2: cls_prob_reshape of model rfcn
    bottom: { name: "rfcn.bbox_pred_reshape" }      //input 3: bbox_pred_reshape of model rfcn
    top: {
        name: "rect_image"                          //report: rect_image
        type: U8                                            data type of rect_image
        shape: {dim:64 dim:227 dim:227 dim:3}               dimension information
    }
}

model {
    name: "alexnet"                                 //alexnet mode name
    bottom: { name: "rfcn_conn_alexnet.rect_image" }    //input: rect_image of rfcn_conn_alexnet
    top: { name: "prob" }                           //report: prob
}
```

从图4-4看出，model，connector通过bottom，top的连接关系构成一张完整的网络拓扑结构。

当前版本对网络拓扑结构的prototxt主要约束如下：

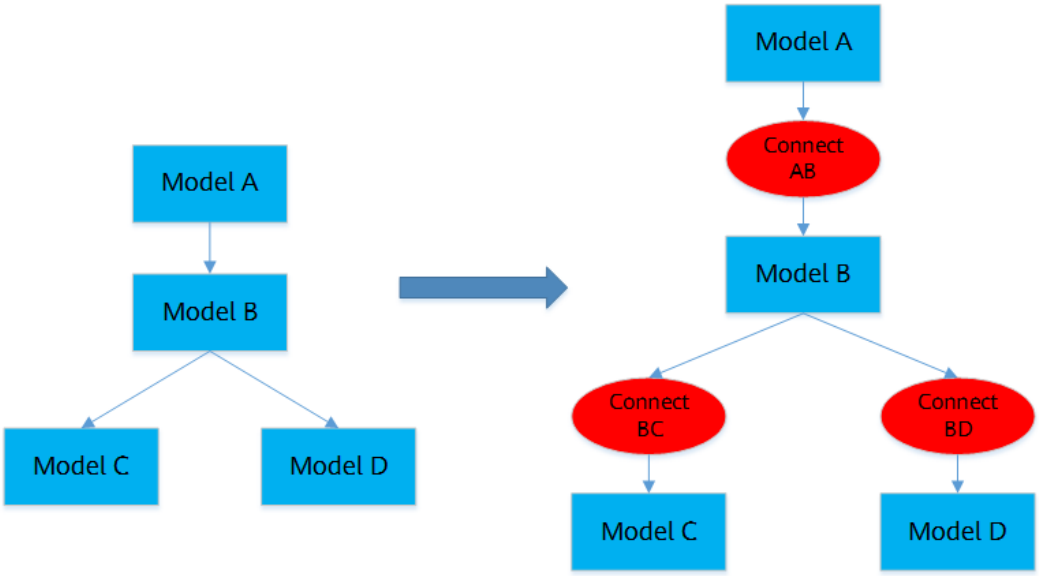
- Model、Connector、Input的名称需要全局唯一。
- bottom为Input输入，其要求如下：
  - 不要求和离线wk文件中input名称一致。
  - 输入的bottom须顺序对应排列，model中的bottom需要和wk中的输入顺序一致，connector需要和函数注册的输入参数顺序一致。
  - 上接input层时，其名称直接为input name，例如：bottom: { name: "data" }
  - 上接非input层时，其名称为上接Model/Connector的名称加上需要使用的report节点名称，例如：bottom: { name: "rfcn.proposal" }
- top为上报节点，其要求如下：
  - 不同模型的top名称可以相同。
  - 必须和离线wk文件中的output名称一致。
  - 输出的top须顺序对应排列，model中的top需要和wk中的输出顺序一致，connector需要和函数注册中的输出参数顺序一致。
- 当前支持model、connector、priority、name、bottom、top、cops、type、shape、frame_depend、max_tmpbuf_size_mb(当前只支持4G以下)等配置选项，具体请参考sample代码。

4.5 级联网络支持

Runtime支持级联网络inference，相同组名标识的网络模型为一个模型组。

- 模型组内模型间通过【连接器】进行连接。
- 【连接器】可以和【连接器】相连接。
- 【连接器】支持多输入多输出。
- 【连接器】可以作为模型组输入。
- 【连接器】为用户自实现代码，以注册函数方式和Model一起在调用LoadModelGroup函数时作为参数传递下来。
- 可通过动态设置【连接器AB】输出中的N维度为0，来控制所有下接模型和连接器不再继续运行，N维度为Runtime Blob结构体中的num来设置。
- 所有的输入输出BLOB内存均由客户进行分配，模型内的辅助内存用户不需要管理。

图 4-5 模型级联图



级联API调用，具体可见《HiSVP API 参考》Runtime章节。

4.6 优先级配置

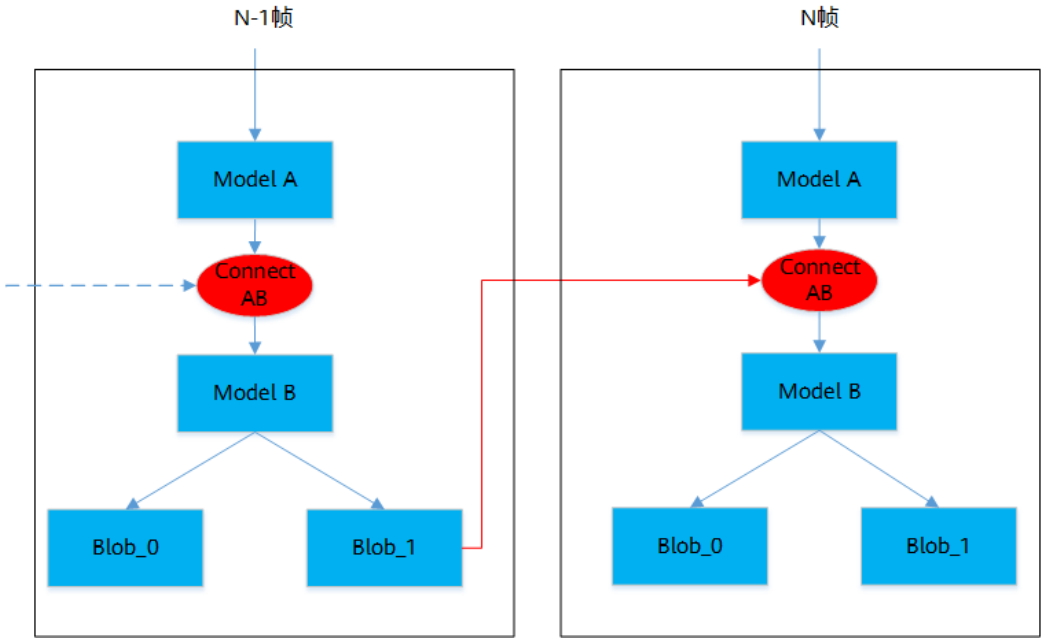
Runtime API支持模型组优先级，按照模型内分段颗粒度，优先分配硬件资源给优先级更高的分段进行运算；优先级相同的分段按照First Come First Serve的方式进行处理。

4.7 帧间依赖网络支持

Runtime API支持帧间依赖网络inference，当前帧的执行需要以前面帧的执行结果为其输入。

- N帧的输入以N-1帧的输出作为其输入，形成一种帧与帧之间相互依赖的网络结构；
- 典型的跟踪网网络结构。

图 4-6 帧间依赖网络示意图



4.8 全局参数可配置

Runtime初始化时，用户可以根据使用环境配置Runtime相关全局参数，用户可以配置如下信息：

- runtime监听线程使用的cpu亲和度
- 客户自定义插件以及connector对象使用的cpu亲和度
- 日志级别
- PROC开关

4.8.1 线程使用的 CPU 亲和度可配置

runtime监听线程、插件库以及连接器对象(CPU上运行)都运行在CPU线程中，用户可以通过配置CPU线程达到性能的最优化。

runtime支持客户配置CPU线程属性，可以通过配置的项如下：

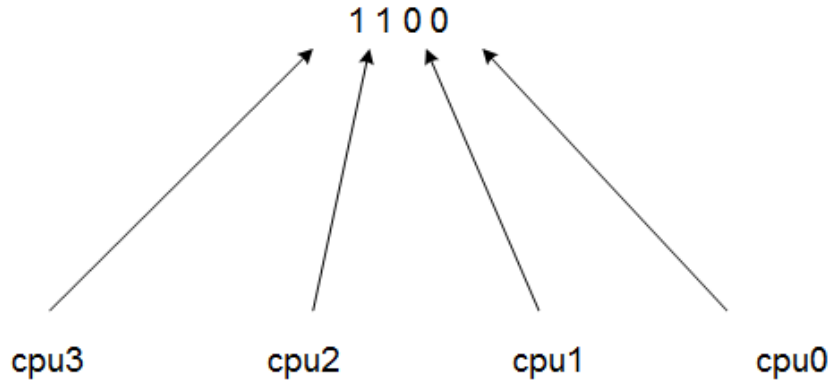
- CPU的个数
- CPU线程在哪个CPU核上运行

说明

对于非运行在CPU上的插件库和连接器，此配置无作用。

- CPU线程配置说明

图 4-7 CPU 配置示意图



多个CPU组成一个二进制xxxx结构，CPU0对应最右边的一位，CPU1对应前一位，依此类推；1代表线程运行此CPU核上，0代表线程不运行在此CPU核上。

CPU线程也可以指定使用的CPU核的范围，比如6，即0110，代表此CPU线程可以运行在CPU1或者CPU2上。

- CPU线程配置举例
  ```
  HiRTInit ("cpu_task_affinity:4 cpu_task_affinity:8", NULL);
  ```
  - 两个cpu_task_affinity代表启动两个CPU线程
  - cpu_task_affinity设置为4、8
    4对应0100，代表使用CPU2
    8对应1000，代表使用CPU3

4.8.2 日志级别可配置

有五种级别：

- VERBOSE
- DEBUG
- INFO
- WARNING
- ERR

日志级别优先级高低：ERR>WARNING>INFO>DEBUG>VERBOSE

不配置日志级别的情况下，默认级别为INFO。

系统运行时，会打印出设置的级别及以上更高级别的所有打印。

例如：

```
HiRTInit ("log_level：WARNING cpu_task_affinity:8", NULL);
```

log_level为WARNING，代表设置的级别为WARNING，则系统运行时，会打印WARNING和ERR的日志信息。

4.8.3 PROC 开关

不设置PROC开关的时候，默认不开启PROC；

如果需要开启PROC，在HiRTInit的globalSetting中设置"proc_enable:true"即可。

例如：

```
HiRTInit ("proc_enable:true", NULL);
```

4.8.4 插件库路径可配置

用户自定义层对应库或者Runtime默认提供的自定义库，用户可以选择放入自行定义的目录下。

例如：

```
HiRTInit("log_level：WARNING cpu_task_affinity:8 custom_plugin_lib_path:\"/tmp/test/runtime_test/\"", NULL);
```

代表用户自定义库的路径为/tmp/test/runtime_test

4.8.5 全局参数解析约束

- 以关键字log_level/cpu_task_affinity/custom_plugin_lib_path进行匹配，自动忽略额外的字符。
- 关键字正确性由用户保证，如果匹配不到按默认值，接口不报错。CPU亲和度默认为随机CPU核，日志级别默认为INFO，插件库路径默认为空。

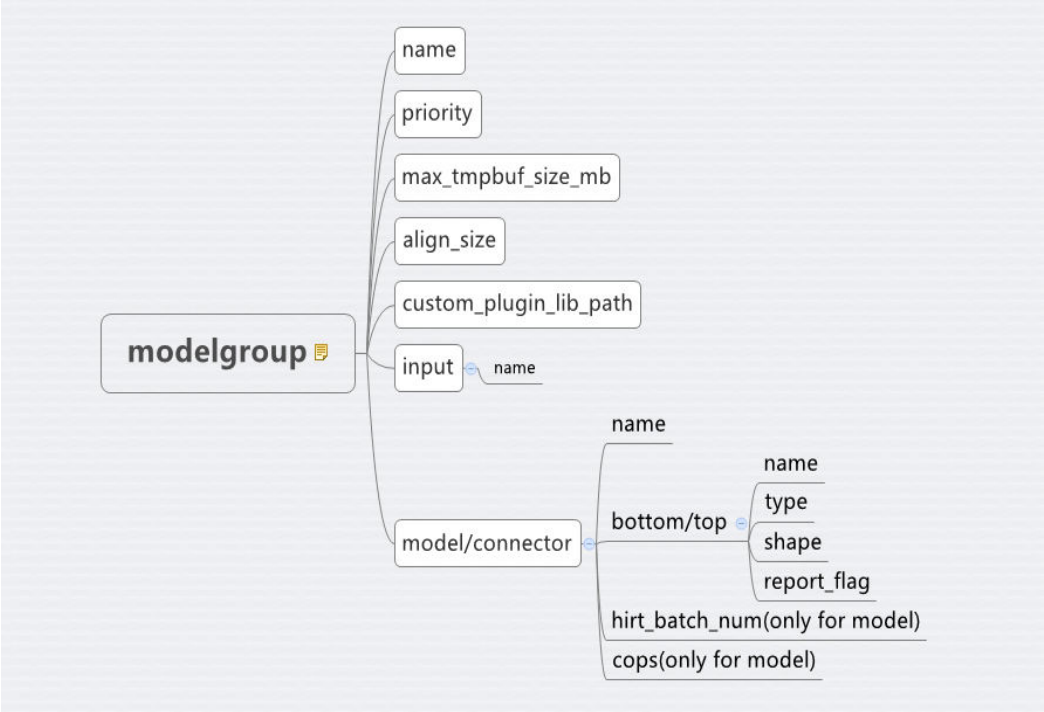
4.9 模型组可配置

Runtime支持传入的模型组可配置，用户可以根据实际需求进行配置。

模型组配置包括：模型组全局配置项、input配置项、model/connector配置项

其配置项关系如下图4-8所示：

图 4-8 Modelgroup 可配置项关系图

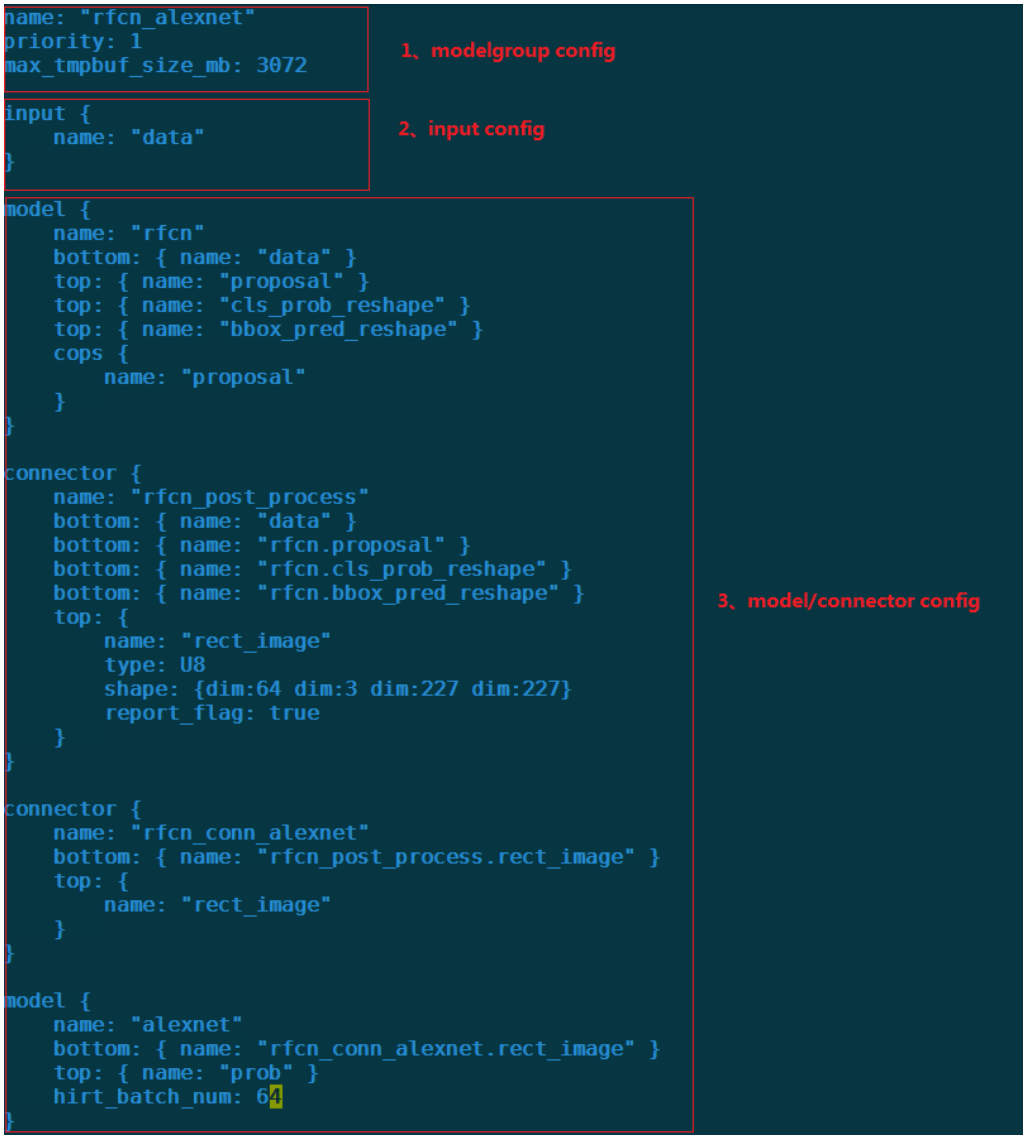


图中，各节点关系如下：

- 模型组全局配置项：name、priority、max_tmpbuf_size_mb、align_size、custom_plugin_lib_path
- input配置项：input
- model配置项：name、bottom、top、hirt_batch_num、cops
- connector配置项：name、bottom、top
- bottom/top：name、type、shape、report

以rfcn+alexnet举例，其modelgroup如图4-9所示：

图 4-9 rfcn+alexnet modelgroup 结构图



图中标注1的部分为模型组全局配置项，标注2的部分为input配置，标注3的部分为model/config配置。

对于custom_plugin_lib_path：

- modelGroup文件中已配置，则HiRTInit中配置的custom_plugin_lib_path不生效。
- modelGroup文件和HiRTInit中都没有配置，则使用运行时的当前目录。

4.9.1 模型组全局配置项

以下配置只针对模型组有效，其配置表如表4-1所示：

表 4-1 模型组全局配置项表

配置项名称	简要描述	类型	取值范围	默认值	必填
name	模型组名称	字符串	最大长度64	无	是
priority	模型组优先级	整型	[0,4]	无	是
max_tmpbuf_size_mb	模型组配置的最大tmpbuff内存	整型	(0,4096)	1024	否
align_size	模型组的对齐方式	整型	16、32	16	否
custom_plugin_lib_path	自定义插件库存放路径	字符串	取决于系统最长全路径名的长度	无	否

- name：模型组名称，其名称对于不同的模型组是唯一的。
- priority：模型组优先级，优先级高低顺序为：0 > 1 > 2 > 3 > 4，0是最高优先级，4是最低优先级
- max_tmpbuf_size_mb：模型组配置最大tmpbuff内存，单位MB。
- align_size：模型组的对齐方式，支持16，32字节对齐；

当模型组的align_size和wk中的align_size冲突时，系统会报错。

4.9.2 input 可配置项

input只包含name字段，其描述如表4-2所示：

表 4-2 input 可配置项

配置项名称	简要描述	类型	取值范围	默认值	必填
name	用户数据输入节点	字符串	最大长度64	无	是

对同一个模型组文件，各个input的名称必须是唯一的。

4.9.3 模型、连接器可配置项

模型、连接器可配置项包括：name、top、bottom信息。其中，name字段和“4.9.2 input可配置项”章节input的name字段的约束一致。

4.9.3.1 top、bottom 信息包含的数据结构

其数据结构如表4-3所示：

表 4-3 bottom/top 数据结构

配置项名称	简要描述	值信息	必填
name	top/bottom 名称	字符串，最大长度129	是
type	数据类型	见《HiSVP API 参考》Runtime章节中数据类型中的Bolb Type， 只写最后一个下划线后的部分即可，如 HI_RUNTIME_BLOB_TYPE_U8，只写U8	否
shape	shape信息	包含n、c、h、w四个维度	否
report_flag	report标记	true、false	否

- name：bottom/top名称，最大长度：129，
  - 如果上接input层，则直接填入input节点的名称
  - 如果上接model，则填入model name + ”.” + top name(model)的格式
  - 如果上接connector，则填入connector name + ”.” + top name(connector)的格式
  - 名称不唯一。
- type：指定top，bottom的数据类型
  - bottom：不需要填入type信息
  - top：如果下接connector，未下接model，则必须填入type信息，如图中的coneector否则，可以不填type信息
- shape：包含四个维度信息，包含n、c、h、w四个维度，各维度为正整型，例如：
  - shape: {dim:64 dim:3 dim:227 dim:227}，shape的各个dim值的大小>=1
  - bottom：不需填入shape信息
  - top：如果下接connector，未下接model，则必须填入shape信息，否则，可以不填shape信息
- report_flag：标志此top是否为report层。
  - true：表示report层，需要客户分配相关内存；
  - false：内存由runtime内部分配；

  以下情况需要设置report_flag为true：
  - 用户需要关注的Blob

- Blob在modelgroup中为叶子节点(无下接节点)
- 帧间依赖的连接点

4.9.3.2 模型、连接器可配置项的信息

其配置项如表4-4所示：

表 4-4 模型、连接器可配置项

配置项名称	简要描述	取值范围	必填
name	model/connector名称	字符串 最大长度64	是
bottom	依赖的数据输入信息	请参考bottom结构	否
top	输出输出信息	请参考top结构	否

- name：名称，最大长度：64，对于模型组内的模型、连接器，其名称必须为唯一。
- bottom/top：请参考“4.9.3.1 top、bottom信息包含的数据结构”章节。

4.9.3.3 模型独有配置

- hirt_batch_num: 正整型，模型配置的最大batch数目。

  0 < =hirt_batch_num <= 256

  - 不设置hirt_batch_num或者hirt_batch_num =0，runtime以wk内部的batch number进行tmpbuff分配。
  - 设置了hirt_batch_num，runtime会取hirt_batch_num和wk中的batch number两种最小值进行tmpbuff分配。
  - 用户需要保证hirt_batch_num大于实际输入batch数。
- cops: 标志模型内部的cop信息，例如proposal如下：

```
model {
    name: "rfcn"
    bottom: { name: "data" }
    top: { name: "proposal" }
    top: { name: "cls_prob_reshape" }
    top: { name: "bbox_pred_reshape" }
    cops {
        name: "proposal"
    }
}
```

  标记proposal为custom op

  这个例子中，proposal即为custom op，又为上报层。

4.9.4 modelgroup 文件解析约束

- 配置以关键字匹配，自动忽略额外的字符，例如：

shape:{dim:2 xxxdim:3 dim:4 dim:5}，自动忽略xxx三个字符，得到的shape信息为n=2、c=3、h=4、w=5。

- 所有数字配置只支持10进制，不支持其他进制类型。
- 当关键字只能有一个时，以第一个匹配的为准，例如：
  model {name: "aaa" name: "bbb"}，则匹配到的model name为"aaa"

4.10 Sample 说明

4.10.1 图片 Resize

检测网输出的目标通常需要resize成分类网要求的分辨率后再输入到分类网进行处理，可以在runtime的connector中实现图片的Resize。Sample中resizeROI接口提供了两种Resize的实现：板端使用IVE；仿真使用OpenCV。目前相同的输入使用IVE和OpenCV得到的结果会有细微差异，这是由硬件的定点运算引起的。因此sample中经过resize后板端和仿真的结果无法保证完全一致。

如果需要保证resize结果一致来对比板端和仿真结果，建议的处理方式是：将仿真得到的检测网输出通过板端resizeROI的IVE接口获取resize后的图片保存下来，再输入到仿真的分类网。

4.10.2 模型输出 BLOB 名称

用户编写sample时，对于模型输出BLOB名称，尽量参考编写Runtime wk时生成的dot图信息（dot图中，橙色节点为输出BLOB，即report节点）。

在编写sample时，sample中的top name对应为dot图中的去掉_report后缀的report节点名称; 例如在dot图中的report节点为xxxxx_report，则在sample中该top name为xxxxx。

4.11 性能最佳配置

本章节描述的最佳配置，是单纯考虑runtime的性能最佳配置，客户实际场景业务可能很多，请综合考虑自己的业务进行配置。

4.11.1 CPU 亲和度配置

请参考4.8.1 线程使用的CPU亲和度可配置来配置CPU的亲和性，sample中已经给出了样例可以参考。

4.11.2 配置内存数

可以设计一组测试用例，起N个线程，每个线程开一份内存。从N=1开始，通过逐步增加线程的方式，测试一段时间(例如30分钟以上)的Forward的帧数来计算吞吐量，同时通过PROC观察NNIE的利用率，当NNIE利用率达到最大的时候，即为该模型相对的最佳配置。

5 RuyiStudio 工具使用指南

RuyiStudio集成windows版的NNIE mapper、Runtime mapper和仿真库，具有生成NNIE wk功能、Runtime wk功能和仿真NNIE功能，同时具有代码编辑、编译、调试、执行功能、网络拓扑显示、目标检测画框、向量相似度对比、调试定位信息获取等功能。

RuyiStudio集成windows版的NNIE mapper基于Visual Studio 2015 64bit版本编译，所以其依赖库也需要使用Visual Studio 2015 64 bit进行编译。

RuyiStudio集成仿真库基于MinGW-W64 7.3.0编译，所以其依赖的编译链环境也需要是MinGW-W64。

5.1 RuyiStudio 安装

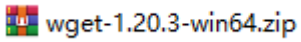
在RuyiStudio工具同目录下会有一个ruyi_env_setup的文件夹，可以进入该文件夹用脚本进行mingw，python及caffe的一键环境配置。本章节提供一键脚本配置方法和手动配制方法。

5.1.1 编译链 MinGW-W64 安装

5.1.1.1 脚本一键安装

步骤1 安装wget，从**https://eternallybored.org/misc/wget/** 网页下载wget压缩包（选择Version 1.20.3），下载得到wget-1.20.3-win64.zip（如**图5-1**所示），将其解压到本地某个目录，建议解压到一个干净的目录下，例如在C:\Program Files下新建一个名为wget的目录，将wget.exe所在目录添加到用户环境变量PATH中，用于从网页下载相关包。建议在用户环境变量中，先创建一个变量名为WGET_PATH的环境变量，变量值即为wget.exe所在路径，也就是C:\Program Files\wget路径，然后再将%WGET_PATH%添加到用户环境变量PATH中。

图 5-1 wget 安装可执行文件



步骤2 点击ruyi_env_setup文件夹下的setup_mingw.bat，就可以自动下载mingw和msys的安装包，解压到指定位置并配置环境变量。

须知

使用代理服务器访问外部网络的用户需配置wget代理。

----结束

5.1.1.2 手动安装

步骤1 进入官方下载网址：https://sourceforge.net/projects/mingw-w64/files/mingw-w64/，选择7.3.0的x86_64-posix-seh版本下载（此版本支持c++11特性）。

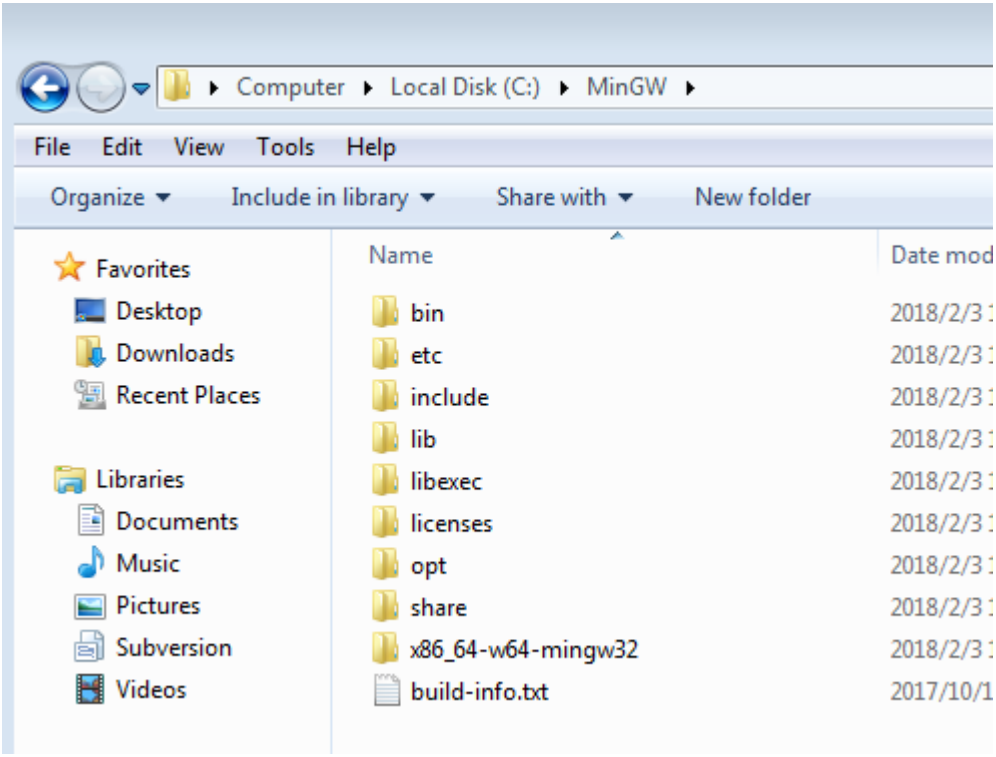
须知

如果安装配置了其他的mingw版本则sample工程下仿真库依赖的opencv库功能可能会异常，因其opencv库是在指定版本上编译出来的。

图 5-2 MinGW-w64 release 版本

下载后解压到安装目录，注意后缀是.7z的压缩格式，推荐C盘根目录。解压后的目录如图5-3所示，后面步骤的路径均以下图路径进行说明。

图 5-3 MinGW-w64 解压后的目录结构

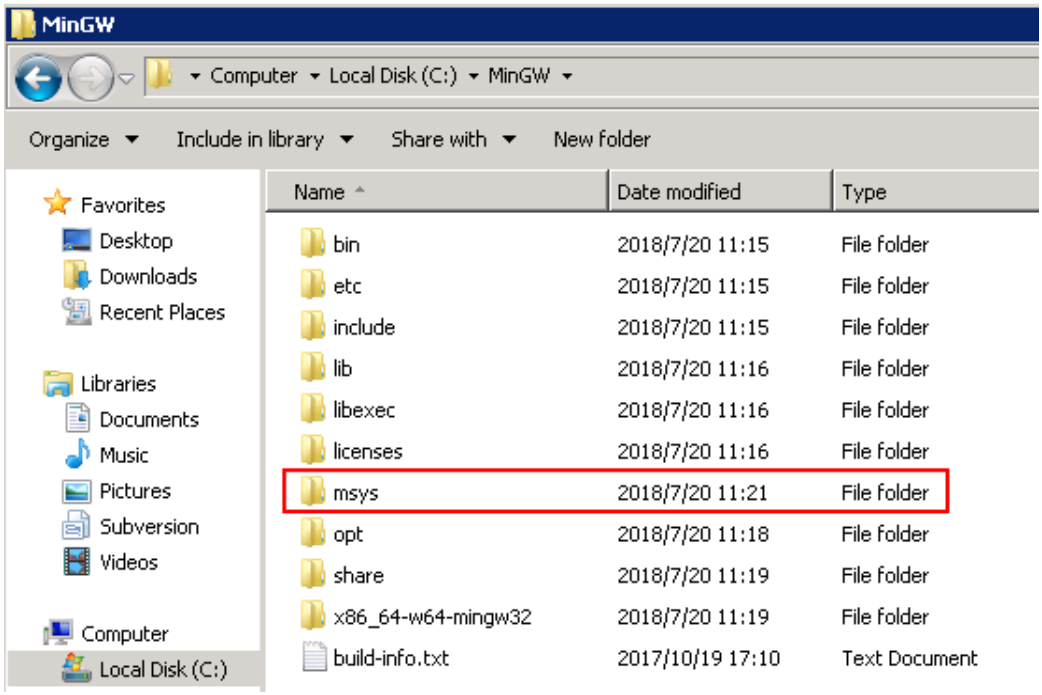


步骤2 从**https://sourceforge.net/projects/mingwbuilds/files/external-binary-packages/**选择**msys+7za+wget+svn+git+mercurial+cvs-rev13.7z**进行下载，msys中会提供用于清理Ruyi工程缓存时用到的rm.exe。

图 5-4 Msys 版本

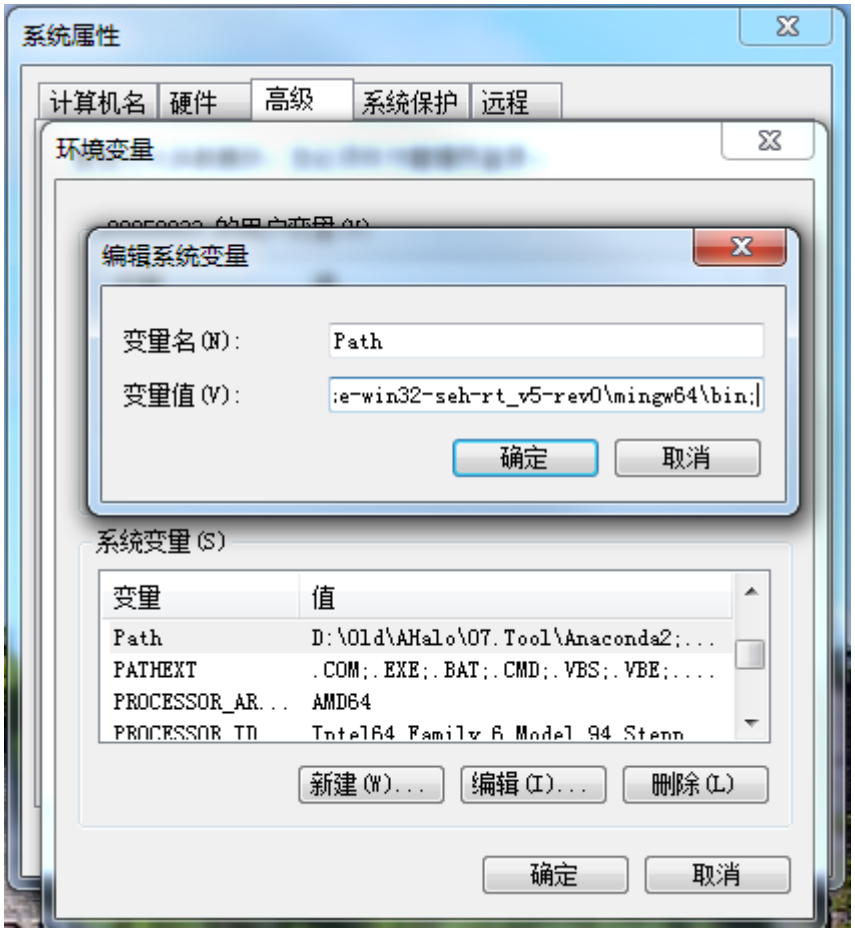
步骤3 将下载得到的**msys+7za+wget+svn+git+mercurial+cvs-rev13.7z**解压到MinGW文件夹下

图 5-5 解压的 msys 放置的位置



步骤4 设置环境变量，计算机上点击右键->属性->高级系统设置->环境变量，在系统变量中添加以下变量（若已有同名变量则添加变量值用分号隔开）：变量名Path，变量值MinGW的bin路径和msys的bin路径C:\MinGW\bin; C:\MinGW\msys\bin，如图5-6所示。

图 5-6 系统环境 MinGW 环境变量设置



步骤5 在C:\MinGW\bin目录下，将x86_64-w64-mingw32-gcc.exe再拷贝一份并重命名为mingw32-gcc.exe，否则RuyiStudio工具不能自动识别到MinGW工具链。

步骤6 重新启动计算机。

----结束

5.1.2 Python3.5+caffe 环境配置

5.1.2.1 脚本一键安装

步骤1 同5.1.1.1 步骤1

步骤2 从**https://github.com/willyd/caffe-builder/releases** 下载libraries_v140_x64_py35_1.1.0.tar.bz2 并放置在ruyi_env_setup 文件夹下。

图 5-7 libraries_v140_x64_py35_1.1.0.tar.bz2

步骤3 点击setup_python.bat进行安装，setup_python.bat脚本会调用python_bat文件夹下的setup_download_python.bat、setup_extract_python.bat和setPath_python.bat，下载python，caffe的依赖包，并配置环境变量。如果环境变量配置失败（如环境变量超过1024个字符），或者在win10系统下配置系统环境变量，可参考**5.1.2.2 手动安装**的描述，进行手动配置。

须知

使用代理服务器访问外部网络的用户需配置wget代理。

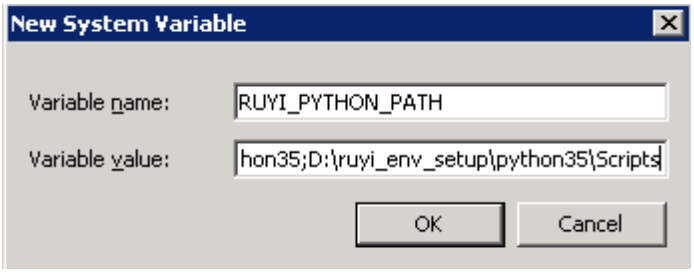
步骤4 重新启动计算机。

----结束

5.1.2.2 手动安装

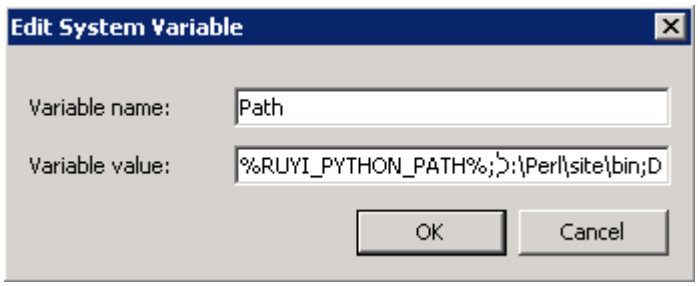
步骤1 在系统环境变量中增加变量RUYI_PYTHON_PATH，如果当前bat脚本放置在D:\ruyi_env_setup目录下，则变量值为D:\ruyi_env_setup\python35; D:\ruyi_env_setup\python35\Scripts; D:\ruyi_env_setup\pyhon35\Library\bin，如**图5-8**所示。

图 5-8 RUYI_PYTHON_PATH



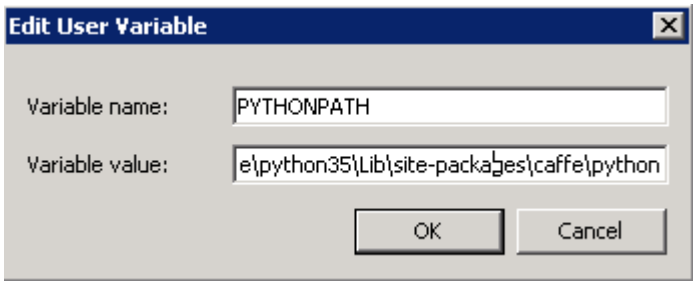
步骤2 添加%RUYI_PYTHON_PATH%到系统环境变量path的开头。

图 5-9 Path



步骤3 手动添加用户环境变量PYTHONPATH，D:\ruyi_env_setup\python35\Lib\site-packages\caffe\python，用于工具识别到caffe。

图 5-10 PYTHONPATH



步骤4 参照requirements.txt中给出的版本号下载相关的包。除过caffe.zip、opencv_python-3.4.2.16-cp35-cp35m-win_amd64.whl和protobuf-3.6.1-cp35-cp35m-win_amd64.whl（详见步骤 5和步骤6），其余包都是直接放到脚本创建的python35文件夹下进行解压。

步骤5 caffe.zip需要放到python35\Lib\site-packages下并解压，如图5-11所示，然后解压libraries_v140_x64_py35_1.1.0.tar.bz2，将libraries\bin目录下的caffehdf5.dll，caffehdf5_hl.dll，caffezlib1.dll，glog.dll，libgcc_s_seh-1.dll，libgfortran-3.dll，libopenblas.dll，libquadmath-0.dll，VCRUNTIME140.dll，libraries\x64\vc14\bin目录下的opencv_core310.dll，opencv_imgcodecs310.dll，opencv_imgproc310.dll以及libraries\lib下的boost_chrono-vc140-mt-1_61.dll，boost_filesystem-vc140-mt-1_61.dll，boost_python-vc140-mt-1_61.dll，boost_system-vc140-mt-1_61.dll，boost_thread-vc140-mt-1_61.dll，gflags.dll拷贝到python35\Lib\site-packages\caffe\python\caffe文件夹下，如图5-12所示。

图 5-11 解压 caffe.zip 到 caffe 文件夹中

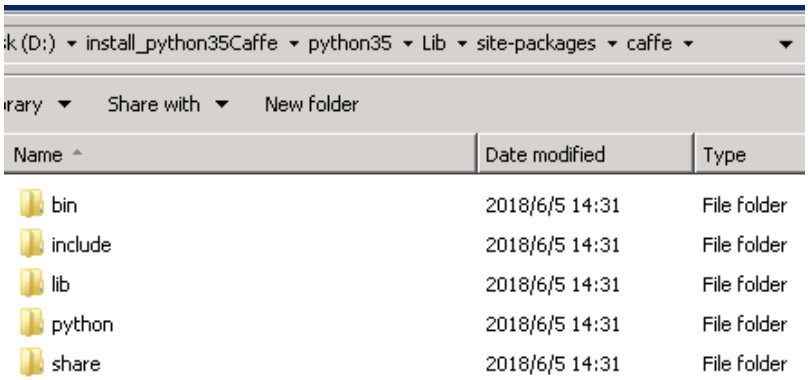
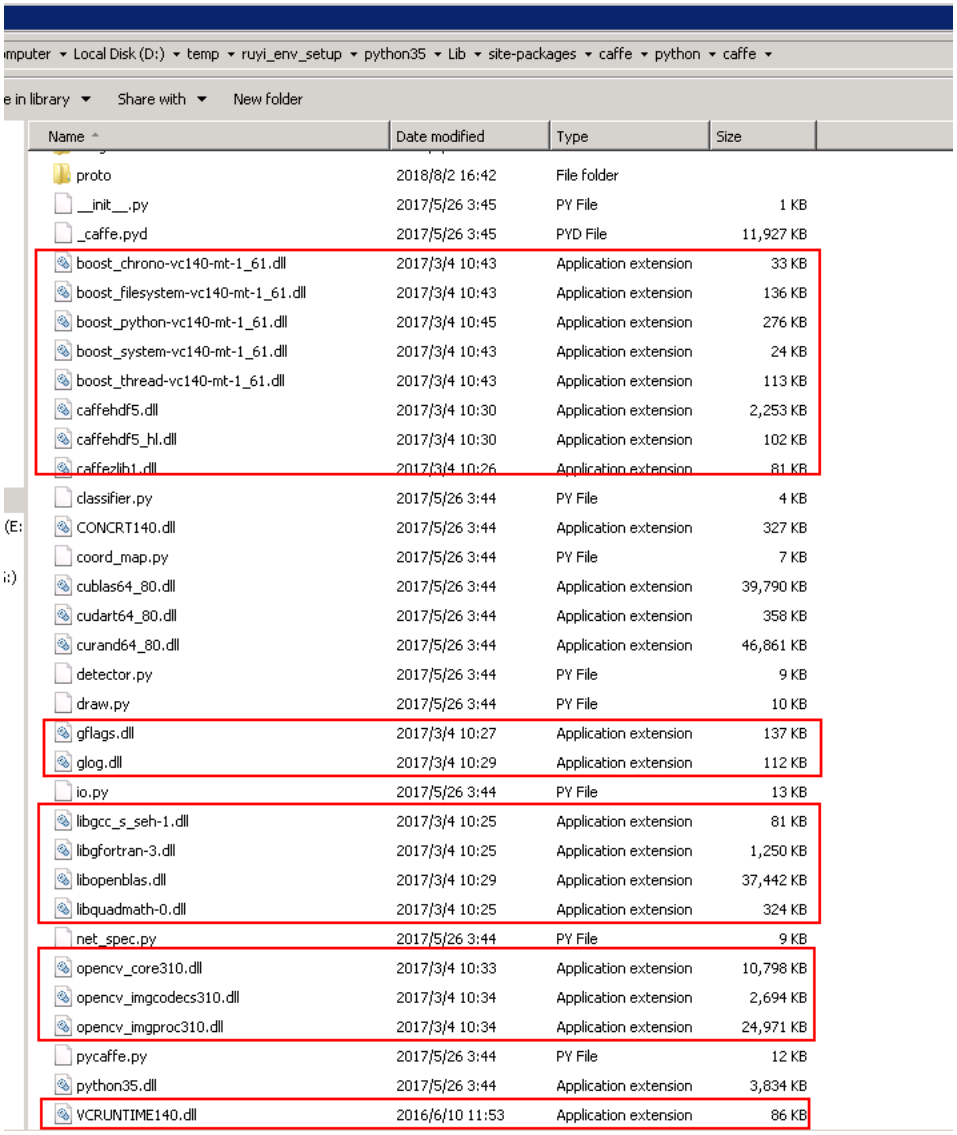


图 5-12 拷贝 libraries 中的库文件到指定文件夹中



步骤6 opencv_python-3.4.2.16-cp35-cp35m-win_amd64.whl和protobuf-3.6.1-cp35-cp35m-win_amd64.whl在python35\Lib\site-packages，需要从cmd进到python35\Lib\site-packages目录，用pip install opencv_python-3.4.2.16-cp35-cp35m-win_amd64.whl和pip install protobuf-3.6.1-cp35-cp35m-win_amd64.whl指令进行安装。

步骤7 当涉及到计算检测网中roi_pooling实现的proposal层的shape信息时或者python层的实现是rpn时，需要客户自行引入相应的模块，才能够识别到。客户需要保证本地机器安装了VS2015，并且保证Visual C++ 2015 Tools for Windows Dektop（Common Tools for Visual C++ 2015和Windows 8.1 SDK and Universal CRT SDK）有正确安装，可以通过能否成功导入sample下的工程识别是否有安装这些库文件。对于win10机器，由于安全限制，需要设置脚本所在的目录权限，选中文件夹，点击右键->属性->安全->编辑>允许完全控制。

步骤8 执行setup_roi_caffe.bat脚本，下载py-faster-rcnn-windows-master.zip包，用VS自带的可执行文件进行编译，并将编译生成的文件拷贝到python35\Lib\site-packages\caffe\roi_pooling文件夹下，由于安装的python脚本的版本号默认是py35，因此相关的python脚本需要遵循python35的规范，需要对脚本进行修改，具体都已经在setup_roi_caffe.bat脚本中做好，可以参考脚本中的内容进行手动操作。

setup_python.bat，setup_mingw.bat，setup_roi_caffe.bat，7z.exe和7z.dll需要放置在同一目录下。

步骤9 重新启动计算机。

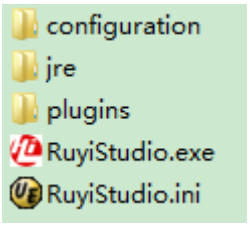
----结束

5.1.3 RuyiStudio 启动方式

步骤1 从SDK包中取到RuyiStudio-X.X.X工具，拷贝到PC上（PC要求安装Win7 64位操作系统或Win10 64位操作系统）的某个本地硬盘，工具中已经集成JRE，无需单独安装。

步骤2 解压RuyiStudio-X.X.X.zip，双击工具目录下的RuyiStudio.exe，打开RuyiStudio工具，如图5-13所示。

图 5-13 打开 RuyiStudio 工具



工具启动界面如图5-14所示。

图 5-14 Ruyi Studio 工具启动界面

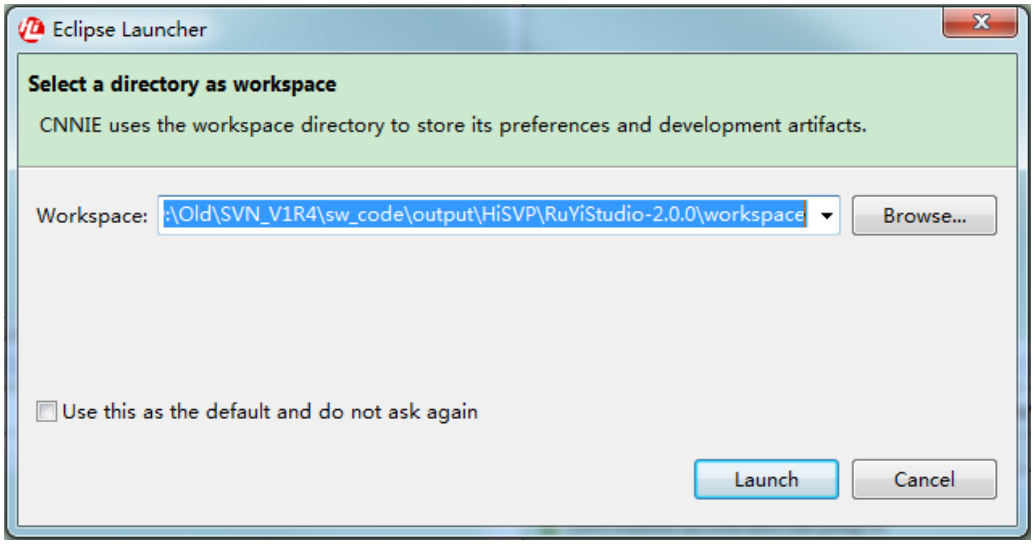
----结束

5.2 RuyiStudio 工具平台公共功能介绍

5.2.1 启动工具设置 workspace

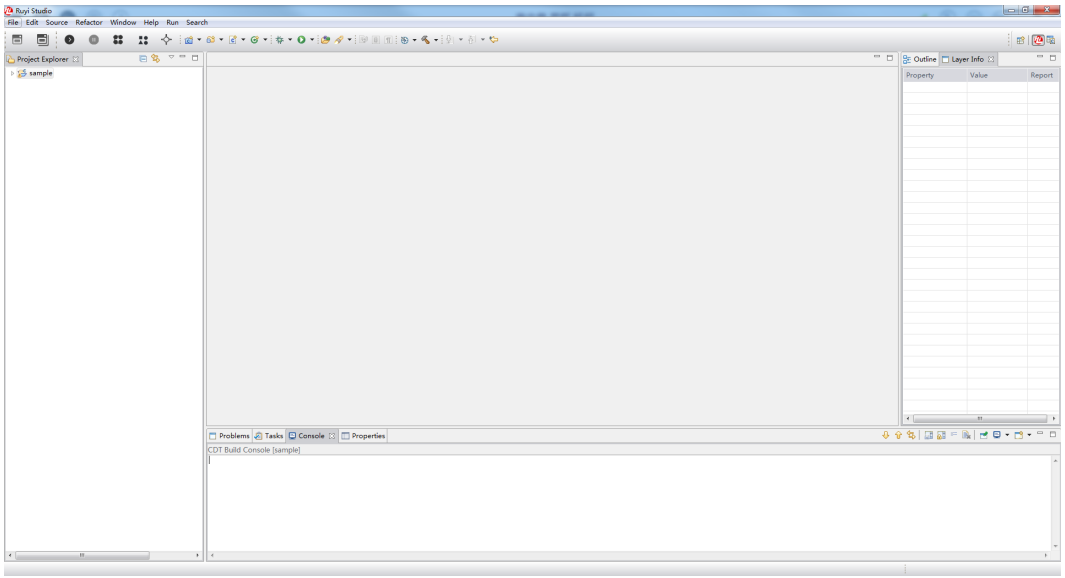
解压RuyiStudio-X.X.X.zip后，双击RuyiStudio.exe，打开工具，选择工作空间，即为选择之后工具中创建的项目的工作空间，如图5-15所示。

图 5-15 选择 workspace 的目录



打开工具后，显示的视图如图5-16，主要由工程视图，主界面视图和控制台视图组成。

图 5-16 工具界面



5.2.2 透视图功能介绍

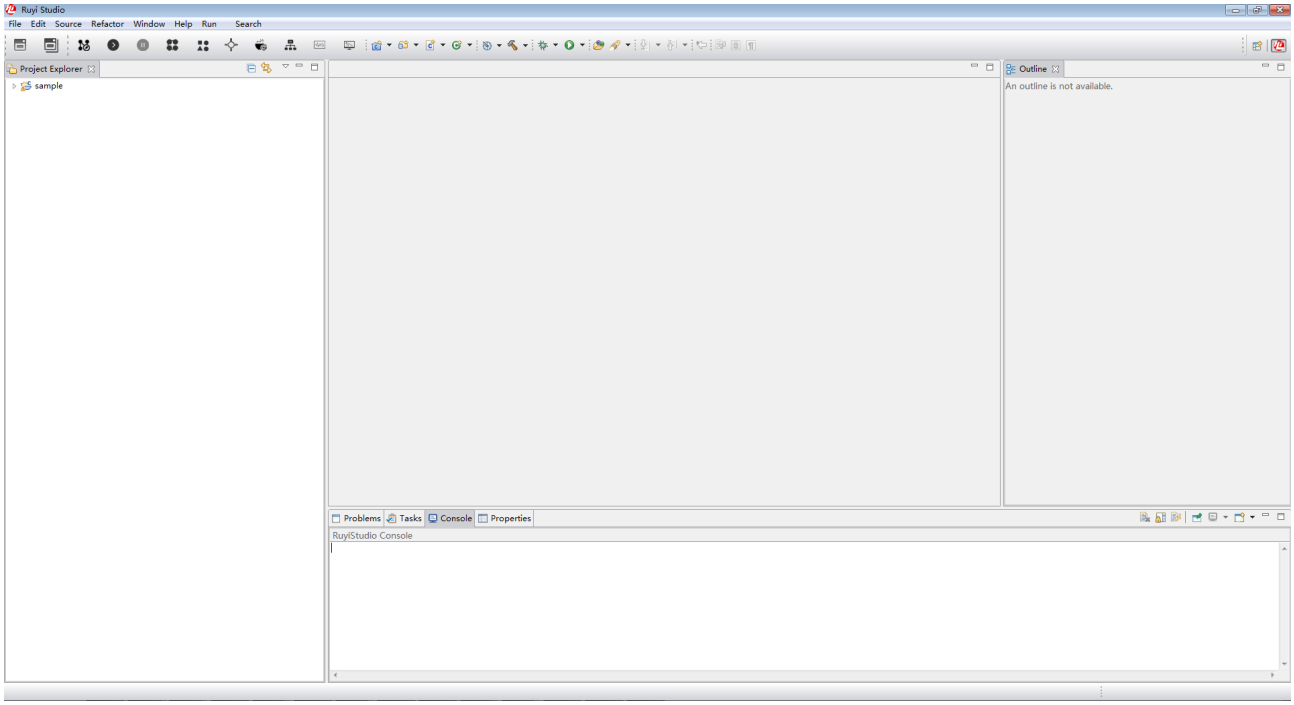
5.2.2.1 Ruyi 透视图

Ruyi透视图是RuyiStudio工具默认的透视图，在工具中右上角，如图5-17，默认选中Ruyi透视图。

图 5-17 选择 Ruyi 透视图

Ruyi透视图是NNIE Make wk专用的透视图，包括RuyiStudio自定义的功能：Make WK功能  ，终止Make WK功能  ，Graph View功能  ，Detection Result功能  ，Vector Comparison统计向量比较功能  ，Get Caffe Output 输出caffe的中间结果  ，Restore Network 还原网络  ，Simulator Performance 性能仿真结果展示  。除此之外，Ruyi透视图还包含了C/C++工程需要使用的Build功能  ，Debug调试功能  ，Run运行功能  。Ruyi透视图默认视图如图5-18。

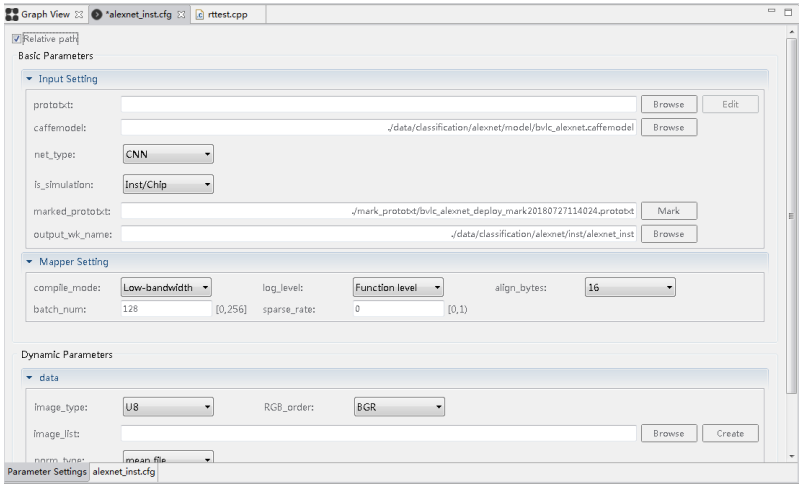
图 5-18 Ruyi 透视图



整个透视图包含4个视图区域：

- 工具左侧为Project Explorer视图，用于显示用户创建或导入的各种工程；
- 下侧为各种信息视图，用于显示工具的问题，后台任务列表，软终端与对象属性；
- 右侧含有Outline视图，在进行C++代码编辑时，显示当前C++文件或头文件的结构及组成部分；还有LayerInfo视图，在标记prototxt文件后，可以显示网络的层属性。
- 中间为工作区，打开的代码文件、cfg文件，DOT图视图等其他工具功能视图均出现在此处。

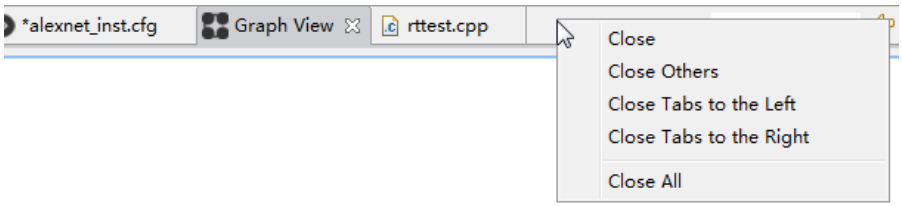
图 5-19 工作区视图



任一视图区域及其中的视图均可执行以下操作：

- 点击最小化按钮，可以将当前视图区域最小化到最近的边栏；
- 点击最大化按钮，可以将当前视图区域最大化，几乎占满整个窗口。其他的视图区域将全部最小化到最近的边栏；
- 点击视图标签上的关闭按钮（对于非选中状态的视图标签，需要鼠标悬停后按钮才会出现）可以关闭对应的视图；
- 拖动视图标签可以将选中的视图移动到别的视图区域；
- 在视图区域标签栏的空白部分点击右键，会弹出关闭标签页的选项菜单：

图 5-20 视图关闭选项



- Close：关闭当前处于选中状态的视图（区域内有视图时才会出现）；
- Close Others：关闭当前未处于选中状态的视图（区域内的视图多于1个时选项才出现）；
- Close Tabs to the Left：关于选中视图左侧的所有视图（选中视图不是最左边的一个视图时选项才出现）；
- Close Tabs to the Right：关闭选中视图右侧的所有视图（选中视图不是最右边的一个视图时选项才出现）；
- Close All：关闭所有视图（区域内的视图多于1个时选项才出现）；

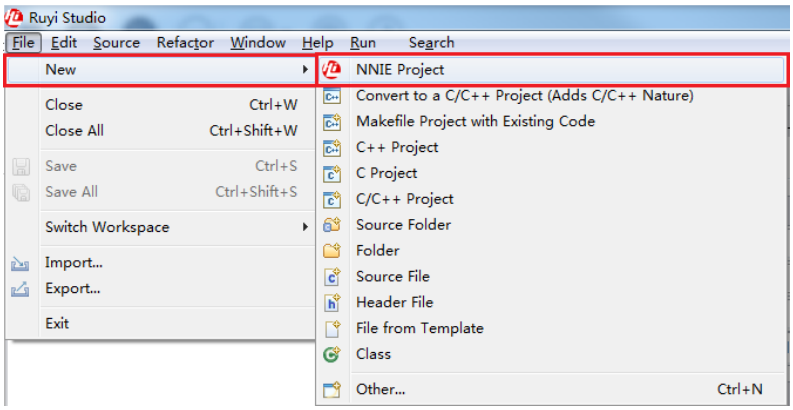
5.2.3 创建工程

5.2.3.1 创建过程

步骤1 工程创建，可以采用以下任一方式进行：

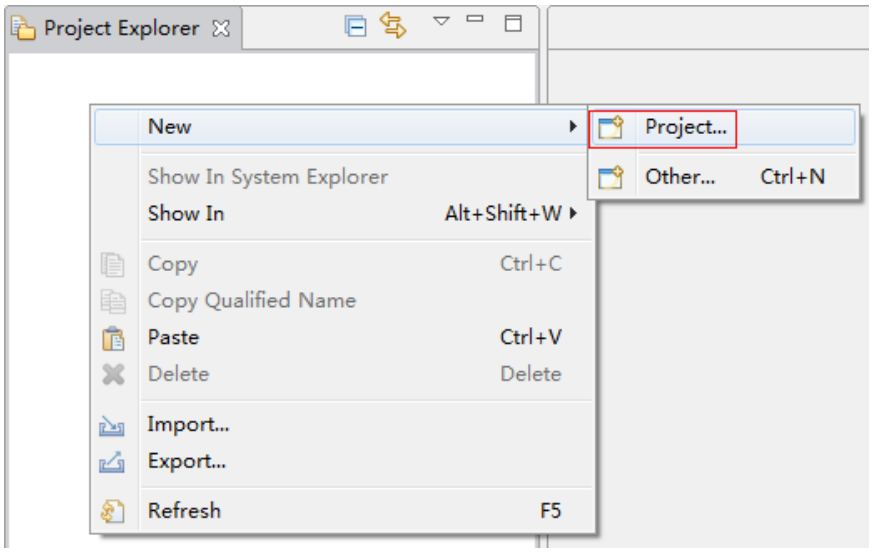
依次点击File → NEW → NNIE Project，如图5-21所示。

图 5-21 通过 File 菜单创建 Project



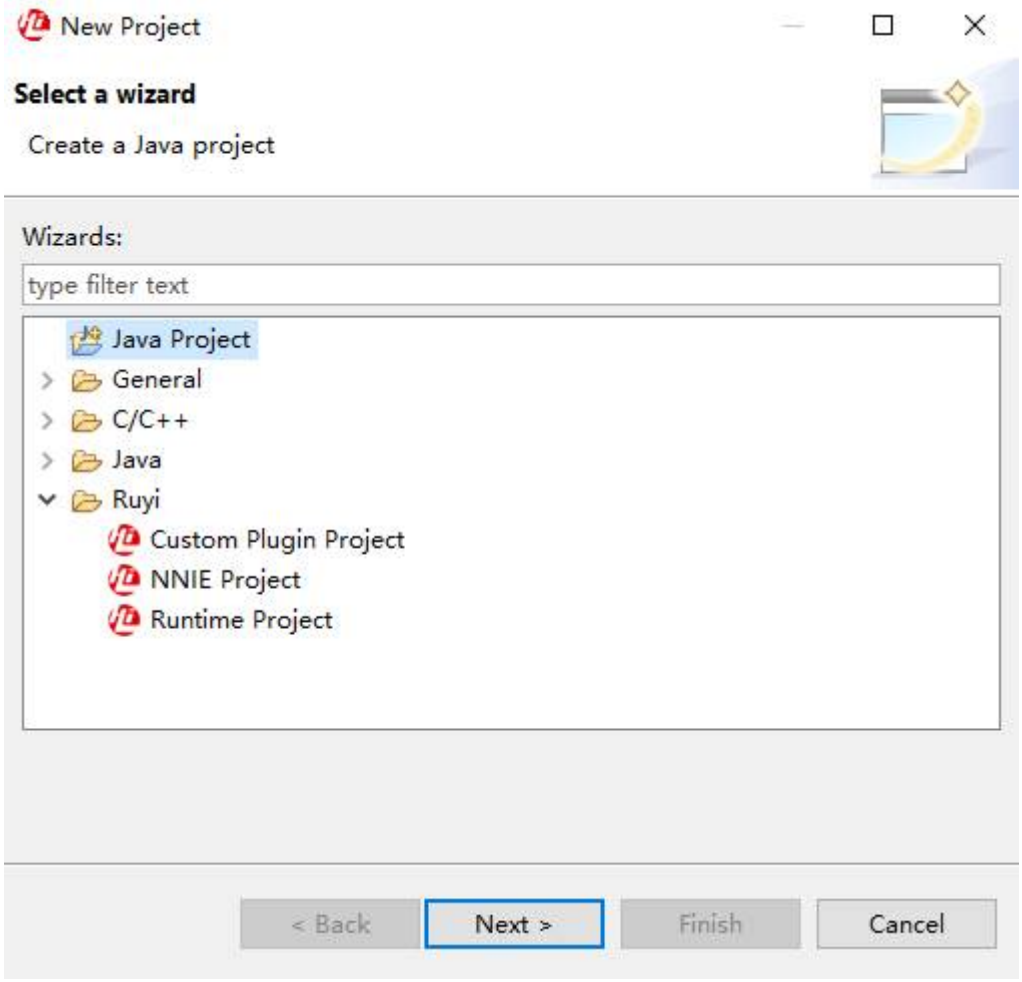
通过Project Explorer视图中鼠标右键创建Project，如图5-22所示。

图 5-22 通过鼠标右键创建 Project



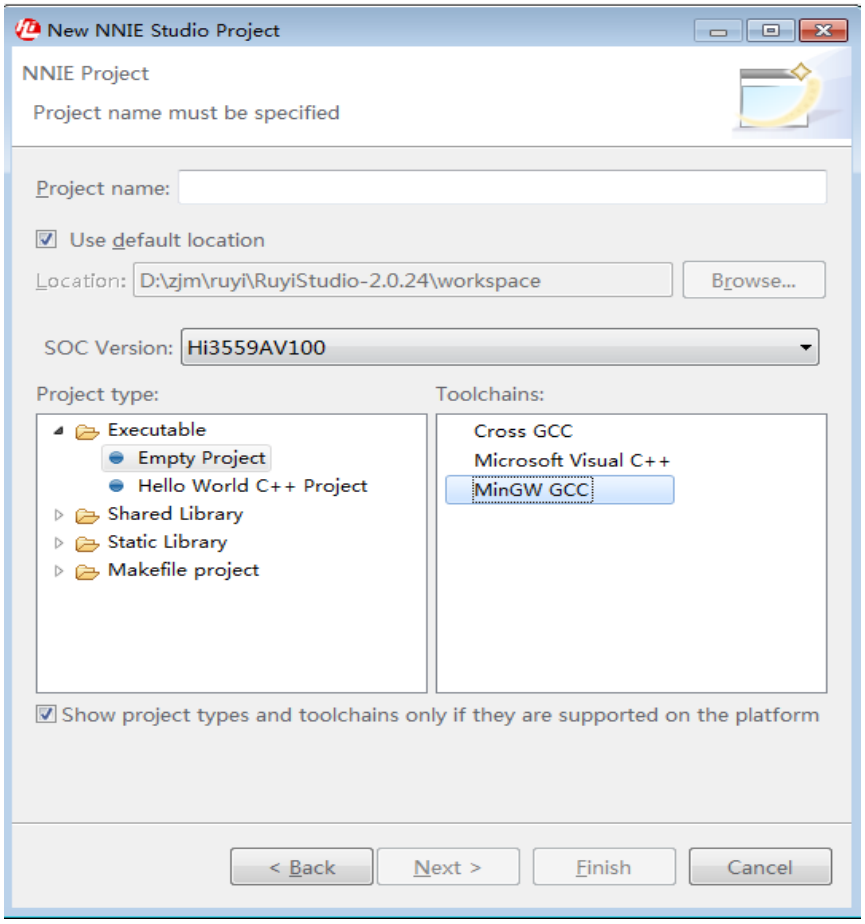
步骤2 选择NNIE project，点击Next，如图5-23所示。

图 5-23 选择 NNIE Project



步骤3 设置Project名称，选择工程所属芯片，默认选择Hi3559AV100；点击Finish完成工程创建；注意只有当MinGW安装好之后才会在Toolchains里显示MinGW GCC，如图5-24所示。

图 5-24 设置 Project 的名字并确认创建工程



----结束

须知

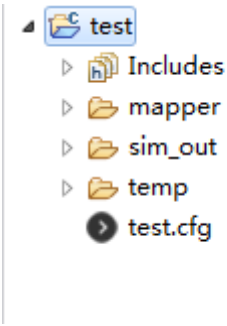
- 若用户的工作空间路径或工程名包含非ASCII字符，工程将无法创建。
- 若用户的工程路径（工作空间路径+工程名整合后的路径）字符数大于操作系统限制（Windows系统下为256），则工程将无法创建。

5.2.3.2 工程文件说明

创建后的工程如图如图5-25所示。包含两个空文件夹和一个配置文件：

- *.cfg：用于配置NNIE mapper运行参数的cfg文件；
- mapper文件夹用于存放NNIE mapper后输出的文件；
- temp文件夹用于存放mapper前准备过程中的输出文件；
- sim_out保存调用仿真库运行时根据hisilicon/nnie_sim.ini配置的仿真输出文件。

图 5-25 创建工程后的工程视图



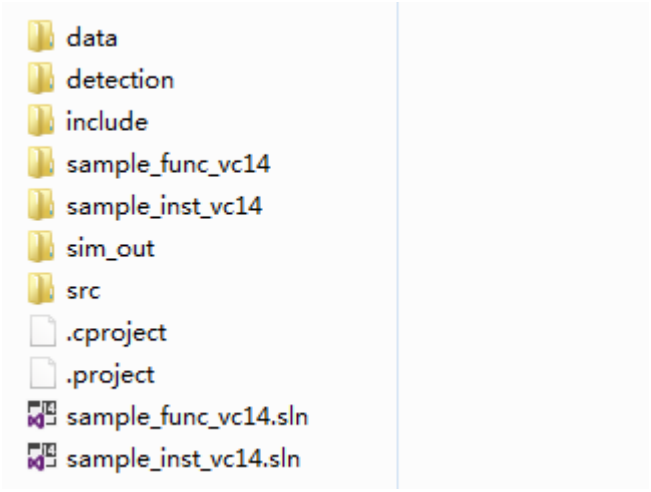
5.2.4 导入工程

HiSVP_PC_Vx.x.x.x\software\sample_xxx是NNIE PC端仿真库的sample工程，可分为两种：

- sample_simulation为功能仿真和指令仿真工程
- sample_runtime为runtime仿真工程

其中包含.cproject和.project文件，表示该工程是可以被导入到RuyiStudio的NNIE工程，如图5-26所示。

图 5-26 NNIE PC 端 Sample 工程



步骤1 鼠标左键单击“File”->“Import”，如图5-27所示；或者鼠标右键点击Project Explore，在弹出菜单中选择“Import”，如图5-28所示。

图 5-27 工程导入方式 1

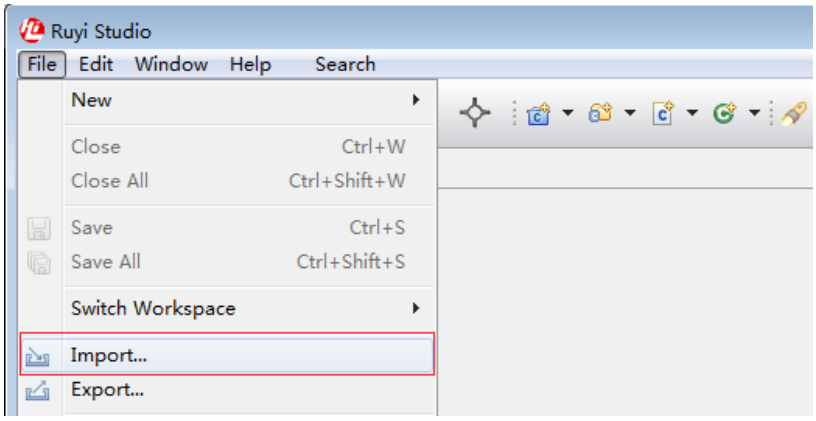
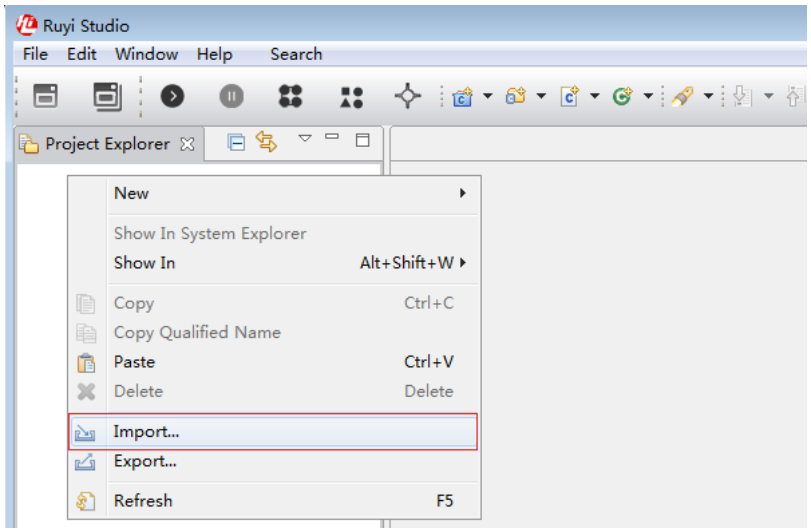
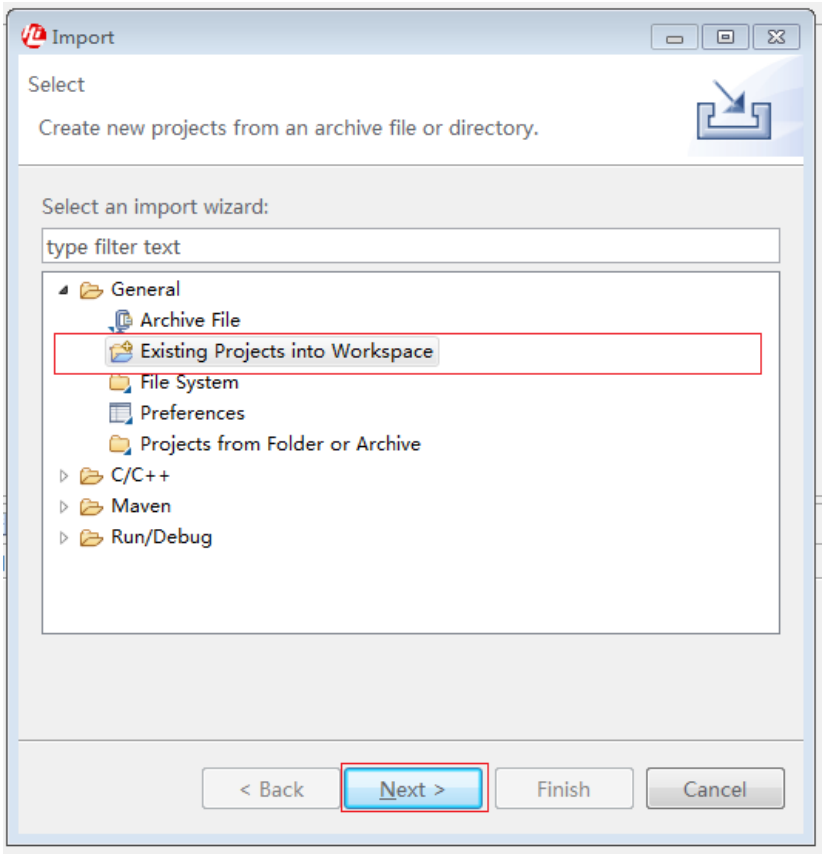


图 5-28 工程导入方式 2



步骤2 在弹出的导入对话框中，选择“Existing Projects into Workspace”，如图5-29所示。

图 5-29 选择外部的项目对话框



步骤3 在弹出的对话框中，如图5-30所示，选择工程所在路径，并选择该路径下要导入的sample工程，点击“Finish”按钮，导入成功后工程将显示在“Project explore”视图中，如图5-31所示。

图 5-30 选择工程对话框

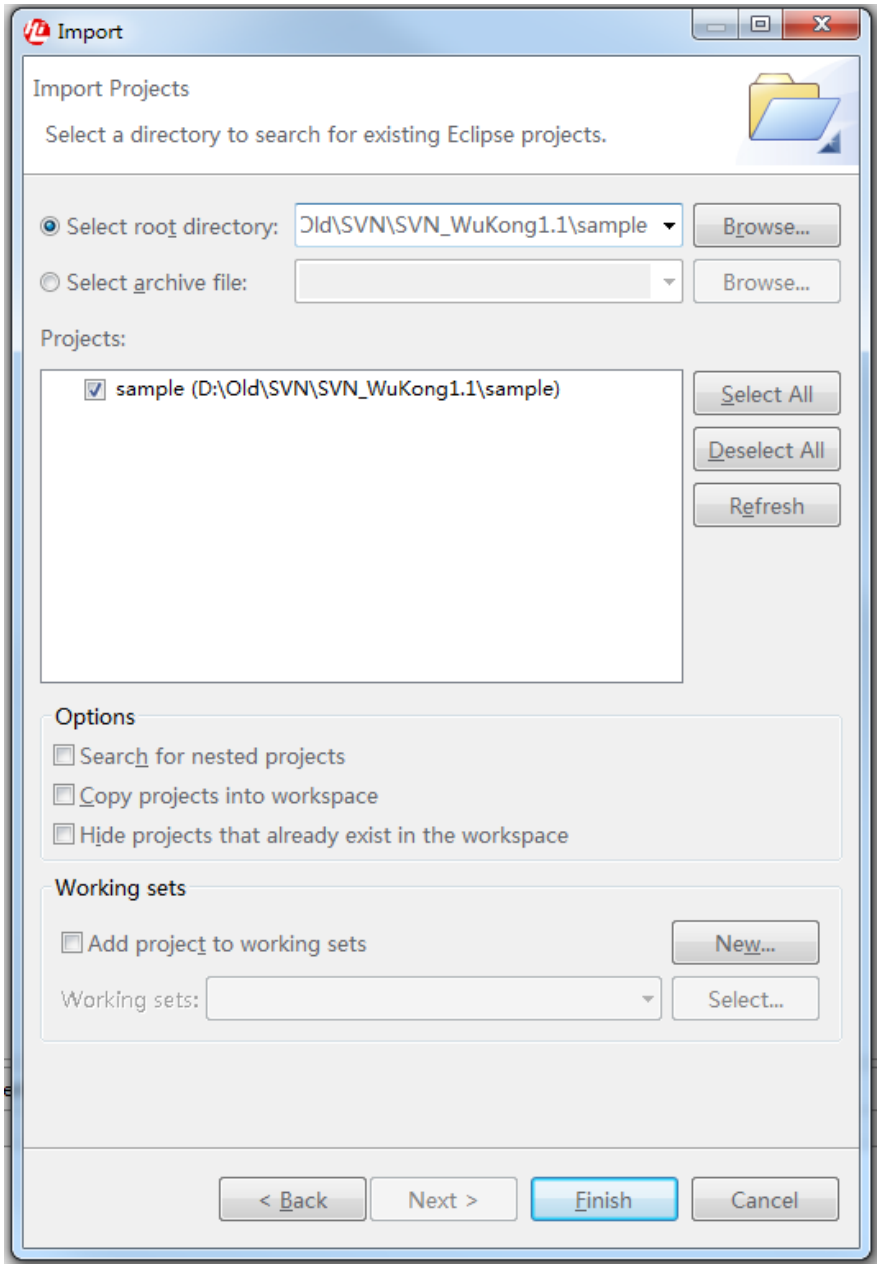
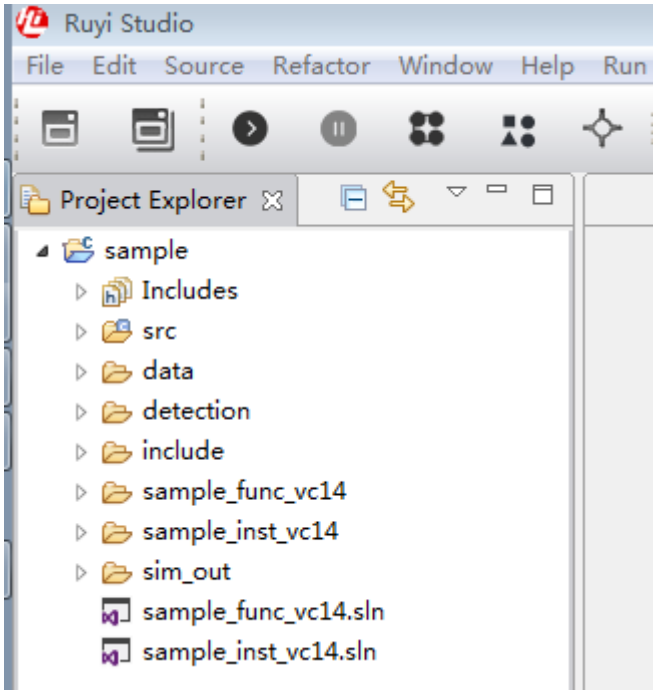


图 5-31 导入 sample 代码后的工程效果



----结束

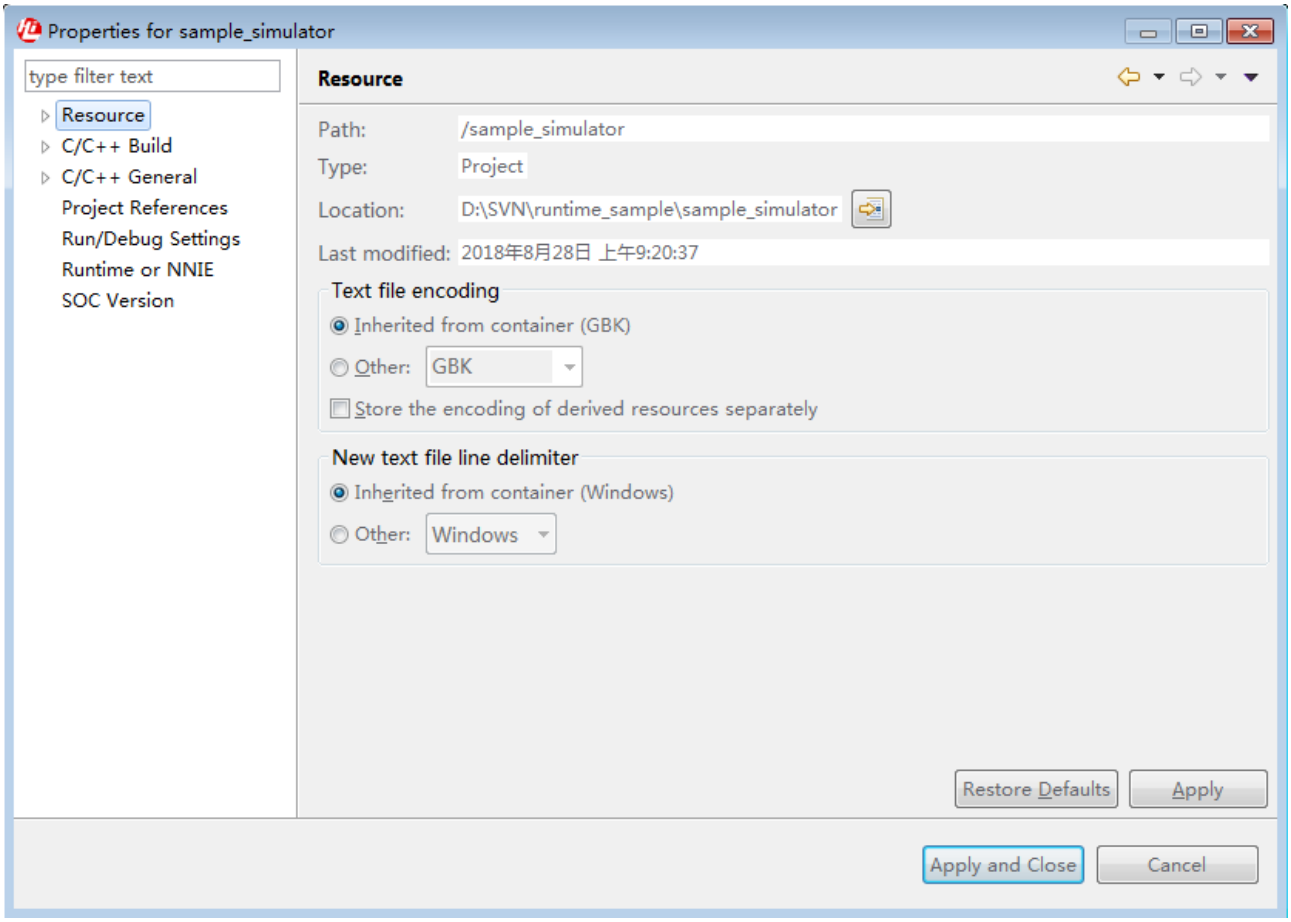
须知

- 请勿导入包含非ASCII字符的工程目录。使用这样的目录可能带来sample无法编译，调试时无法查看源代码等一系列问题。
- 若导入后工程内存在绝对路径字符数超出系统限制（Windows系统下为256）的文件，则这些文件可能无法在工具中被读取或用于编译。

5.2.5 工程属性设置

选中工程右键选择Properties，可以对工程属性进行配置，如图5-32所示。

图 5-32 工程属性视图



5.2.5.1 C/C++ Build 中 Environment 配置

在工程属性的C/C++ Build-> Environment属性页中，可以进行项目自定义生成环境配置，该属性页可以读取系统环境变量，设置变量名，用于C/C++工程的依赖或编译。

如果当前C/C++工程是使用MinGW编译链，按“**5.1.1 编译链MinGW-W64安装**”小节安装后不再需要进行设置。

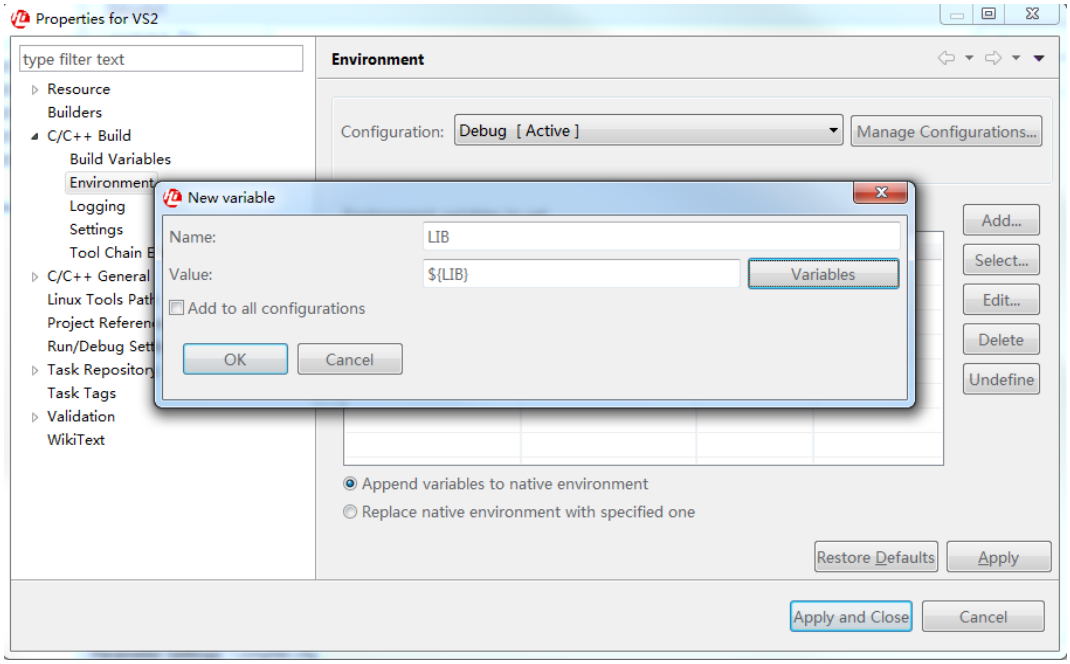
如果当前C/C++工程是使用VS编译链，编译需要使用到VS的cl.exe工具，C++库函数及Include头文件，故需要设置LIB, INCLUDE, PATH环境变量。

- LIB主要指定VS相关的LIB路径
- INCLUDE 主要制定VS相关库的INCLUDE头文件路径
- PATH环境变量主要目的是指定MinGW中的cl.exe的bin路径

设置的方式有以下两种：

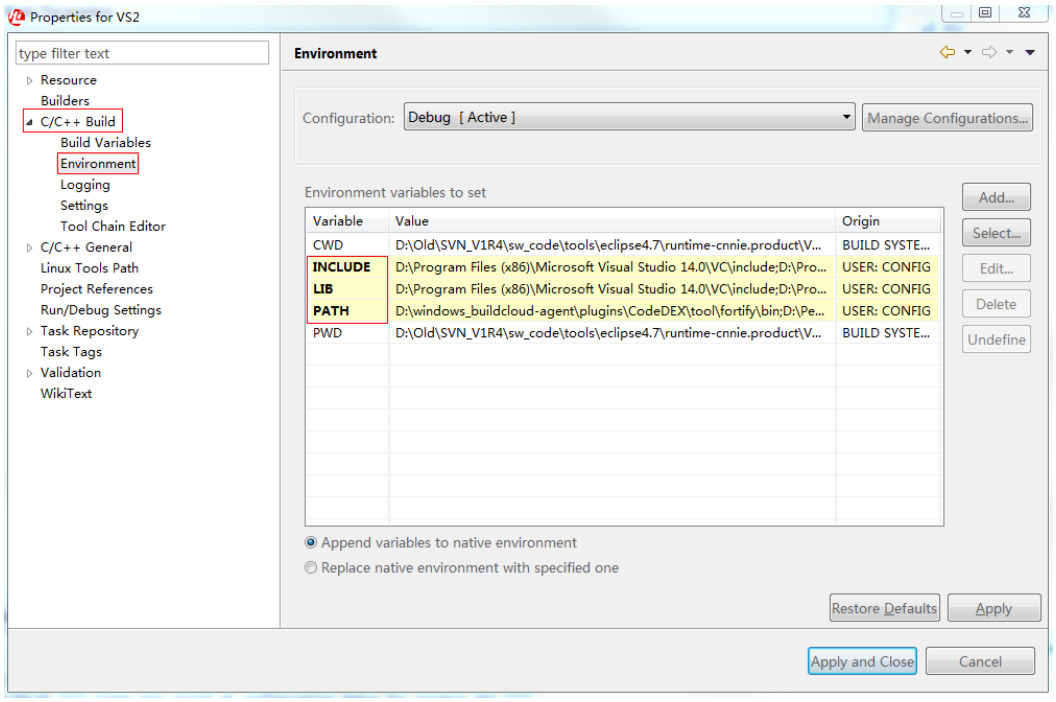
- 采用系统环境变量，点击Add，分别新增LIB，INCLUDE, PATH，在Variables中搜索对应的系统环境变量，然后选中，如**图5-33**所示。

图 5-33 采用系统环境变量



- 自定义环境变量，点击Add，分别新增LIB，INCLUDE, PATH，在Value中输入对应的环境变量路径，如图5-34所示。

图 5-34 采用自定义环境变量



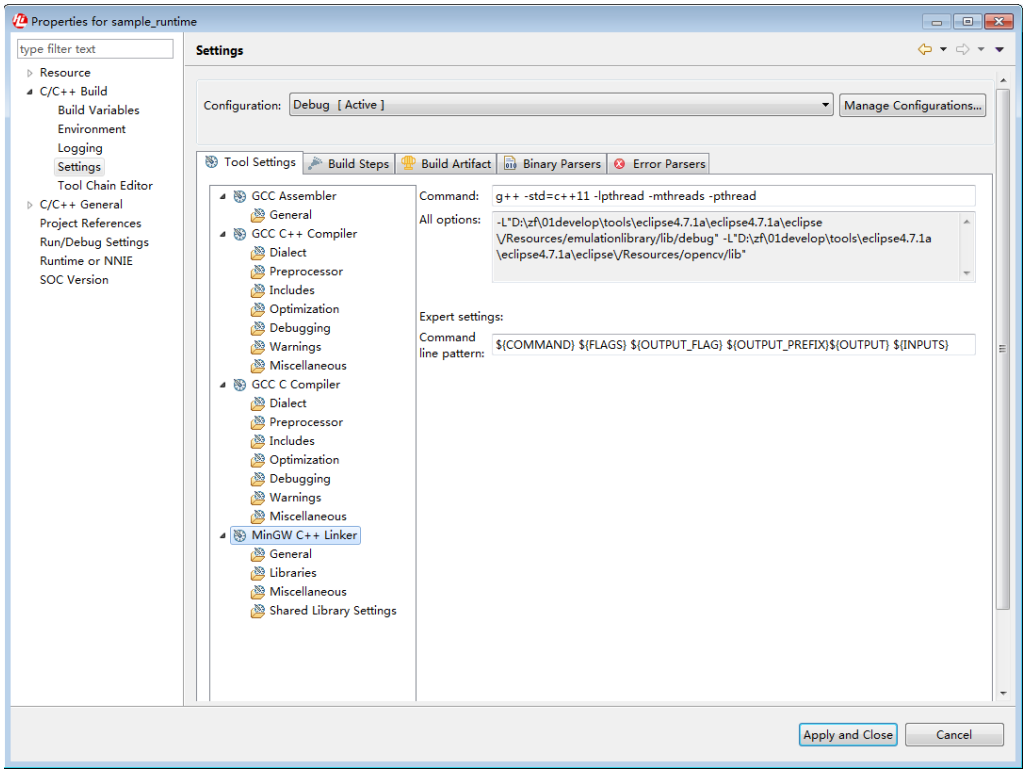
须知

PATH默认为PC环境变量中的PATH路径，需要注意如果PATH中有之前配置过其他的MinGW的bin路径，则新增配置的MinGW-W64 7.3.0的bin路径可能会不生效，用户可以在图5-34的PATH中确认是否有多处MinGW路径，如果有则需要把此处PATH中的MinGW路径保证指定唯一MinGW-W64 7.3.0的bin路径。

5.2.5.2 C/C++ Build 中 Setting 配置

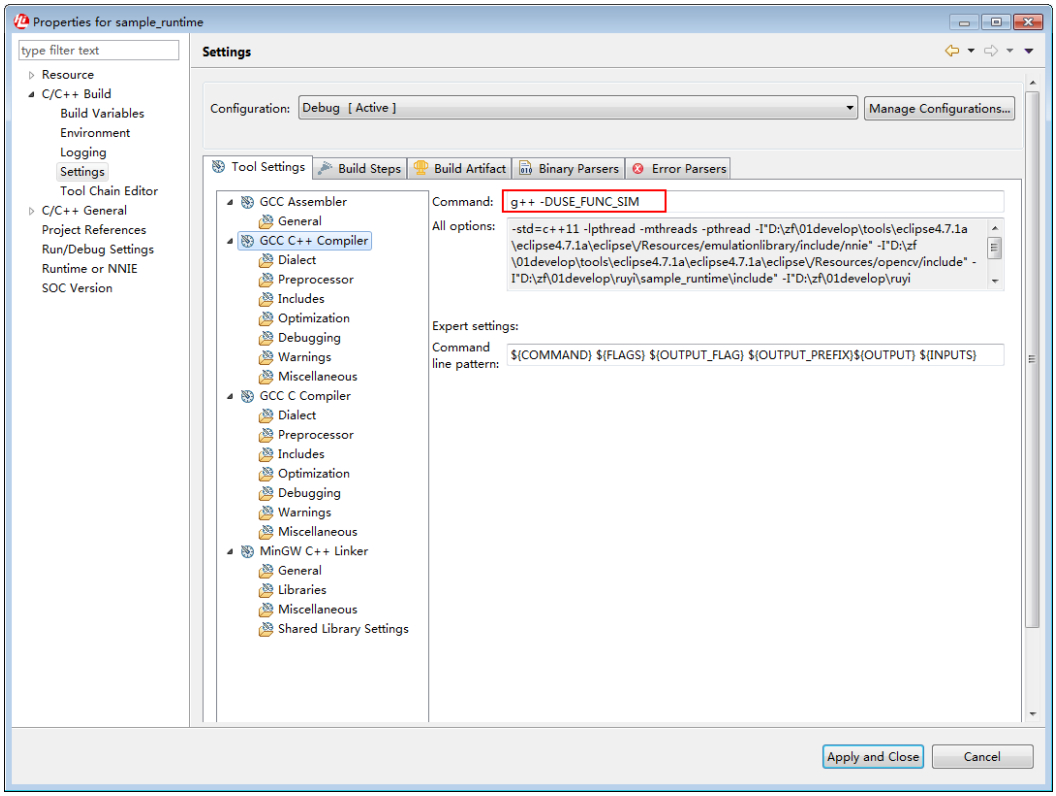
使用 "Settings" 属性页中的Tool Settings，选项卡自定义在生成配置中使用的工具和工具选项。可以在此处对编译发送的命令进行配置，用于C/C++工程的依赖或编译，如图5-35设置MinGW C++ Linker。

图 5-35 设置 MinGW C++ Linker



当编译前需要指定g++的调试参数时，需要在Seting处的GCC C++ Compiler页签中g++命令后添加，如图5-36所示。

图 5-36 设置 g++的编译参数

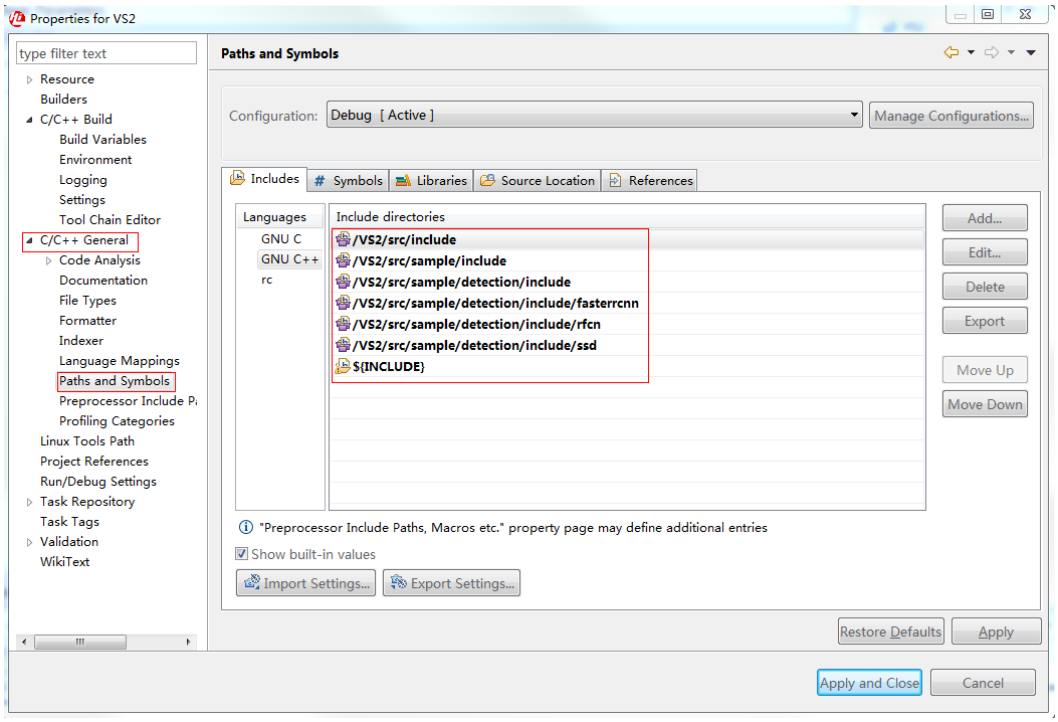


5.2.5.3 C/C++ General 中 Paths and Symbols 配置

在工程属性的C/C++ General->Paths and Symbols中的Includes和Libraries选项卡中，这里配置了NNIE工程编译相关头文件和路径和依赖的库：

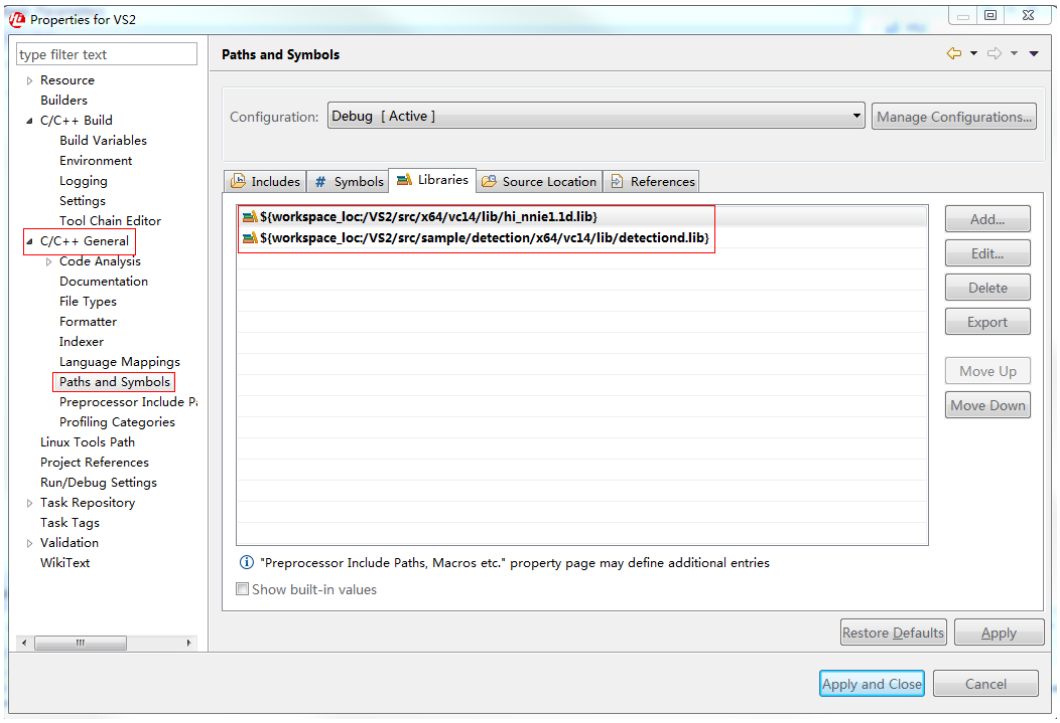
- Include配置中点击Add后有三种引入类型可以选择：
  - 选择Workspace，可以将workspace中的include文件路径都引入到此处；
  - 选择Variables，可以选择系统环境变量中的变量，如INCLUDE引入到此处；
  - 选择File System则可以将任意路径的文件目录引入到此处，如图5-37所示。

图 5-37 工程所需的 Include 路径配置



- Libraries配置中点击Add后同样有三种引入类型可以选择：
  - 选择Workspace，可以将workspace中的lib或dll文件路径都引入到此处；
  - 选择Variables，可以选择系统环境变量中的变量，如LIB引入到此处；
  - 选择File System则可以将任意路径的文件目录引入到此处，如图5-38所示。

图 5-38 工程所需的 Lib 或 Dll 文件配置



说明

C/C++ Properties属性页配置较多，此处介绍的主要和NNIE工程编译相关，如需了解其他配置请参考《C/C++ Development User Guide》文档。网址如下：http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.cdt.doc.user%2Fconcepts%2Fcdt_o_home.htm

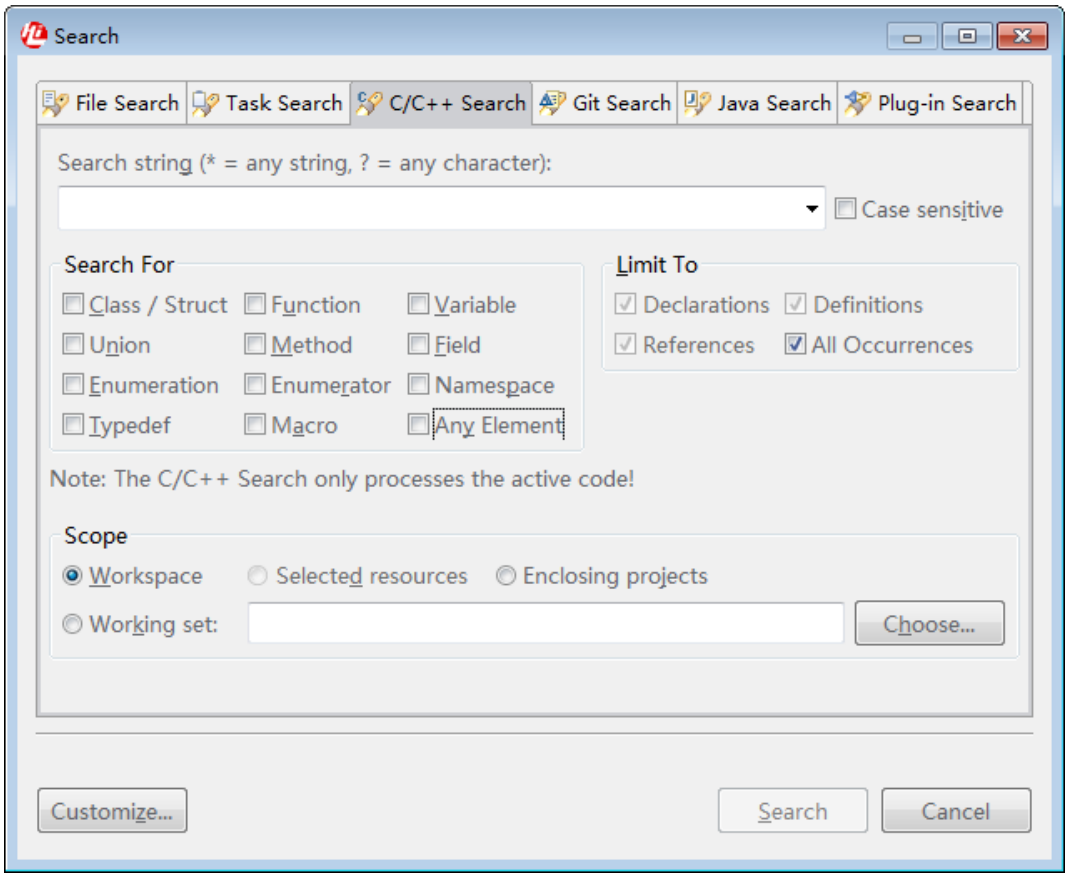
5.2.6 C/C++代码编辑功能介绍

在创建NNIE工程后，可以添加在工程中添加C/C++代码。

5.2.6.1 搜索功能

在创建的C++项目中，打开一个C文件，在该编辑器界面按住Ctrl+H键或者在菜单栏点击Search→Search，会弹出一个搜索框，如图5-39所示。

图 5-39 搜索框

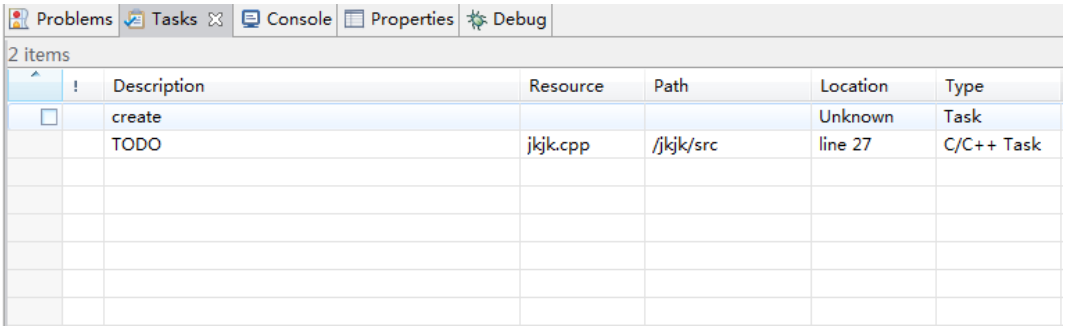


在输入框输入需要查询的类、函数、变量、枚举的关键字，选择需要查询的类、函数、变量、枚举，点击Search按钮，就能定位到需要查询的位置。

5.2.6.2 任务功能

步骤1 在创建的工程中，有一个Tasks视图，如图5-40所示。

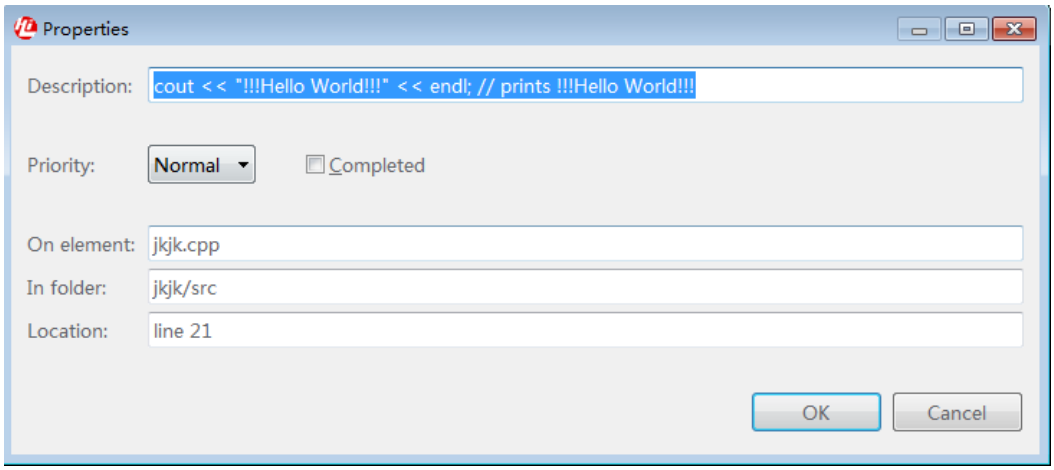
图 5-40 Tasks 视图



步骤2 在Tasks视图中可以查看所有的任务，点击其中一个任务，可以定位到任务的位置。

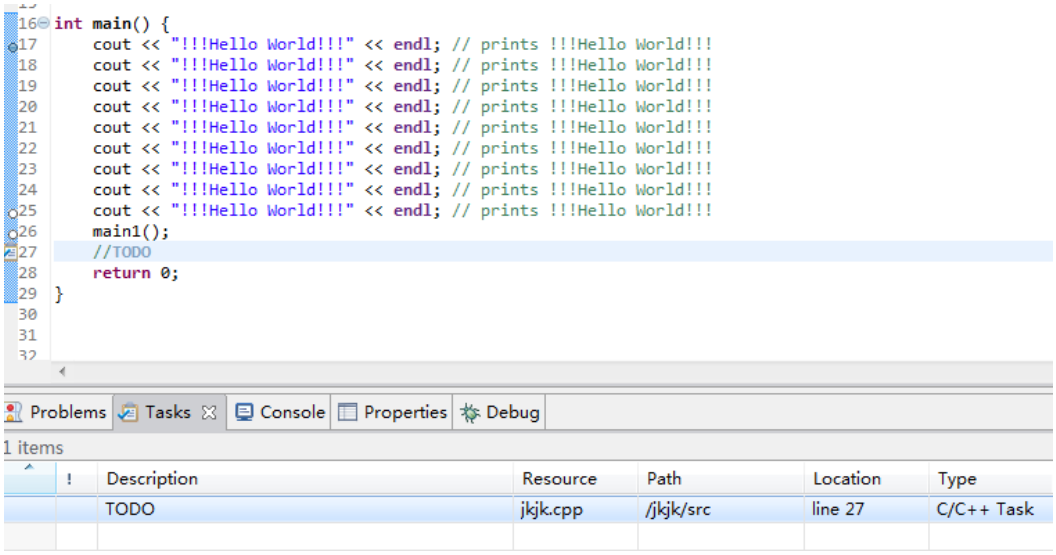
步骤3 在编辑器的左边框，选择需要加断点的行，点击鼠标右键，选择Add Task，会弹出一个对话框如图5-41所示。

图 5-41 创建任务框



步骤4 在代码中添加注释如“//TODO”格式的注释，会在Tasks视图添加一条任务。如图5-42所示。

图 5-42 在代码中创建任务

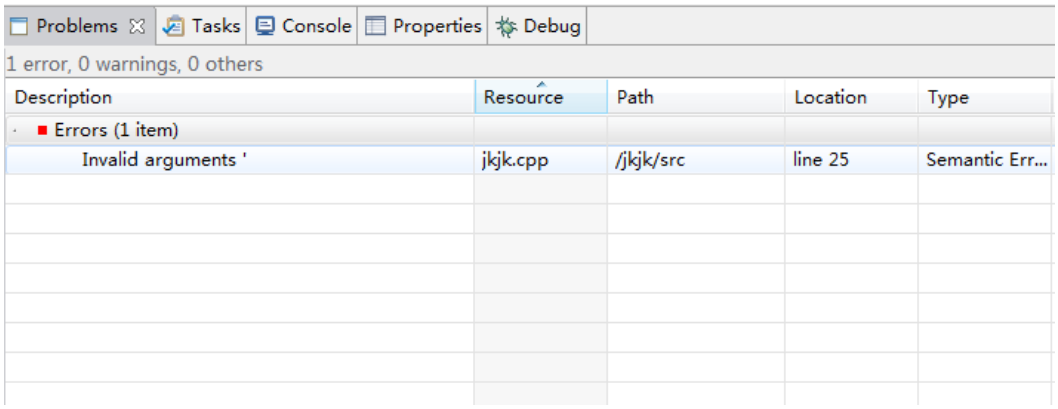


----结束

5.2.6.3 定位问题功能

在创建的C++工程中，有一个Problem视图，如图5-43所示。

图 5-43 问题视图

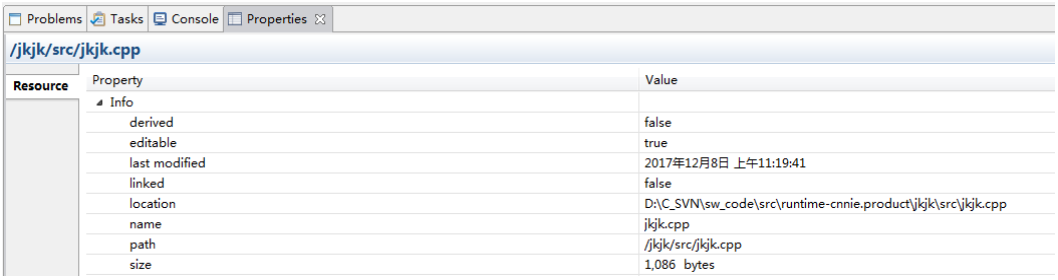


在视图中会显示所有工程中出现的问题，在视图中会显示详细的报错信息，报错的文件，路径以及位置。点击该问题，能在快速在代码中定位到当前错误的位置。

5.2.6.4 查看文件属性功能

在创建的C++工程中，有一个Properties视图。在工程中，选中一个文件，Properties视图会显示该文件的信息，包括名字、路径、权限、文件修改记录和文件大小。如图5-44。

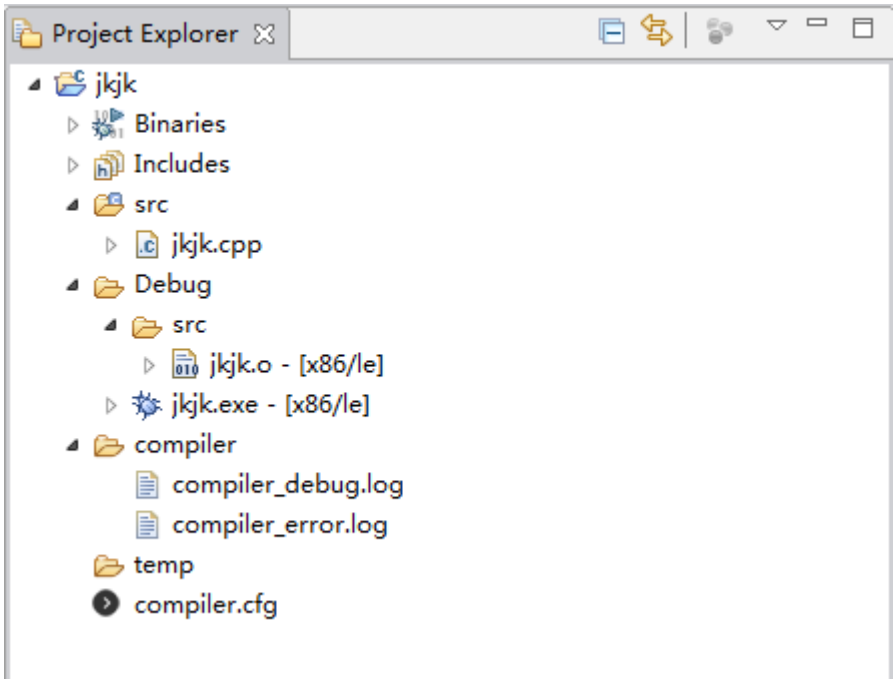
图 5-44 属性视图



5.2.6.5 项目资源管理功能

在工具的界面中，有一个Project Explorer视图，如图5-45所示。

图 5-45 项目管理视图

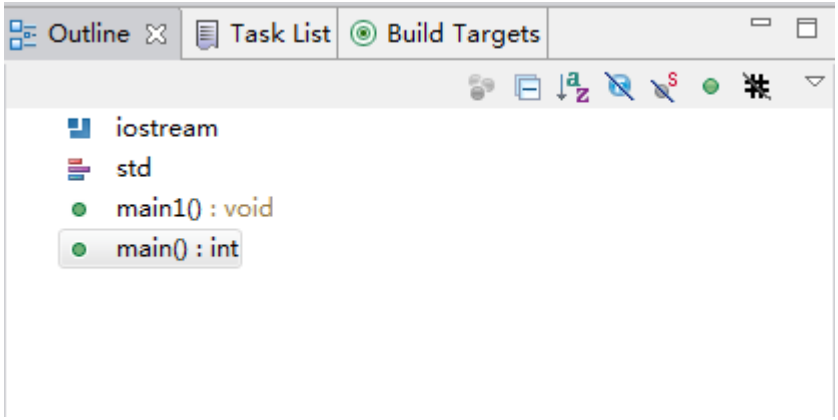


在该视图中，可以查看当前工作空间所有创建的工程，同时也可以对创建的工程进行资源理，可以对工程的文件进行创建、修改、删除和查看。

5.2.6.6 大纲视图功能

在工具的界面中，有一个Outline视图，在创建的工程中，打开一个C文件，在Outline视图中，可以显示该文件的所有的函数、头文件以及变量，如图5-46所示。

图 5-46 大纲视图

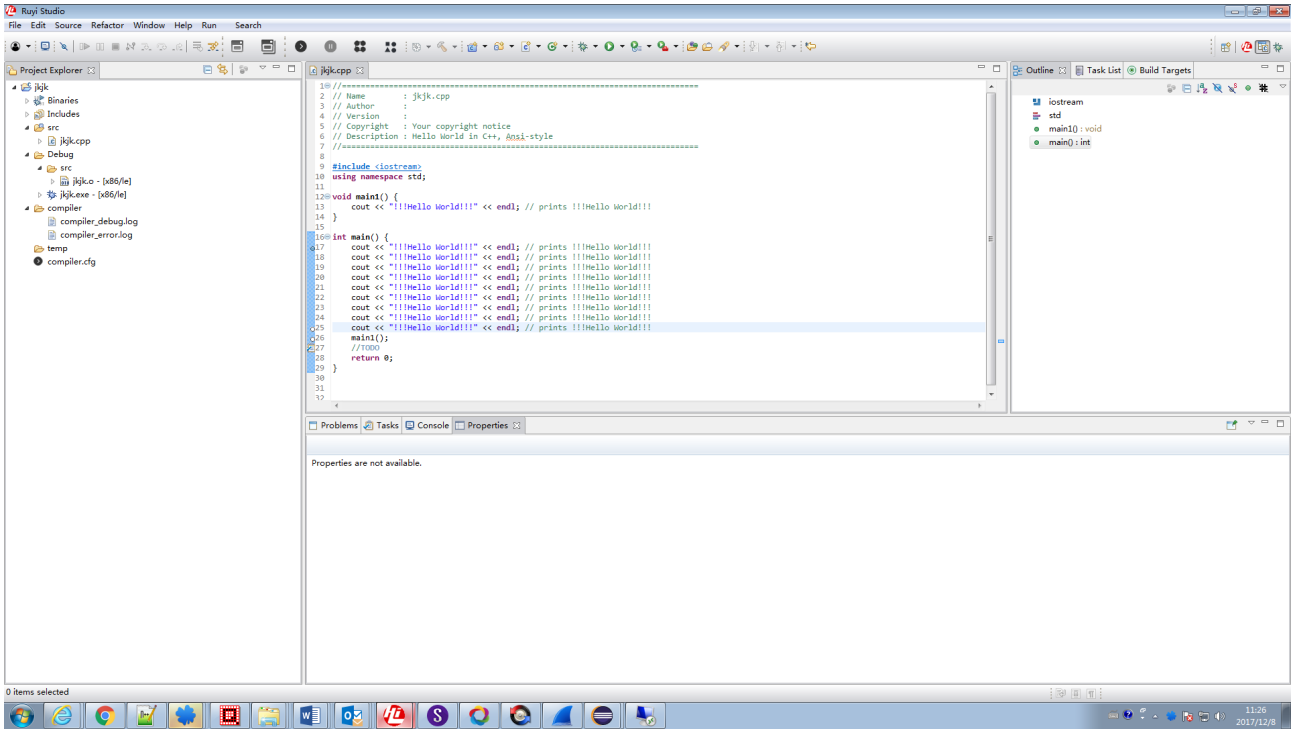


在该视图中可以选择需要查看的函数、头文件和变量，双击选中的属性，可快速定位到对应的代码中。

5.2.6.7 引用关系查看功能

打开工程中创建的一个C文件，选择当前文件中的函数或者头文件，按住Ctrl键，选中的函数或者头文件会出现下划线。如图5-47所示。

图 5-47 引用关系查看

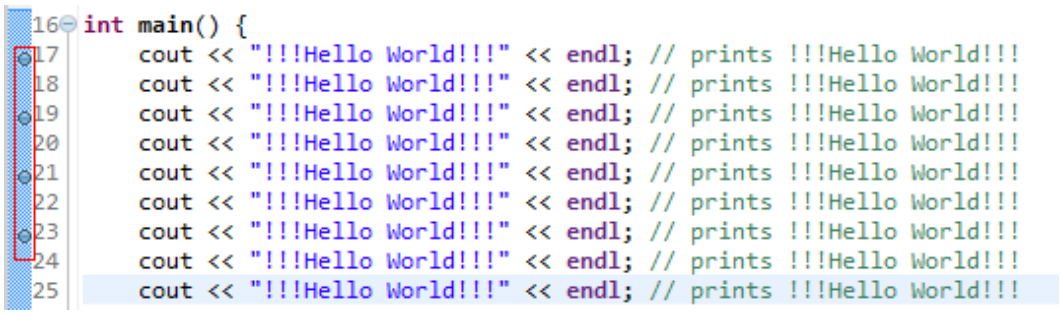


此时按住Ctrl键不放，点击鼠标左键，会跳转到选择函数或者头文件的定义处。

5.2.6.8 集成调试功能

打开工程中创建的一个C文件，在打开文件的编辑处，在编辑器的左边框，选择需要加断点的行，点击鼠标右键，选择Toogle BreakPoint，会在该处出现一个圆点，如图5-48所示。

图 5-48 断点显示

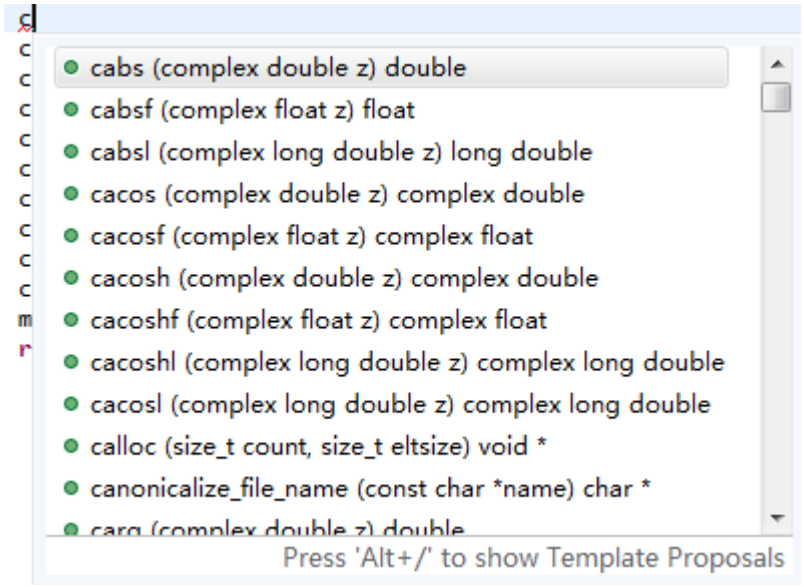


该圆点功能为调试Debug的断点，当使用Debug调试的时候，程序走到这里就会暂停。

5.2.6.9 代码内容辅助功能

打开工程中创建的一个C文件，再打开文件的编辑处，对文件进行编译，在编辑的过程中没有输入完整的关键字，此时可以按住Alt+/ 键，会弹出该当前输入关键字有关的函数、变量的对话框，如图5-49所示。

图 5-49 代码辅助功能



在弹出的框中选择所需的函数或者变量。选择的对象会输入到当前的编辑行。

5.2.6.10 代码前进后退功能

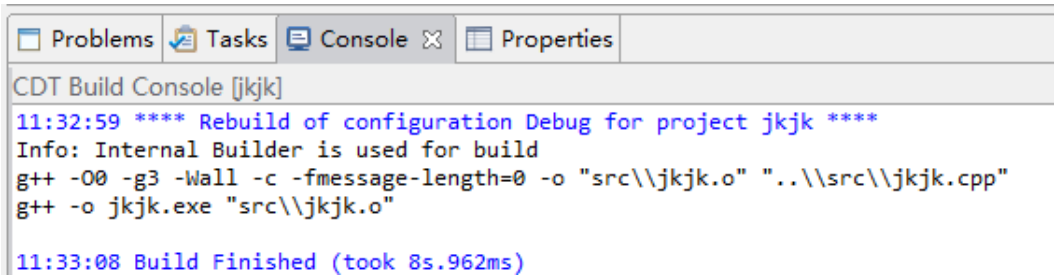
- 支持回退到上一次代码访问的位置。
- 支持前进到上一次代码访问的位置。

5.2.7 C/C++代码编译功能

编译NNIE的C/C++工程时需要提前配置工程的Include，Lib及Path路径，请参考“5.2.5 工程属性设置”章节。配置完成后，方可执行如下编译操作。

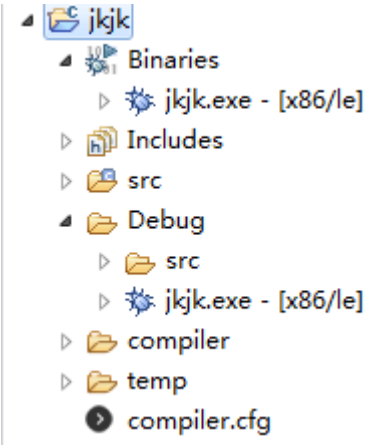
在项目资源中选择一个C++项目，点击 按钮，点击完按钮之后会在Console视图打印当前项目的编译日志如图5-50。

图 5-50 编译日志



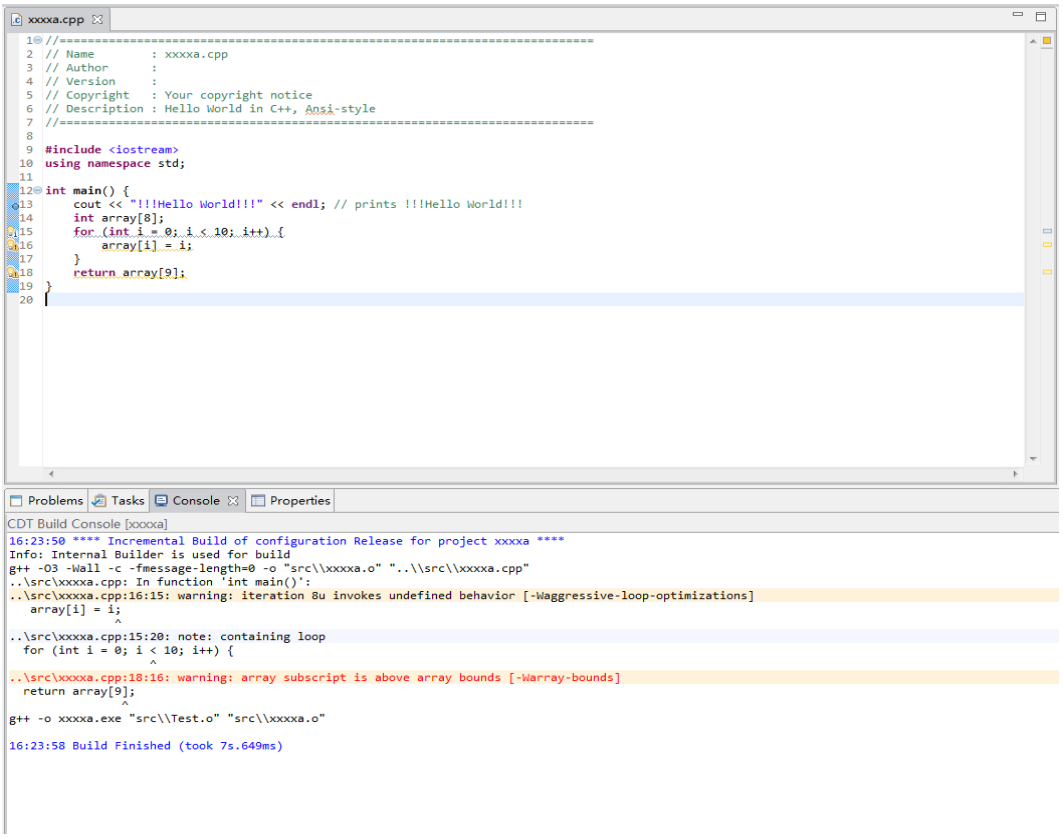
编译成功后会在当前的项目中生成一个Release或者Debug文件夹，文件夹包含一个src文件夹和对应的工程文件名的exe文件。如图5-51。

图 5-51 编译成功后的文件夹



在对选中的文件进行编译时，当出现错误时，Console会打印编译报错的日志。如图5-52。

图 5-52 编译报错的日志



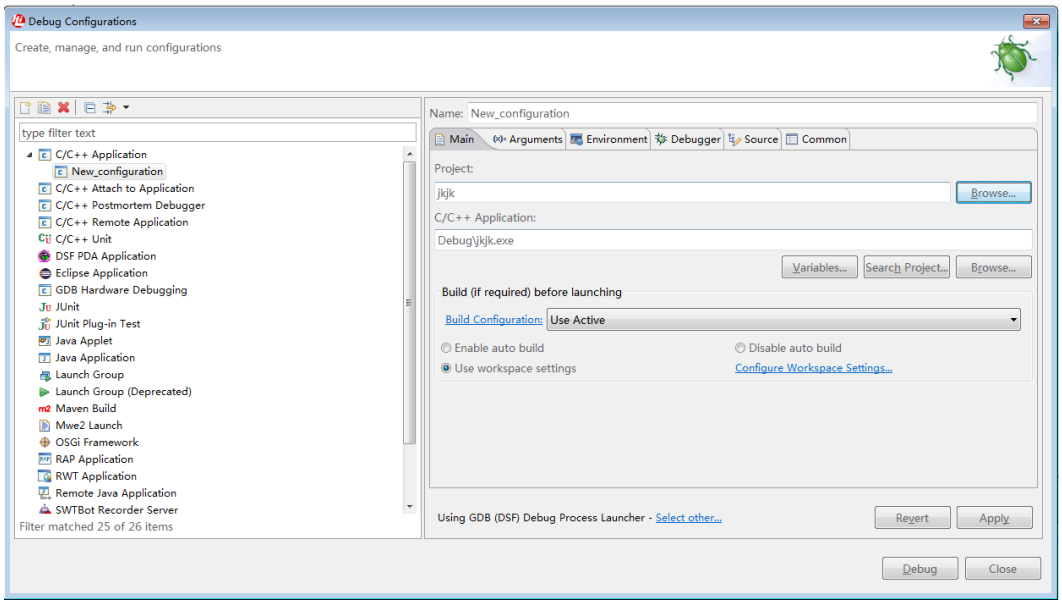
在打印的报错日志中，点击对应的报错日志，可定位到编译出错的代码中。

5.2.8 C/C++代码调试功能

5.2.8.1 创建 C/C++ Application

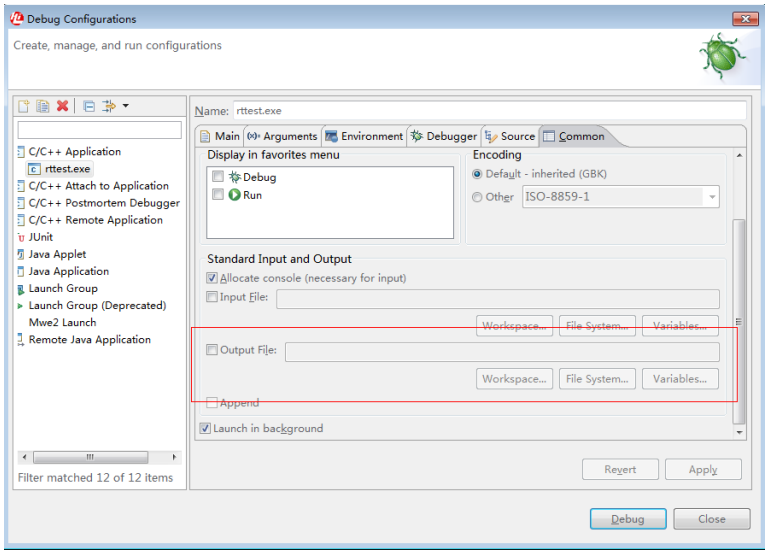
步骤1 代码编译通过后，点击Debug按钮  ，点击小三角，选择Debug Configure，弹出对话框，如图5-53。

图 5-53 创建 Application 设置视图



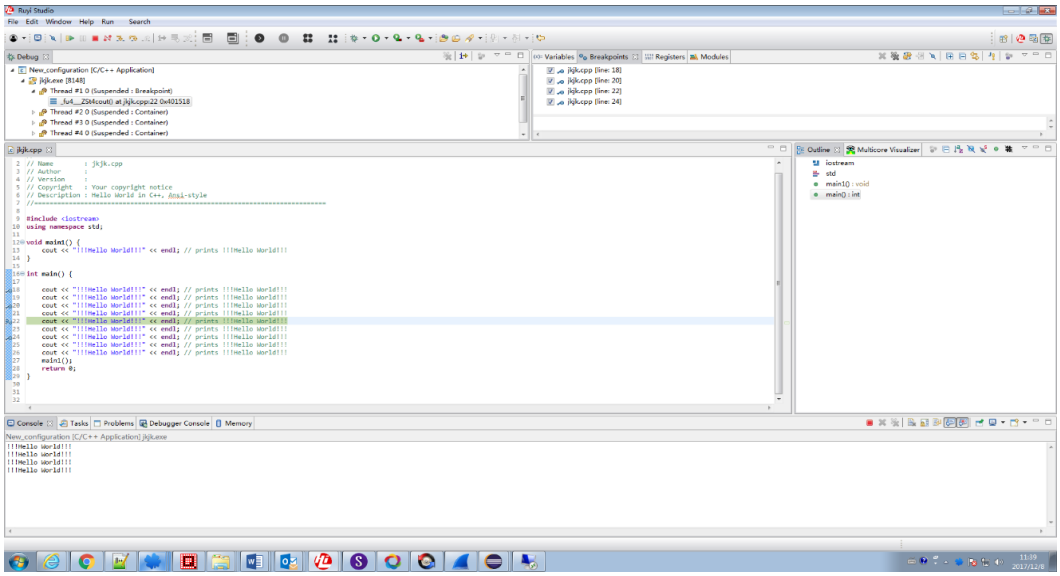
步骤2 由于工具界面Console视图能容纳的日志行数有限，因此一般情况下需要重定向标准输出。步骤1的Application设置视图中选择Common页签，在Output File中指定一个文件作为日志输出文件即可。

图 5-54 重定向标准输出



步骤3 设置完成后点击Debug按钮，可以选择跳转到debug透视图，如图5-55所示。

图 5-55 Debug 透视图



----结束

5.2.8.2 Debugger 视图按钮功能

- 调试实例----调试名称和状态。
- 线程实例----线程数量和状态。
- 堆栈框架和实例----堆栈框架数量，函数，文件名，文件行数
- 移除所有中止的发布----在调试视图中清除所有的中止进度。
- 跳过断点----在调试的过程中跳过所有的断点，不在打了断点的地方停止。
- 恢复----运行程序。
- 暂停----暂停执行当前选中的调试目标线程。
- 终止----结束选中的调试会话或者进度。影响的行为依赖于在调试视图中选中的条目的类型。
- 步进执行----执行当前语句，但会跟踪到子程序的内部。
- 单步执行----执行当前语句，但是跳过子程序的调用。
- 指令步进模式----启用指令步进模式，当按下此按钮，单步和步进操作是按汇编指令一条条执行，否则，是按源码的语言形式一条条执行。

5.2.8.3 C/C++工程相关调试说明

C/C++工程具体调试功能，请参考《C/C++ Development User Guide》文档，链接如下：

http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.cdt.doc.user%2Fconcepts%2Fcdt_o_home.htm

5.3 生成 NNIE wk 功能

新建工程或者导入已有工程后，会出现后缀为“.cfg”的NNIE mapper配置文件。

5.3.1 cfg 文件配置模式

5.3.1.1 界面配置模式

默认双击左侧Project Explorer视图中xxx.cfg文件，cfg文件将以Mapper Configuration Editor的模式打开界面模式，或者选中cfg文件右键选择Open with-> Mapper Configuration Editor，如图5-56，打开后的界面如图5-57。

图 5-56 Mapper Configuration Editor 界面模式打开 cfg 文件

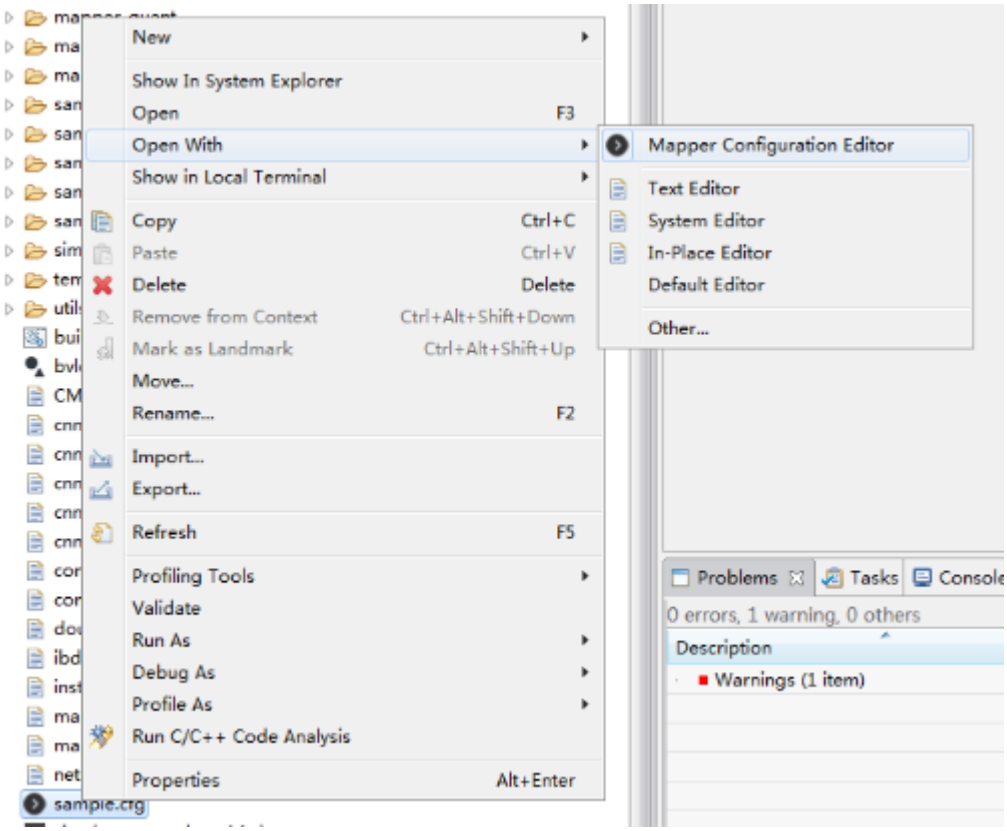
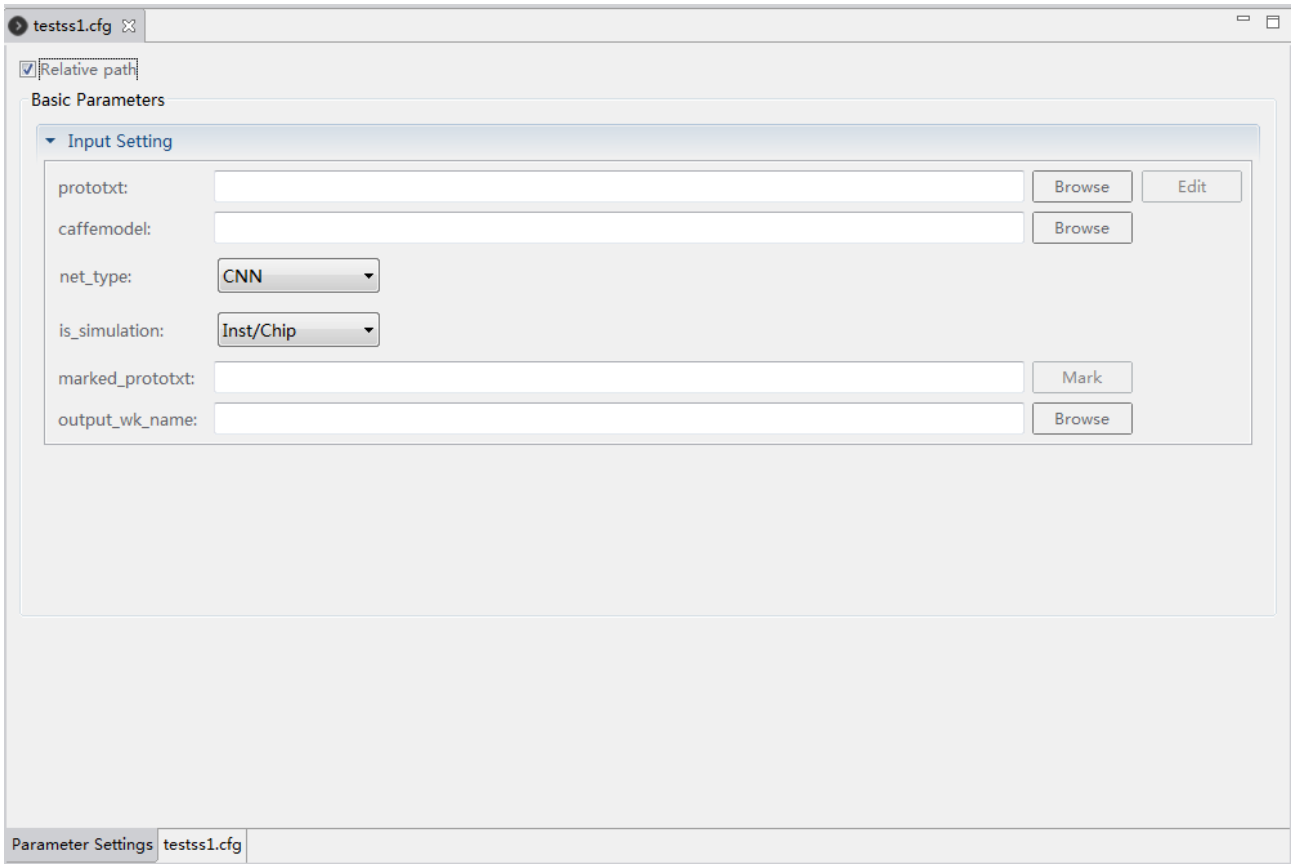


图 5-57 Mapper Configuration Editor 视图

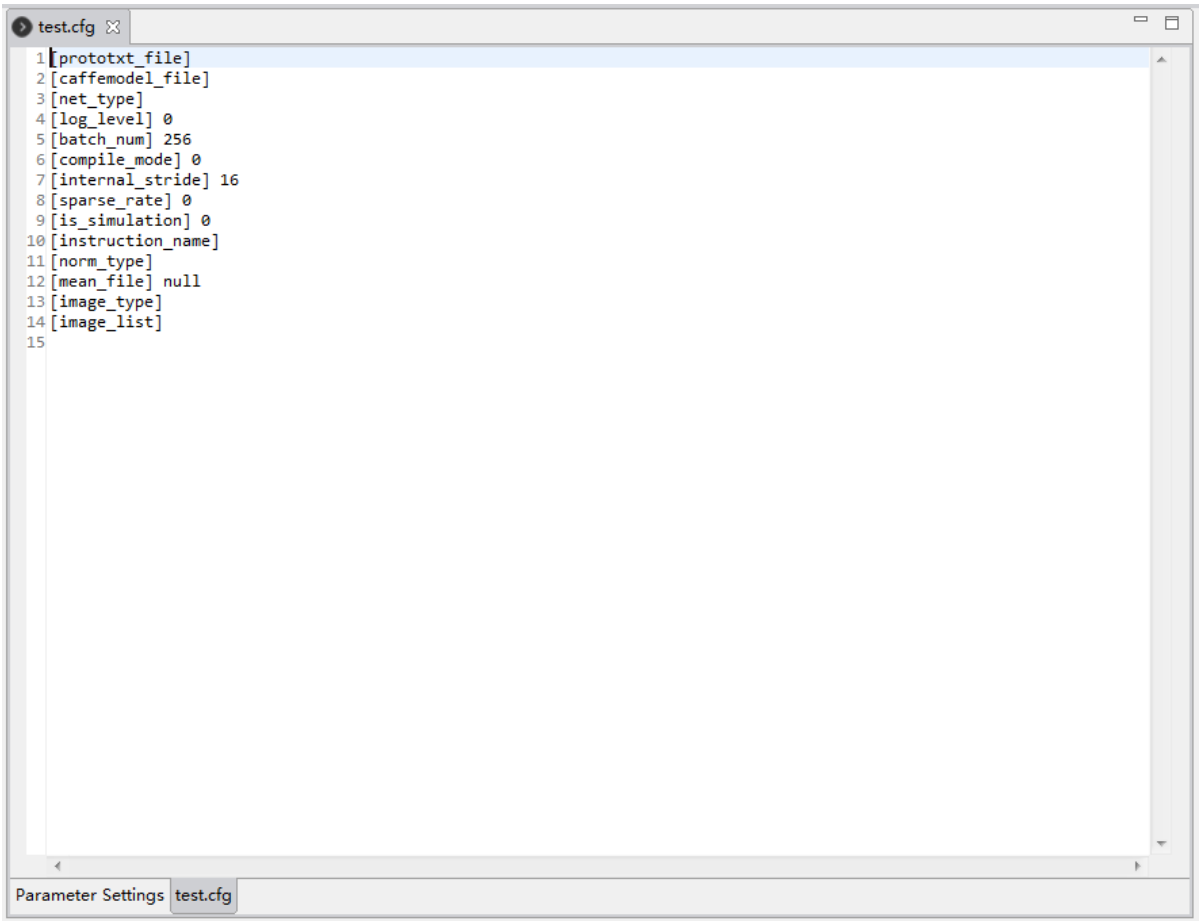


5.3.1.2 文本编辑模式

选中cfg文件右键选择Open with->Text Editor，或者点击图5-58左下角的test.cfg按钮可以切换到cfg文件的文本编辑视图；

cfg文本文件可以在linux版本的nnie_mapper中编译。

图 5-58 cfg 文件 Text Editor 视图



5.3.2 Prototxt 文件自动标记功能

在cfg文件界面视图中，如图5-59，点击Browse按钮导入prototxt，导入prototxt后，需要选择net_type/caffemodel才会进行预处理，RuyiStudio自动将NNIE不支持的层修改为Custom或Proposal层，该功能为预处理的第一个步骤。

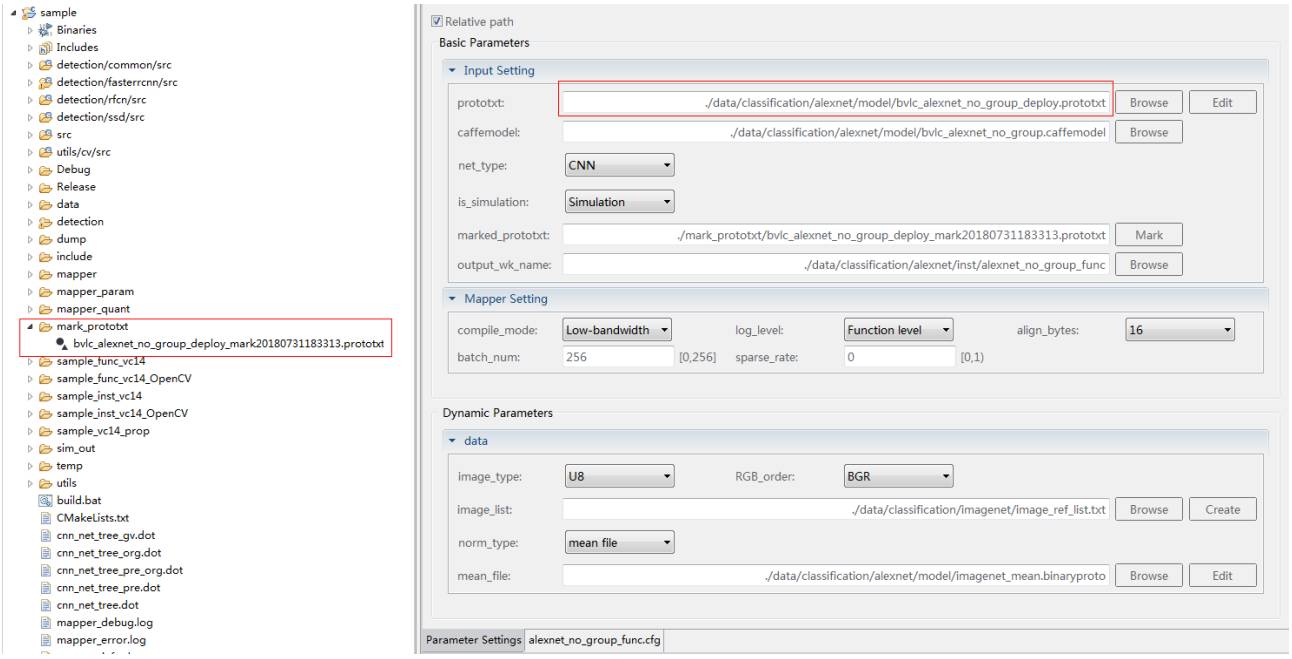
因用户导入的Prototxt文件中可能有nnie_mapper不支持的层类型，RuyiStudio自动识别prototxt层并根据NNIE mapper Non-support层处理方式将不支持的层修改为Custom或者Proposal层，且生成一个标记后的prototxt文件，路径存放在Project工程mark_prototox文件目录下，命名规则为原Prototxt文件名_mark_时间戳.prototxt文件，如图5-59所示。

RuyiStudio将作为ROIPooling或者PSROIPooling层输入的Non-support层则修改为“Proposal”层，其他则作为“Custom”层。

RuyiStudio支持解析prototxt的层类型如下：

"Convolution", "Deconvolution", "Pooling", "DepthwiseConv", "InnerProduct", "LRN", "BatchNorm", "Scale", "Bias", "Eltwise", "ReLU", "PReLU", "AbsVal", "TanH", "Sigmoid", "BNLL", "ELU", "CReLU", "LSTM", "RNN", "Softmax", "Exp", "Log", "Reshape", "Flatten", "Split", "Slice", "Concat", "SPP", "Power", "Threshold", "MVN", "Reduction", "Proposal", "ROIPooling", "PSROIPooling", "Normalize", "PassThrough", "Upsample", "Permute", "MatMul", "Custom", "Input", "Dropout"。

图 5-59 Prototxt 自动标记功能



5.3.3 标记后的 Prototxt 可视化功能

点击如图5-60中Mark按钮进入工具自动标记后的Prototxt的网络拓扑图，如图5-61。

图 5-60 编辑标记后 Prototxt 文件

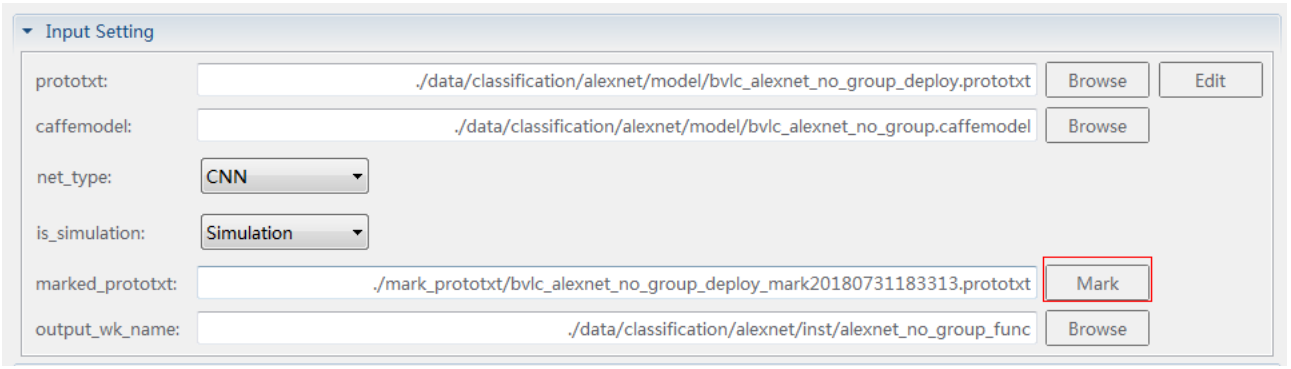
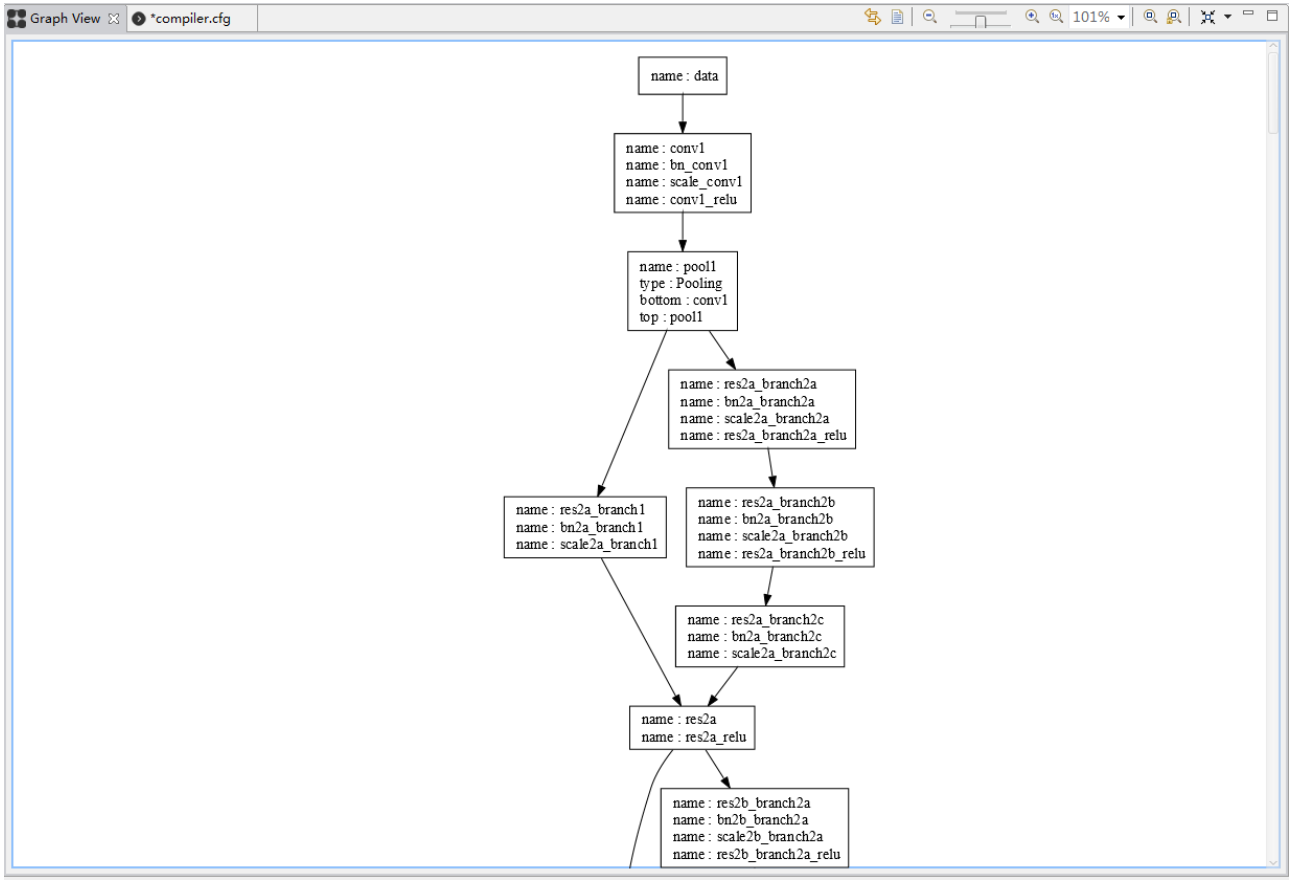
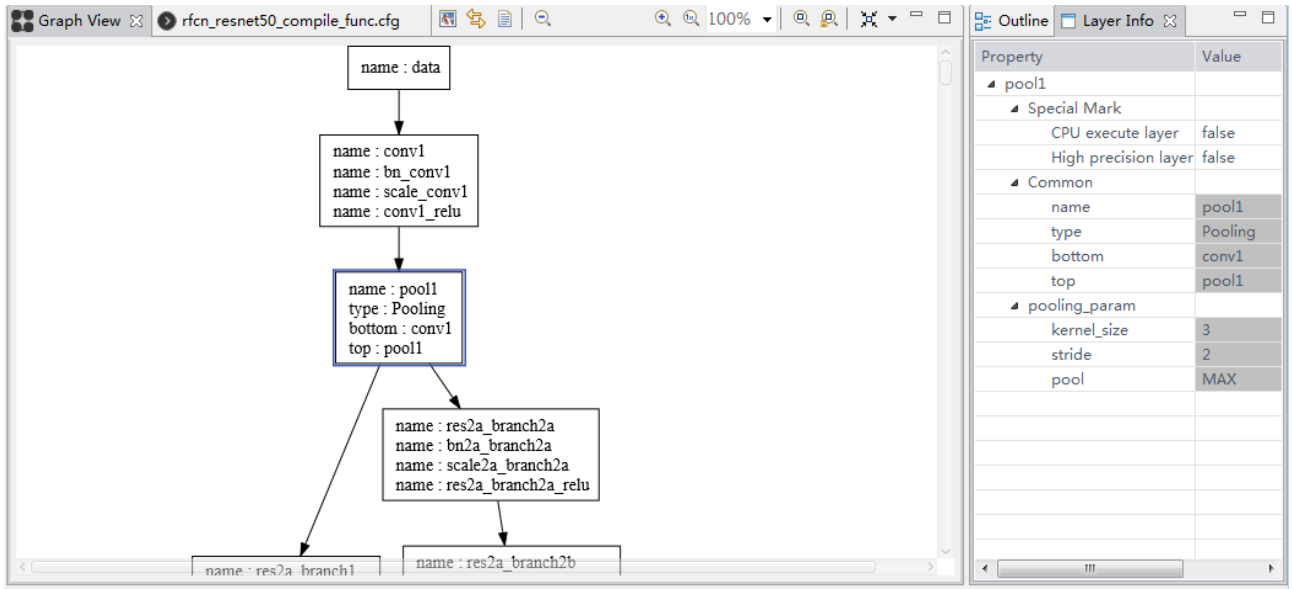


图 5-61 标记后的 Prototxt 的网络拓扑图



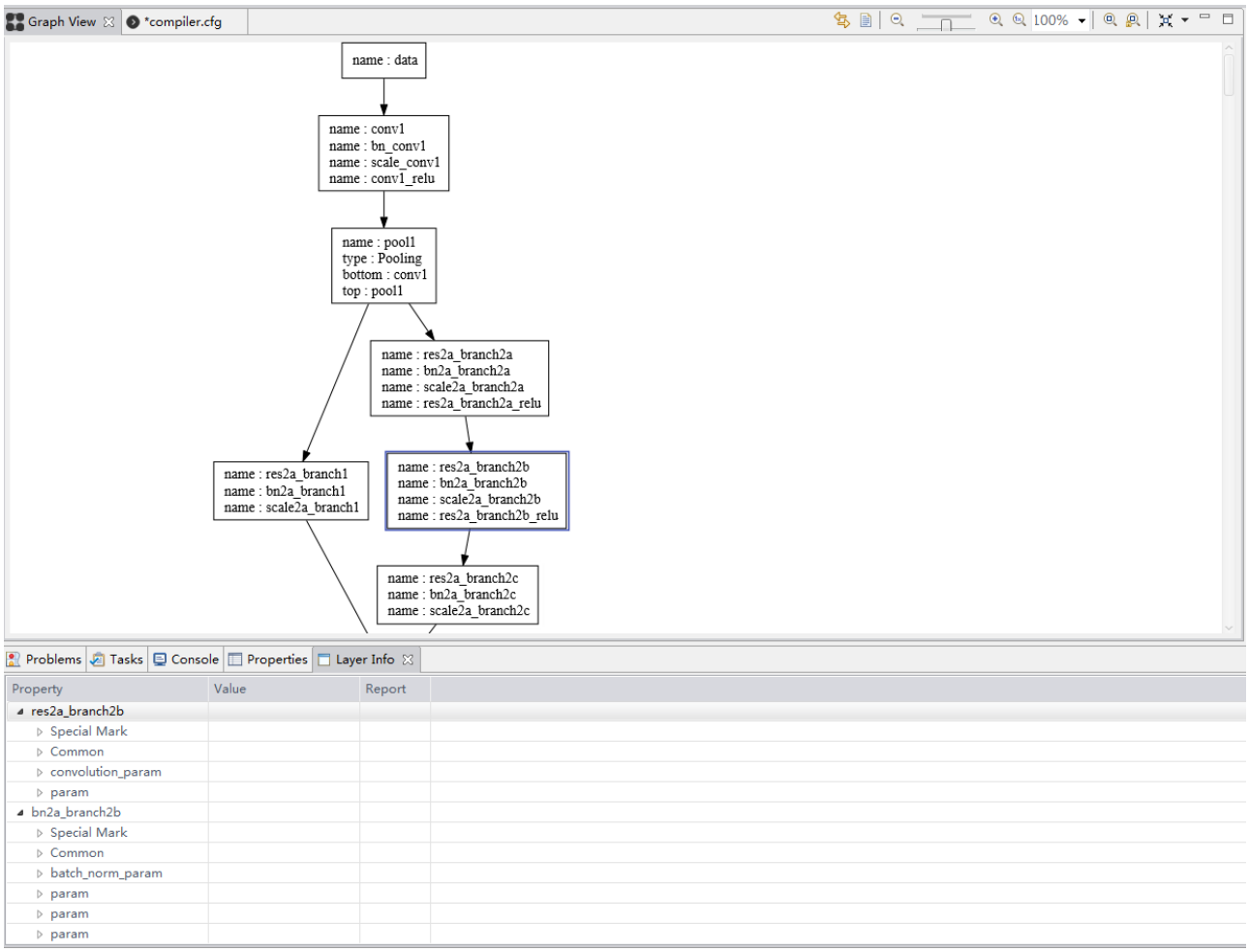
选中网络拓扑图中的Layer节点，被选中的层的外框会被标记为蓝色，如果当前层对应的Type是NNIE不支持的Type类型，则当前层的Type会被改为Custom，且会将其他参数删除，双击选中的层可以在右侧Property视图中查看和修改当前Layer对应的Common基本参数，包括name, type, bottom, top等信息，如图5-62。

图 5-62 标记后的 Prototxt 中 Layer 对应的属性图



Inplace的层（Inplace即为相互连接的几个层中，top是相同的，bottom是不完全相同的）在网络拓扑图中会被放在一个框节点中，选中一个这样的框，所有层的属性都会显示在Layer Info 视图中，如图5-63。

图 5-63 Inplace 层节点的 Layer Info 视图

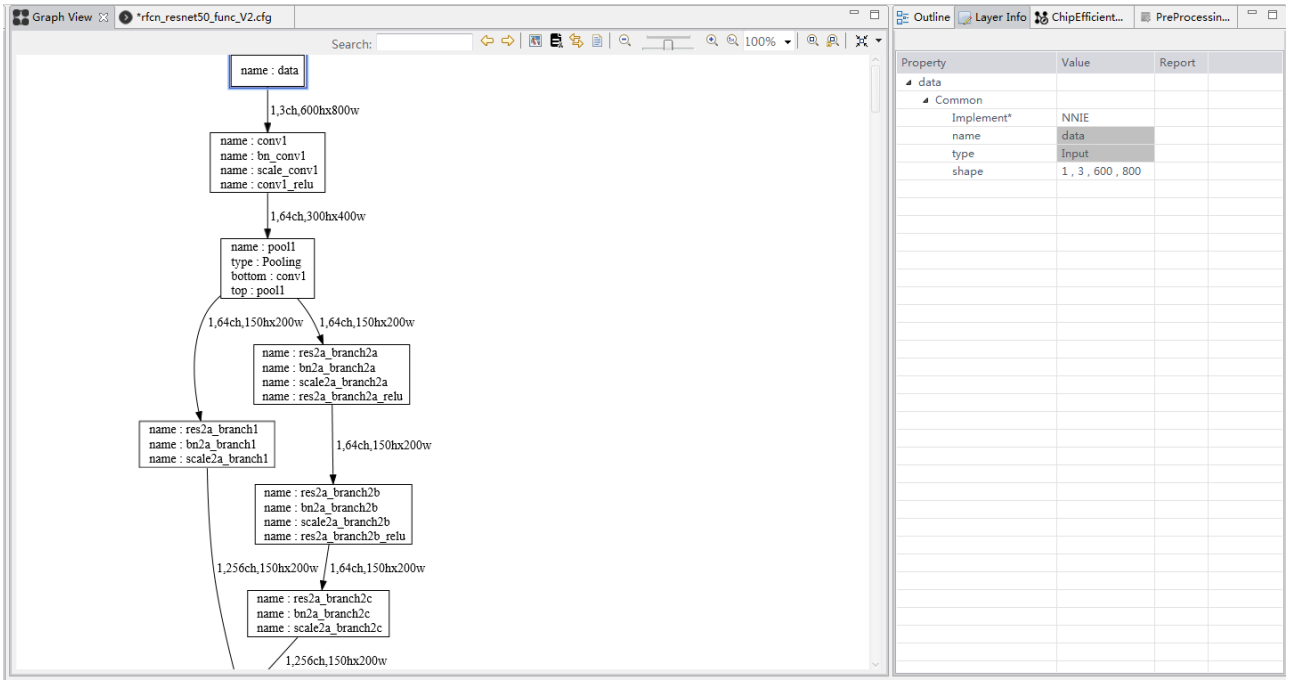


5.3.4 网络拓扑图中显示各原生 caffe 支持层的 Shape 信息

5.3.4.1 Shape 信息显示

支持当前PC如果安装了Python和Caffe环境下（可参考5.1.2 Python3.5+caffe环境配置章节），在按照5.3.3 标记后的Prototxt可视化功能操作后，网络拓扑图中可以显示出各层的shape信息，如图5-64。

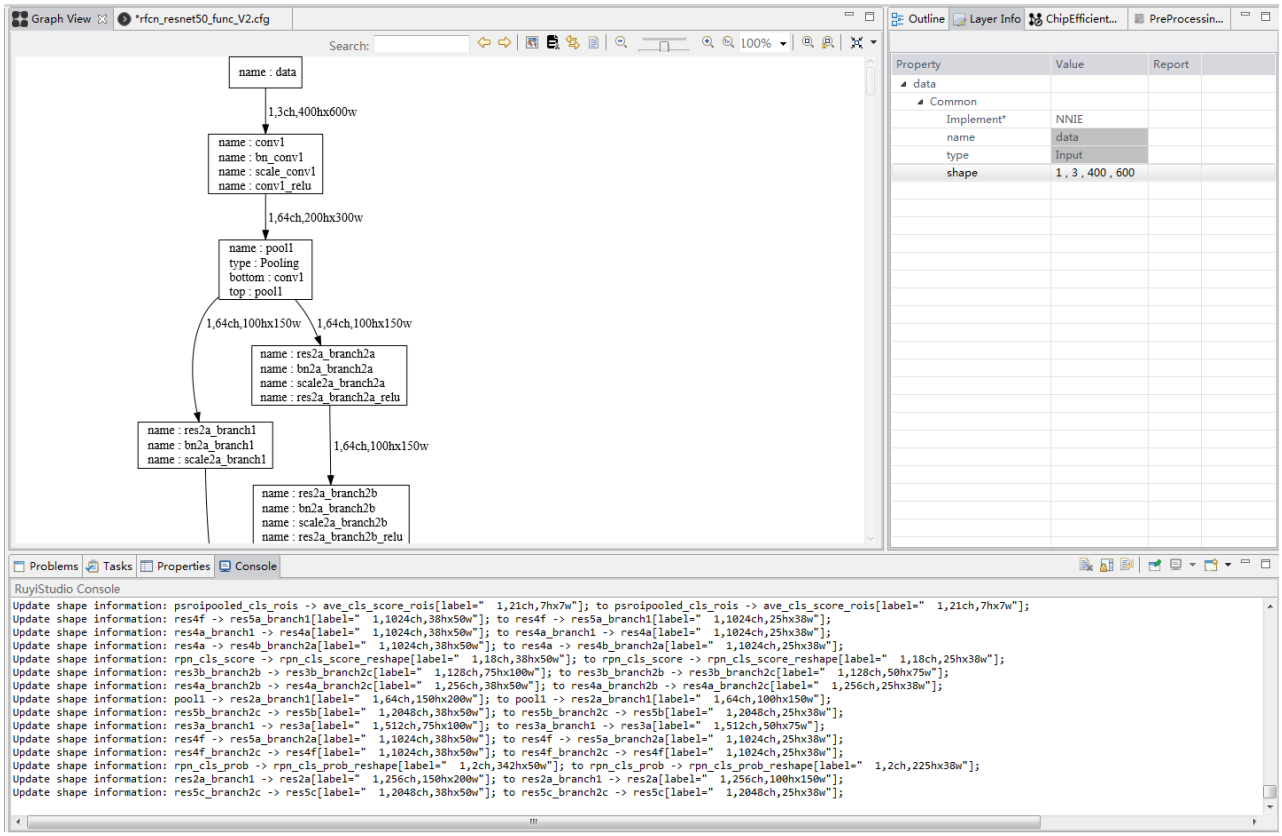
图 5-64 显示 shape 信息的网络拓扑图



5.3.4.2 Shape 信息更新

在GraphView中双击input层或custom层，然后在层属性LayerInfo界面中，修改shape信息，然后点击其他空白处确认后，工具会自动更新当前网络拓扑图中各层的shape信息并刷新到界面上且修改内容会打印在控制台中，如图5-65所示。

图 5-65 更新 shape 信息后的网络拓扑图



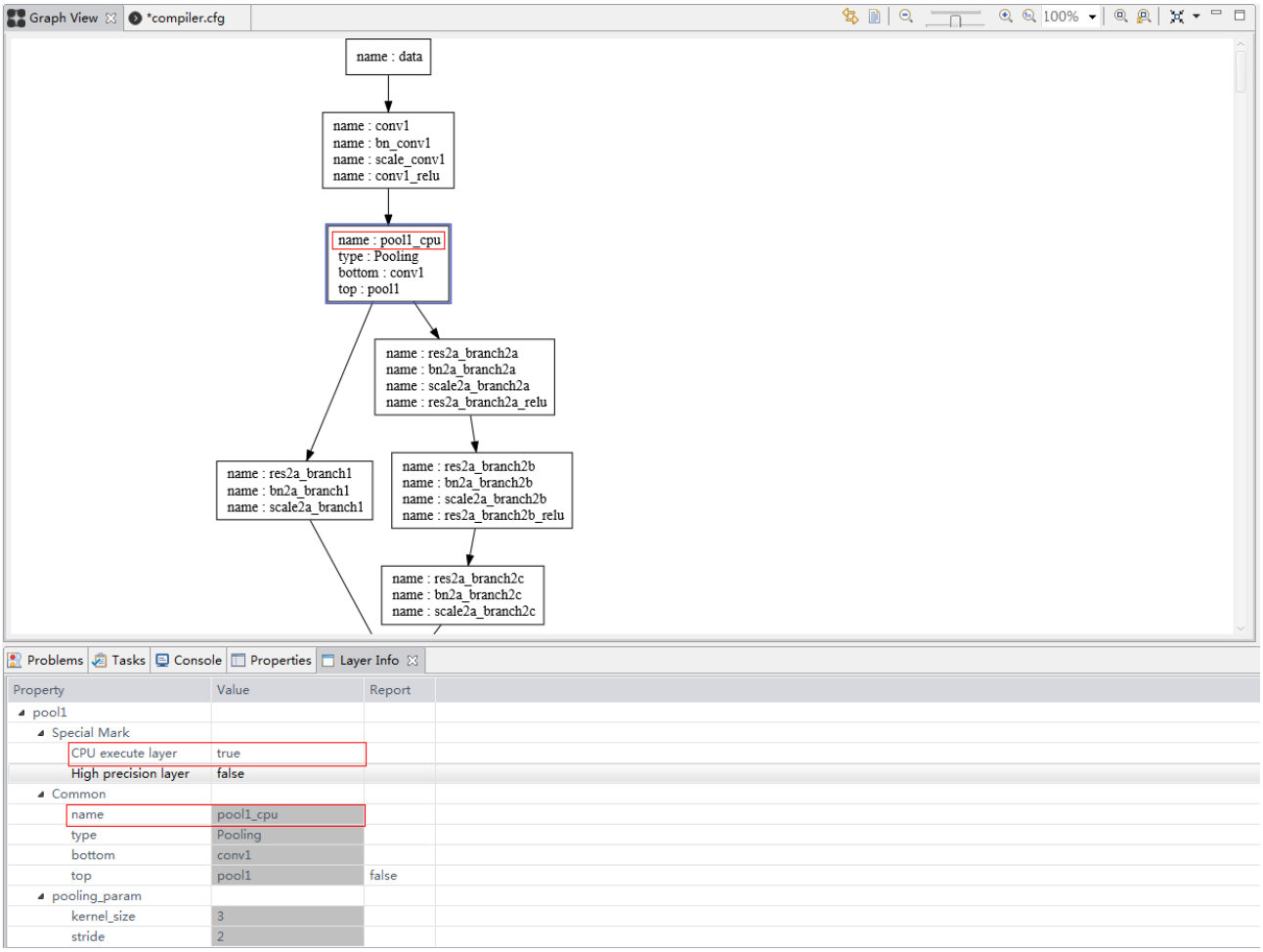
5.3.5 Prototxt 层属性可视化编辑

NNIE mapper支持将任意层切换为CPU运算，支持自定义高精度层，支持任意层中间上报。

在工具中通过修改层对应的属性，可以将当前层标记为以上三种类型。

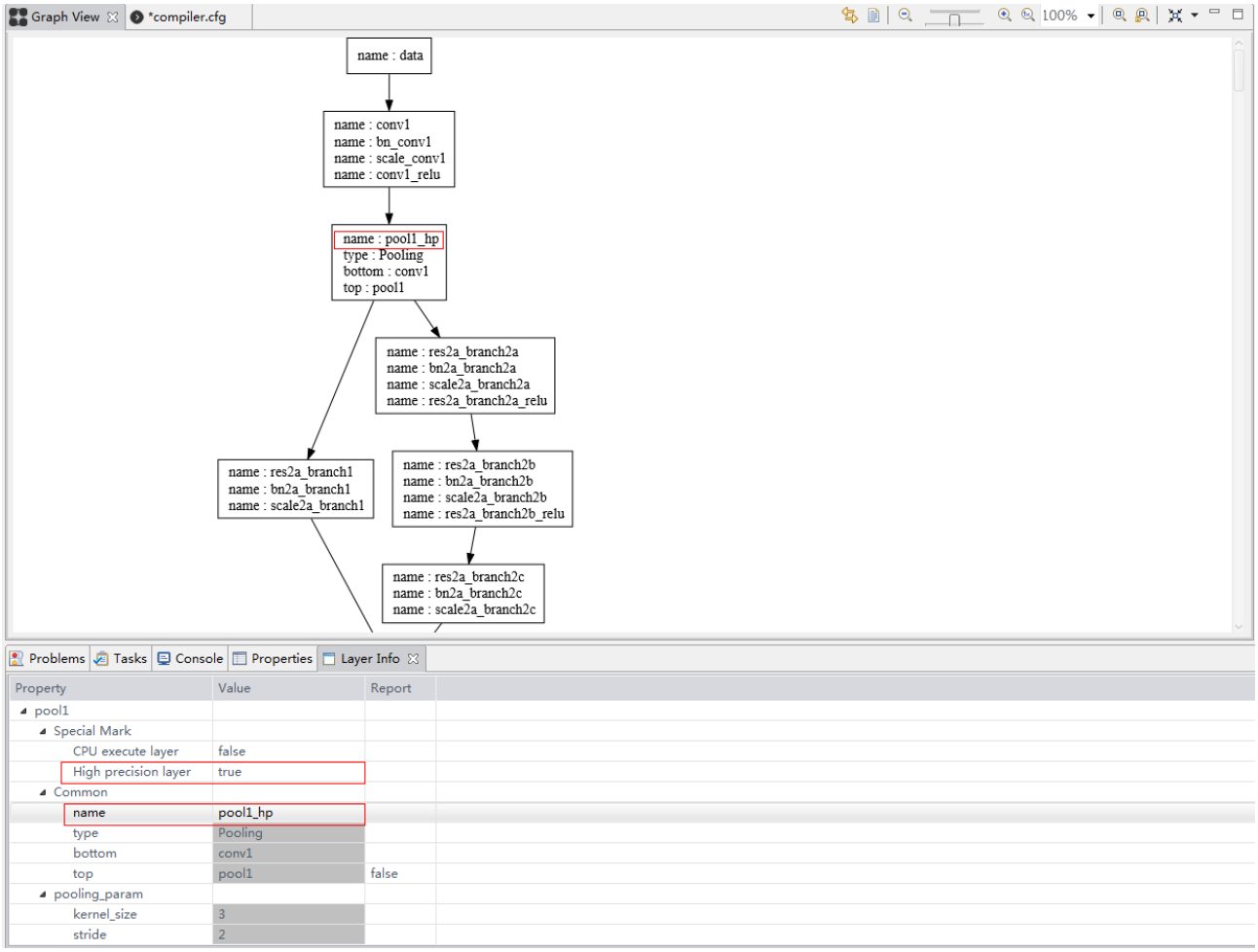
- 将当前Layer指定为CPU运算，在右侧Property视图中选择true后，工具自动修改name添加_cpu字段，如图5-66所示。

图 5-66 将当前 Layer 指定为 CPU 运算



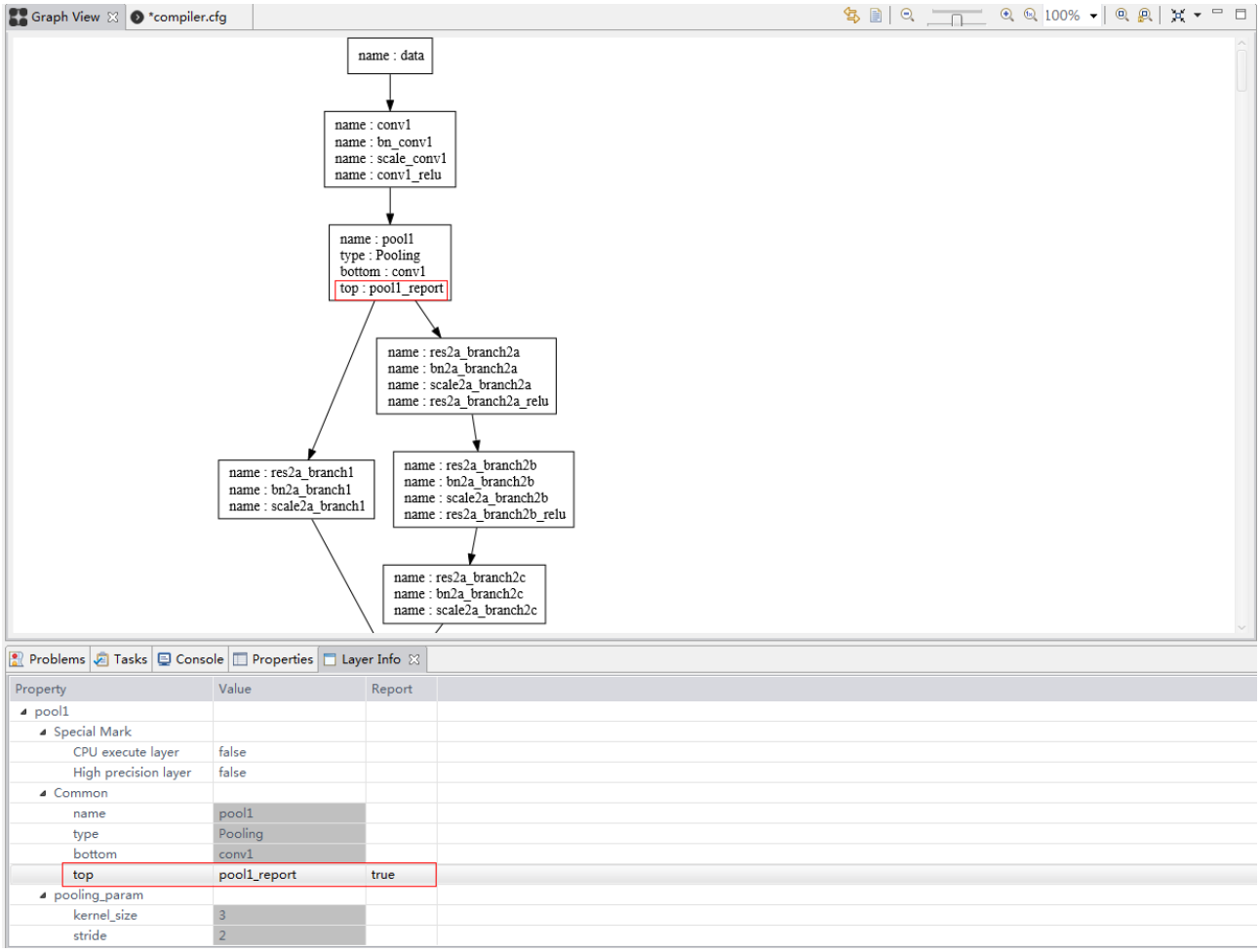
- 将当前Layer指定为高精度层，在右侧Property视图中选择true后，工具自动修改name添加_hp字段，如图5-67。

图 5-67 将当前 Layer 指定为高精度层



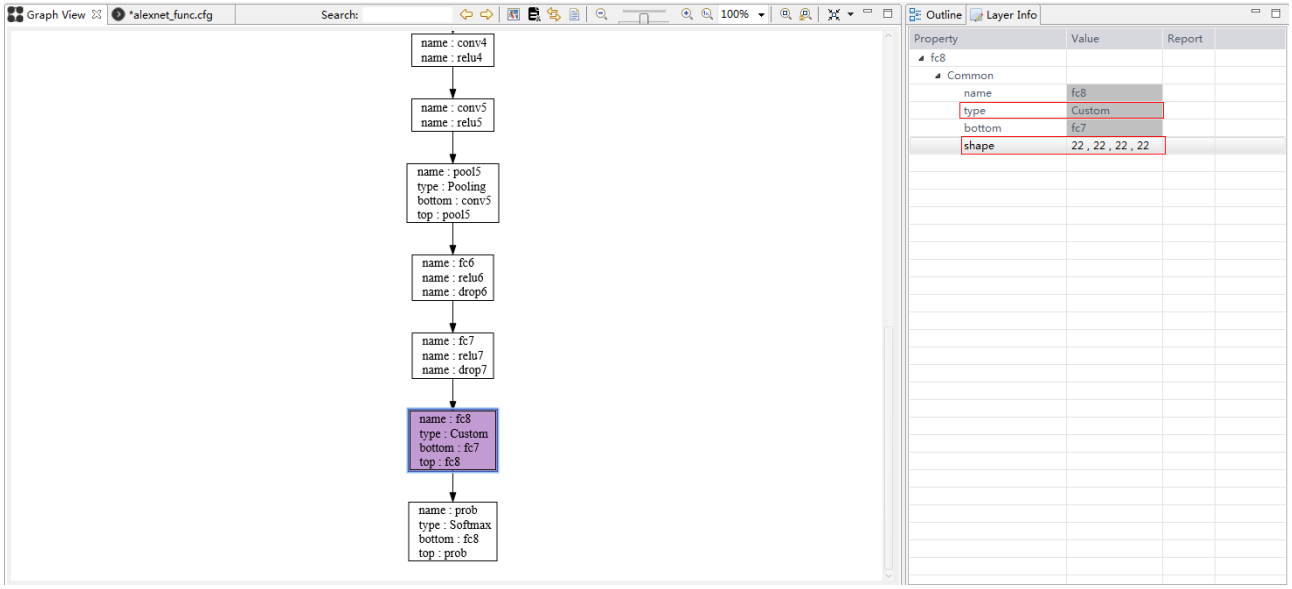
- 将当前Layer指定为中间上报层，在右侧Property视图中选择true后，工具自动修改top添加_report字段，如图5-68。

图 5-68 将当前 Layer 指定为中间上报层



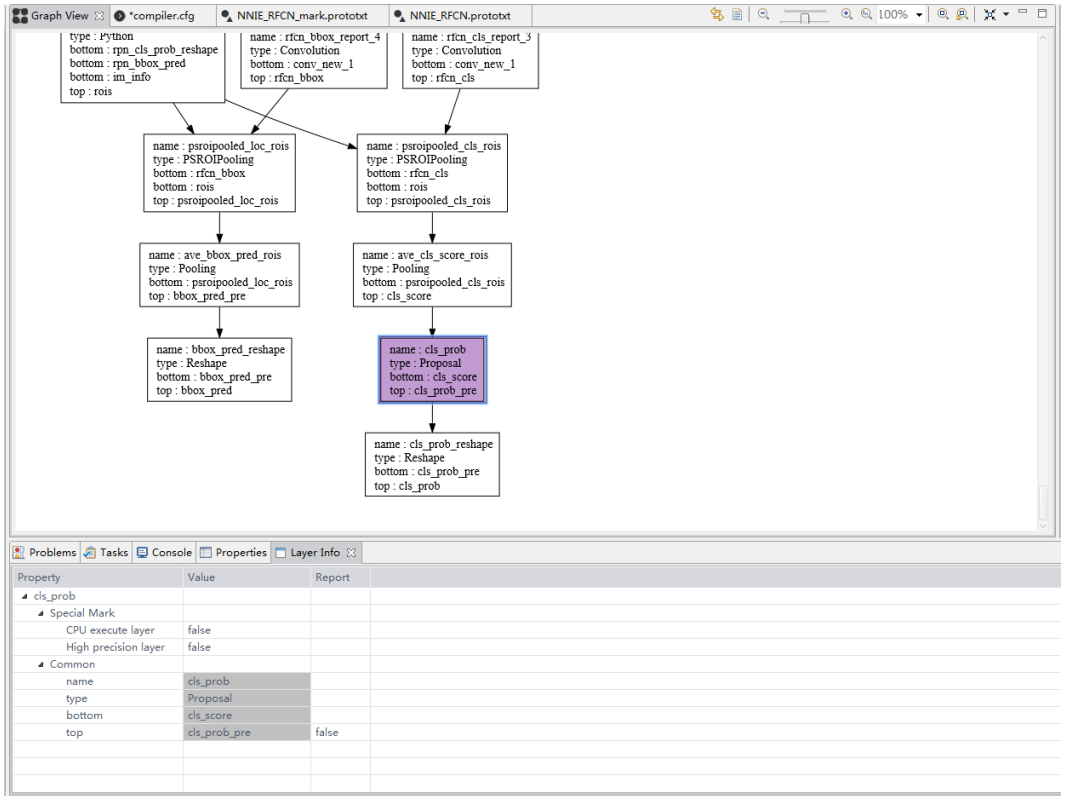
- 当前选中的Layer为Custom时，除过基本参数name、type、bottom、top，其他参数因nnie_mapper可能无法识别，工具在自动标记Prototxt时会删除这些参数，同时会新增shape参数，当前Custom层中有多少个Top属性就会对应有几个Shape参数，shape默认的参数配置为0,0,0,0，分别表示N、C、H、W，这时用户必须修改shape参数的值；nnie_mapper支持的shape参数值至少有2位，如果填0，工具默认当做此dim值为无效值不传给nnie_mapper，即（*，*，0, 0），（*，*，*，0），（*，*，*，*）此三种格式，如图5-69。

图 5-69 当前 Layer 为 Custom 层



- 当前选中的Layer为Proposal时，除过基本参数name、type、bottom、top，其他参数因nnie_mapper可能无法识别，工具在自动标记会删除这些参数，如图 5-70。

图 5-70 当前 Layer 为 Proposal 层



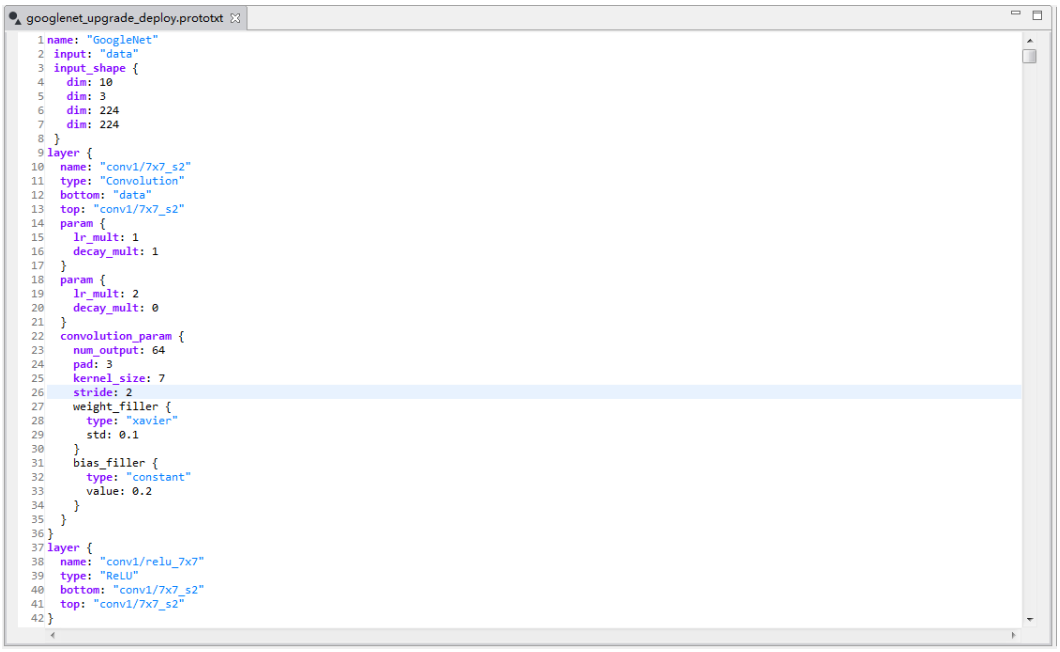
须知

- input层后的第一个层，不能标记为_cpu
- Type为input的层不能打_cpu,_hp标记
- Custom对应shape有2个值，3个值和4个值的情况，不允许非0值中间出现0值
- Custom层不能打标记
- Proposal层不能打_hp标记

5.3.6 原始 prototxt 编辑功能

在Mapper Configuration Editor视图中，点击Edit按钮，显示如图5-71，可以对原始的prorotxt进行编辑；

图 5-71 Prototxt 编辑界面



5.3.7 nnie_mapper 配置项自动生成功能

用户在编辑完原始的prototxt后，切换回Mapper Configuration Editor页面，因Browse prototxt操作有标记custom和proposal的功能，所以如果在Edit中prototxt文件有内容修改则需要重新点击Browse导入prototxt，工具会根据自动标记后的Prototxt生成动态的nnie_mapper参数界面，且有些配置按默认值配置，如图5-72所示。

自动生成的配置项包含：

1）静态配置项，即任意prototxt都会有的配置；

2）动态配置项，主要有3类：

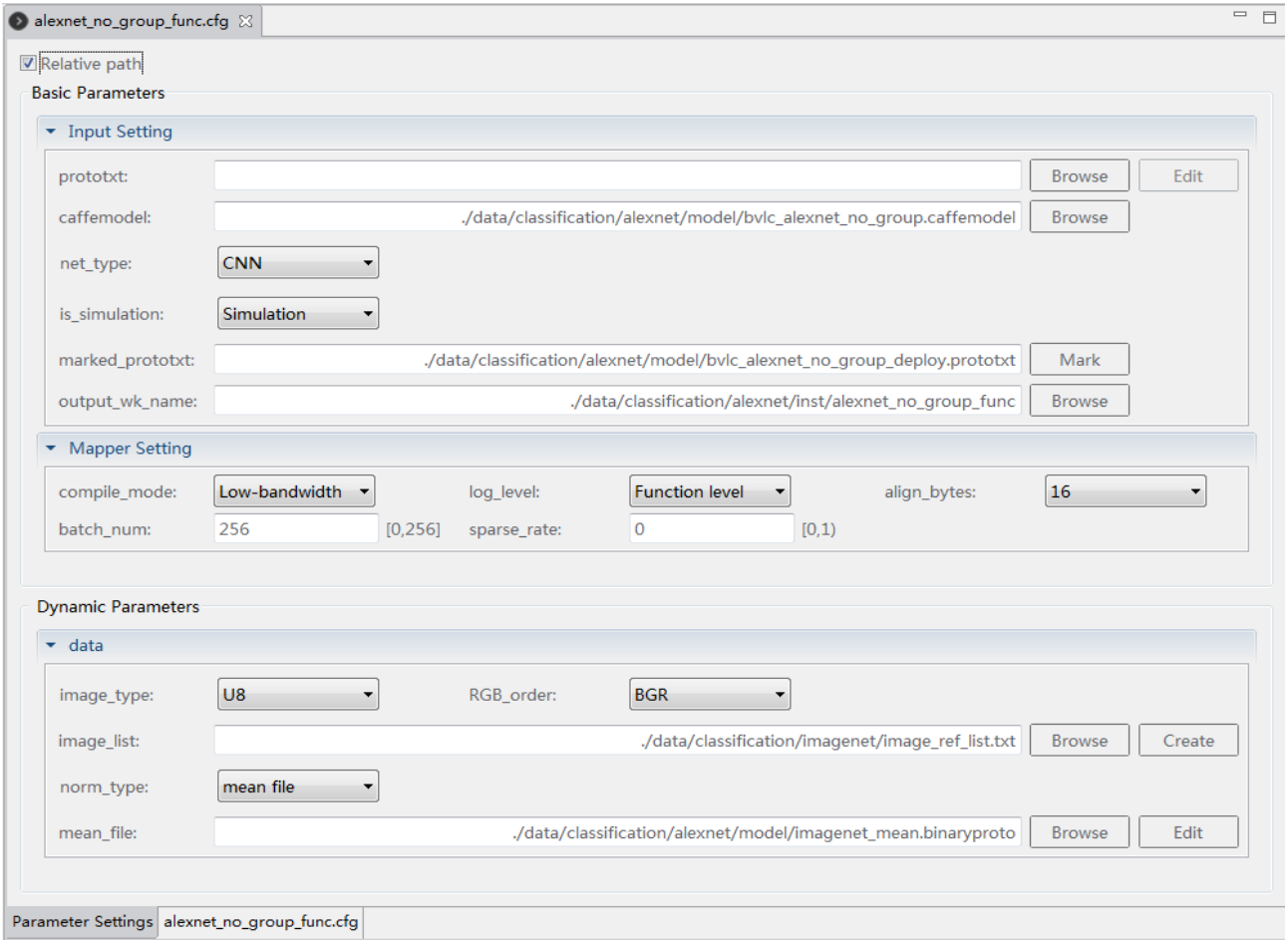
- 根据网络的输入节点数目，生成image_type\image_list\norm_type\mean_file配置项；

- 有“Proposal”时，会生成Proposal层连接的层所需要的roi_coodinate_file（Bbox）配置，“Proposal”层连接了多少个层就会有多少个roi_coodinate_file配置；该配置项配置要求见“3.5.2 配置文件说明”的roi_coordinate_file说明；
- 有“Custom”层时，会生成“Custom”层连接的层所需要的feature_map_file配置，“Custom”层连接了多少个层就会有多少个feature_map_file配置；该配置项配置要求见“3.5.2 配置文件说明”的【注意】说明。

须知

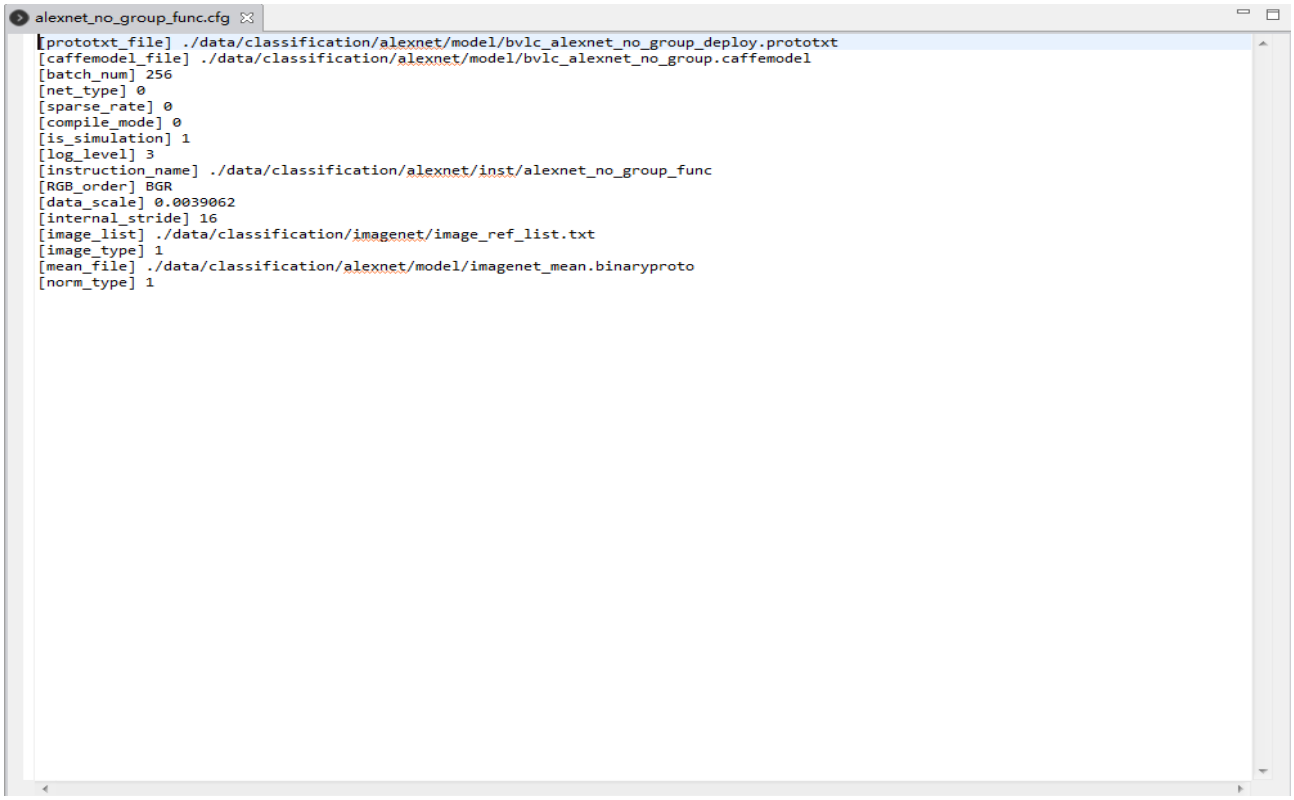
- roi_coordinate_file配置的文件，要求用户必须使用nnie_mapper中配置的参考图像，在caffe中运行该网络直到对应的Proposal层，并按“3.5.2 配置文件说明”的roi_coordinate_file格式要求输出。
- feature_map_file对应cfg文件的四个配置项image_type\image_list\norm_type\mean_file，其中默认image_type为0、norm_type为0、mean_file 为ignore，feature_map_file配置值对应image_list配置值。

图 5-72 Mapper Configuration Editor 参数界面



在Mapper Configuration Editor参数界面中设置好后，点击右下角的alexnet_mapper_func.cfg按钮，切换到Mapper Text Editor视图，可以查看cfg文件的文本配置变化，如图5-73所示。

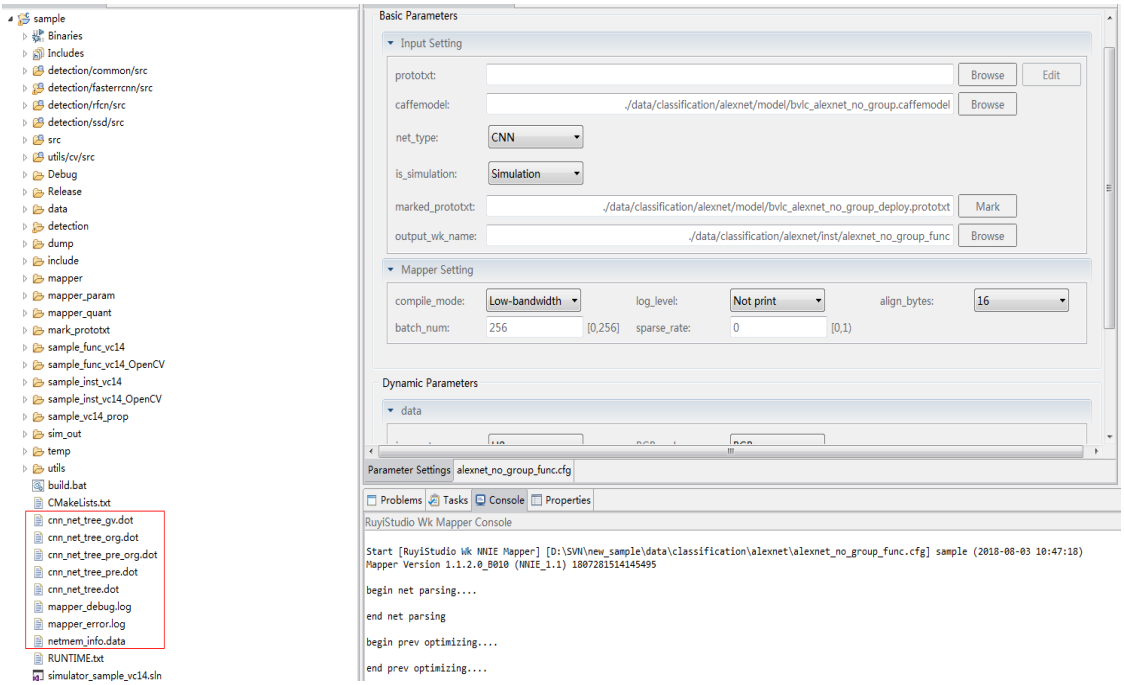
图 5-73 *.cfg Text Editor 界面



5.3.8 生成 NNIE wk 文件

配置好cfg之后，点击左上角工具栏中的Make WK按钮，工具会将当前的xxx.cfg传给nnie_mapper，转化过程会在控制台显示，转化成功或失败后会在Project Explorer中的当前工程的目录下生成多个dot文件及编译过程中的debug和error对应的log，如图5-74所示。

图 5-74 Make WK 过程的工具界面



5.3.9 Make wk 时产生的文件描述

当log_level选择为Function Level时，执行make wk时会在工程文件夹下生成mapper_quant和mapper_param两个文件，在mapper_quant中生成的数据是：

- $(layer_name)_weight-anti.float 是反量化后的weight，浮点。
- $(layer_name)_output$(layer_id)_$C_$H_$W_quant.linear.hex 是对input量化反量化，然后再做forward的output，十六进制，20.12格式。
- mapper_param中的数据是网络各层参数，来源于传入的caffemodel。

5.4 仿真 NNIE 功能

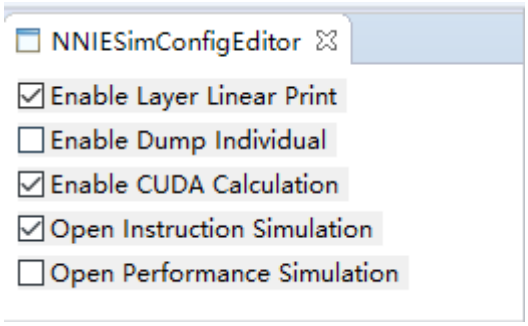
5.4.1 仿真配置文件编辑

用户双击位于工程sim_out目录下的nnie_sim.ini文件，如果没有nnie_sim.ini文件，用户可以创建一个nnie_sim.ini文件，nnie_sim.ini的文件编辑内容为：

```
#===== CONFIG =====
[LAYER_LINEAR_PRINT_EN]
0
[DUMP_INDIVIDUAL_EN]
0
[CUDA_CALC_EN]
0
[OPEN_INSTSIM]
1
[OPEN_PERFSIM]
0
```


其中0表示不可用，1表示可用，双击可以打开仿真配置文件可视化编辑器，如图5-75所示。

图 5-75 仿真配置文件可视化编辑



可以设置以下开关：

- Enable Layer Linear Print：激活此选项后，可以在进行指令仿真时输出caffe各层的中间线性结果。生成的文件在func_layer_output_linear目录下。
- Enable Dump Indivudal：激活此选项后，可将多个模型的推理结果在0,1,2…文件夹依次独立存放，彼此不覆盖。
- Enable CUDA Calculation：激活此选项后，可以在仿真时使用CUDA加速特性。需要安装CUDA Toolkit才可以使用。四个选项的下方会显示CUDA运行版本与驱动版本，显示为Invalid时，表示当前的CUDA安装无效。
- Open Instruction Simulation：激活此选项以支持指令仿真运行。
- Open Performance Simulation：激活此选项以支持性能仿真运行，如果需要激活此选项，必须先要激活指令仿真运行。

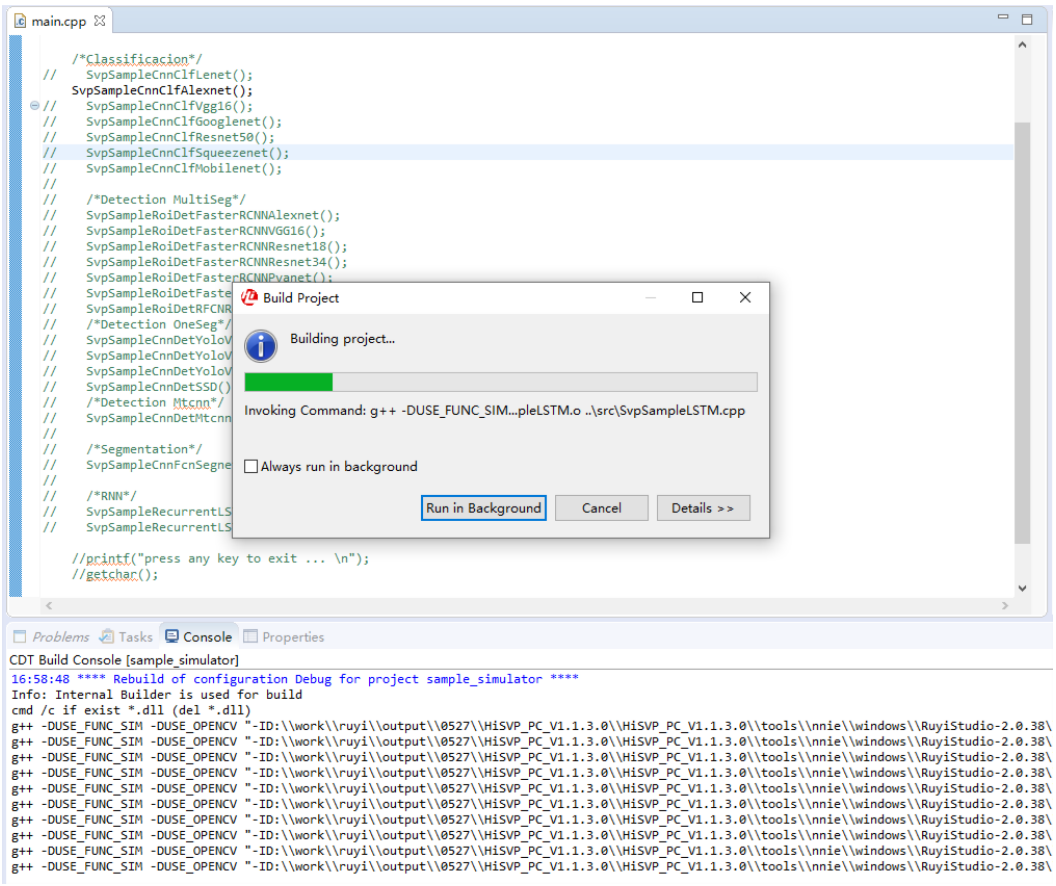
详见“3.6.4 **配置文件说明**”。

编辑完成后，点击RuyiStudio工具栏的保存按钮  可将设置保存到所选的nnie_sim.ini文件。

5.4.2 仿真工程编译与运行

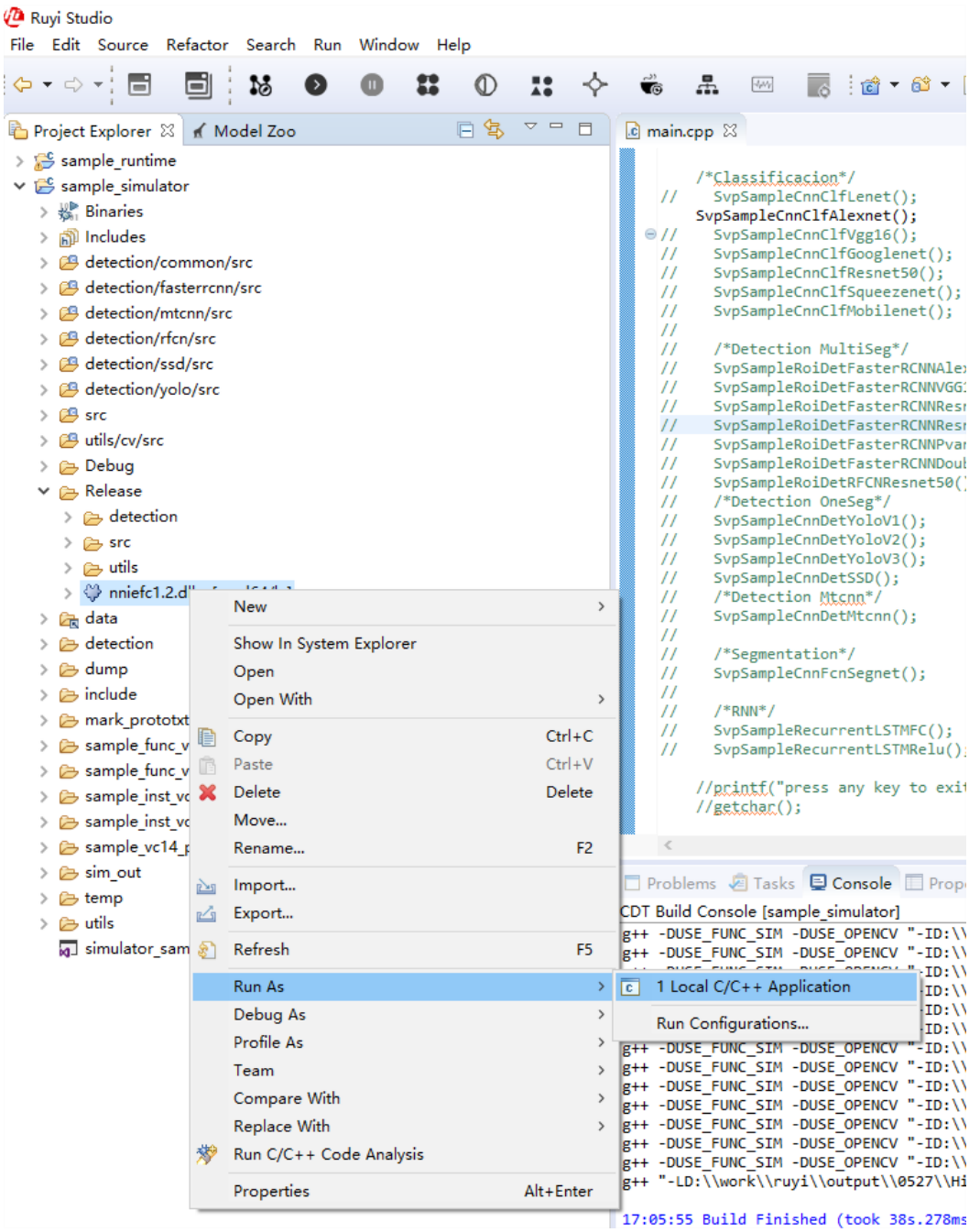
用户选择工作区的项目，双击打开src目录下的main.cpp文件，点击  按钮，选择编译Debug版本或者Release版本，点击完按钮之后会在Console视图打印当前项目的编译日志，如图5-76所示。

图 5-76 仿真工程编译



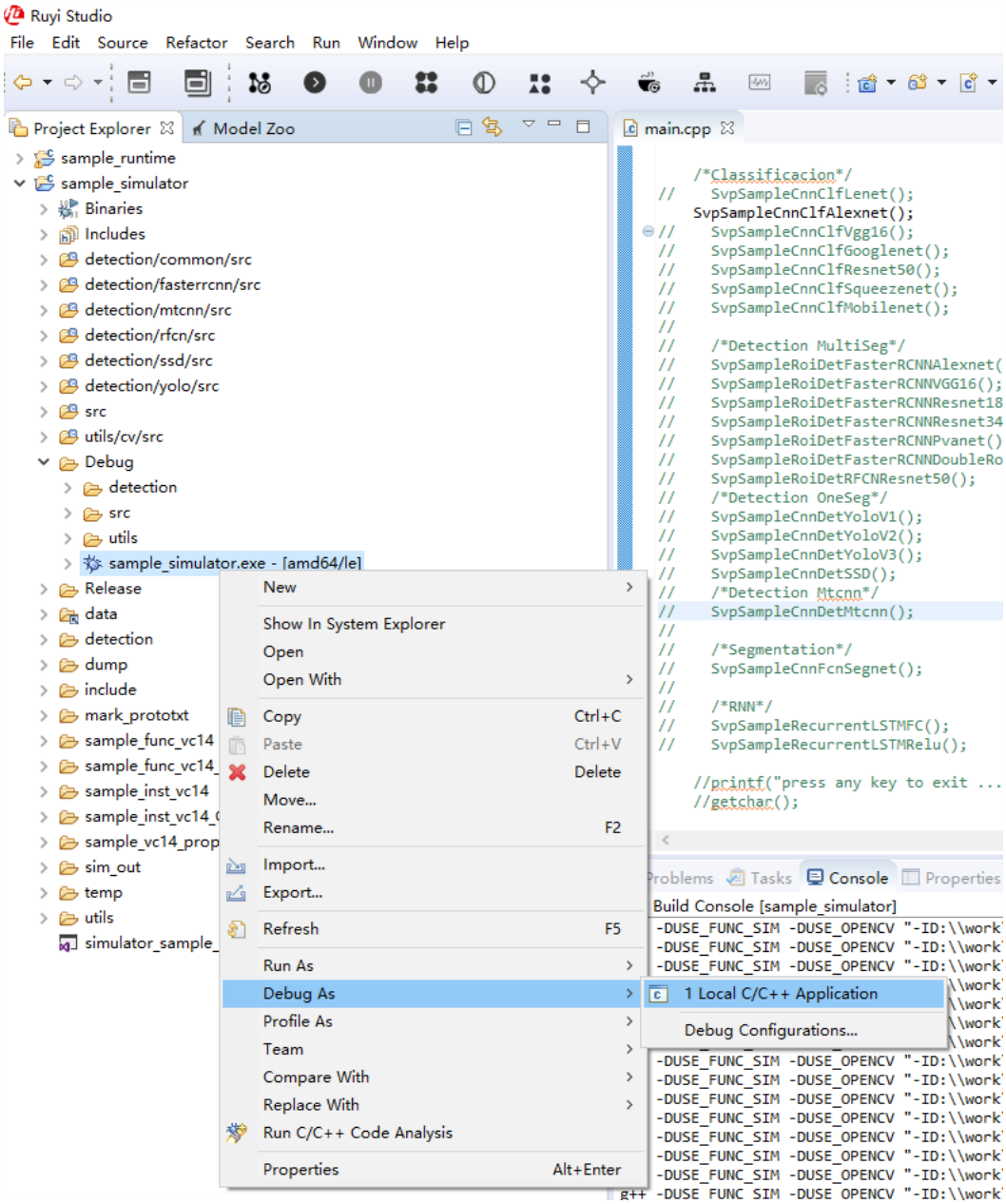
编译成功后，在项目的Debug或者Release目录会生成可执行文件xxx.exe，如图5-77所示，右键可执行文件，选择Run As->Local C/C++ Application。

图 5-77 仿真工程编译生成可执行文件



右键可执行文件，选择Debug As->Local C/C++ Application可进入断点调试。

图 5-78 仿真工程调试



5.4.3 Sample 代码中一键切换功能仿和指令仿

导入HiSVP_PC_Vx.x.x.x\software\sample工程，默认配置了功能仿的库和使用功能仿wk文件的编译宏，如果需要将sample目录从指令仿切换到功能仿，仅需要在工程上点击右键->Switch Emulation Library->Function Lib如图5-79，即可将当前的工程中Debug和Release版本对应依赖的库统一切换到功能仿库，切换过程修改的编译参数与依赖的库会在控制台中打印，如图5-80所示。

图 5-79 切换功能仿/指令仿的操作

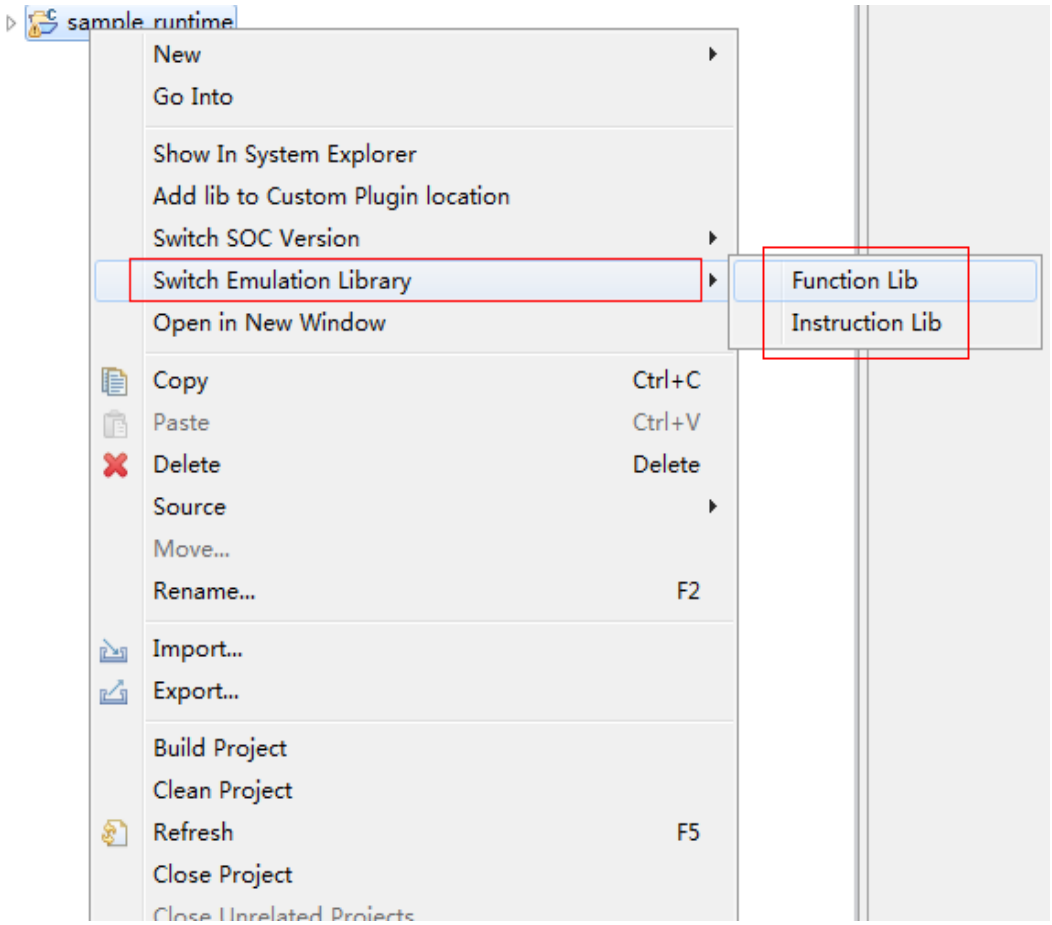
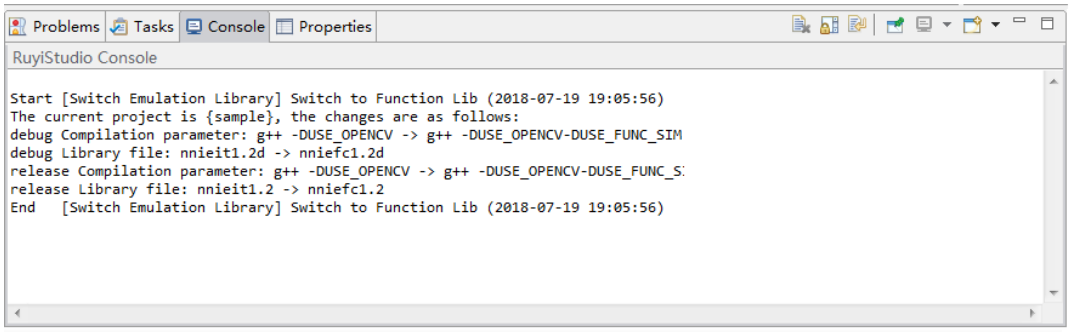


图 5-80 控制台打印切换仿真库对应的修改信息

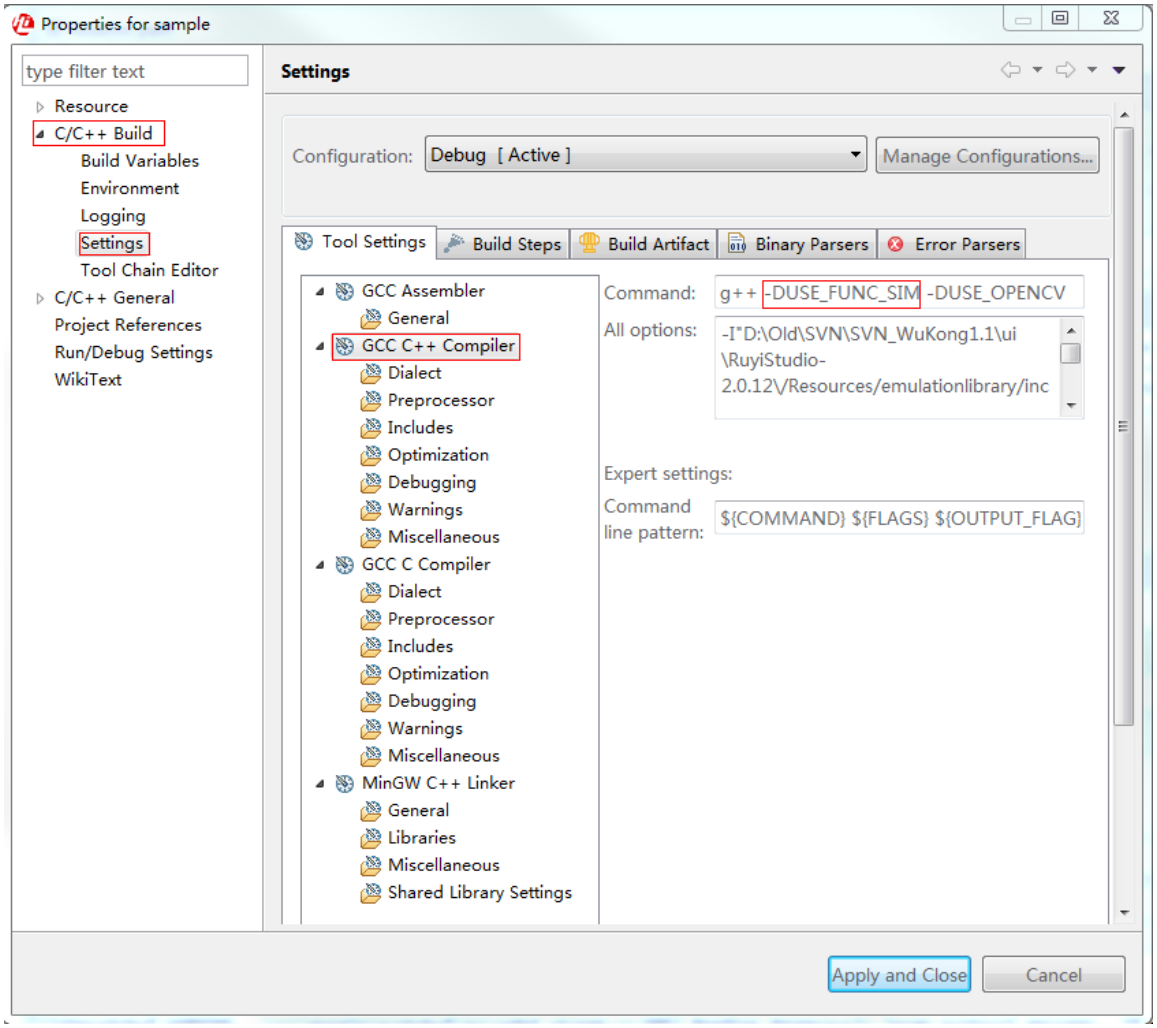


5.4.4 Sample 代码中手动切换功能仿和指令仿切换

导入HiSVP_PC_Vx.x.x.x\software\sample工程，默认配置了功能仿的库和使用功能仿wk文件的编译宏，如果需要将sample目录从功能仿切换到指令仿，需要进行如下步骤：

步骤1 在项目上点击右键->Properties->C/C++ Build->Settings->GCC C++ Compiler中Command:中去掉-DUSE_FUNC_SIM编译宏，以Debug为例，如图5-81所示。

图 5-81 修改 g++编译宏



步骤2 编译宏对应的位置在各个不同网络的sample代码中，以SvpSampleClassification.cpp为例，如图5-82所示。

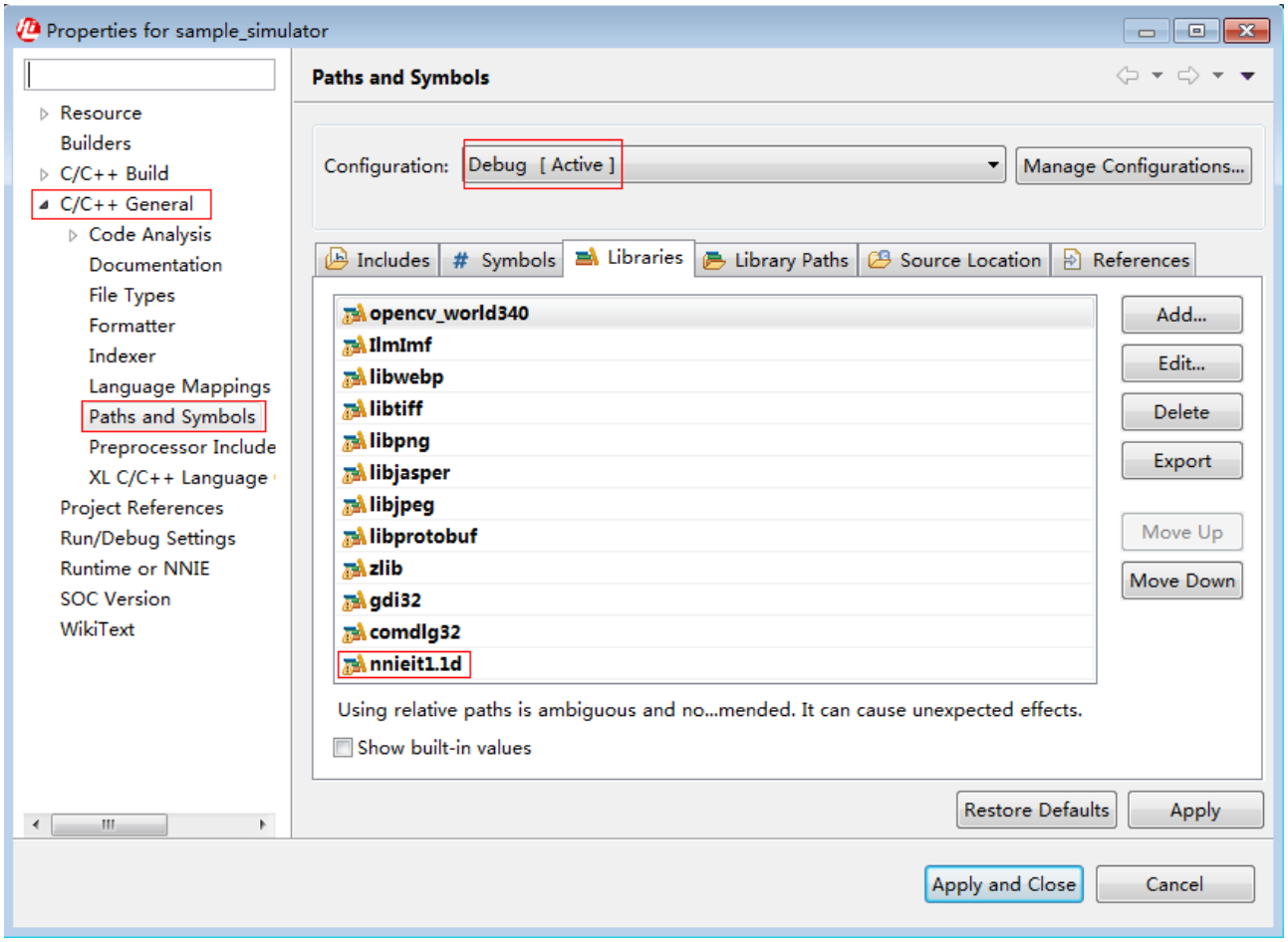
图 5-82 编译宏对应的代码位置

SvpSampleClassification.cpp

```
15
16 #if USE_FUNC_SIM
17 const HI_CHAR *g_paszModelName_c[] = {
18     "../data/classification/lenet/inst/inst_lenet_func_batch.wk",
19     "../data/classification/alexnet/inst/alexnet_no_group_func.wk",
20     "../data/classification/vgg16/inst/vgg16_upgrade_deploy_func.wk",
21     "../data/classification/googlenet/inst/googlenet_upgrade_deploy_func.wk",
22     "../data/classification/resnet50/inst/ResNet50_model_func.wk",
23     "../data/classification/squeezenet/inst/squeezenet_v1.0_func.wk"
24 };
25 #else
26 const HI_CHAR *g_paszModelName_c[] = {
27     "../data/classification/lenet/inst/inst_lenet_inst_batch.wk",
28     "../data/classification/alexnet/inst/alexnet_no_group_inst.wk",
29     "../data/classification/vgg16/inst/vgg16_upgrade_deploy_inst.wk",
30     "../data/classification/googlenet/inst/googlenet_upgrade_deploy_inst.wk",
31     "../data/classification/resnet50/inst/ResNet50_model_inst.wk",
32     "../data/classification/squeezenet/inst/squeezenet_v1.0_inst.wk"
33 };
34 #endif
```

步骤3 在项目上点击右键->Properties->C/C++ General->Paths and Symbols->Libraries，将功能仿的.a文件删除，新增指令仿的.a，如图5-83所示。

图 5-83 修改工程依赖的 Libraries



----结束

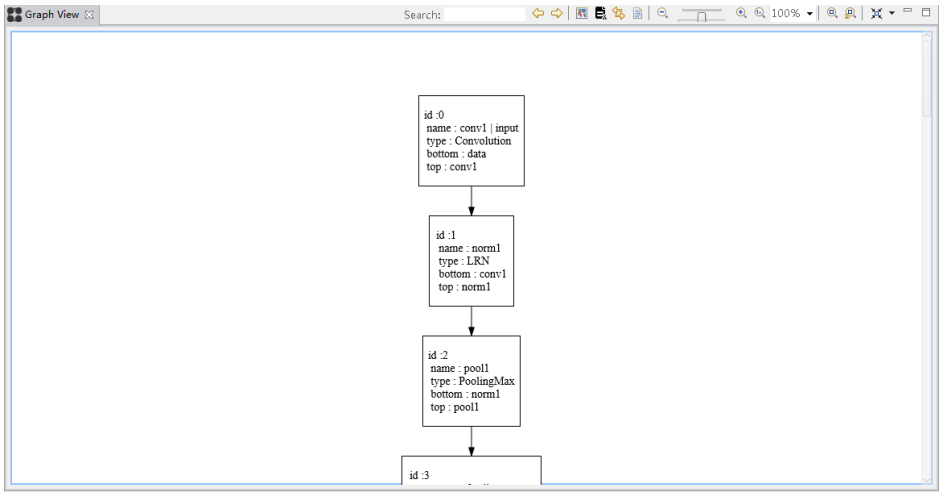
工程依赖的功能仿和指令仿存放在RuyiStudio工具目录下的Resources \emulationlibrary\lib目录中，Release版本名称为功能仿libnniefc1.1.a，指令仿libnnieit1.1.a，Debug版本名称为功能仿libnniefc1.1d.a，指令仿libnnieit1.1d.a。需要注意在步骤2中添加.a文件时需要去掉原文件名称中的lib和后缀.a。

5.5 数据分析工具

5.5.1 网络拓扑图显示功能

在工具栏中点击Graph View按钮  可以打开网络拓扑图视图，该视图可以显示由DOT视图语言描述的网络拓扑图，如图5-84所示：

图 5-84 网络拓扑图视图

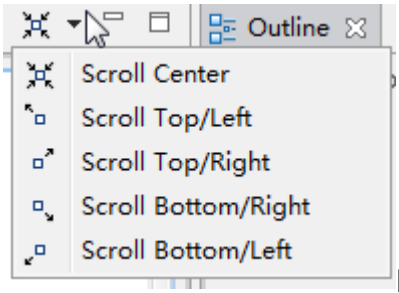


从GraphView自带工具栏中可以启用网络拓扑图的各项功能：

 联动按钮：激活联动按钮后，从Project Explorer工程视图选择dot文件，会将该dot文件显示于网络拓扑图视图中；

 打开按钮：可选择本地硬盘上的dot文件显示；

缩放工具：可以将当前的网络拓扑图缩放显示。



自动滚动工具：可以自动滚动网络拓扑图到指定的方位。


截取按钮：可以截取网络拓扑图中的一层或多层并保存数据到指定文件夹。


保存截图：可以将当前的网络拓扑图保存为png文件。

搜索工具：可以按关键字搜索指定的层。在Search文本框中输入文字，按回车键即可搜索。网络拓扑图中标签文字包含关键字的层将被高亮显示，并自动定位，在标记prototxt后的网络拓扑图上还会在Layer Info视图显示层属性。如果有多个层标签匹配，使用切换按钮  可切换层显示。

5.5.2 向量相似性比较功能

提供不同功能配置下的每一层的向量的统计量两两比较的余弦相似度，目的是方便的定位具体是哪个环节引入精度的误差等。

不同配置（建议使用）包括：

- caffe推理结果

  使用发布包中的脚本CNN_convert_bin_and_print_featuremap.py，在caffe环境下运行，会输出各层的中间文件在脚本当前路径的output目录下，例如：conv1_output0_96_55_55_caffe.linear.float的文件；

- nnie_mapper量化及反量化bypass后推理结果

  在cfg文件中配置log_level=3(Function level)后点击make wk按钮，生产wk文件，在$(project_root)/mapper_quant中也会输出各层的中间文件，例如：conv1_output0_96_55_55_quant.linear.hex

- 指令仿真推理结果

  在nnie_sim.ini配置选项[LAYER_LINEAR_PRINT_EN]，在func_layer_output_linear文件夹下也会输出各层的中间文件，例如：seg0_layer0_output0_inst.linear.hex

- 上板推理结果

  上板结果只能打印用户上报输出的层，按segid_layerid_output $id_board.linear.hex的文件格式打印（打印的sample已提供，见sample_nnie.c文件中的segnet sample，在SAMPLE_SVP_NNIE_Segnet 函数中通过调用SAMPLE_SVP_NNIE_PrintReportResult将结果打印输出）

- 功能仿真推理结果

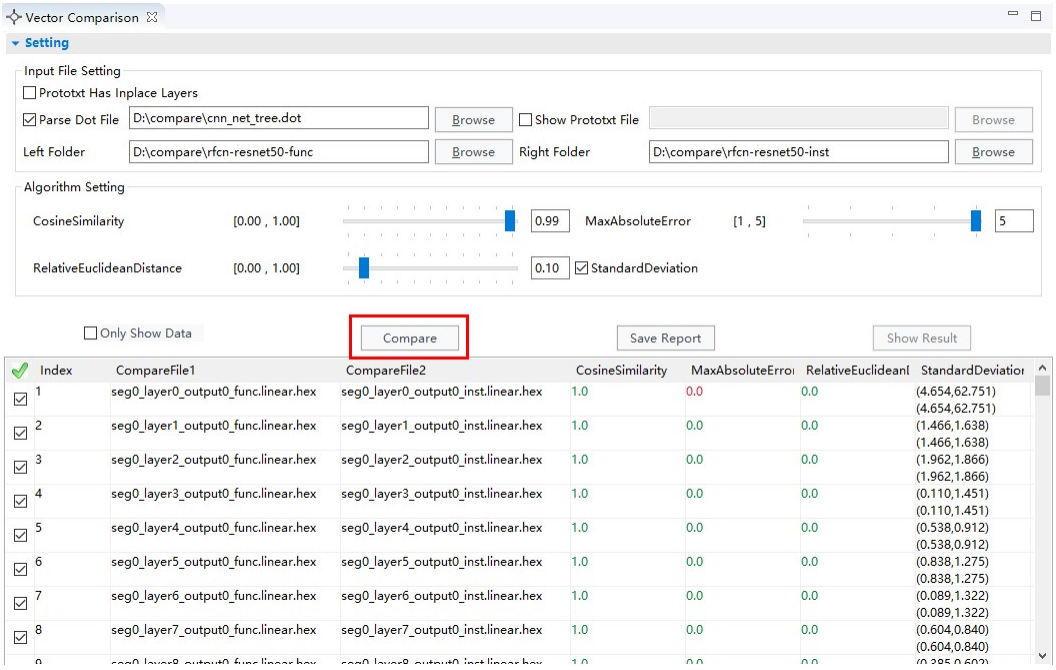
  同指令仿的配置，输出的中间文件例如：seg0_layer0_output0_func.linear.hex

5.5.2.1 展示比较结果到列表功能

操作方式如下：

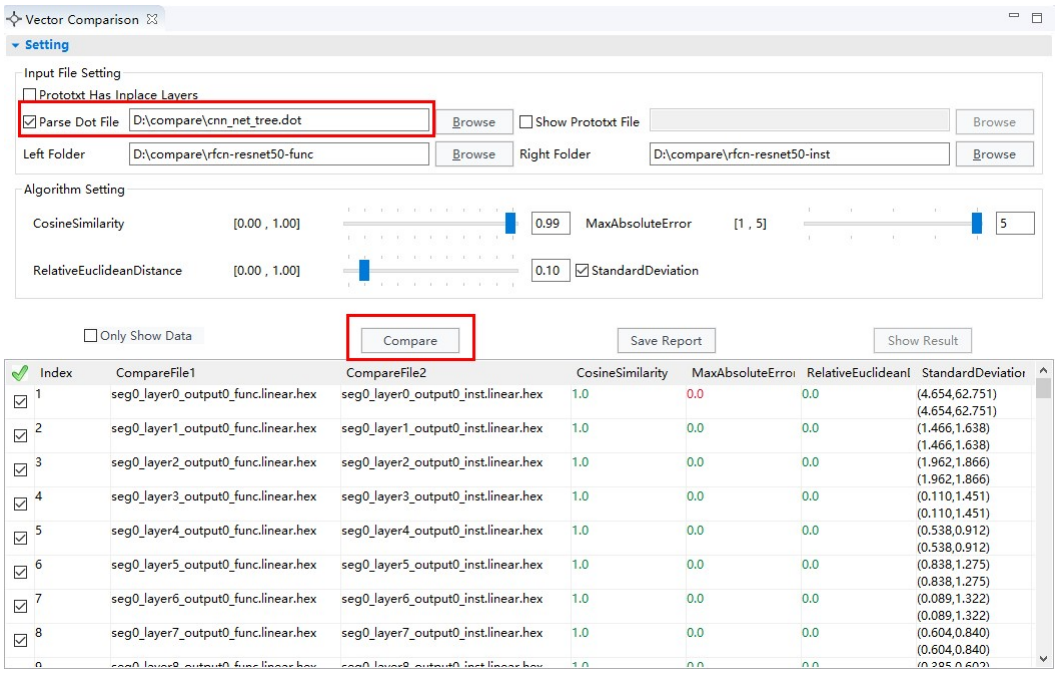
步骤1 点击工具栏向量比较按钮  ，弹出向量比较的视图，在compare folders中选择你需要比较的不同配置的中间输出结果所在的文件夹，如果其中一个是caffe或者nnie_mapper的推理结果的数据，而另一个是仿真数据，还需要额外的选择nnie_mpper生成的dot文件来进行层的匹配对应，如果Prototxt存在Inplace写法，需要勾选上Prototxt Has Inplace Layers，并选择相应的Prototxt文件。如图5-85所示。

图 5-85 向量对比视图



如果当前选择的比较文件中名称为seg开头且名称中没有具体的layer名称时，则需要先选择Parse Dot File再导入比较的文件后点击Compare进行比较，这样工具就会根据选择的dot文件解析seg文件中实际的layerName，从而匹配layerName相同的文件来进行比较，如图5-86所示。

图 5-86 选择 dot 文件的向量对比视图



步骤2 空白区域为未匹配上的文件，用户可以通过选中任意CompareFile列中的文件框来可以手动选择文件进来进行比较，点击Compare按钮，比较结果会显示到视图的列表中。

算法选择区的主要功能如下：

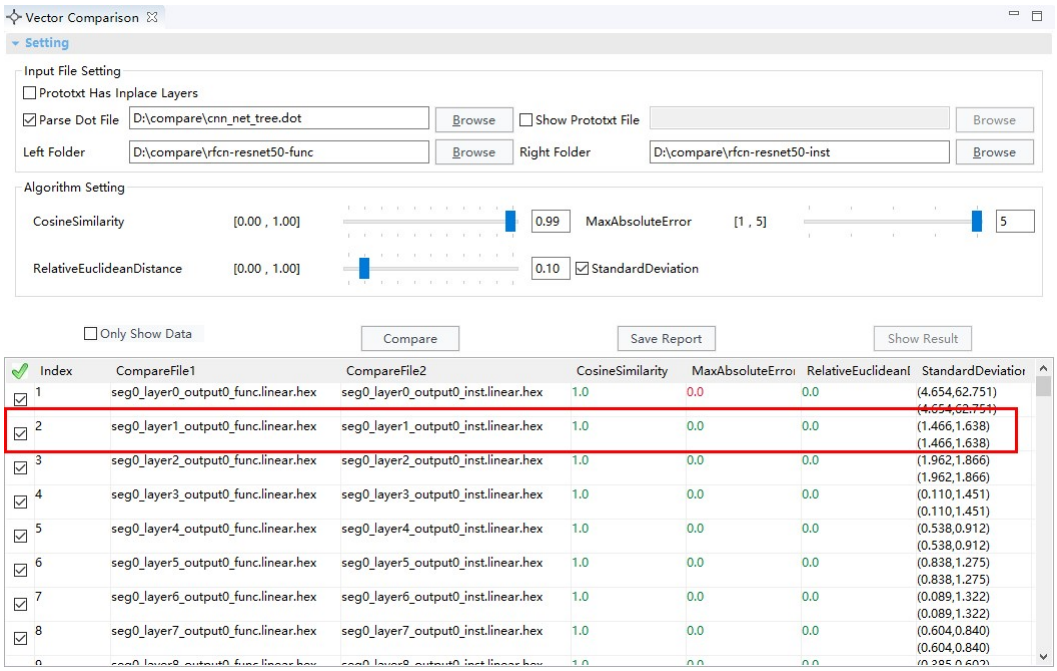
- CosineSimilarity滑块选择的阈值用于表示将小于阈值的值在比较后标记为红色。
- RelativeEuclideanDistance滑块选择的阈值用于表示将大于阈值的值在比较后标记为红色。
- MaxAbsoluteError滑块选择的阈值用于表示显示列表中最大的绝对误差的个数。
- StandardDeviation 勾选框意为，勾选时比较后显示均值与标准差的结果。

----结束

5.5.2.2 展示详细比较结果功能

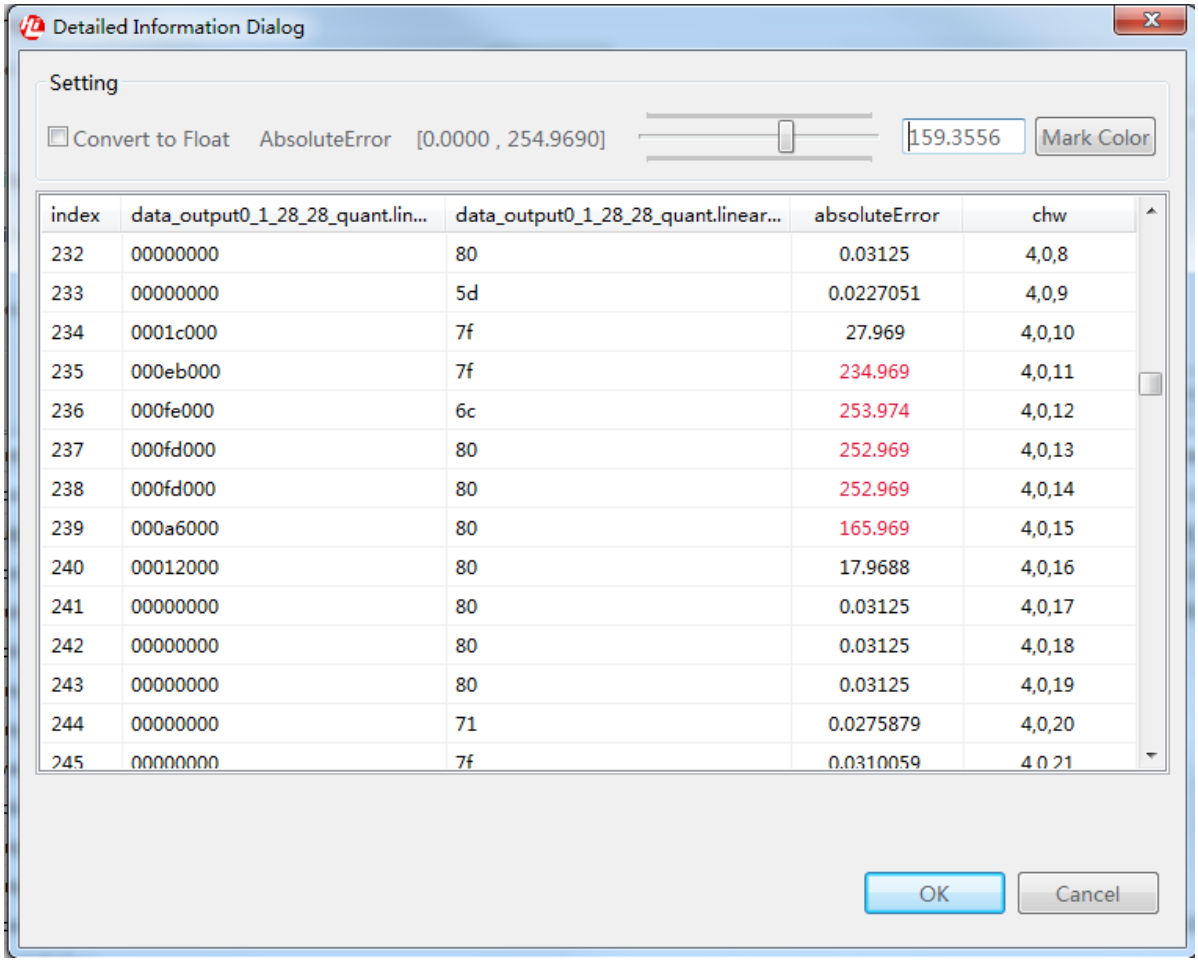
在Compare完后，双击选择需要查看的行，会弹出详细对比界面，如图5-87所示。

图 5-87 双击需要查看的行



如图5-88，得到详细比较结果，其中Convert to Float按钮功能为将列表中每行数据的hex值转换为float类型显示，方便用户查看。另外，绝对误差对应的滑块和文本框控件用于选择将一个阈值，大于此阈值的absoluteError值再点击Mark Color会被标记为红色。

图 5-88 详细比较结果



5.5.2.3 展示比较结果到 GraphView 功能

当用户选择了Show Prototxt File时，点击Compare比较成功后，再点击Show Result按钮，如图5-89所示，结果会在工具中的console和Graph View展示结果，预期结果>=0.99不标色；(0.95~0.99)给出橙色提示；<=0.95给出红色预警，如图5-90所示。

图 5-89 点击 Show Result 按钮

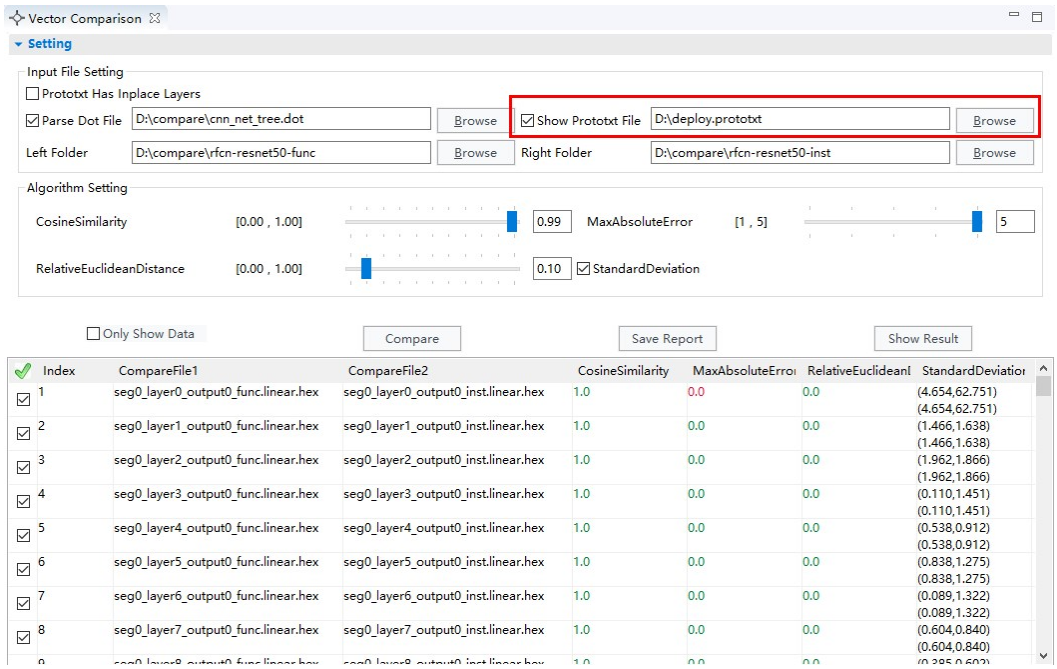
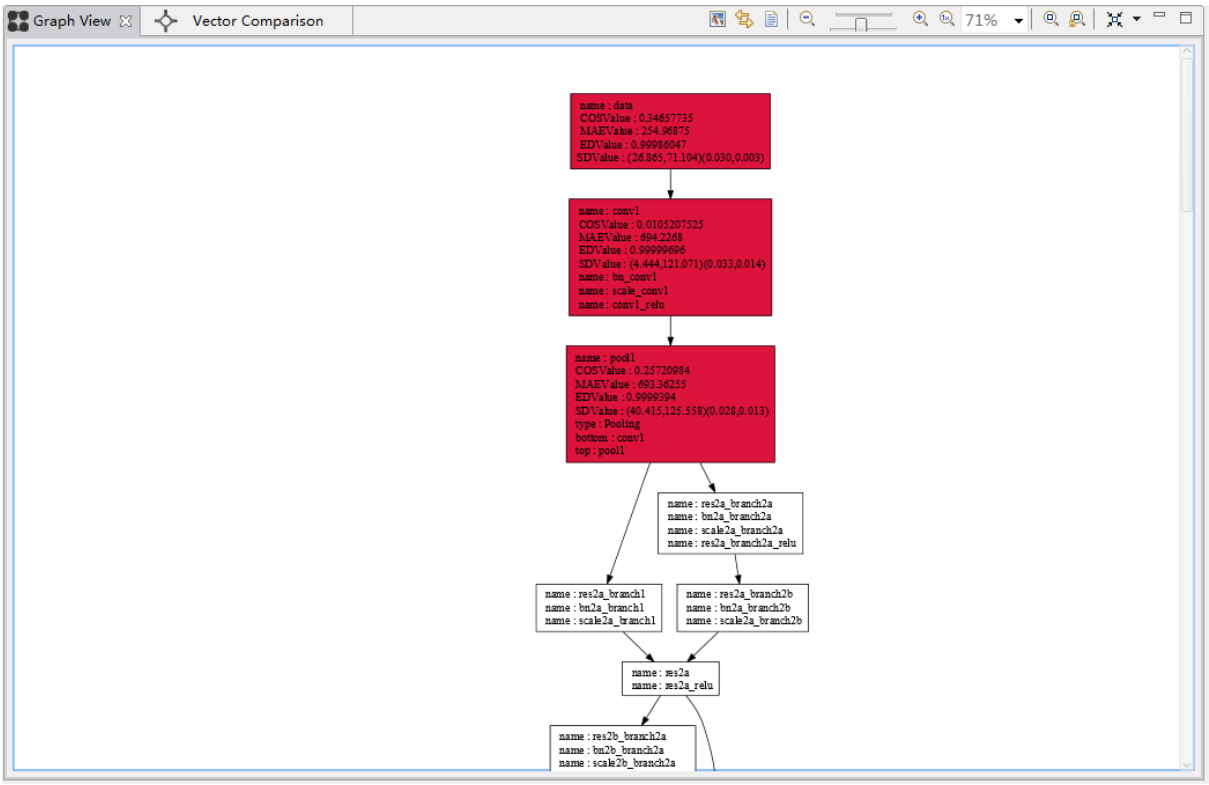


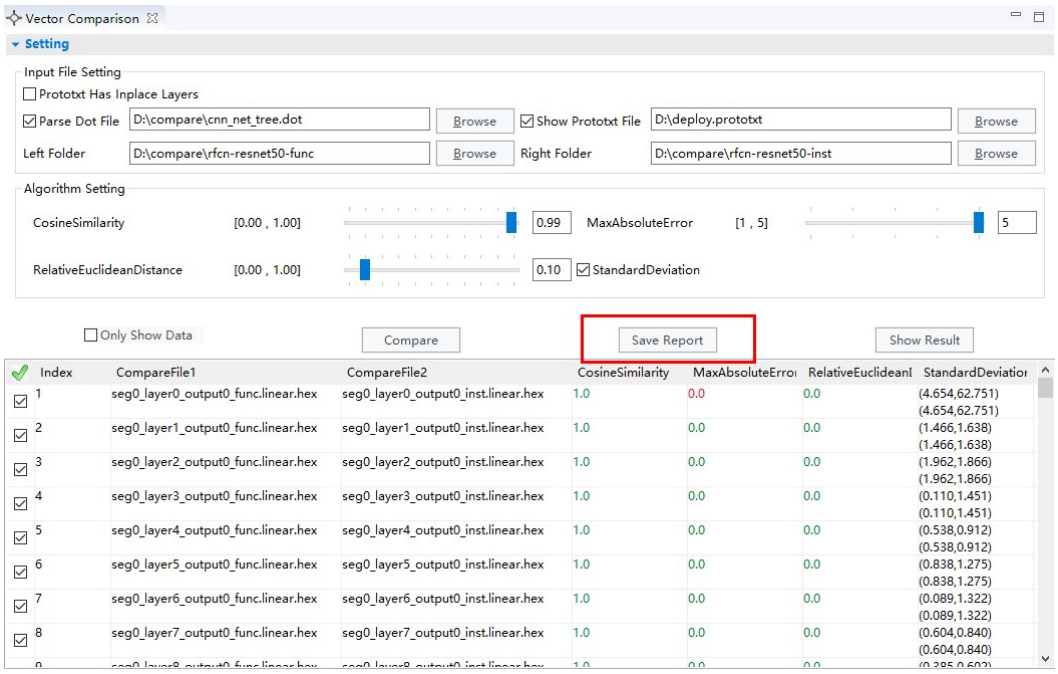
图 5-90 Graph View 显示对比结果



5.5.2.4 保存比较结果生成 csv 报告功能

点击Save Report按钮如图5-91，并选择需要保存报告的路径，点击确定即可将当前列表中的数据保存到csv文件中以备查看。

图 5-91 保存 csv 格式的报告



5.5.3 调试定位信息获取功能

背景

在用户使用了相似度比较工具确认精度问题不是nnie_mapper量化导致的，且定位到某个或者某些非用户自定义层出现精度下降问题（一种情况是断崖式下降，一种情况是相似度不是断崖式的下降，精度逐渐下降或者忽高忽低但是差异不大但整体偏差精度较大），需要将怀疑有问题的层的网络数据发送给上海海思定位。

此功能提供两种导出场景：

- 从首层开始到问题层的数据，包括：
  - 从data层开始到出问题层的test.prototxt；
  - 从data层开始到出问题层的weight\bias数据，或者是截断的test.caffemodel（前者用工具来转为test.caffemodel）；
  - 给NNIE的首层数据（这里是data层的）input.bin和input.txt (inpu.bin是二进制，input.txt每行表示一个8bit的数据，共有w*h*c行，input就是test.jpg中的裸数据，是为了规避caffe和nnie解析test.jpg时的差异)；
  - 跟网络预处理方式相关的文件test.binaryproto或者mean.txt等；
  - 提供一个Readme说明相关数据情况，包括数据格式、当前层的问题情况说明等有用的信息。
- 提供中间层数据，包括：

- 相关层test.prototxt，示例中为norm1->conv2->pool2层的prototxt；
- 前一层输出的shape信息，示例中为relu1层输出shape信息(c, h, w)；
- 相关层的weight\bias参数，或者是截断的test.caffemodel，前者需要一个工具来生成caffemodel，示例中 指norm1->pool1->conv2的weight\bias参数；
- 给NNIE的上板输入board_input.bin和board_input.txt ，示例中为上板跑relu1的输出，borad_input.bin是二进制打印，board_input.txt每行表示一个32bit的定点数据，共有w*h*c行；
- 给nnie_mapper量化输入的featuremap.txt，客户在使用nnie_mapper转化网络时，配置log_level=3；由于conv1->relu1是inplace写法(一个进一个出，进出的name相同)，在mapper运行生成的文件中找到conv1_output0.linear.hex（ ）作为featuremap.txt；若是conv1->relu1是non-inplace写法，则在mapper运行生成的文件中找到relu1_output0.linear.hex作为featuremap.txt；
- 提供一个Readme说明相关数据情况，包括数据格式、当前层的问题情况说明等有用的信息。

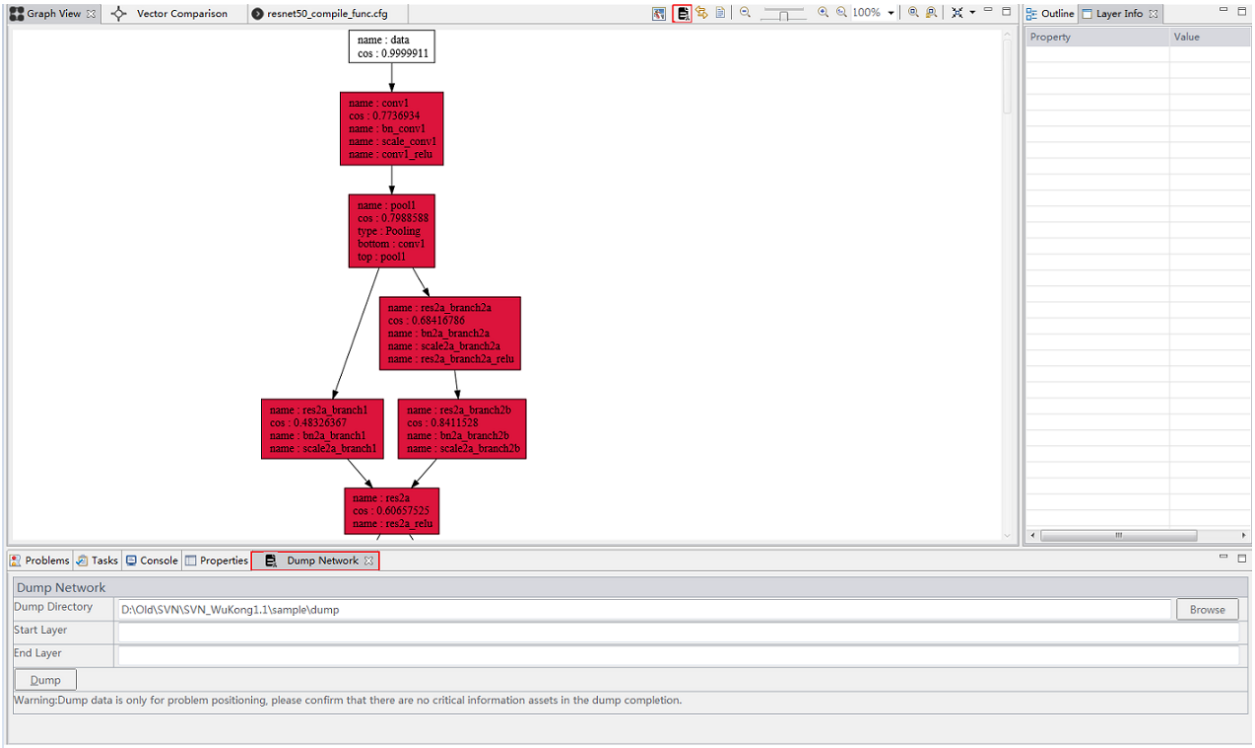
须知

只支持截取一个单输入（起始节点）和单输出（结束节点）的有向无环网络，否则不支持dump。

操作方式如下：

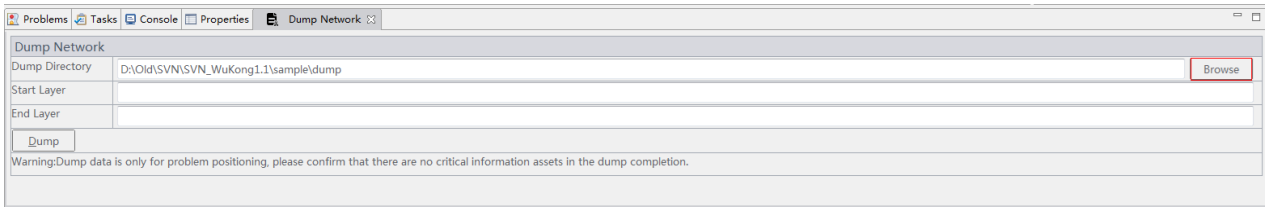
步骤1 打开Dump Network视图：在向量比较视图中点击“Show Result”按钮后，工具会将算法的值标记在Graph View视图对应的layer上，点击“Graph View”视图的右上角“Dump Network”按钮  ，在工具界面视图下侧弹出Dump Network视图，如图5-92所示。

图 5-92 Dump Network 视图



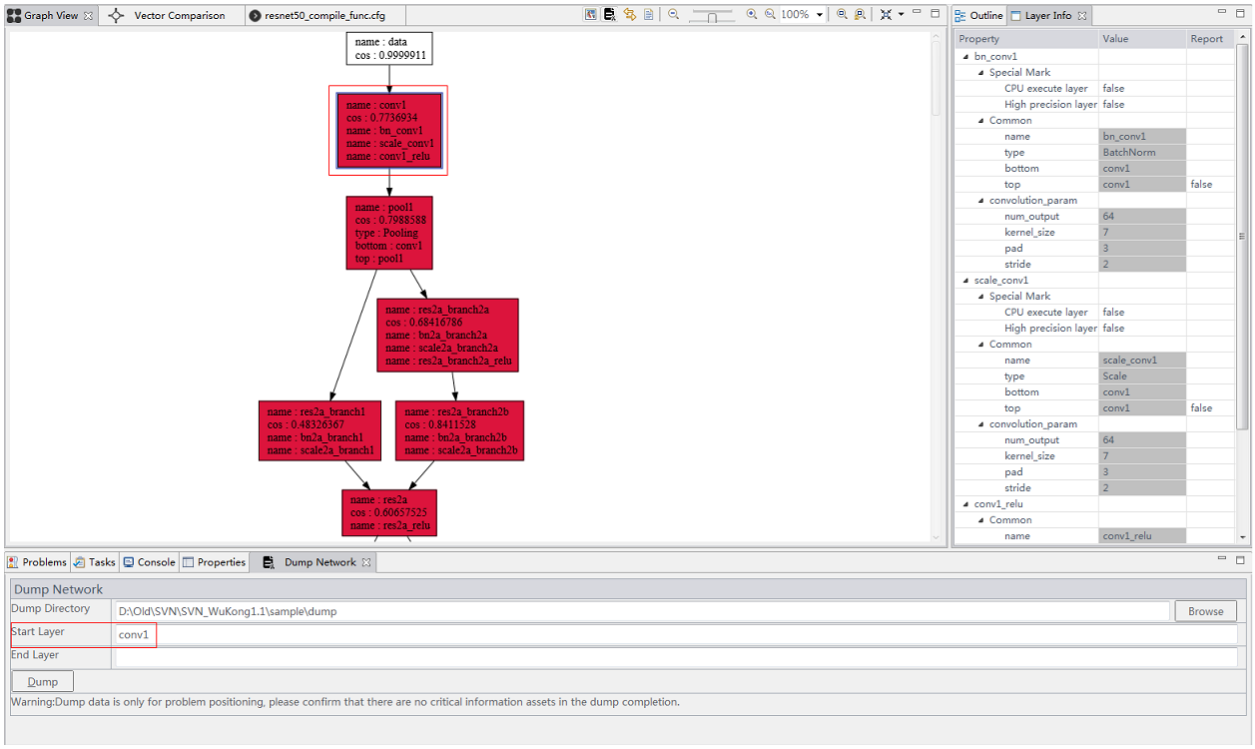
步骤2 选择保存导出数据的文件夹：当用户默认选择了工程路径时，Dump Directory会存在默认路径再当前工程目录下的dump文件夹，用户可点击“Browse”按钮自行配置导出数据的目录，如图5-93所示。

图 5-93 选择保存导出数据的文件夹



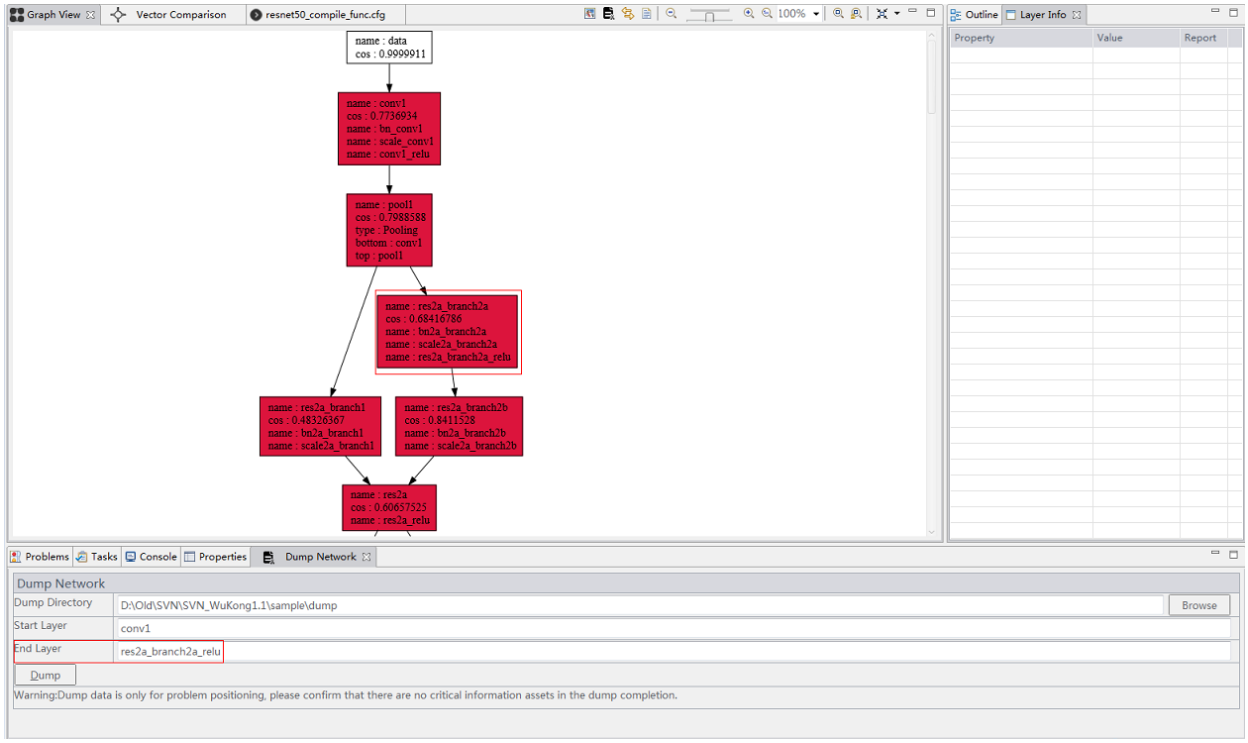
步骤3 选择导出的开始层：选中“Start Layer”文本框，然后在Graph View中双击选择需要导出的开始层时，以conv1为例，如图5-94所示。

图 5-94 选择需要 Dump 的开始层



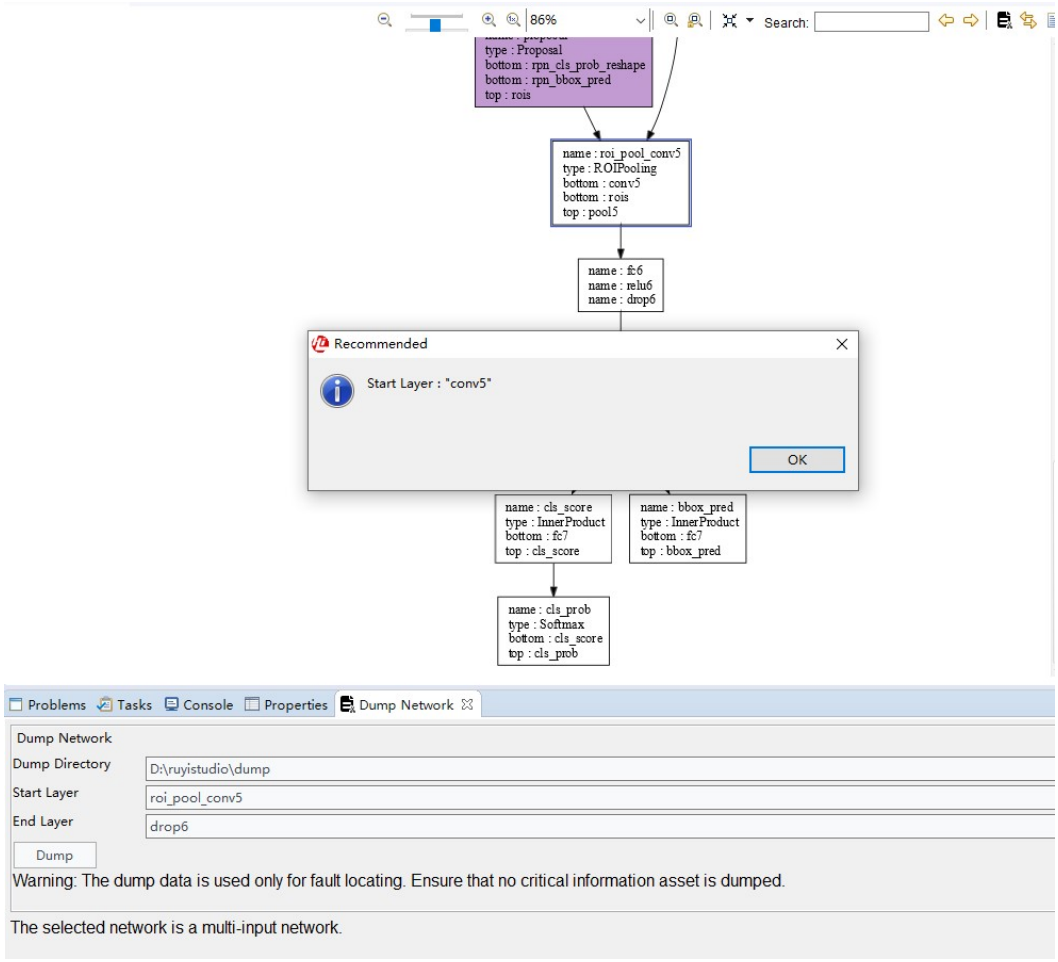
步骤4 选择导出的结束层：选中“End Layer”文本框，然后在Graph View中双击选择需要导出的结束层时，以res2a_branch2a为例，如图5-95所示。

图 5-95 选择需要 Dump 的结束层



步骤5 工具检测当前用户选择的Dump层，当选择的起始节点或结束节点不满足单输入单输出的节点时，工具会给出提示，并给出建议的导出层，如图5-96所示。

图 5-96 Dump 提示推荐层



步骤6 查看导出的dump信息，以导出的说明信息文件为例，如图5-97所示。

图 5-97 Dump 导出的信息说明

dumpInfo.txt

```
The following files were exported:
1. feature_map file: data_output0_3_224_224_quant.linear.hex
2. feature_map file: data_output0_3_227_227_quant.linear.hex
3. prototxt file: data->prob.prototxt
4. The bias/weight/param of each layer after parse:
(1) conv1:
    bn_conv1_blob0_param.float;
    bn_conv1_blob1_param.float;
    bn_conv1_blob2_param.float;
    conv1_blob0.float;
    conv1_blob0_weight.float;
    conv1_blob1.float;
    conv1_blob1_bias.float;
    scale_conv1_blob0_weight.float;
    scale_conv1_blob1_bias.float;
(2) bn_conv1:
    bn_conv1_blob0_param.float;
    bn_conv1_blob1_param.float;
    bn_conv1_blob2_param.float;
(3) scale_conv1:
    scale_conv1_blob0_weight.float;
    scale_conv1_blob1_bias.float;
(4) conv1_relu:
(5) pool1:
(6) res2a_branch1:
    res2a_branch1_blob0_weight.float;
(7) bn2a_branch1:
    bn2a_branch1_blob0_param.float;
    bn2a_branch1_blob1_param.float;
    bn2a_branch1_blob2_param.float;
```

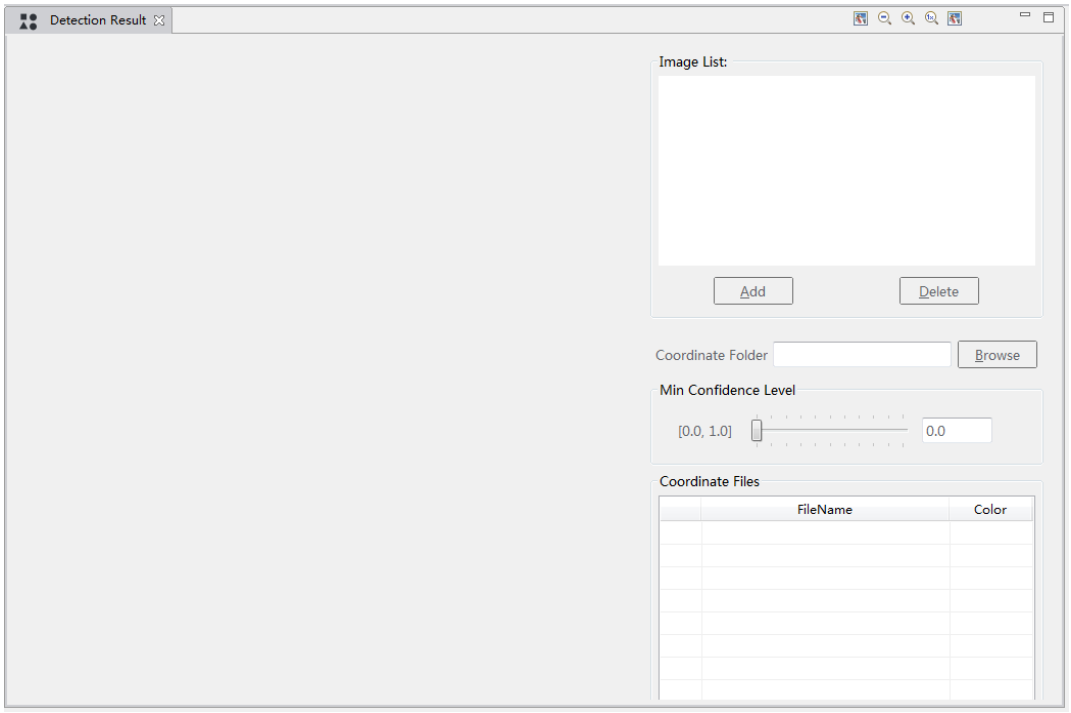
----结束

5.5.4 目标检测功能

运行目标检测视图用来查看框选结果，操作如下：

步骤1 点击左上角工具栏的目标检测按钮，通过add或delete按钮添加或删除需要检测的图片加入到Image List列表中，add支持单选和多选，delete支持ctrl/shift选中列表中的ImageList后删除点击Detection Result按钮，打开Detection Result视图，如图 5-98。

图 5-98 目标检测界面



步骤2 导入坐标文件夹Coordinate Folder，Coordinate 文件的规则如下：

坐标文件首行表示图片缩放到模型输入要求的像素宽和高，从第二行开始，每行由6列组成，每列分别对应图片名、类别、置信度、选框左上角x坐标、选框左上角y坐标、选框右下角x坐标、选框右下角y坐标。以horse_dog_car_person_224x224_roi_pos_out_ALL.txt为例，如图5-99。

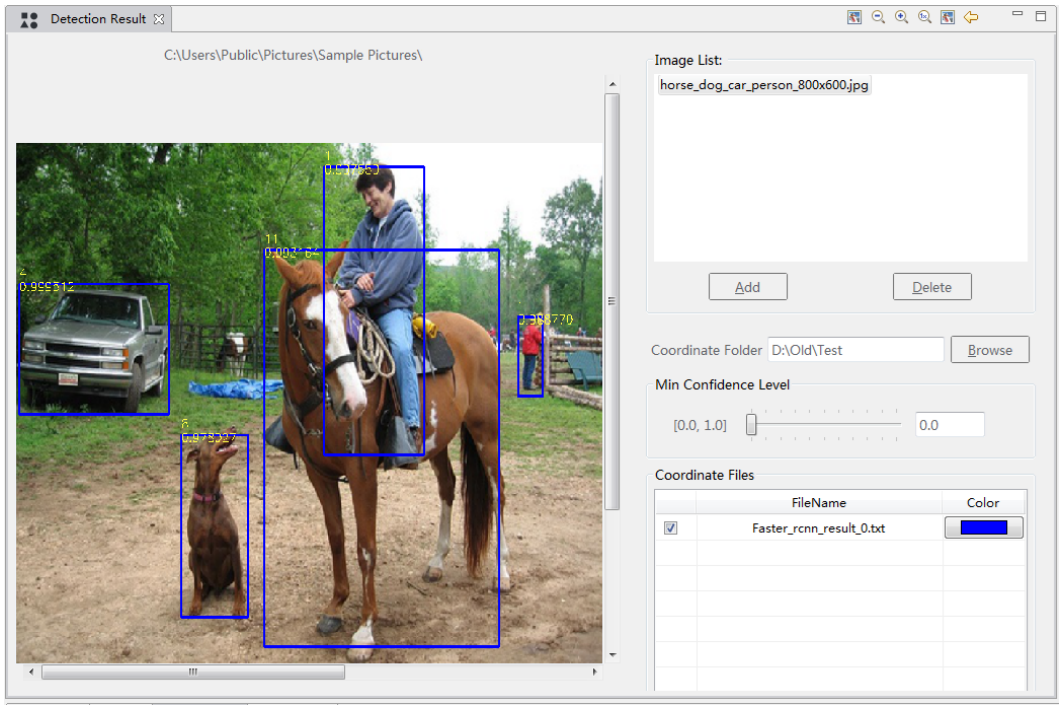
图 5-99 坐标检测文件示例

horse_dog_car_person_224x224_roi_pos_out_ALL.txt

```
1  224  224
2  horse_dog_car_person_224x224     7  0.92163086    0   59   62  107
3  horse_dog_car_person_224x224    13  0.81738281   90   30  185  215
4  horse_dog_car_person_224x224    13  0.46826172  150   70  192  192
5  horse_dog_car_person_224x224    15  0.94116211  115    6  159  142
6  horse_dog_car_person_224x224    15  0.62402344  192   67  208  111
```

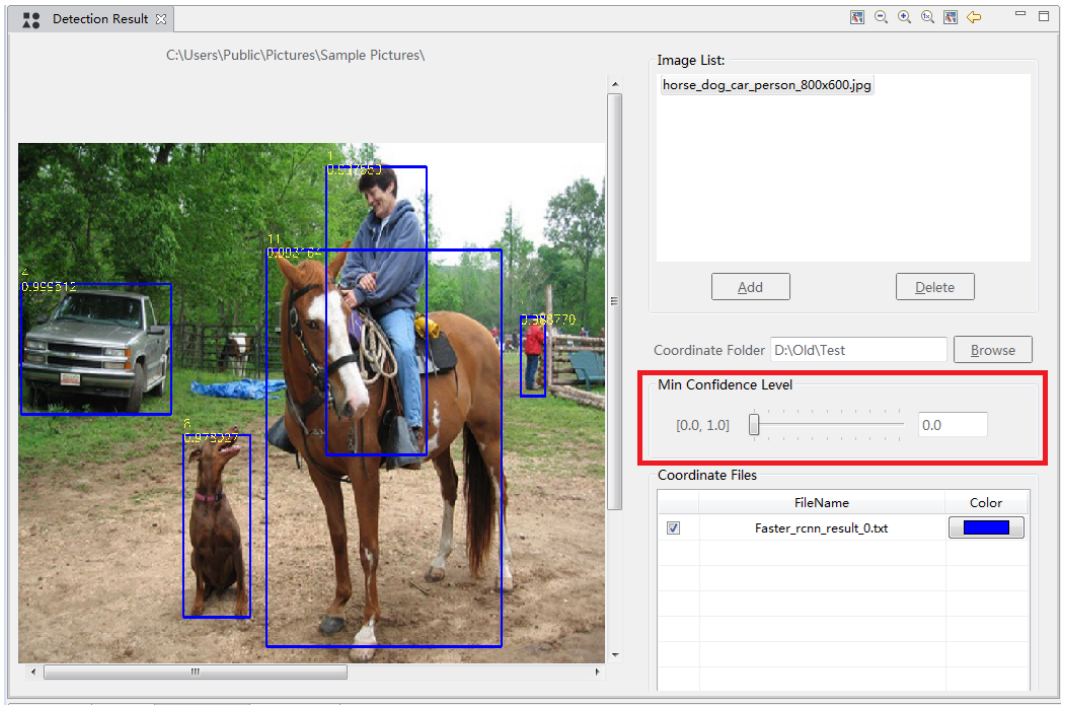
步骤3 导入坐标文件后，坐标文件会与当前的imagelist中的图片的名称进行匹配，匹配到的坐标文件才会显示到Coordinate Files里，如图5-100。

图 5-100 框选结果



步骤4 动态调整显示框图的最小置信度，通过修改滑块，可以支持根据选择的最小置信度来过滤框选图，如图5-101；

图 5-101 修改置信度后的框选结果



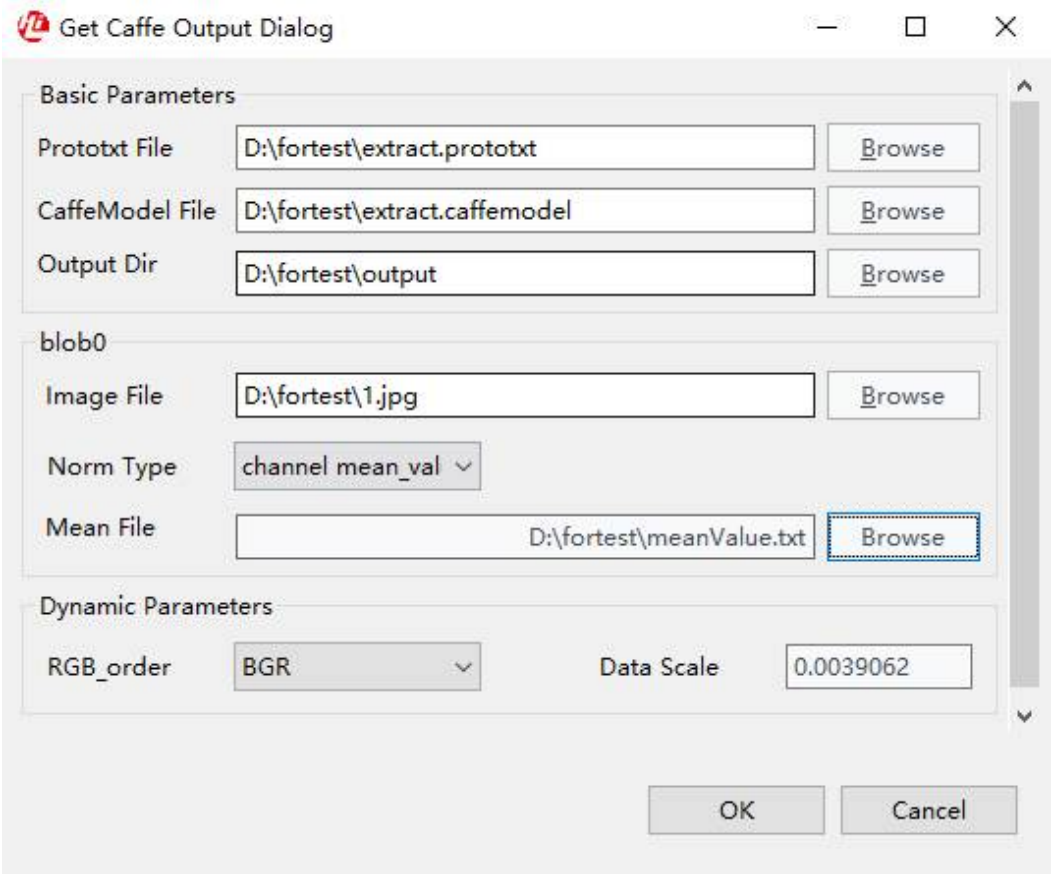
----结束

5.5.5 输出 caffe 的中间结果

本章节和**5.5.6 还原网络**章节的功能依赖python3，caffe，需要客户环境已经具备，配置方式请参考“**5.1.2 Python3.5+caffe环境配置**”章节的描述。对于本章节和“**5.5.6 还原网络**”章节，如果要使用GPU加速，需要保证按照“**3.6.4 功能仿真CUDA加速配置**”配置好相关环境，保证CUDA加速使能。如果工具检测到有CUDA加速，会使用CUDA加速，否则使用CPU生成Caffe结果。

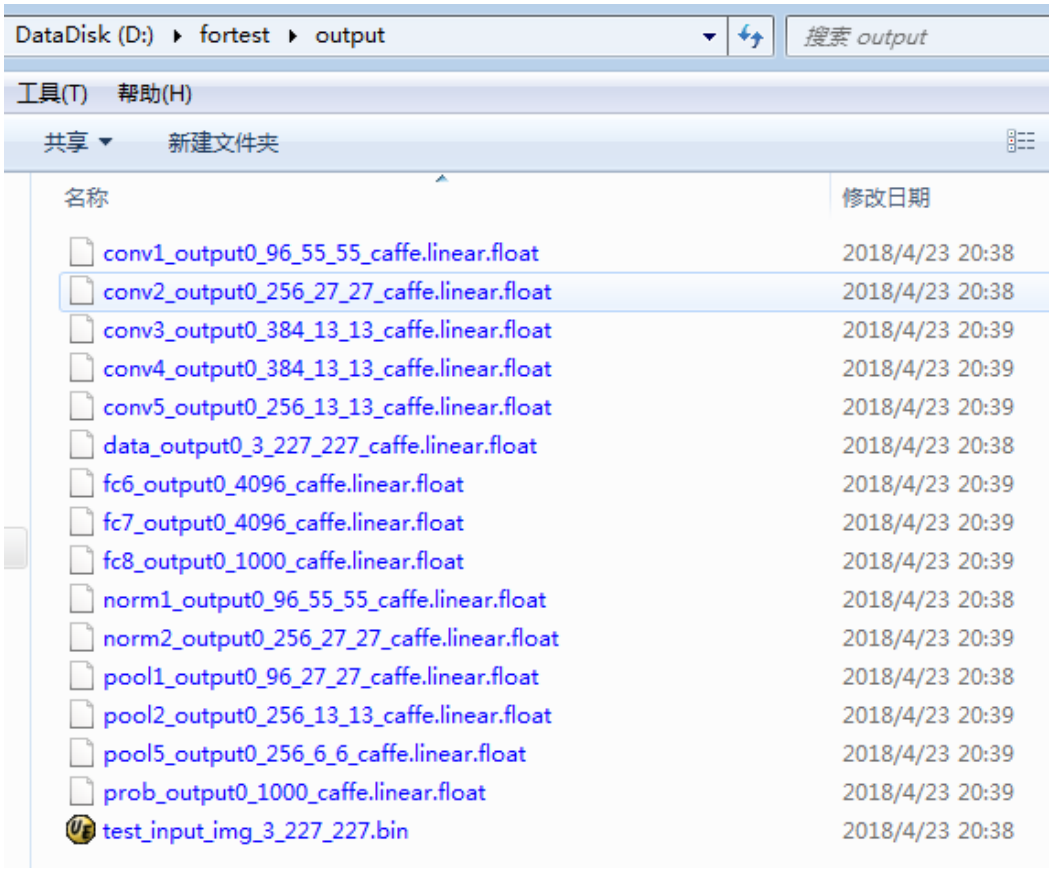
步骤1 点击工具栏上的  按钮，弹出如下对话框，依次设置输出中间结果的网络的prototxt, caffemodel, ImageFile, 如果要对图片做预处理操作，需要配置Norm Type, Data Scale, RGB_order和MeanFile, 具体含义可参考**3.5.2 配置文件说明**章节中配置文件项的相关介绍，并且设置caffe输出结果所在的路径Output Dir，如**图5-102**所示。

图 5-102 输出 caffe 中间结果



步骤2 点击OK键，在output文件夹下得到如下输出，如**图5-103**所示。

图 5-103 在配置的 output Dir 下面得到 caffe 中间结果



----结束

5.5.6 还原网络

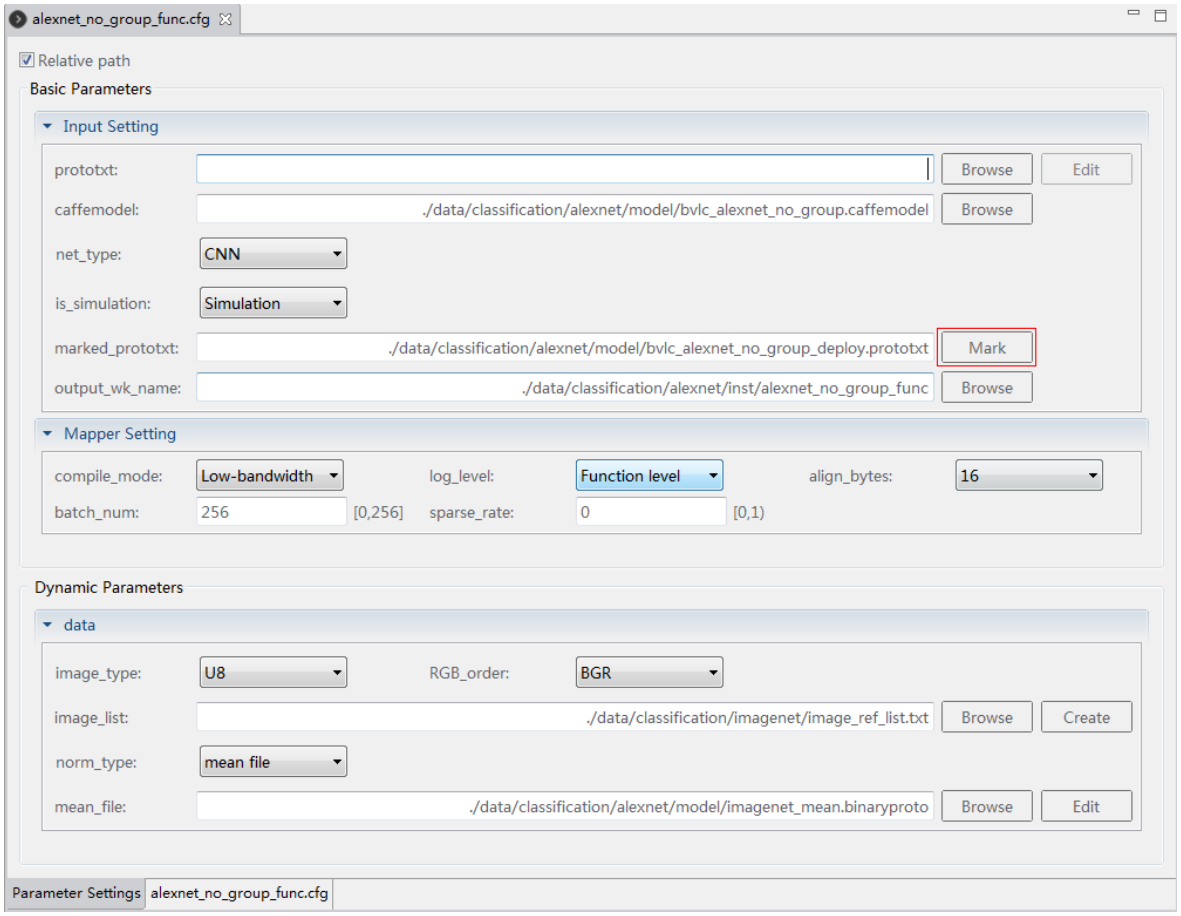
5.5.6.1 背景

客户已使用相似度比较工具确认精度问题不是nnie_mapper量化导致的，且定位到某个或者某些非用户自定义层出现精度下降问题（一种情况是断崖式下降，一种情况是相似度不是断崖式的下降，精度逐渐下降或者忽高忽低但是差异不大但整体偏差精度较大），需要客户将怀疑有问题的局部网络的相关数据截取出来并提供给上海海思定位。工具提供网络模型的导出功能方便还原网络定位问题。

5.5.6.2 操作步骤

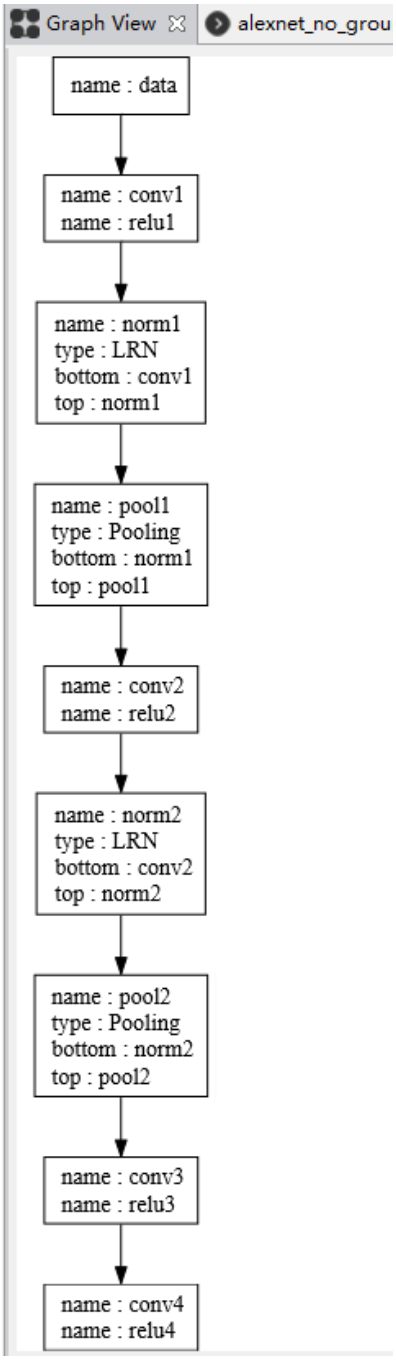
步骤1 利用**5.5.3 调试定位信息获取功能**章节介绍的调试定位信息获取功能dump出想要还原的网络数据，如**图5-104**所示。

图 5-104 标记 prototxt



步骤2 得到视图，如图5-105所示。

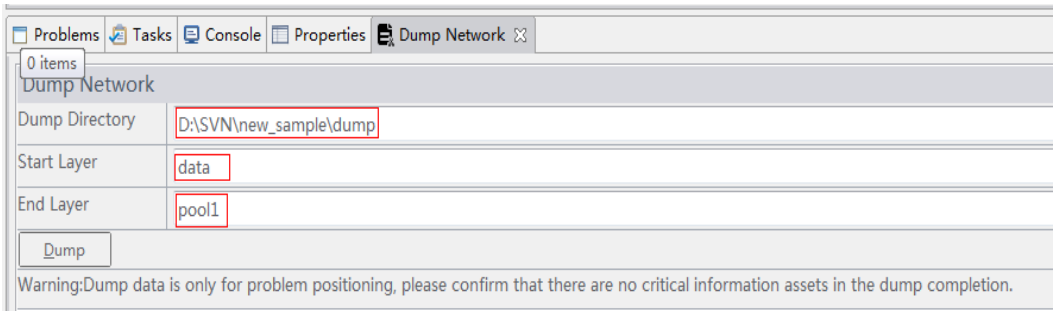
图 5-105 在 Graph View 上画出对应视图



步骤3 点击右上角dump network按钮。

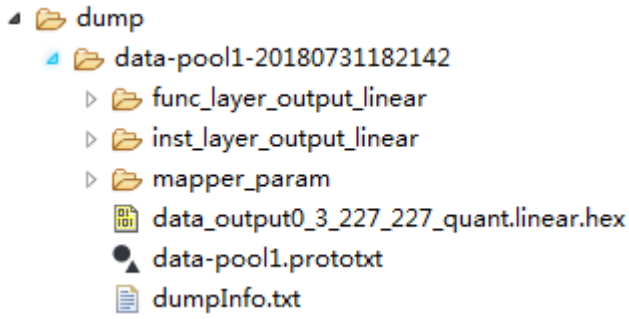
步骤4 选择如下需要dump的层，如图5-106所示。

图 5-106 Dump Network 视图



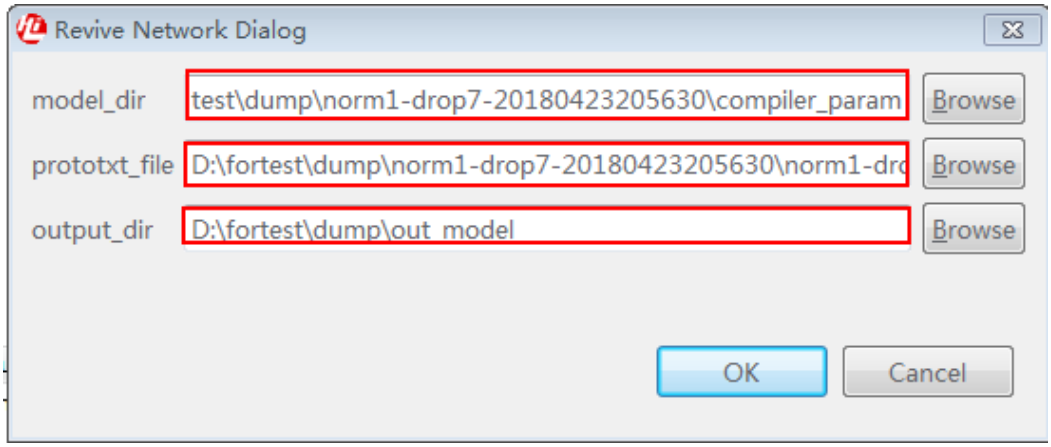
步骤5 点击dump按钮，生成dump出的数据，如图5-107所示。

图 5-107 Dump 出的数据



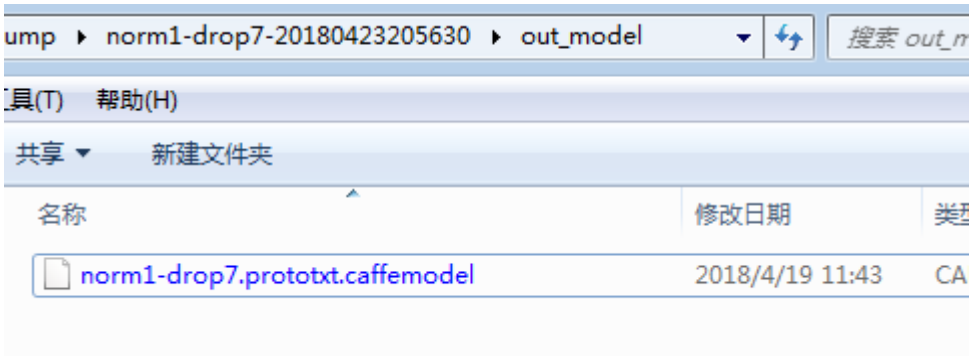
步骤6 点击工具栏上的还原网络  按钮打开还原网络弹出框，将mapper_quant，dump出的prototxt作为还原网络的输入，如图5-108所示。

图 5-108 还原网络



步骤7 得到还原出来的caffemodel，如图5-109所示。

图 5-109 还原出 caffemodel



----结束

5.5.7 芯片高效模式和 Un-inplace 功能

5.5.7.1 背景

把激活层合并到前面的NNIE层，在cycle仿或者板端环境下跑网络时候，在计算的时候就把激活执行，不需要一个单独的层来实现激活了，从而减少了内存的分配。

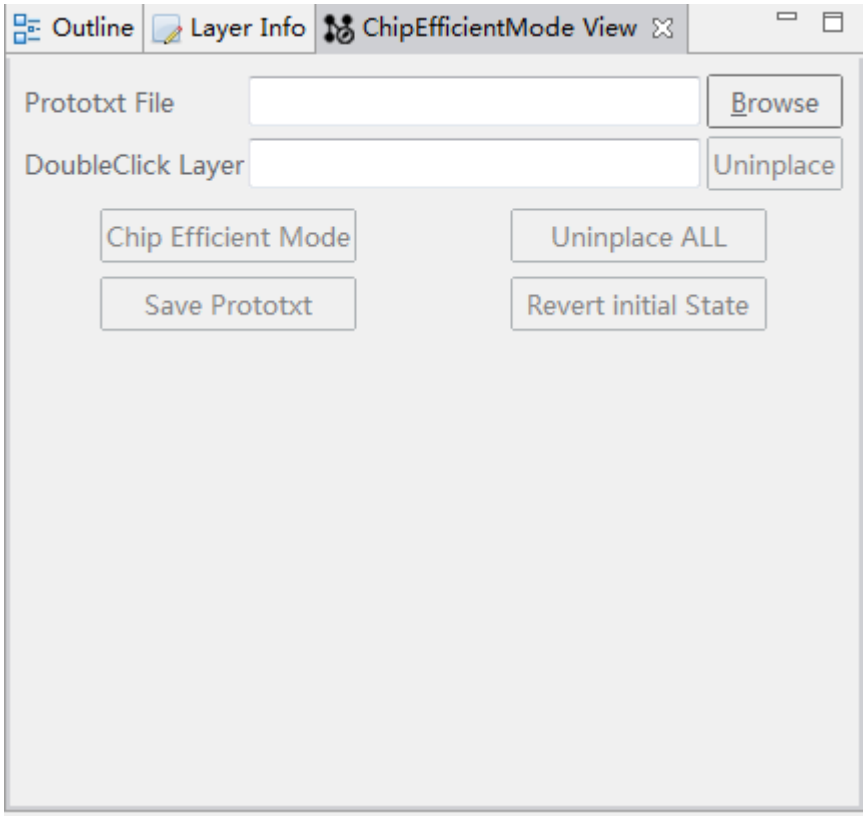
5.5.7.2 芯片高效模式功能介绍

支持将Prototxt文件中符合un-inplace规范且芯片支持的层集合，改写为inplace层。

操作方式如下：

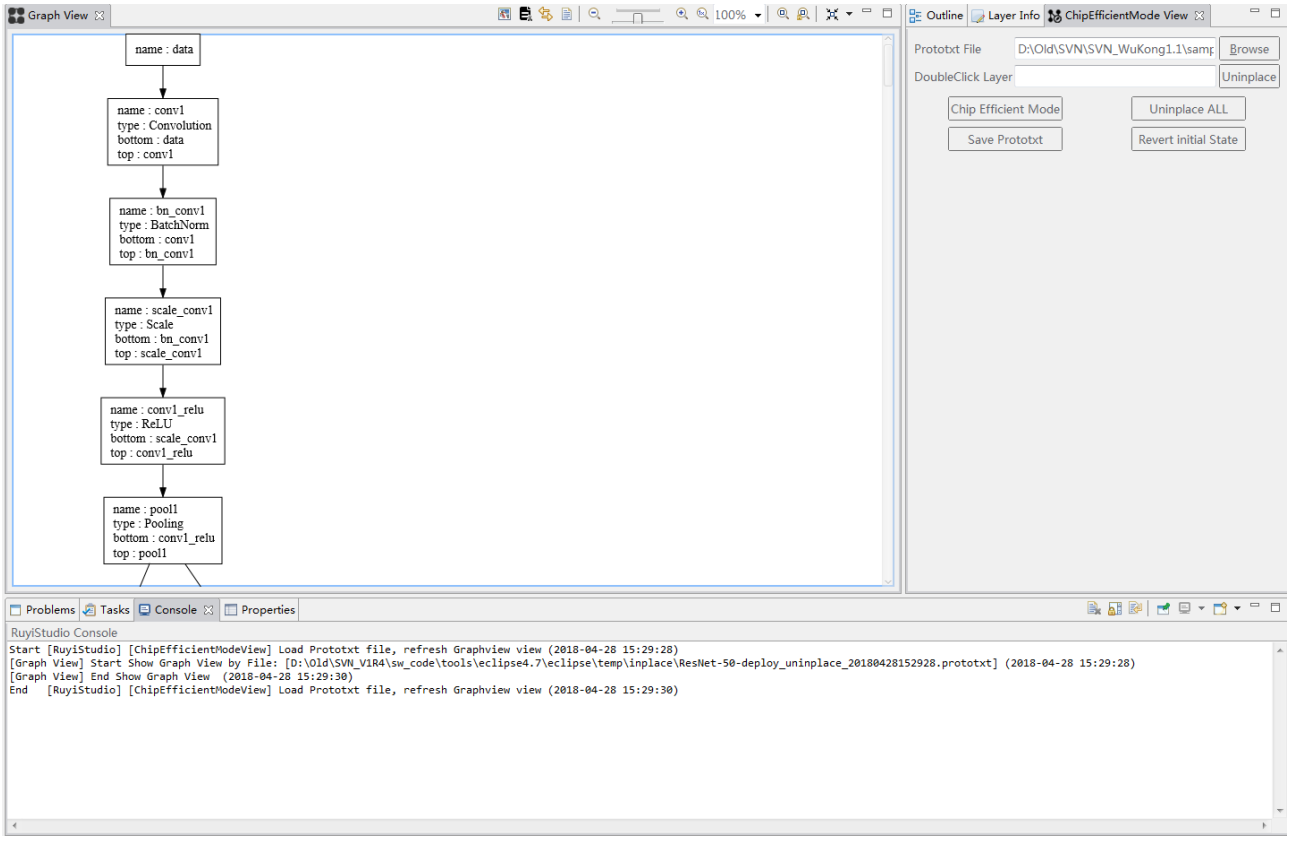
步骤1 点击工具栏芯片高效模式按钮  ，打开芯片高效模式视图，如图5-110所示。

图 5-110 芯片高效模式视图



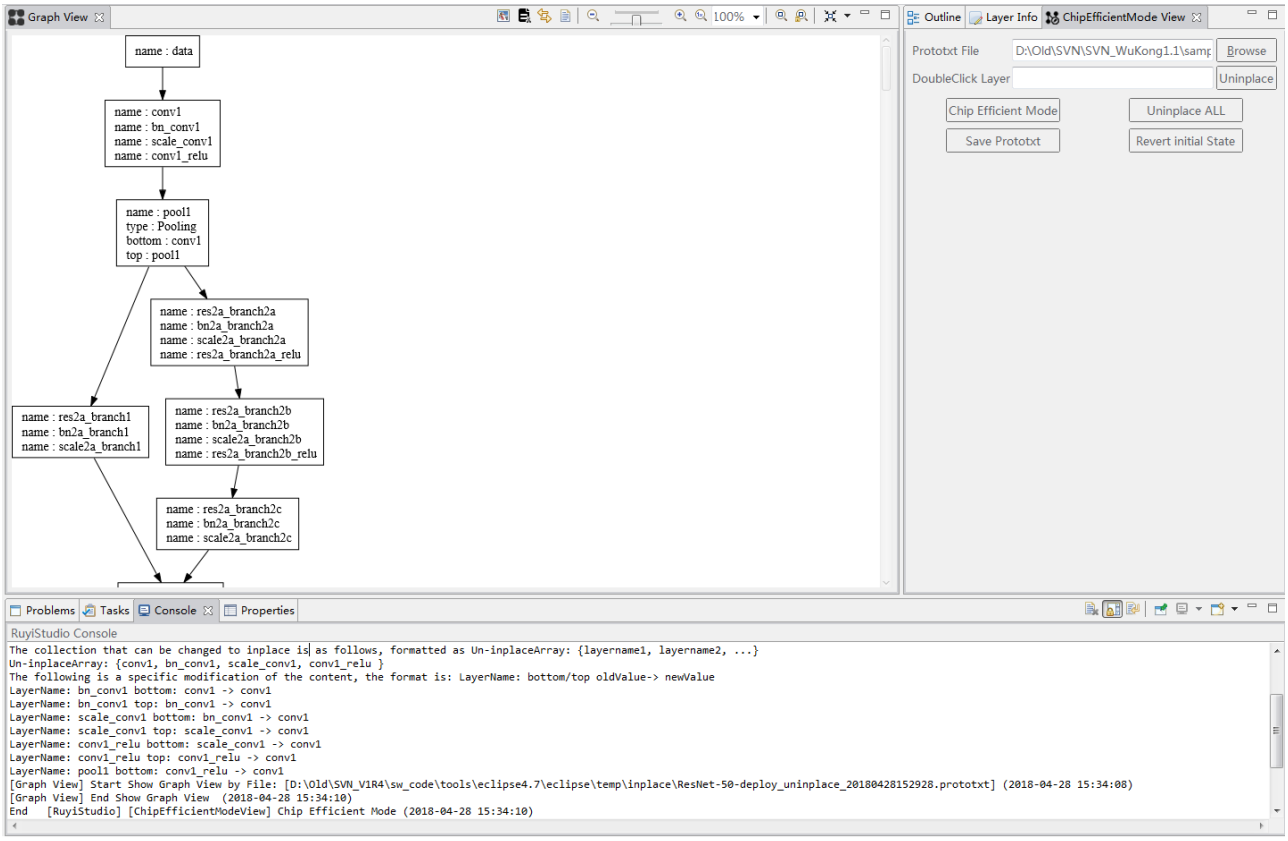
步骤2 将需要修改的Prototxt导入到Prototxt File的文本框，工具会自动加载Prototxt显示到GraphView中，如图5-111所示。

图 5-111 打开 Prototxt 显示到 Graph View 中



步骤3 点击右侧ChipEfficientMode View中的Chip Efficient Mode按钮，在确认弹出框中点击确认，工具就会去查找当前Prototxt中符合芯片支持的改写为Inplace的层的集合，然后将其改写为Inplace写法刷新到Graph View视图中，下方Console控制台会打印Un-inplace操作的过程信息，如图5-112，用户可通过此信息了解工具查找到的符合条件的可改写为inplace的层集合，以及当前操作修改的对应层中的top或bottom信息。

图 5-112 芯片高效模式执行效果



----结束

5.5.7.3 Un-inplace 所有 inplace 层的功能介绍

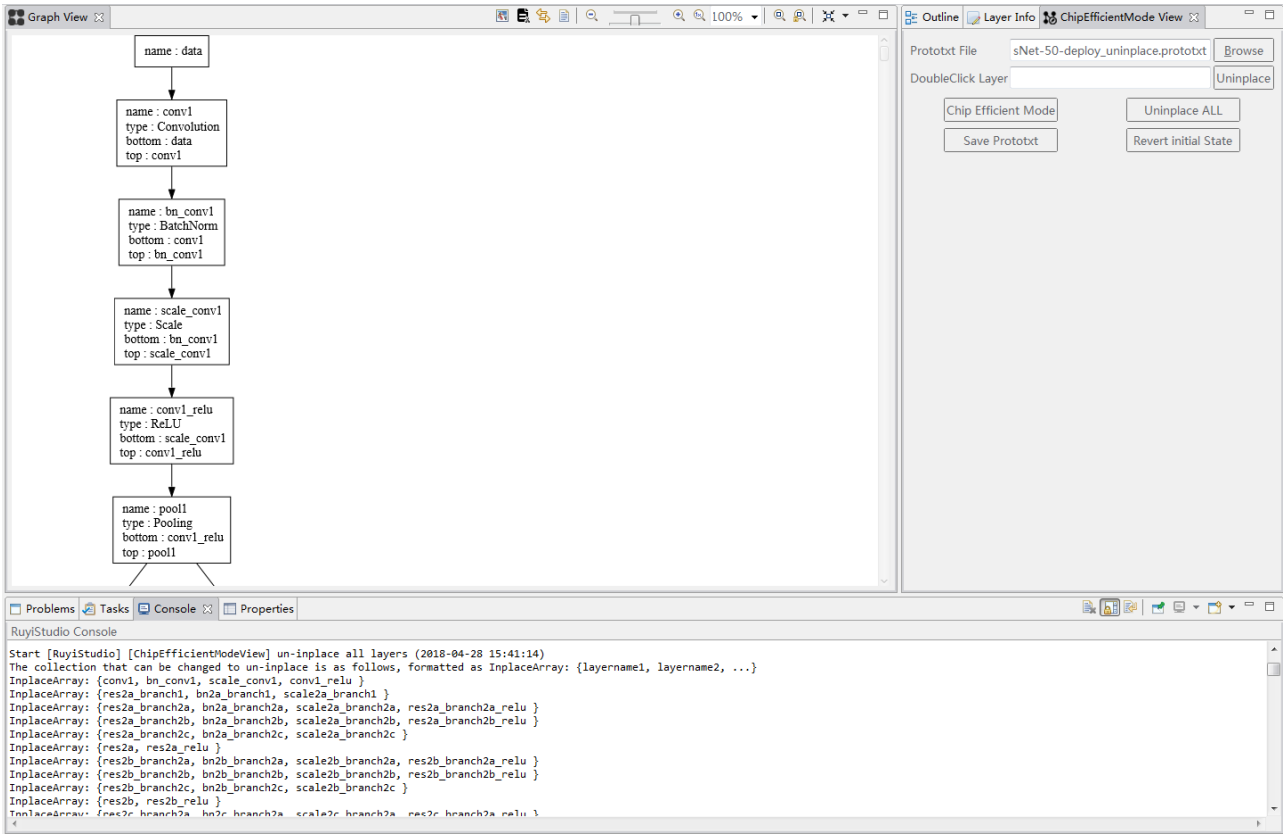
支持将Prototxt文件中符合inplace规范的层集合，改写为un-inplace层。

操作方式如下：

步骤1 导入需要修改的Prototxt文件，操作方式同**5.5.7.2 芯片高效模式功能介绍**章节的步骤2；

步骤2 点击Uninplace ALL按钮，左侧GraphView会刷新为Un-inplace ALL后的图像，下方Console控制台会打印Un-inplace ALL操作的过程信息，如**图5-113**，用户可通过此信息了解工具查找到的符合条件的可改写为unInplace的inplace层的集合，以及当前操作修改的对应层中的top或bottom信息。

图 5-113 Un-inplace 所有 inplace 层的功能



----结束

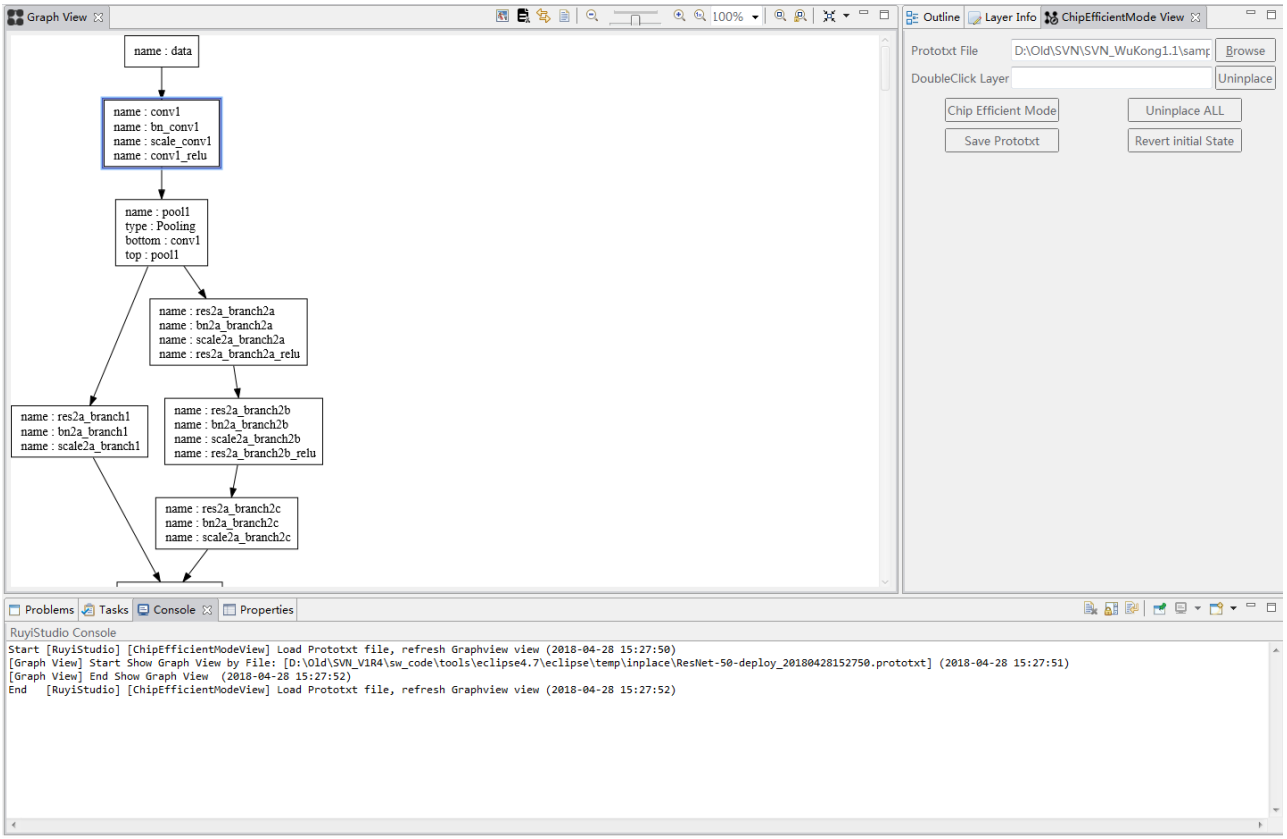
5.5.7.4 Un-inplace 单个 inplace 层功能介绍

支持将Prototxt中芯片支持改写为Un-inplace的用户指定的单个inplace层自动改写为Un-inplace层。

操作方式如下：

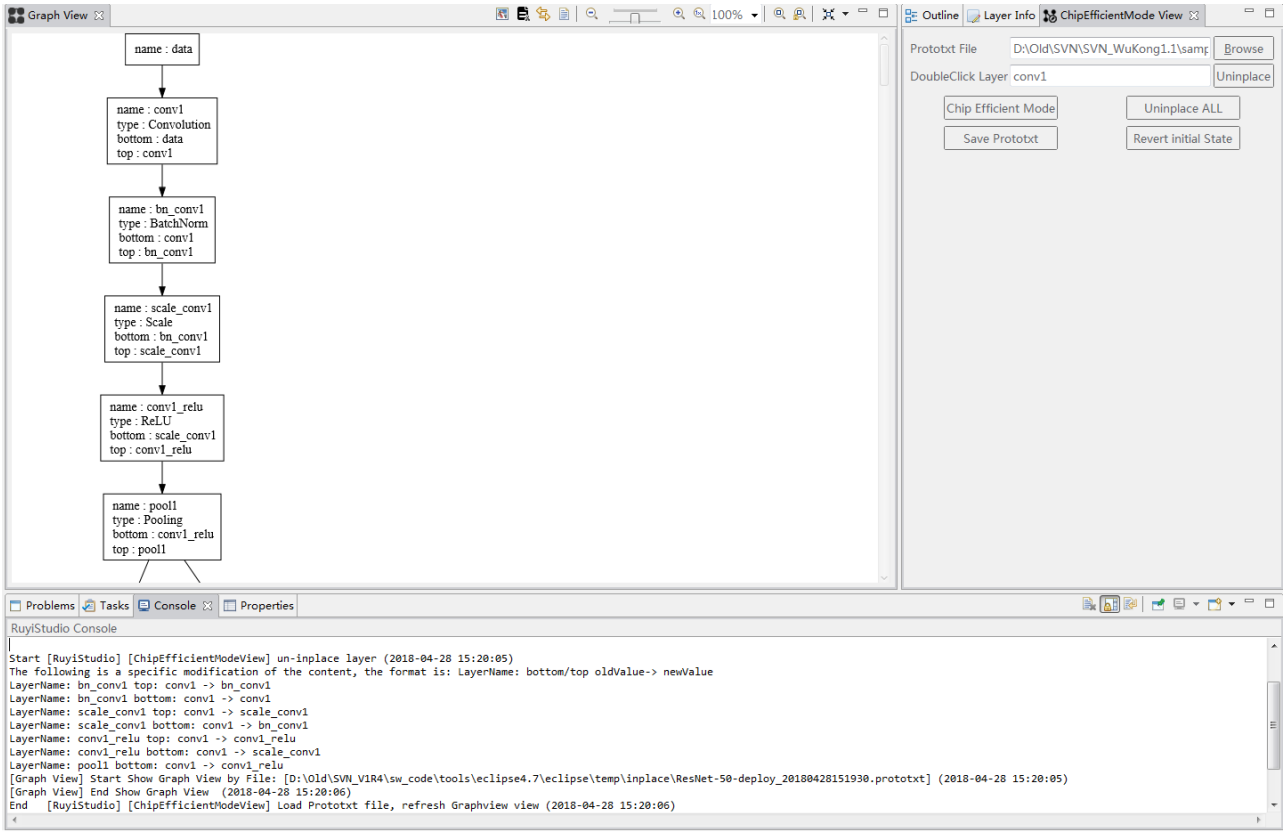
步骤1 在左侧GraphView中双击需要执行Un-inplace操作的inplace层，DoubleClick Layer文本框中会显示选中的层名，如图5-114所示。

图 5-114 双击选中需要 un-inplace 的层



步骤2 点击Uninplace按钮，左侧GraphView会刷新为Un-inplace后的图像，下方Console控制台会打印Un-inplace操作的过程信息，如图5-115，用户可通过此信息了解当前操作修改的对应层中的top或bottom信息。

图 5-115 Un-inplace 单个 inplace 层的功能



----结束

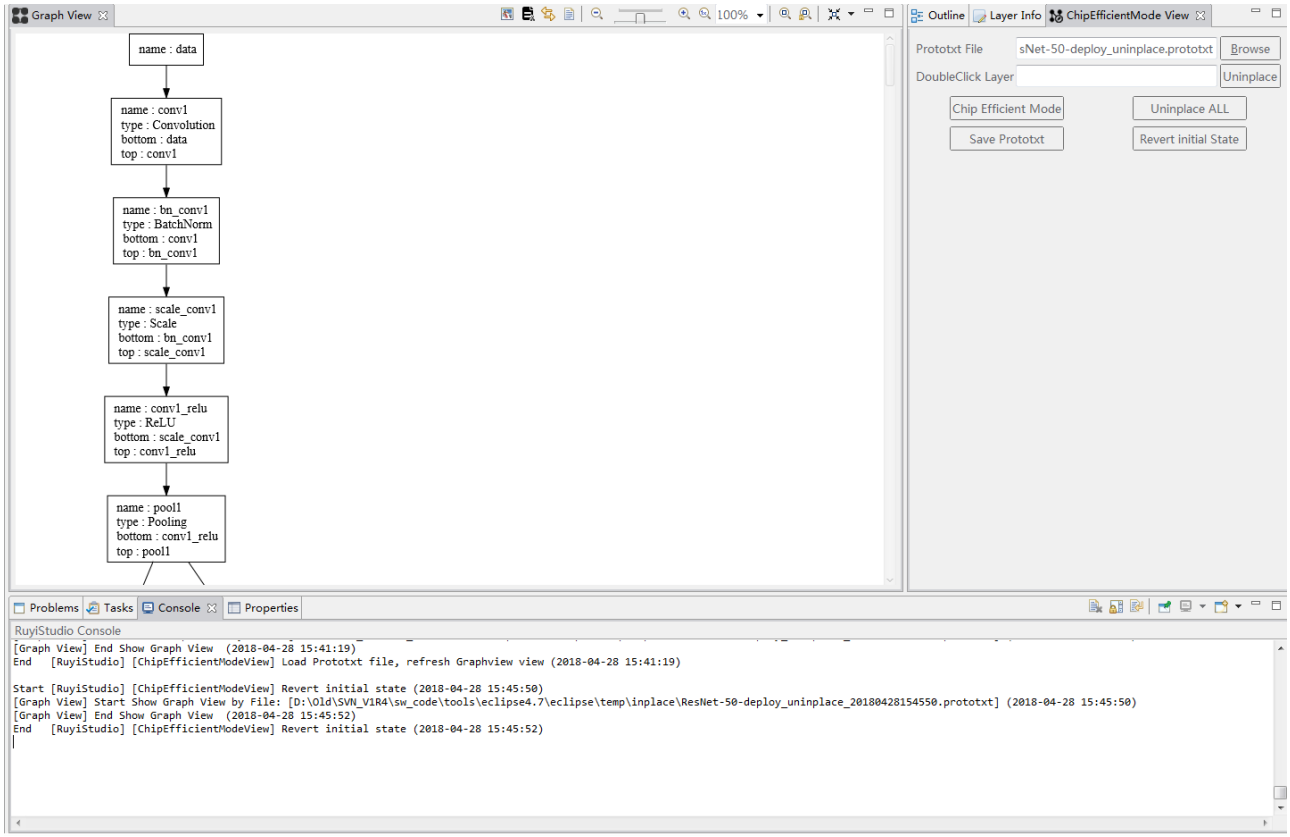
5.5.7.5 恢复 Prototxt 到原始状态的功能介绍

支持将当前GraphView中显示的Prototxt恢复到原始导入的Prototxt。

操作方式如下：

步骤1 点击Revert initial State按钮，左侧GraphView会刷新为原始的Prototxt导入后的图像，如图5-116。

图 5-116 Revert initial State 功能



----结束

5.5.7.6 保存当前 GraphView 中修改过的 Prototxt 的功能介绍

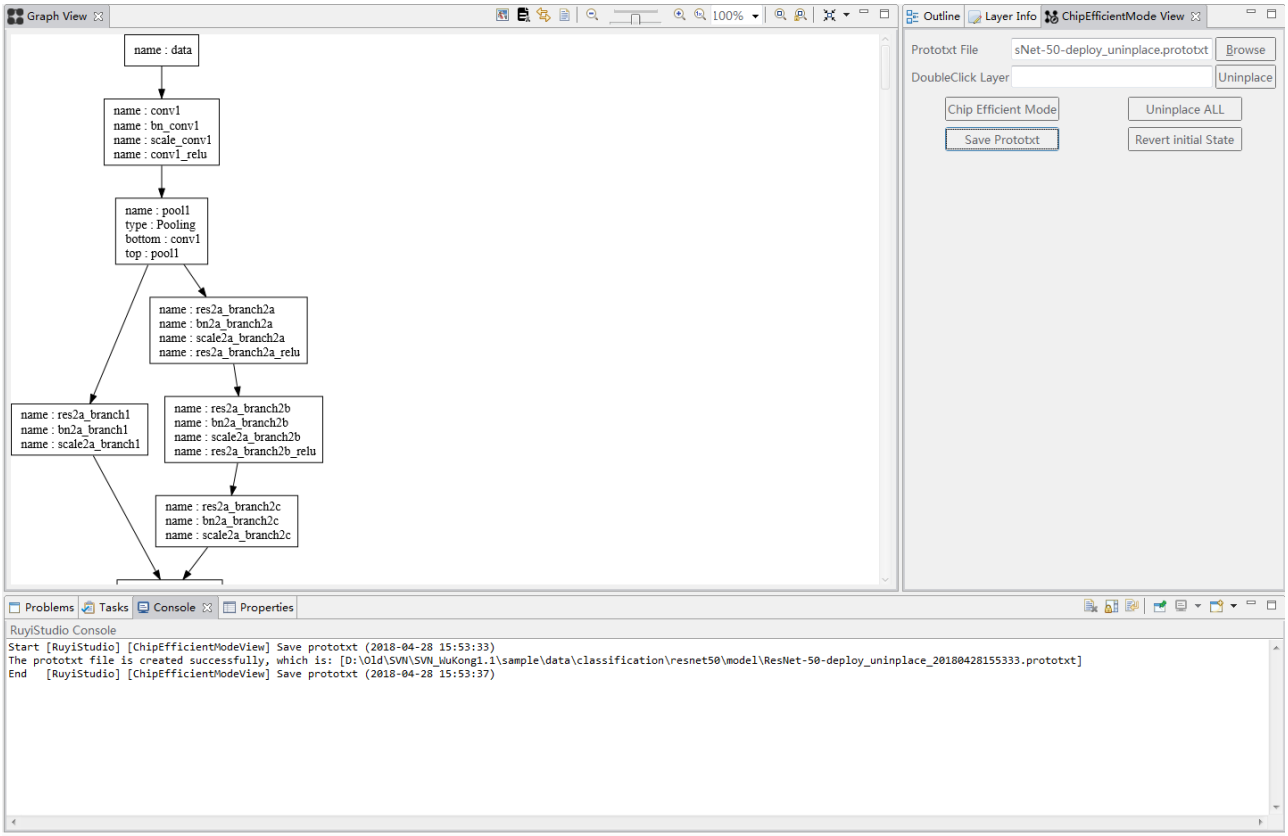
支持将当前GraphView中显示的修改过的Prototxt保存到新的Prototxt文件的功能。

操作方式如下：

步骤1 点击Save Prototxt按钮，设置保存路径；

步骤2 点击确认，保存当前GraphView中的Prototxt文件成功，如图5-117。

图 5-117 Save Prototxt 功能



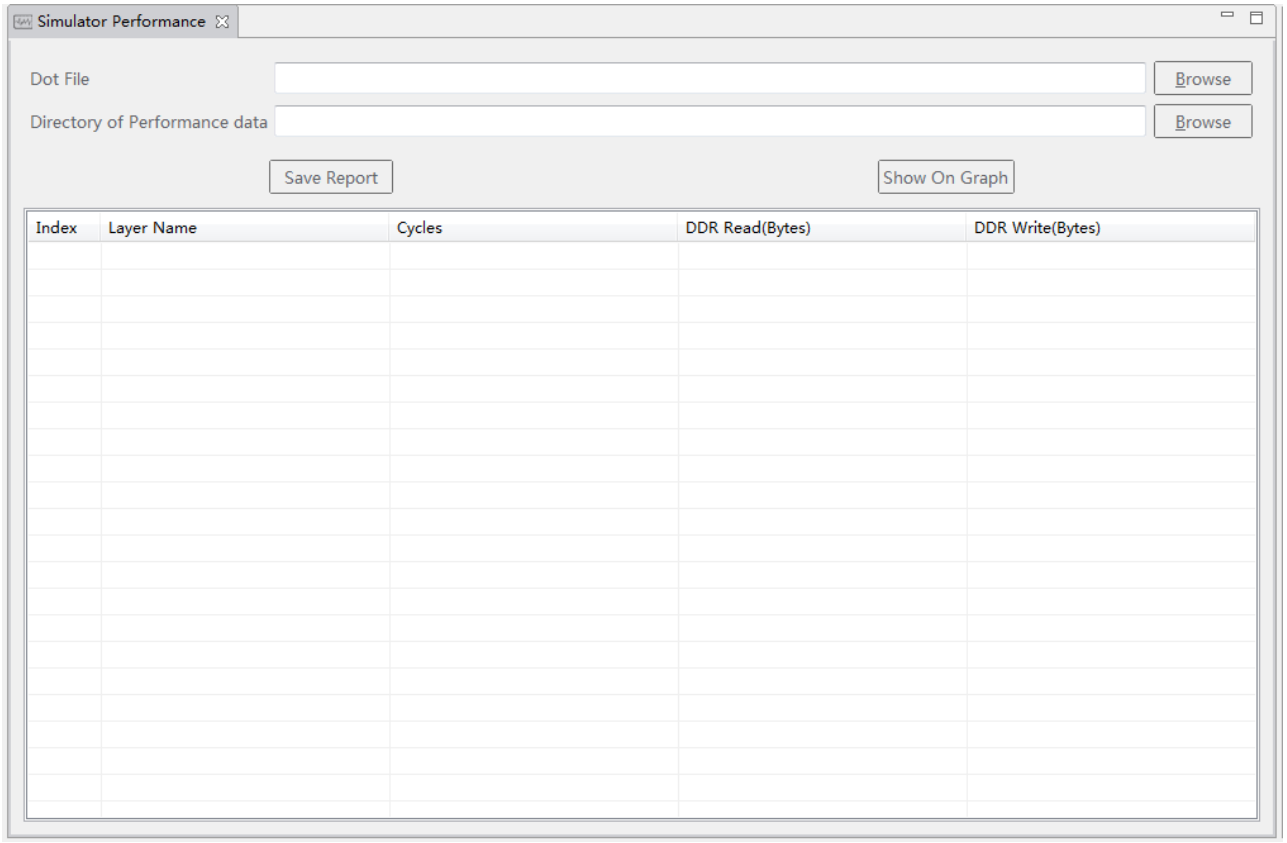
----结束

5.5.8 性能仿真结果展示

提供仿真库中cycle仿执行过程输入的性能仿真数据的UI展示，操作如下：

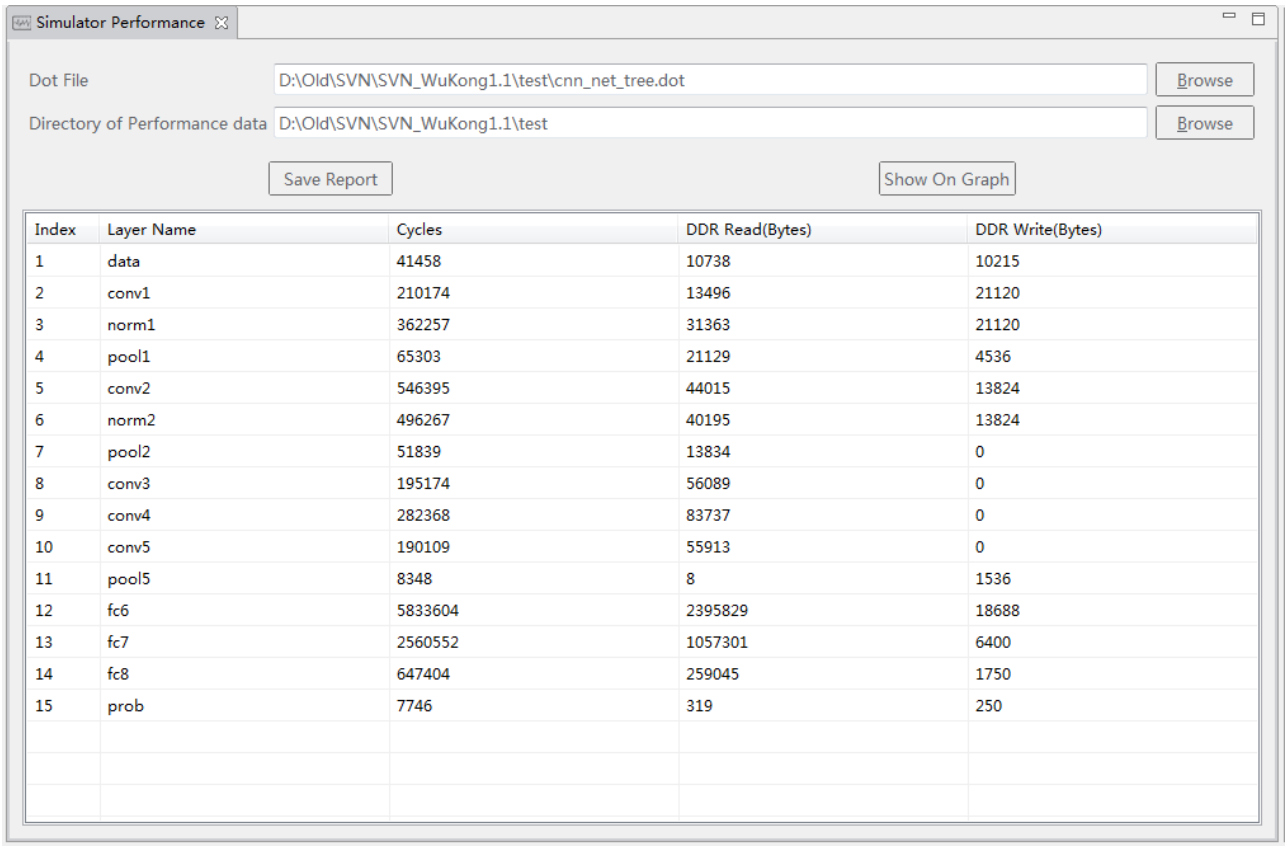
步骤1 点击左上角工具栏的性能仿真结果展示按钮  ，在工具编辑区显示性能仿真结果展示视图Simulator Preformance，如图5-118。

图 5-118 性能仿真结果展示界面



步骤2 导入nnie_mapper生成的Dot文件用于解析层名，再导入性能仿真结果数据存在的文件夹，导入操作如图5-119，性能仿真数据文件存在于cycle仿运行后生成的目录下，以sample工程为例，在sim_out下的perf_info目录中，如cycs_result_0文件，示例如图5-120。

图 5-119 导入 Dot 文件和性能仿真结果文件夹



Index	Layer Name	Cycles	DDR Read(Bytes)	DDR Write(Bytes)
1	data	41458	10738	10215
2	conv1	210174	13496	21120
3	norm1	362257	31363	21120
4	pool1	65303	21129	4536
5	conv2	546395	44015	13824
6	norm2	496267	40195	13824
7	pool2	51839	13834	0
8	conv3	195174	56089	0
9	conv4	282368	83737	0
10	conv5	190109	55913	0
11	pool5	8348	8	1536
12	fc6	5833604	2395829	18688
13	fc7	2560552	1057301	6400
14	fc8	647404	259045	1750
15	prob	7746	319	250

图 5-120 性能仿真结果文件示例

cycs_result_0.log

```
1  ***************************[Layer inst cycles]*****************************
2  [Layer000]: Layer_cycs:     41458    DDR_read:     10738    DDR_write:     10215
3  [Layer001]: Layer_cycs:    210174    DDR_read:     13496    DDR_write:     21120
4  [Layer002]: Layer_cycs:    362257    DDR_read:     31363    DDR_write:     21120
5  [Layer003]: Layer_cycs:     65303    DDR_read:     21129    DDR_write:      4536
6  [Layer004]: Layer_cycs:    546395    DDR_read:     44015    DDR_write:     13824
7  [Layer005]: Layer_cycs:    496267    DDR_read:     40195    DDR_write:     13824
8  [Layer006]: Layer_cycs:     51839    DDR_read:     13834    DDR_write:         0
9  [Layer007]: Layer_cycs:    195174    DDR_read:     56089    DDR_write:         0
10 [Layer008]: Layer_cycs:    282368    DDR_read:     83737    DDR_write:         0
11 [Layer009]: Layer_cycs:    190109    DDR_read:     55913    DDR_write:         0
12 [Layer010]: Layer_cycs:      8348    DDR_read:         8    DDR_write:      1536
13 [Layer011]: Layer_cycs:   5833604    DDR_read:   2395829    DDR_write:     18688
14 [Layer012]: Layer_cycs:   2560552    DDR_read:   1057301    DDR_write:      6400
15 [Layer013]: Layer_cycs:    647404    DDR_read:    259045    DDR_write:      1750
16 [Layer014]: Layer_cycs:      5885    DDR_read:       317    DDR_write:         0
17 [Layer015]: Layer_cycs:      1861    DDR_read:         2    DDR_write:       250
18 ***************************[Net inst cycles]*******************************
19 Net inst cycles: 11498998    DDR_read:  4083011    DDR_write: 113263
20
```

----结束

5.5.9 预处理功能

5.5.9.1 背景

将对Prototxt文件进行格式修改的功能整合到预处理界面下，包含以下功能：

- Convert Data Layerformat
- Convert Efficient Mode
- Add Quant Adaptation Layer
- Remove PriorBox Layer
- Detection Network NNIE1.1 Converted to NNIE1.2

5.5.9.2 功能介绍

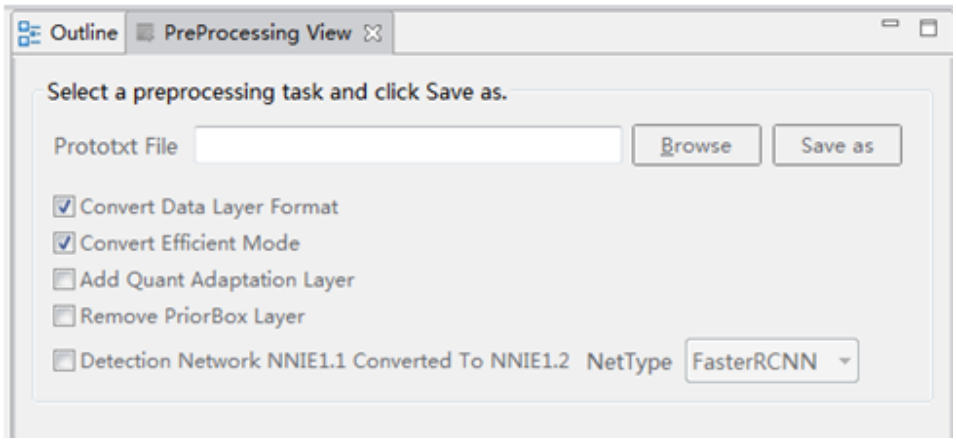
- Convert Data Layerformat，支持将caffe1.0标准的prototxt文件修改为NNIE支持的写法。
- Convert Efficient Mode，支持将Prototxt文件中符合un-inplace规范且芯片支持的层集合，改写为inplace层。
- Add Quant Adaptation Layer，支持在做重训练时，涉及到修改prototxt的操作，如果要增加量化层，可以使用工具增加量化层。
- Remove PriorBox Layer，由于SSD网络下的PriorBox层NNIE不支持也不属于caffe原生的层，手动改动比较繁琐，效率比较低，故工具支持在在预处理的时候去掉PriorBox层。
- Detection Network NNIE1.1 Converted to NNIE1.2，此功能应用于检测网，对NNIE1.1不支持的Proposal和DectectionOutput层添加适配层模板（用户需要自己调整参数）以支持NNIE1.2的硬件加速特性。

5.5.9.3 操作步骤

操作方式如下：

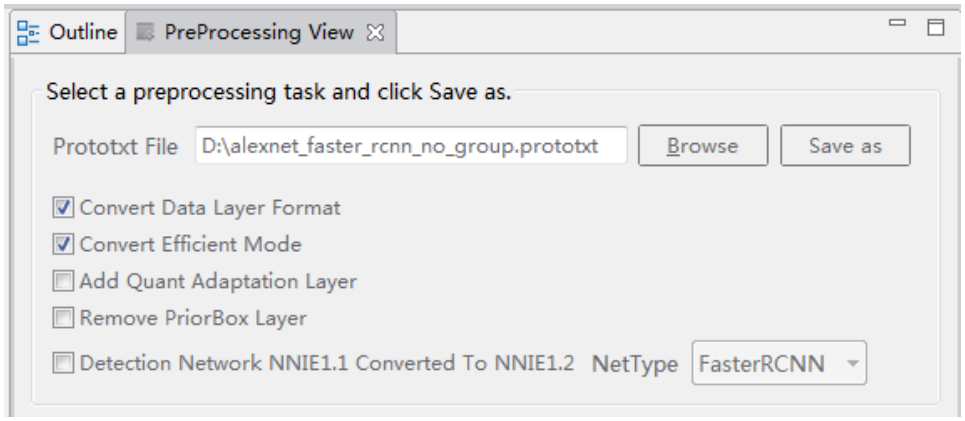
步骤1 点击工具栏预处理按钮  ，打开预处理视图，如图5-121所示。

图 5-121 预处理视图



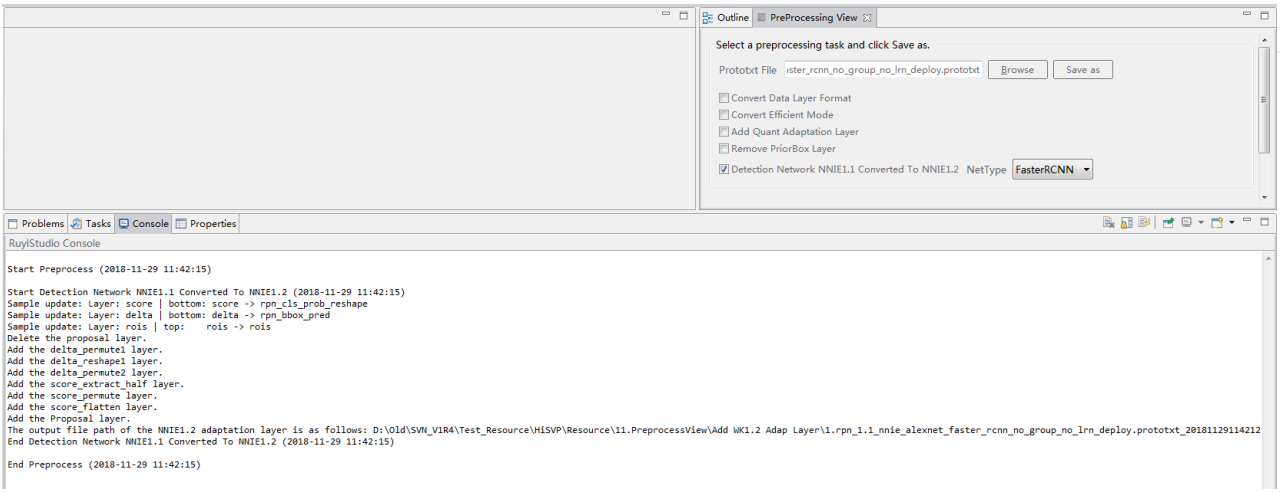
步骤2 将需要修改的Prototxt导入到Prototxt File的文本框，如图5-122所示。

图 5-122 导入 Prototxt



步骤3 选择需要进行预处理修改的选项，点击Save as选择需要保存Prototxt的路径，点击确认即可将修改的prototxt另存，且工具控制台会输出修改详细日志，如图5-123。

图 5-123 进行预处理另存 Prototxt

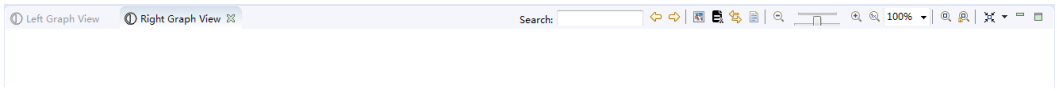


----结束

5.5.10 拓扑图对比功能

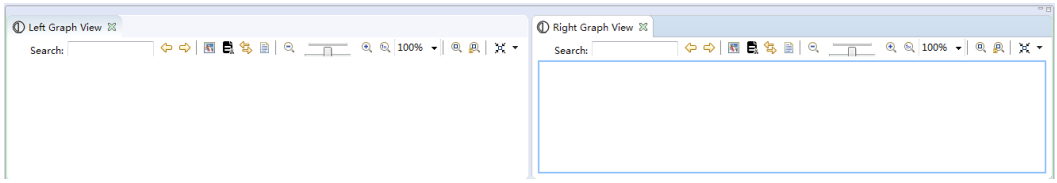
步骤1 在工具栏点击 按钮，会出现如图5-124所示初始化界面。

图 5-124 拓扑图对比初始化界面



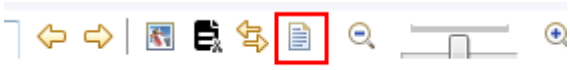
步骤2 手动选中Right GraphView页签拖动视图布局，切为对比视图，如图5-125所示。

图 5-125 拓扑图对比视图



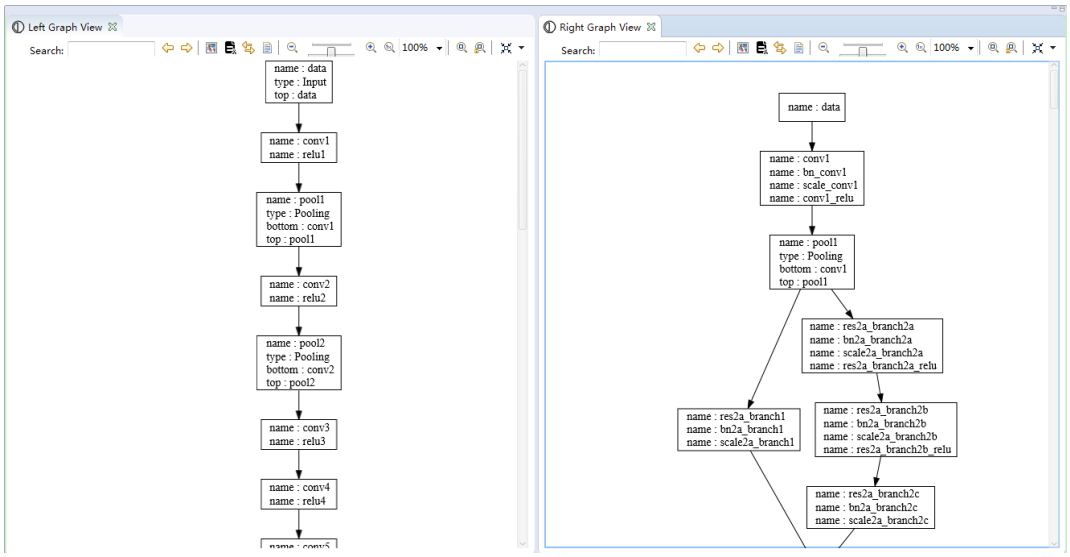
步骤3 点击上方的导入按钮，导入对应的prototxt或者dot文件。

图 5-126 导入文件



步骤4 对比网络拓扑图，界面显示如图5-127所示。

图 5-127 网络拓扑图

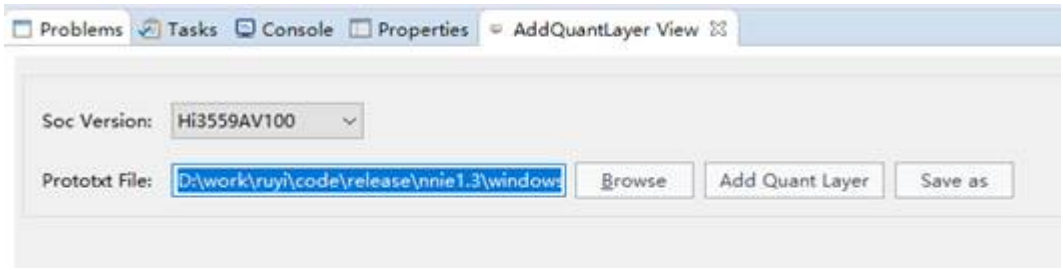


----结束

5.5.11 添加量化层功能

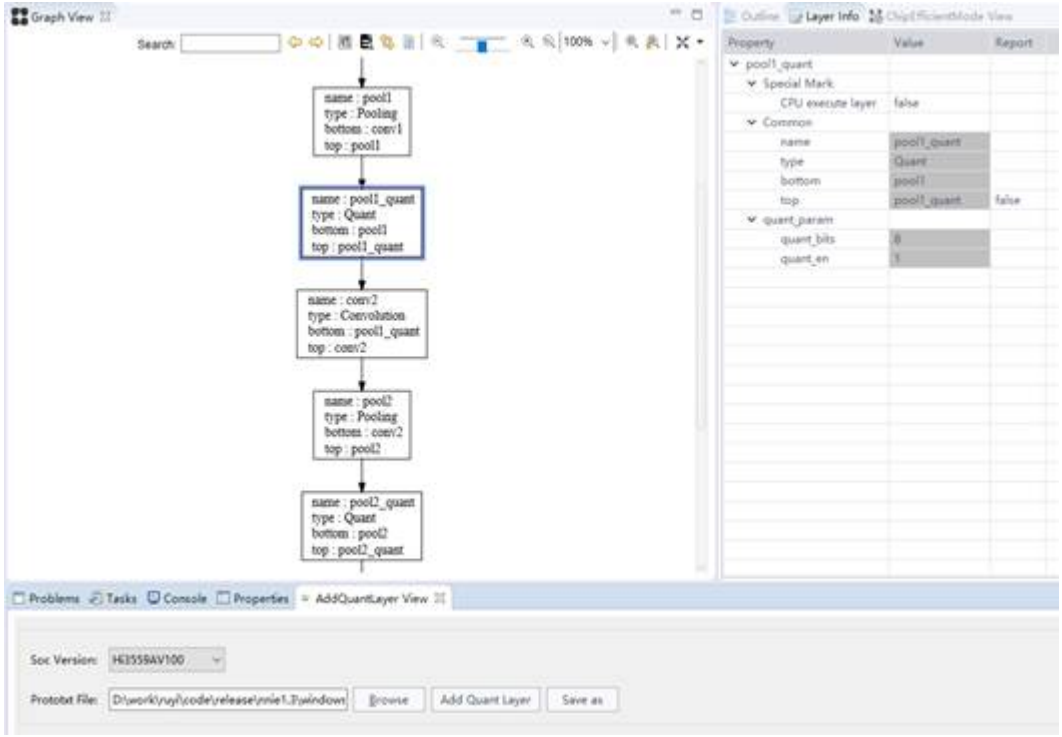
步骤1 在工具栏点击  按钮，会出现如图5-128所示初始化界面。

图 5-128 添加量化层界面



步骤2 选择芯片型号，将需要修改的Prototxt文件导入到Prototxt File的文本框，点击Add Quant Layer进行添加量化层，界面显示如图5-129所示。

图 5-129 Prototxt 显示到 Graph View 中



步骤3 点击 Save as选择需要保存Prototxt的路径，点击确认即可将修改的prototxt报存。

----结束

5.6 多芯片支持功能

因不同的芯片使用了不同的nnie_mapper及仿真库，故RuyiStudio工具提供多芯片支持的功能，用户可以指定当前使用的项目工程匹配的是哪一款芯片，具体的支持表格如表5-1。

表 5-1 芯片名称与 mapper 和仿真库的对应关系

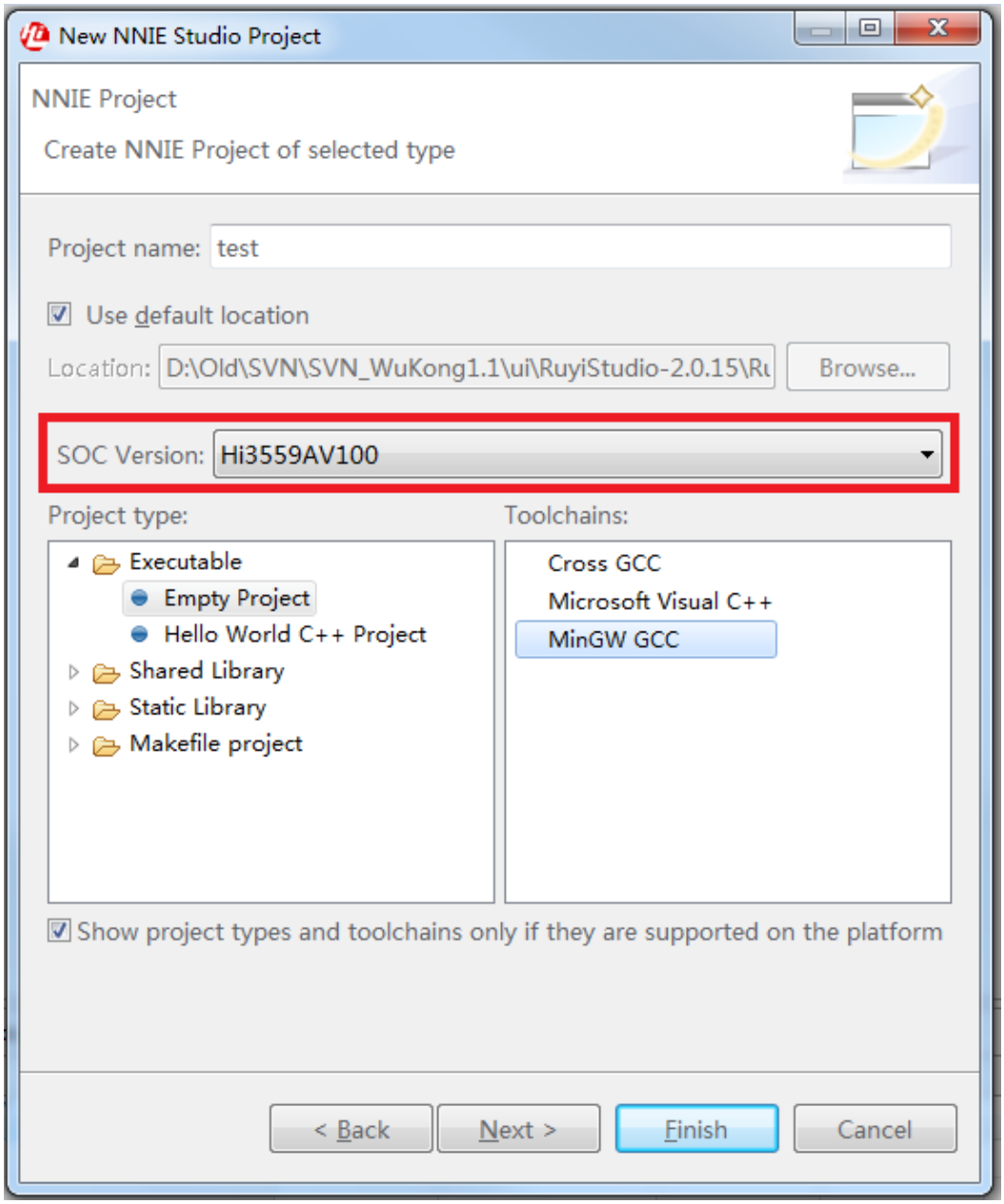
芯片名称	与mapper和仿真库的对应关系
Hi3559AV100 Hi3559CV100 Hi3569V100	nnie_mapper_11.exe libnniefc1.1.a, libnnieit1.1.a, libnniefc1.1d.a, libnnieit1.1d.a nniefc1.1.dll, nniefc1.1d.dll, nnieCUDA1.1.dll, nnieCUDA1.1d.dll
Hi3519AV100	nnie_mapper_12.exe libnnieit1.2.a, libnniefc1.2.a, libnnieit1.2d.a, libnniefc1.2d.a nniefc1.2.dll, nniefc1.2d.dll, nnieCUDA1.2.dll, nnieCUDA1.2d.dll
Hi3556AV100	不支持
Hi3568V100	nnie_mapper_12.exe libnnieit1.2.a, libnniefc1.2.a, libnnieit1.2d.a, libnniefc1.2d.a nniefc1.2.dll, nniefc1.2d.dll, nnieCUDA1.2.dll, nnieCUDA1.2d.dll
Hi3516DV300	nnie_mapper_12.exe libnnieit1.2.a, libnniefc1.2.a, libnnieit1.2d.a, libnniefc1.2d.a nniefc1.2.dll, nniefc1.2d.dll, nnieCUDA1.2.dll, nnieCUDA1.2d.dll
Hi3516CV500	nnie_mapper_12.exe libnnieit1.2.a, libnniefc1.2.a, libnnieit1.2d.a, libnniefc1.2d.a nniefc1.2.dll, nniefc1.2d.dll, nnieCUDA1.2.dll, nnieCUDA1.2d.dll
Hi3559V200	nnie_mapper_12.exe libnnieit1.2.a, libnniefc1.2.a, libnnieit1.2d.a, libnniefc1.2d.a nniefc1.2.dll, nniefc1.2d.dll, nnieCUDA1.2.dll, nnieCUDA1.2d.dll
Hi3516AV300	nnie_mapper_12.exe libnnieit1.2.a, libnniefc1.2.a, libnnieit1.2d.a, libnniefc1.2d.a nniefc1.2.dll, nniefc1.2d.dll, nnieCUDA1.2.dll, nnieCUDA1.2d.dll
Hi3531DV200	nnie_mapper_13.exe libnniefc1.3.a, libnniefc1.3d.a nniefc1.3.dll, nniefc1.3d.dll, nnieCUDA1.3.dll
Hi3535AV100	nnie_mapper_13.exe libnniefc1.3.a, libnniefc1.3d.a nniefc1.3.dll, nniefc1.3d.dll, nnieCUDA1.3.dll

芯片名称	与mapper和仿真库的对应关系
Hi3562V100	nnie_mapper_12.exe libnnieit1.2.a, libnniefc1.2.a, libnnieit1.2d.a, libnniefc1.2d.a nniefc1.2.dll, nniefc1.2d.dll, nnieCUDA1.2.dll, nnieCUDA1.2d.dll
Hi3566V100	nnie_mapper_12.exe libnnieit1.2.a, libnniefc1.2.a, libnnieit1.2d.a, libnniefc1.2d.a nniefc1.2.dll, nniefc1.2d.dll, nnieCUDA1.2.dll, nnieCUDA1.2d.dll
Hi3521DV200	nnie_mapper_13.exe libnniefc1.3.a, libnniefc1.3d.a nniefc1.3.dll, nniefc1.3d.dll, nnieCUDA1.3.dll
Hi3520DV500	nnie_mapper_13.exe libnniefc1.3.a, libnniefc1.3d.a nniefc1.3.dll, nniefc1.3d.dll, nnieCUDA1.3.dll

5.6.1 创建新工程项目时选择对应的芯片

在工具栏上点击File->New->NNIE Project，打开创建视图，在SOC Version处选择项目绑定的芯片名称，如图5-130。

图 5-130 创建 NNIE Project 时设置 SOC Version

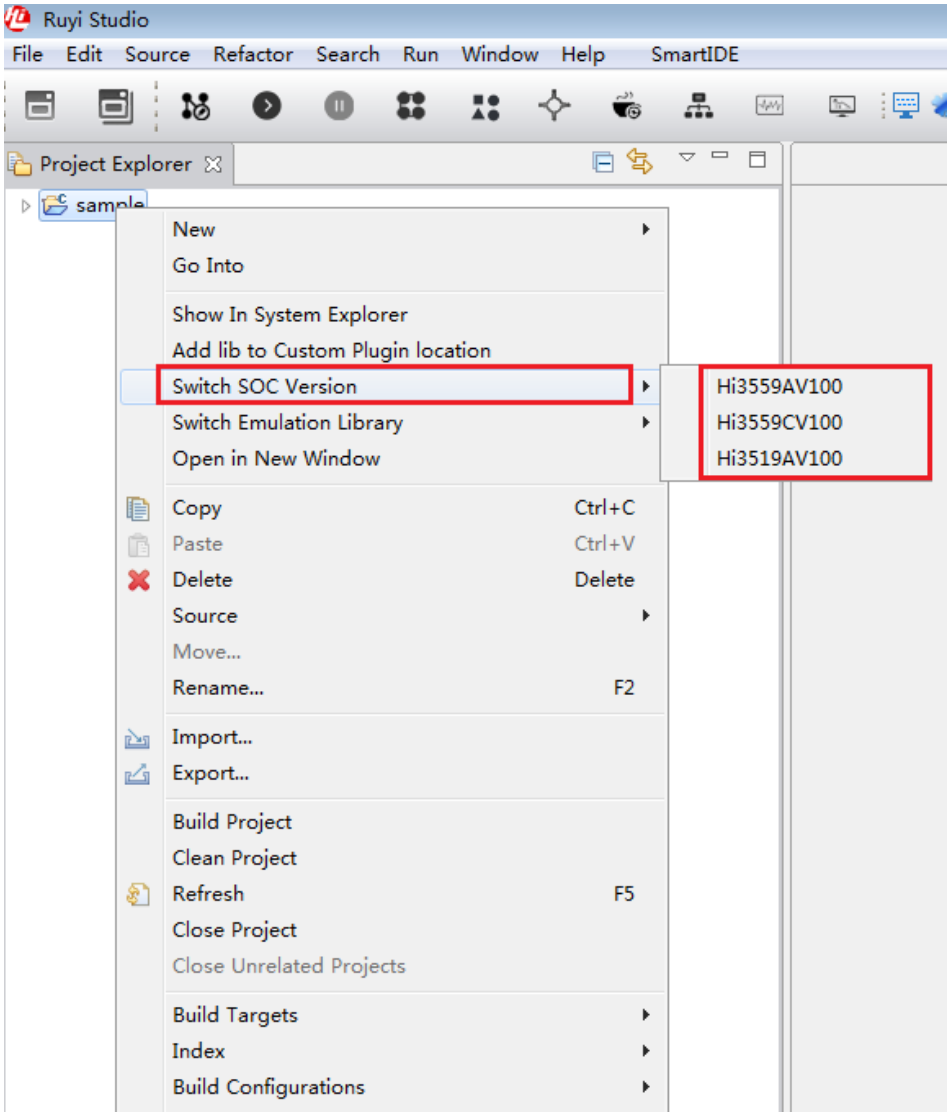


5.6.2 在已有的工程项目中修改对应的芯片

步骤如下：

步骤1 在项目上点击右键-> SOC Version选择对应的SOC Version，如图5-131所示。

图 5-131 修改已有工程中对应的芯片



步骤2 另外，导入到RuyiStudio的工程中若默认没有指定项目对应的芯片，工具会自动添加默认芯片Hi3559AV100给当前的工程。

----结束

5.7 生成 Runtime wk 功能

在Runtime工程中，打开一个cfg。

5.7.1 cfg 文件配置模式

同配置NNIE一样。

须知

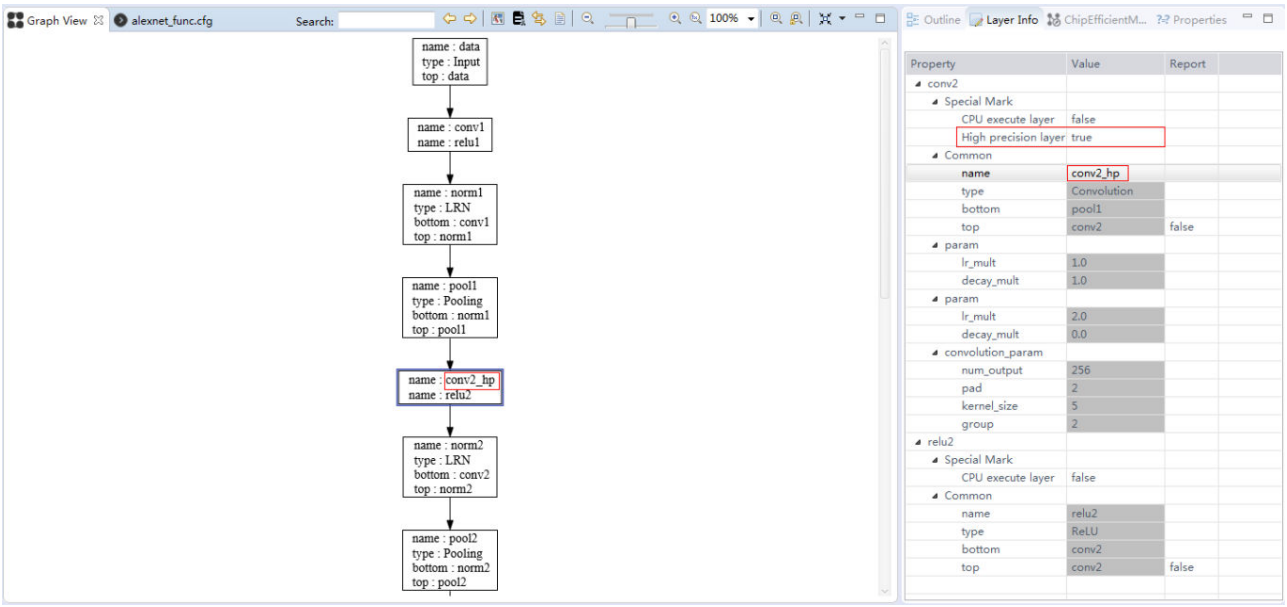
在打开cfg的时候，如果该cfg有prototxt路径，则会在prototxt的路径下生成跟prototxt同名的rtcfg文件。

5.7.2 Prototxt 层属性可视化编辑

Runtime mapper支持将指定的层切换为高精度层，支持任意层中间上报。支持任意层在层属性Implement选项中选择客户自定义实现的插件库。

将当前Layer指定为HP运算，在右侧Property视图中选择true后，工具自动修改name添加_hp字段，如图5-132。

图 5-132 将当前 Layer 指定为高精度运算



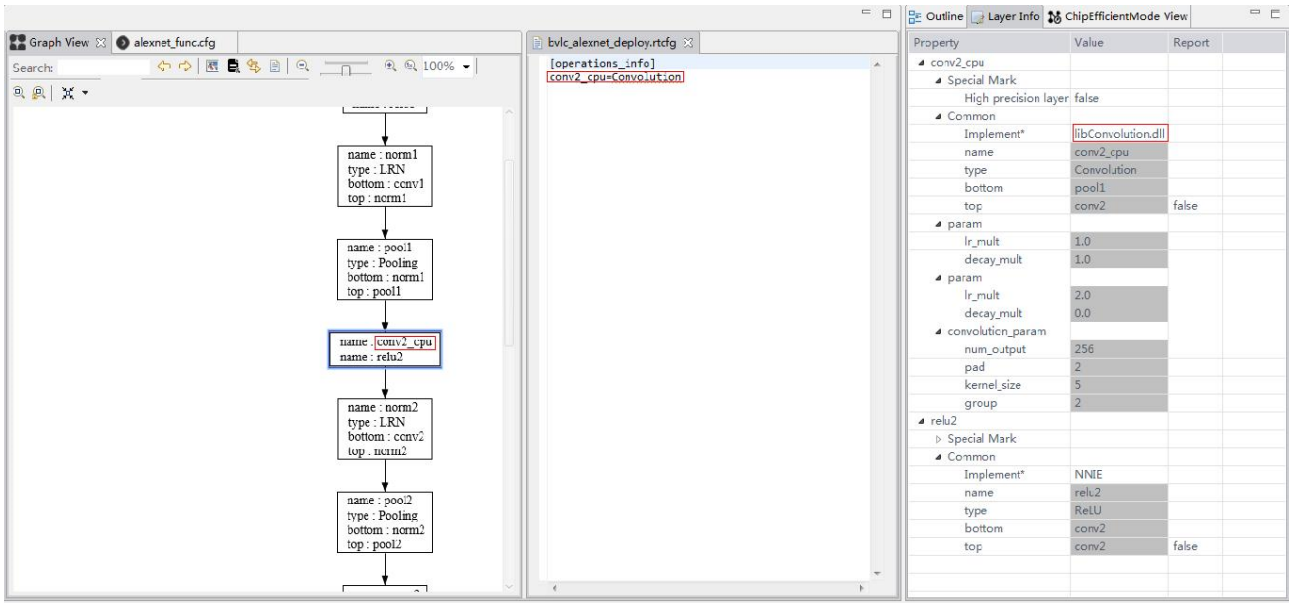
须知

只有当选择层的类型是下列类型中，才可以设置为HP运算：

"Convolution", "Deconvolution", "InnerProduct", "LSTM", "Pooling", "RNN", "PSROIPooling", "ROIPooling", "MatMul", "DepthwiseConv", "PassThrough"，其他的层类型不能设置为HP运算。

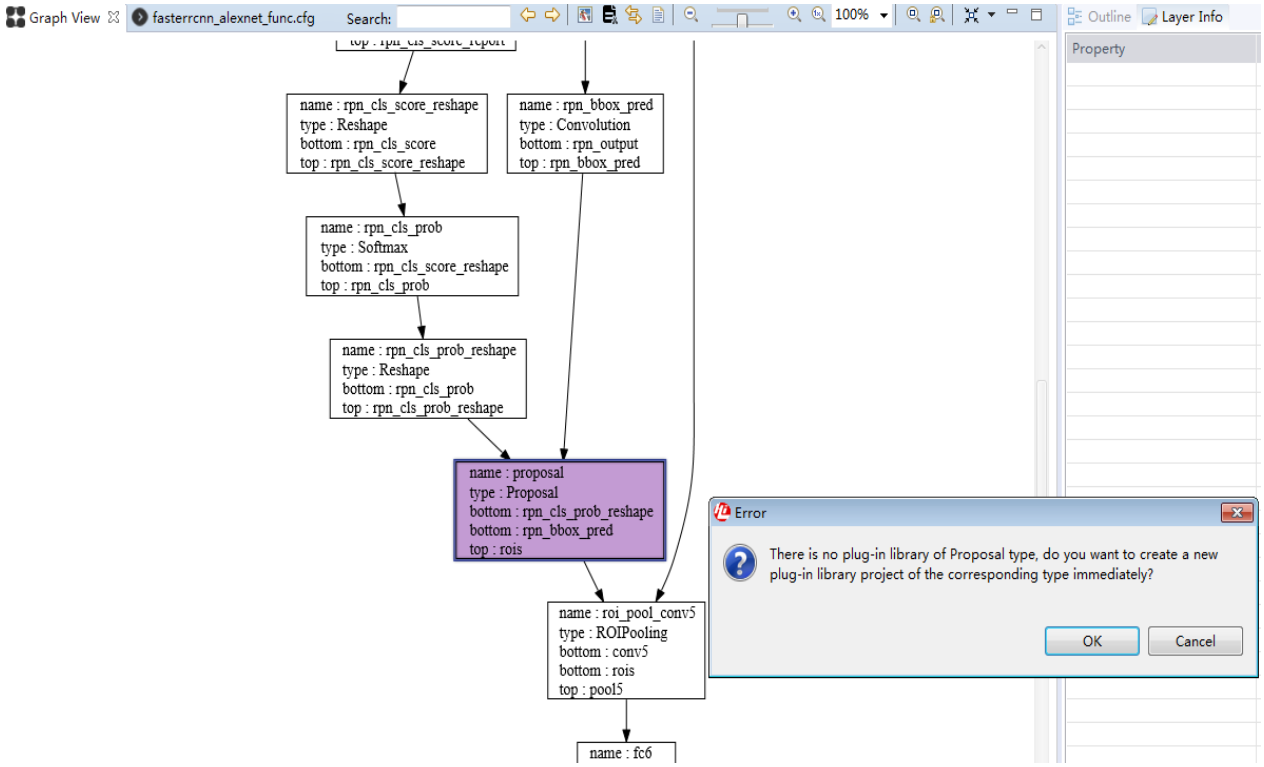
- 将当前Layer指定为CPU运算，在右侧Property视图中的Implement，选择了客户自定义插件后，工具会做下面两个操作：
  - 如果当前选择的层不是Proposal或者Custom，工具会自动给层名添加_cpu字段，
  - 工具会把层名和当前选择的客户自定义插件的名字，写到rtcfg文件中，如图5-133。

图 5-133 将当前 Layer 指定为 cpu 运算



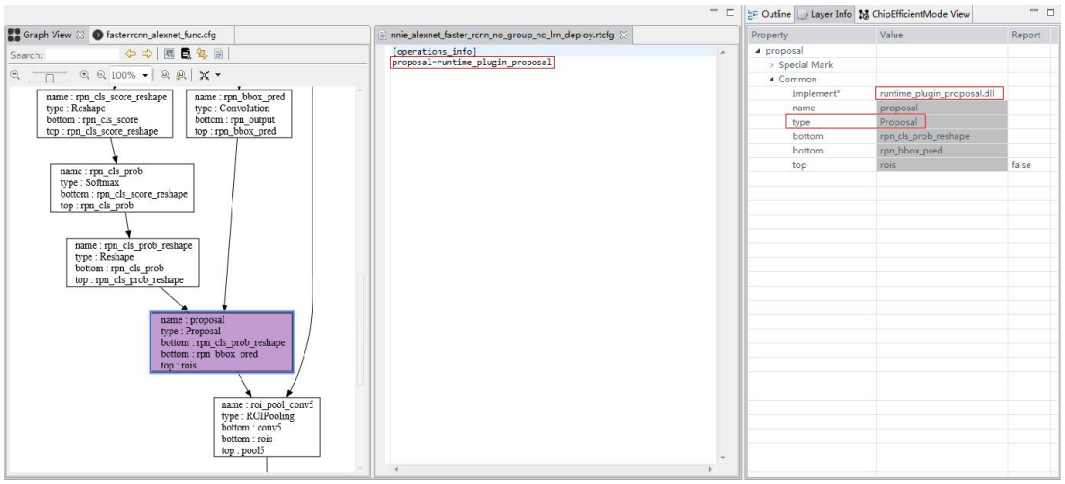
- 让客户自动创建插件，当用户选择Proposal层时，此时如果用户自定义的插件目录中没有Proposal实现的插件库，会弹出是否需要创建插件库的提示如图5-134。

图 5-134 提示是否需要创建 Proposal 类型的插件库



- 让客户自动创建插件，当用户选择Proposal层时，此时如果用户自定义的插件目录有Proposal实现的插件库，会直接选中一个存在的插件库，并把该插件库写到rtcfg文件中，并且proposal不能设置高精度，如图5-135。

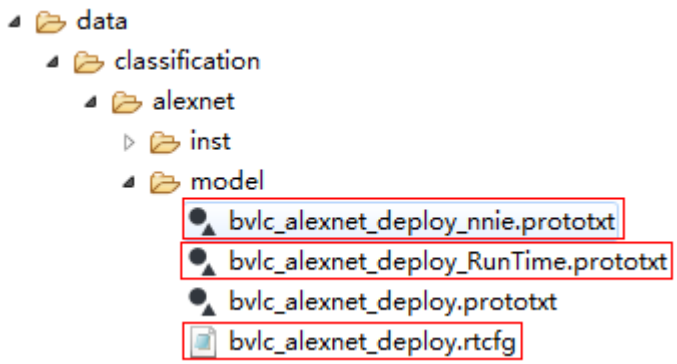
图 5-135 打开 proposal 层的显示



5.7.3 生成 Runtime wk 文件

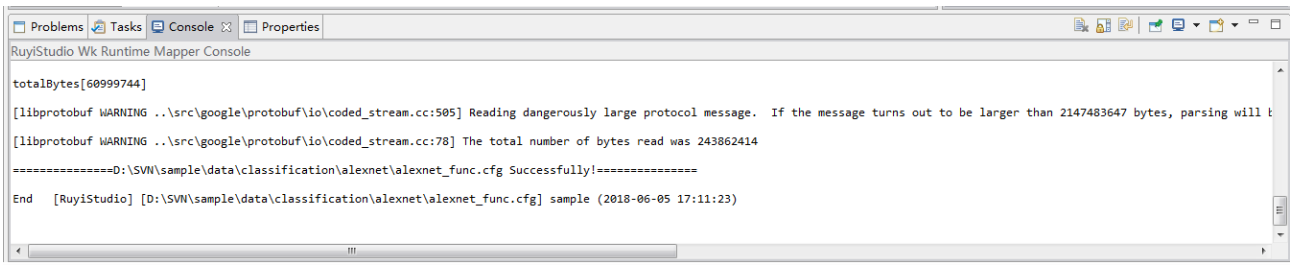
配置好cfg之后，点击左上角工具栏中的Make WK按钮，工具会将当前的xxx.cfg传给runtime_mapper，还需传入NNIE prototxt、runtime prototxt和rtcfg文件，NNIE prototxt和runtime prototxt都是在运行的时候会生成，这些要传入的文件都和prototxt在同一个目录。如图5-136所示：

图 5-136 运行 Runtime 所需要传入的文件



转化过程会在控制台显示，转化成功或失败后会在Project Explorer中的当前工程的目录下生成多个dot文件及编译过程中的debug和error对应的log，如图5-137所示。

图 5-137 运行 Runtime 的界面



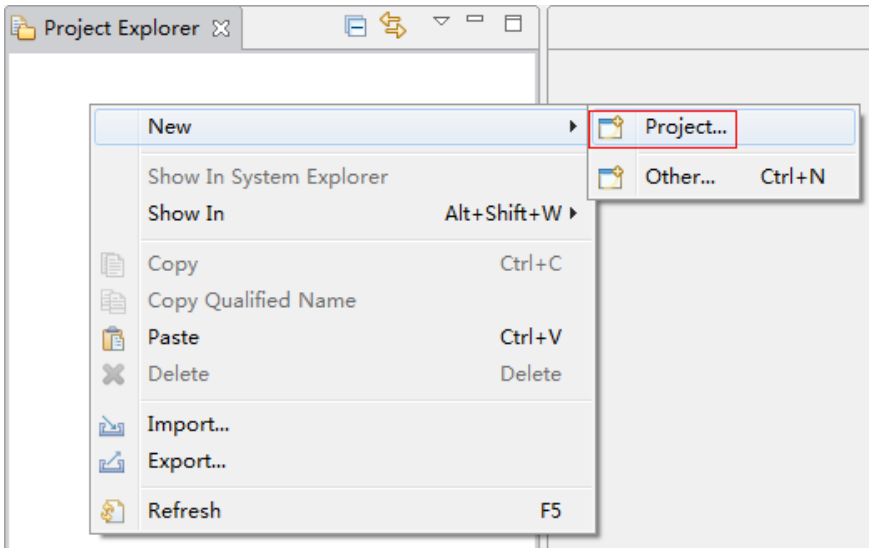
5.8 客户自定义插件

提供用户自己创建自定义插件库，并且把生成的库添加到用户自定义路径下，供runtime调用。

5.8.1 创建工程

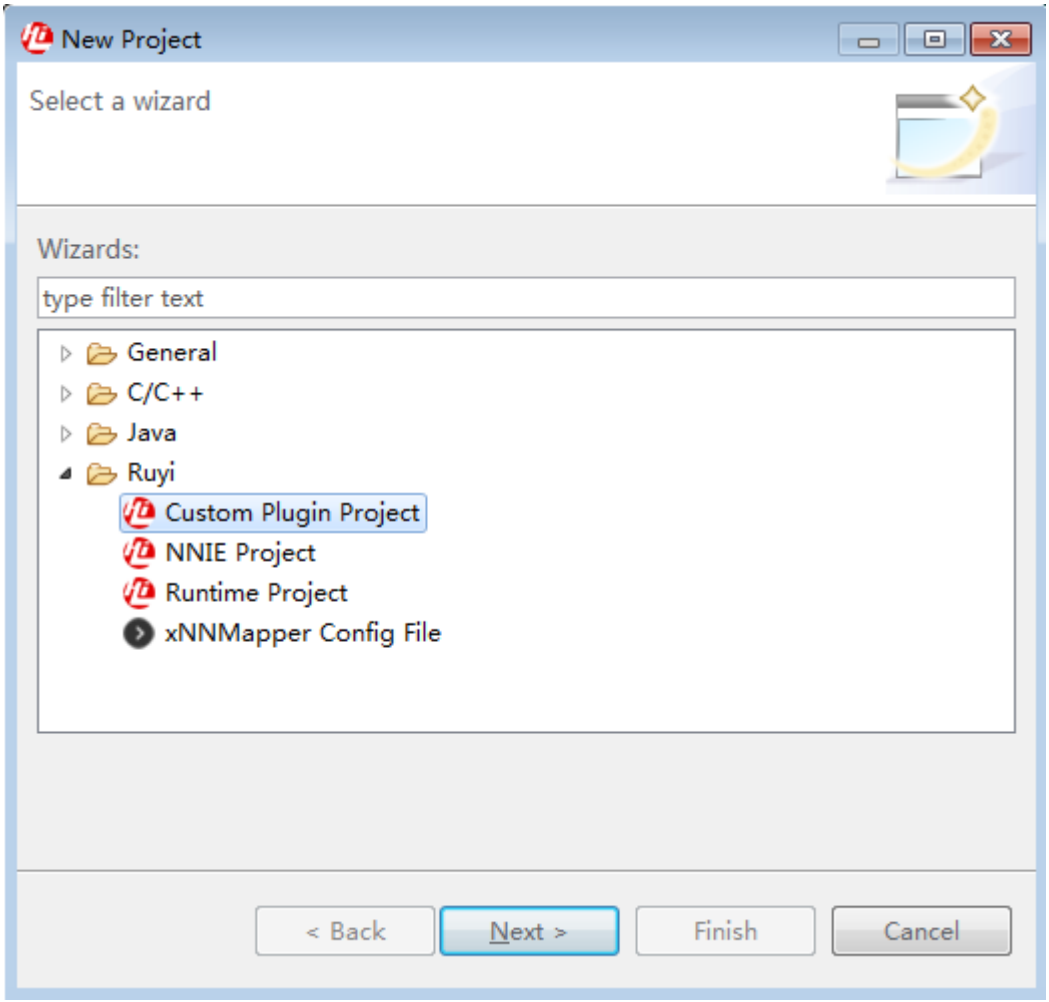
步骤1 点击鼠标右键，弹出图5-138：

图 5-138 通过鼠标右键创建 Project



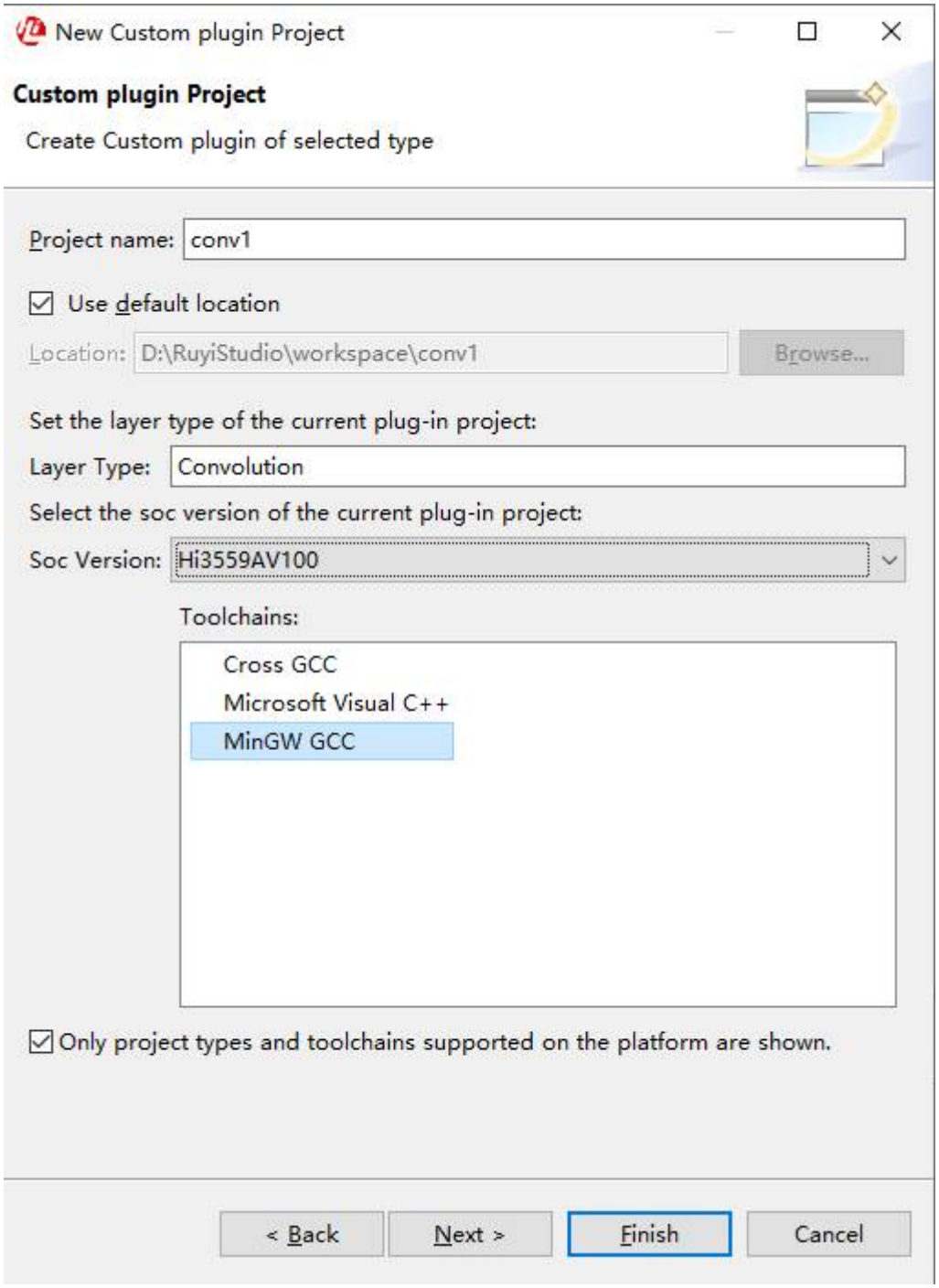
步骤2 选择NNIE project，点击Next，如图5-139所示。

图 5-139 选择 Custom plugin Project



步骤3 设置Project名称，设置 layer type，选择芯片型号，选择MinGW GCC，点击Finish完成工程创建；注意只有当MinGW安装好之后才会在Toolchains里显示MinGW GCC，如图5-140所示。

图 5-140 设置 Project、layer type 的名字和芯片型号并确认创建工程



----结束

须知

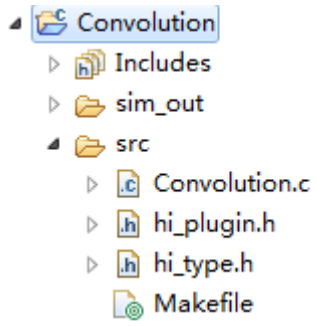
创建自定义插件的编译链只支持minGW GCC编译链。

5.8.1.1 工程文件说明

创建后的工程如图如图5-141所示。包含两个空文件夹和一个配置文件：

- *.c：用于实现接口的文件；
- hi_plugin.h：接口的头文件；
- hi_type.h：自定义的头文件；
- Makefile：用于生成该项目的文件。

图 5-141 创建工程后的工程视图



其中*.c文件已经实现了HI_NodePlugin_getNodeType的接口，HI_NodePlugin_Compute需要客户实现，生成的*.c，如图5-142所示。

图 5-142 生成的*.c 文件

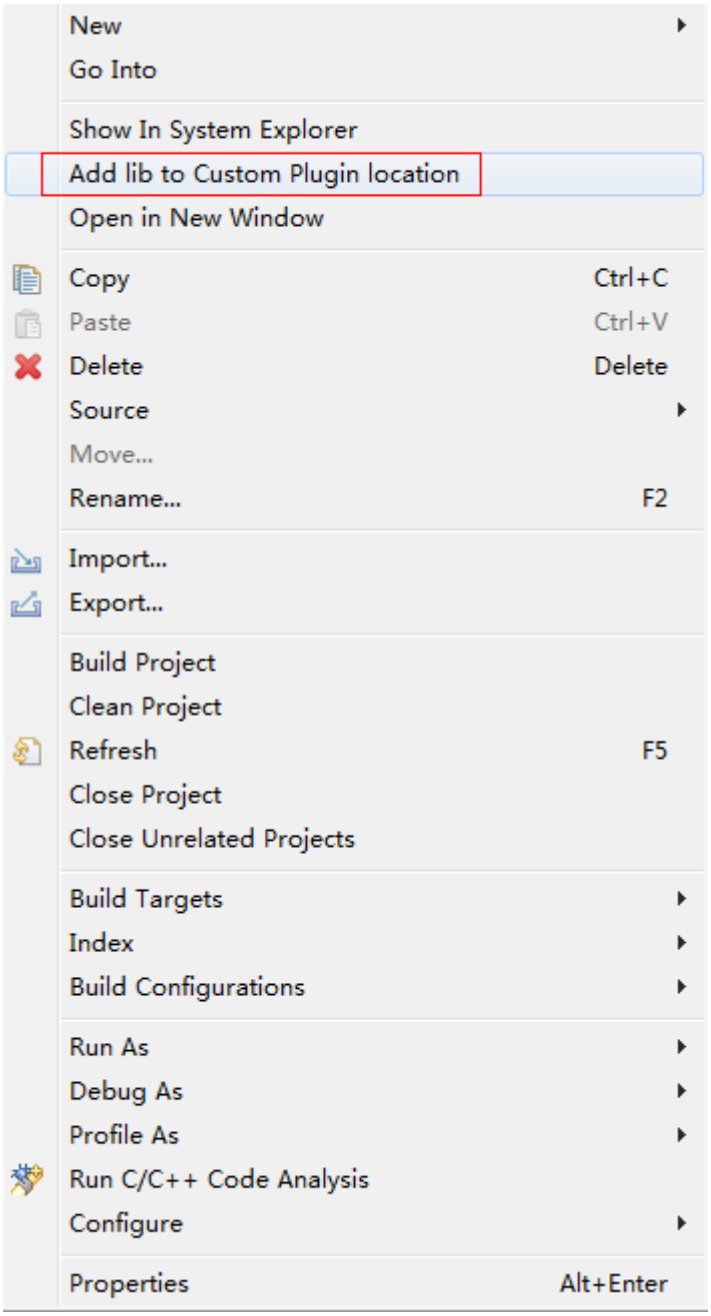
Convolution.c

```c
#include "hi_plugin.h"
#include "string.h"

HI_S32 HI_NodePlugin_Compute(const HI_NodePlugin_Operand_S *pstInputs, HI_U32 u32InputNum,
        HI_NodePlugin_Operand_S *pstOutputs, HI_U32 u32Outputs, HI_NodePlugin_NodeParam_S* pstHyperParam, HI_NodePlugin_NodeParam_S* pstTrainingParam)
{
    return HI_SUCCESS;
}

/*****************************************************************************
    *   Prototype    : HI_NodePlugin_getNodeType
    *   Description  : return the type of layer, such as convolution. proposal
    *                        please do not modify the code of this fucntion
    *   Parameters   : HI_CHAR pszNodeType[]:  output the type of layer
    *   Return Value : HI_SUCCESS: Success; Error codes: Failure.
    *****************************************************************************/
HI_S32 HI_NodePlugin_getNodeType(HI_CHAR *pszNodeType)
{
    strcpy(pszNodeType, "Convolution");
    return HI_SUCCESS;
}
```

5.8.2 添加客户自定义插件到客户自定义插件路径下

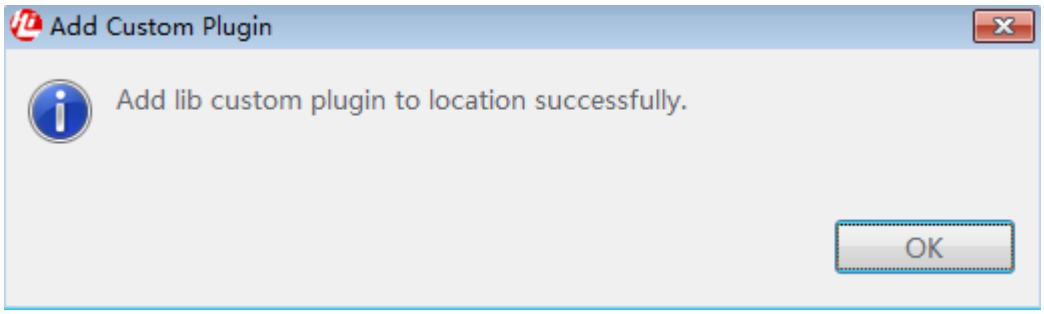
步骤1 选择项目，点击鼠标右键，选择Add lib to Custom plugin location，如图5-143所示。

图 5-143 通过鼠标右键选择 Add lib to Custom plugin location



步骤2 弹出选择dll的对话框，选择项目生成的dll文件。添加成功后会弹出一个提示框。如图5-144所示。

图 5-144 添加 dll 成功之后的提示

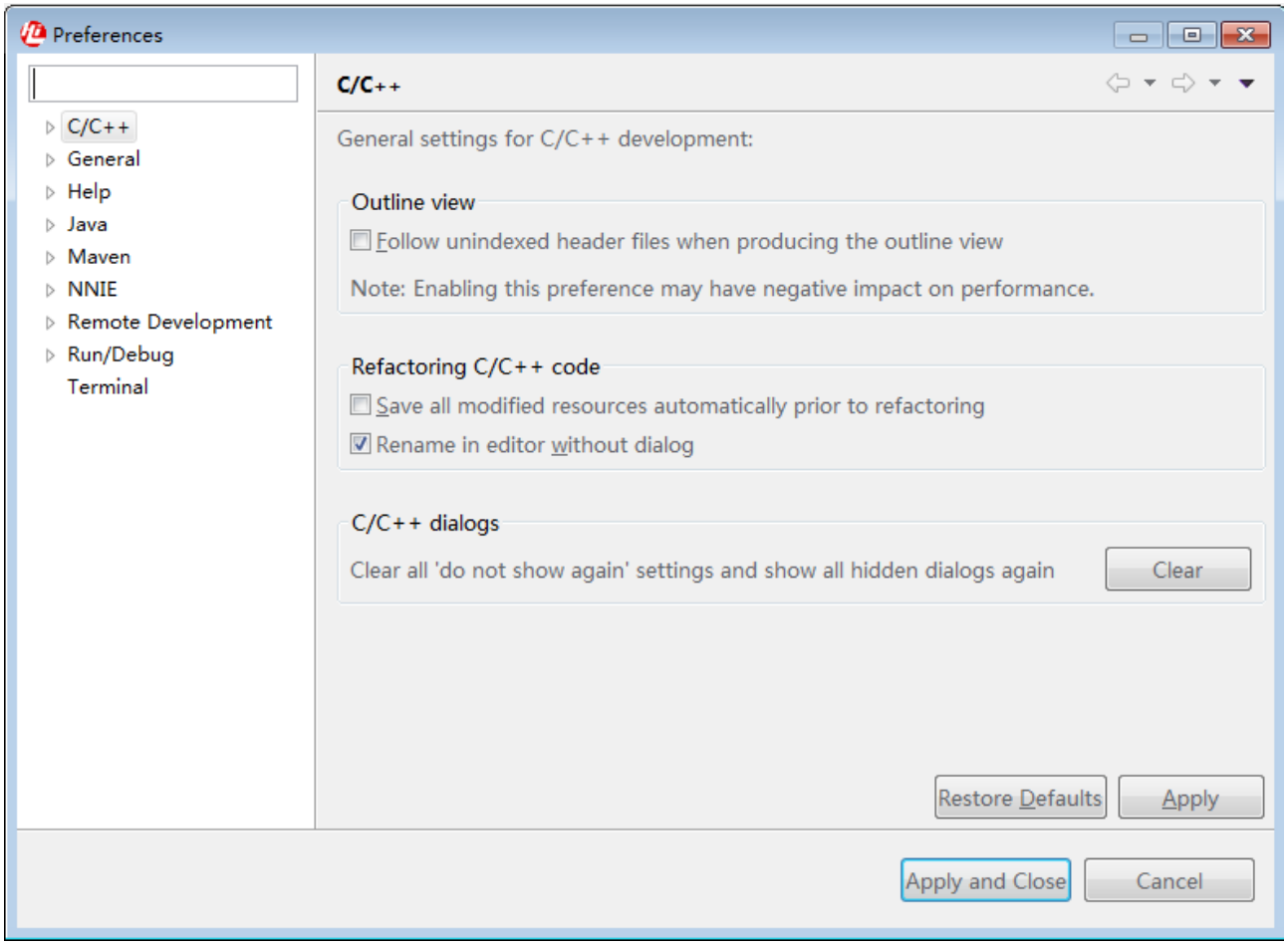


----结束

5.8.3 设置客户自定义插件路径

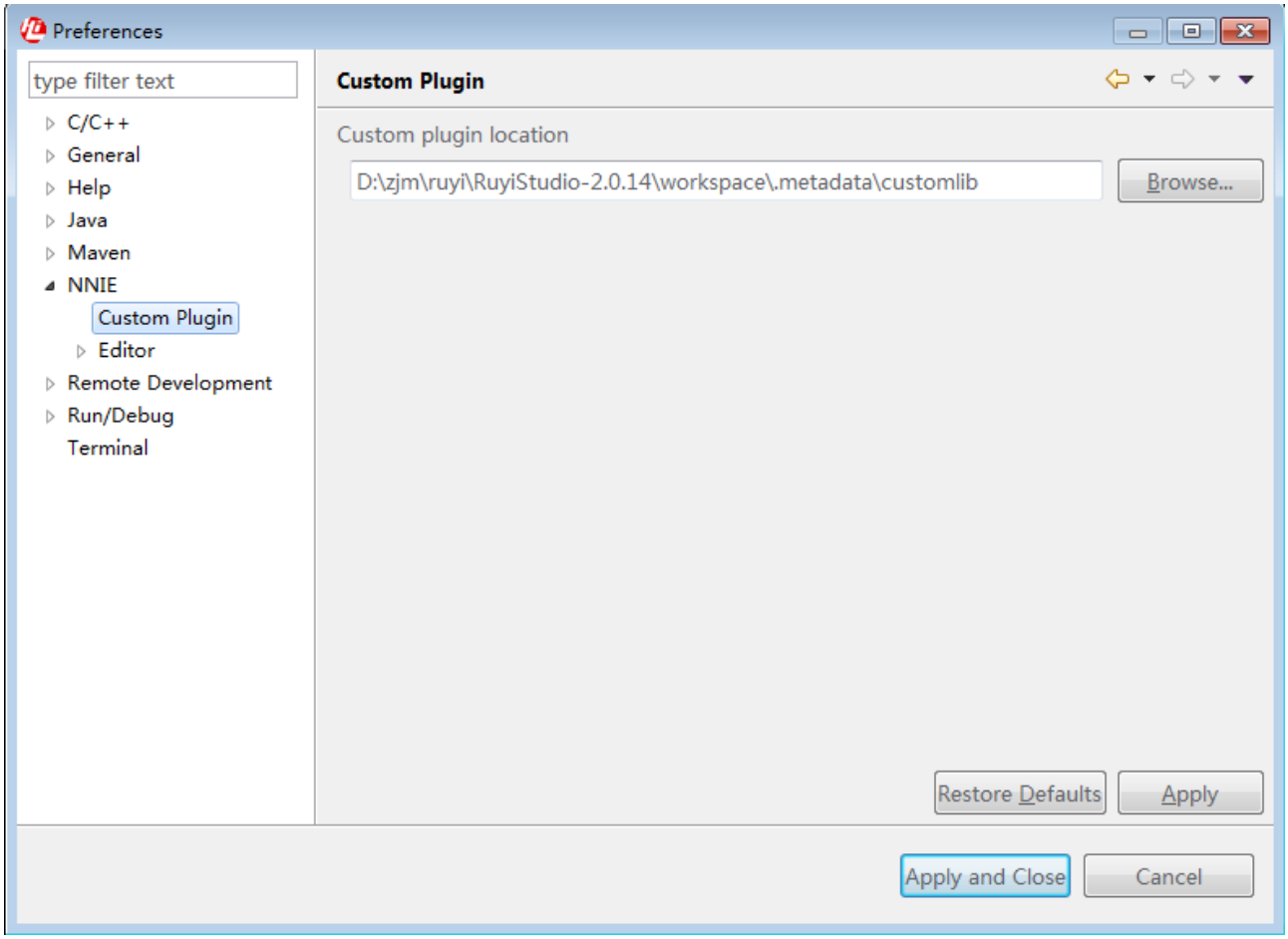
步骤1 依次点击WindowàPreferences，弹出图5-145：

图 5-145 打开 preferences 页面



步骤2 选择NNIE->Custom Plugin。如图5-146：

图 5-146 打开设置 Custom Plugin 页面



步骤3 点击Browse按钮，设置路径，设置完路径后点击Apply 或者Apply and Close就完成了设置。

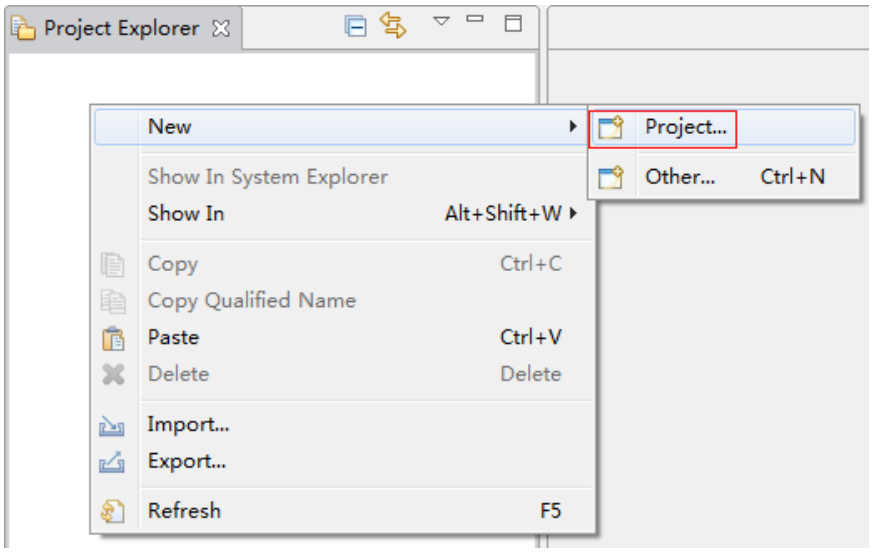
----结束

5.9 创建 Runtime 工程和切换芯片

5.9.1 创建 runtime 工程

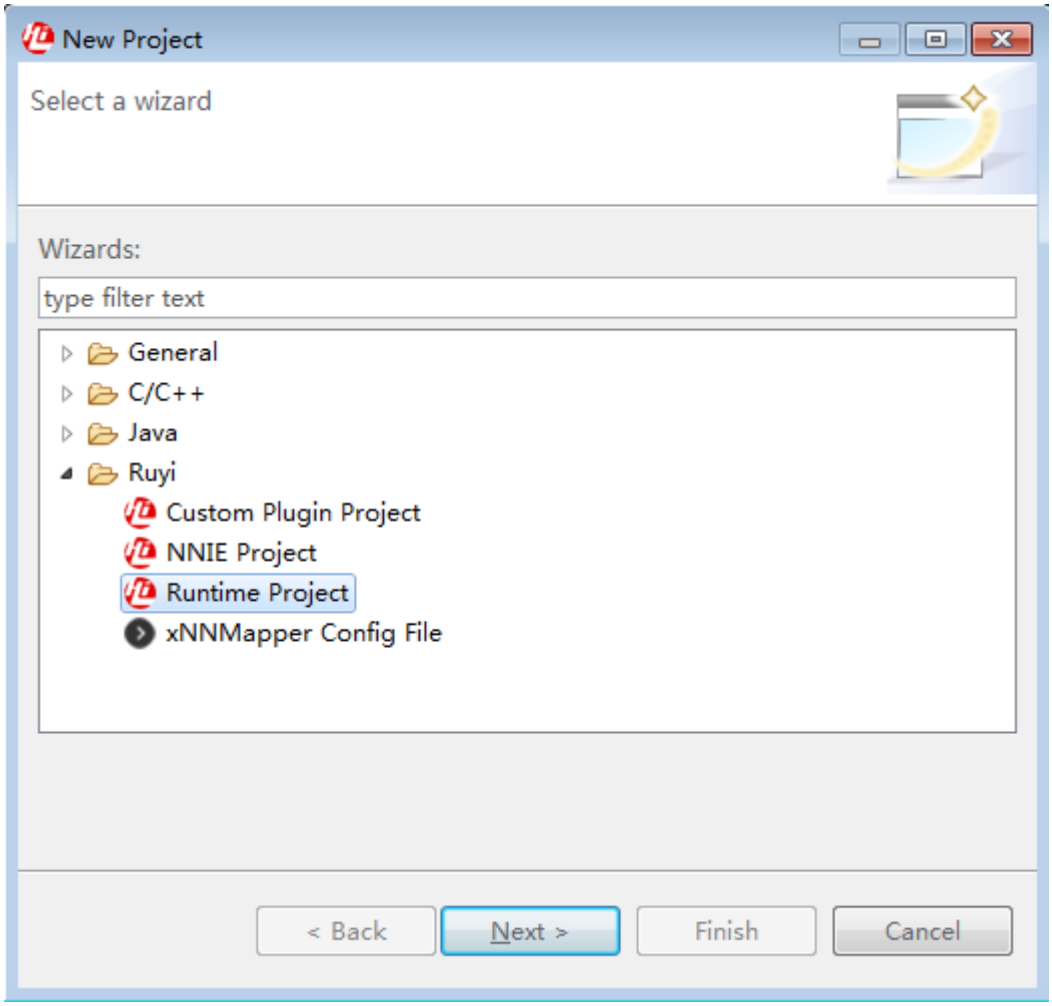
步骤1 工程创建，通过Project Explorer视图中鼠标右键创建Project，如图5-147所示。

图 5-147 通过鼠标右键创建 Project



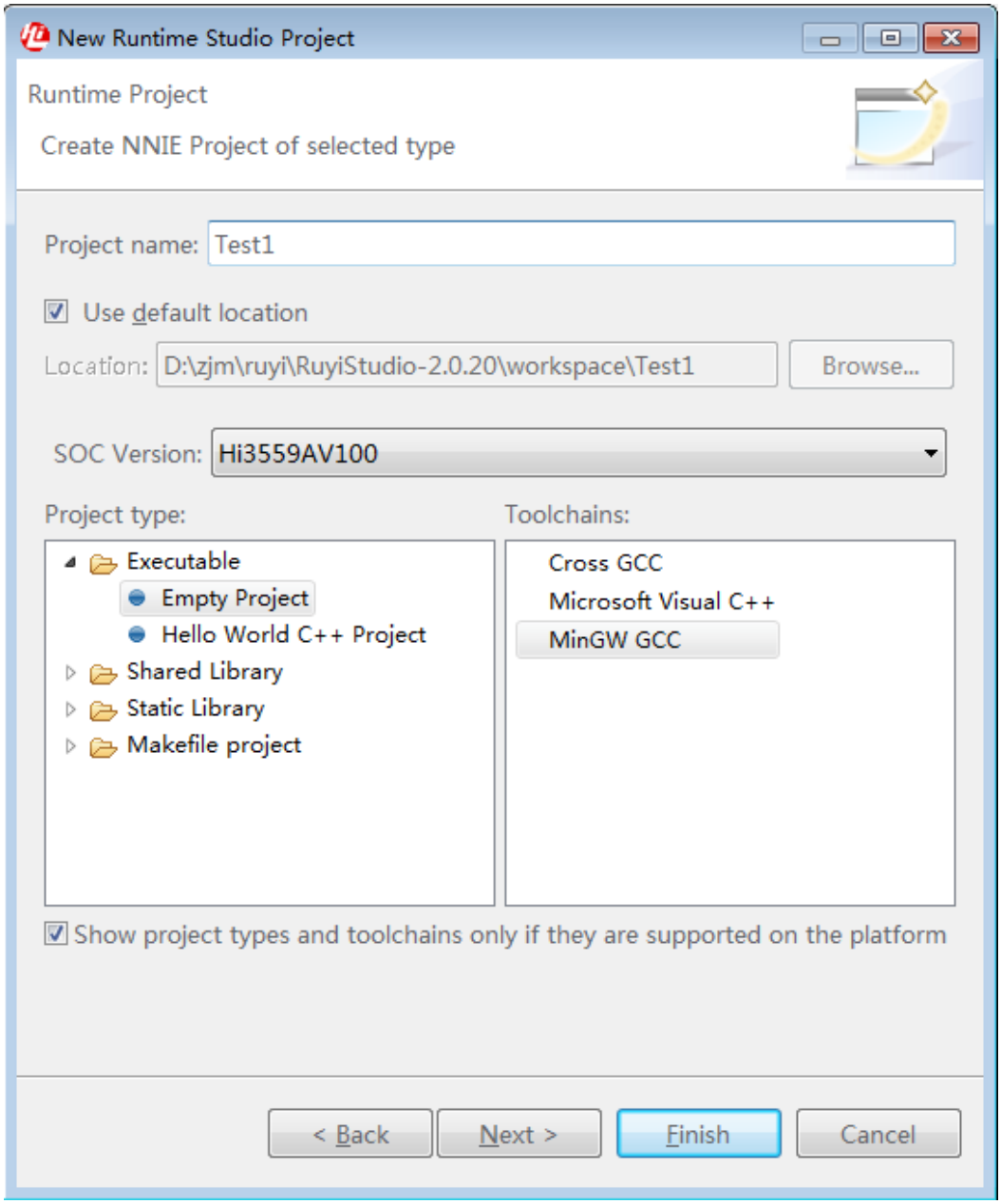
步骤2 选择Runtime project，点击Next，如图5-148所示。

图 5-148 选择 Runtime Project



步骤3 设置Project名称，选择Hi3559AV100芯片型号，选择MinGW GCC，点击Finish完成工程创建；注意只有当MinGW安装好之后才会在Toolchains里显示MinGW GCC，如图5-149所示。

图 5-149 设置 Project 的名字并确认创建工程

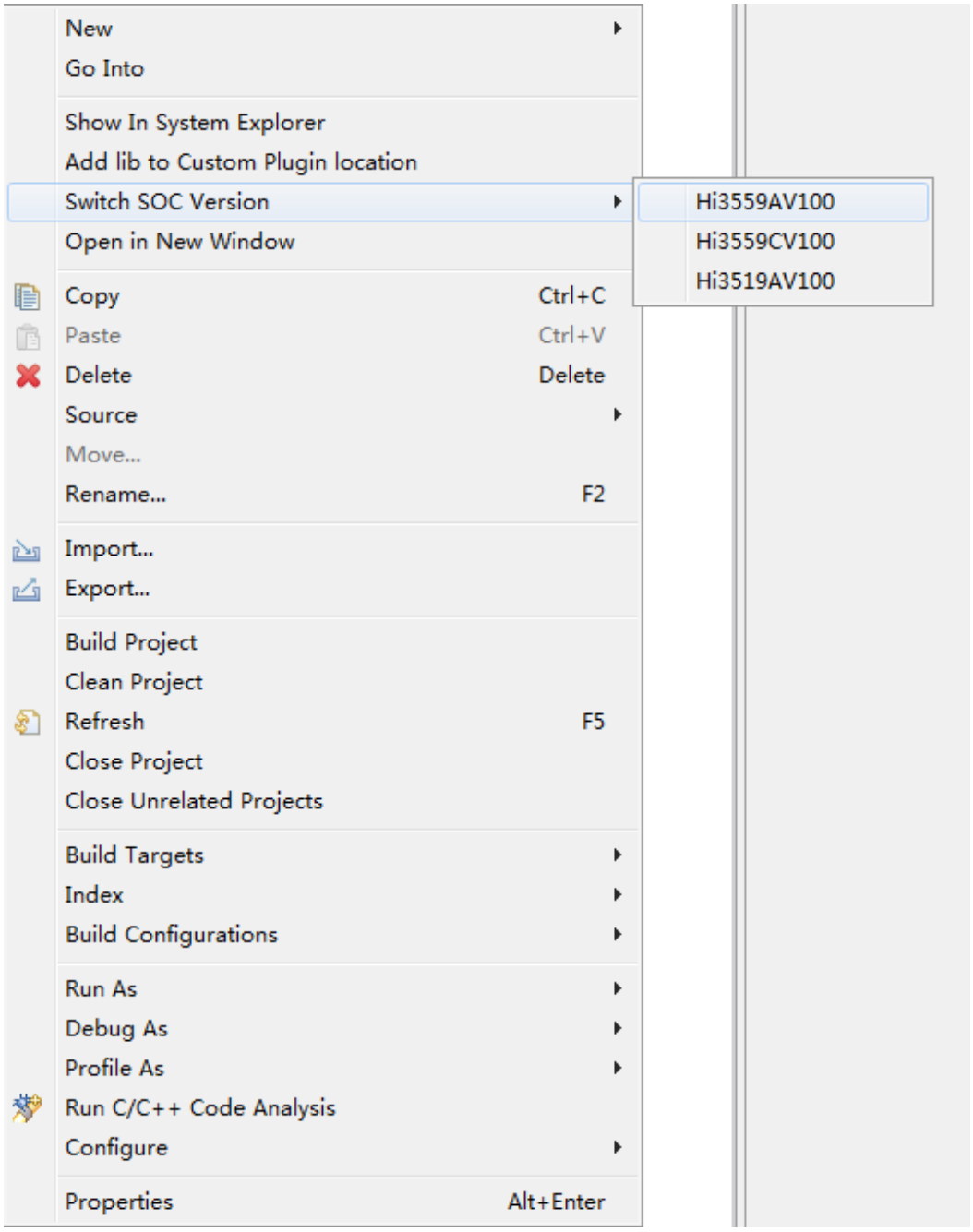


----结束

5.9.2 切换芯片

步骤1 选择项目，点击右键弹出，菜单栏，选择Switch SOC Version菜单，会显示芯片列表，如图5-150所示。

图 5-150 切换芯片



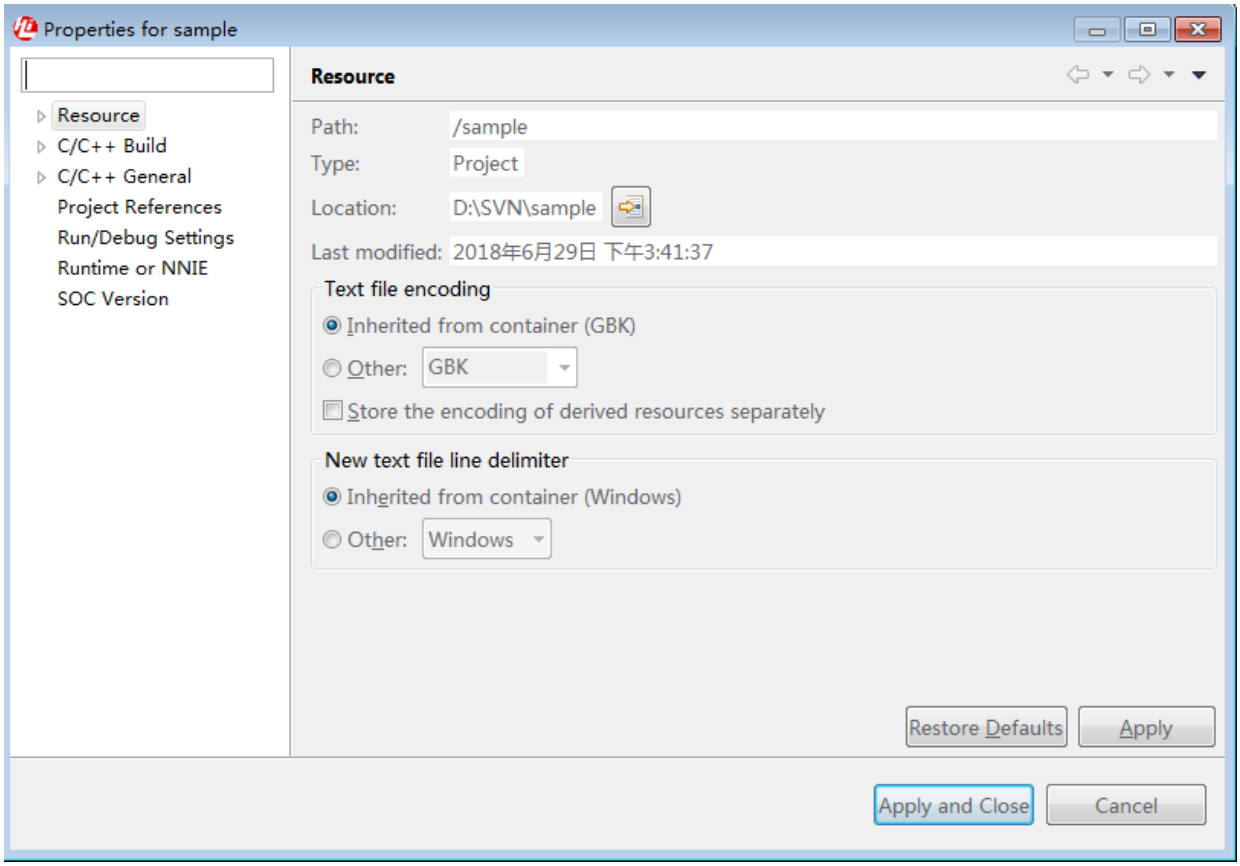
步骤2 选择对应的芯片，如果该项目是NNIE项目，则该项目会切换到当前的芯片。

----结束

5.9.3 查看项目 Model 和芯片型号

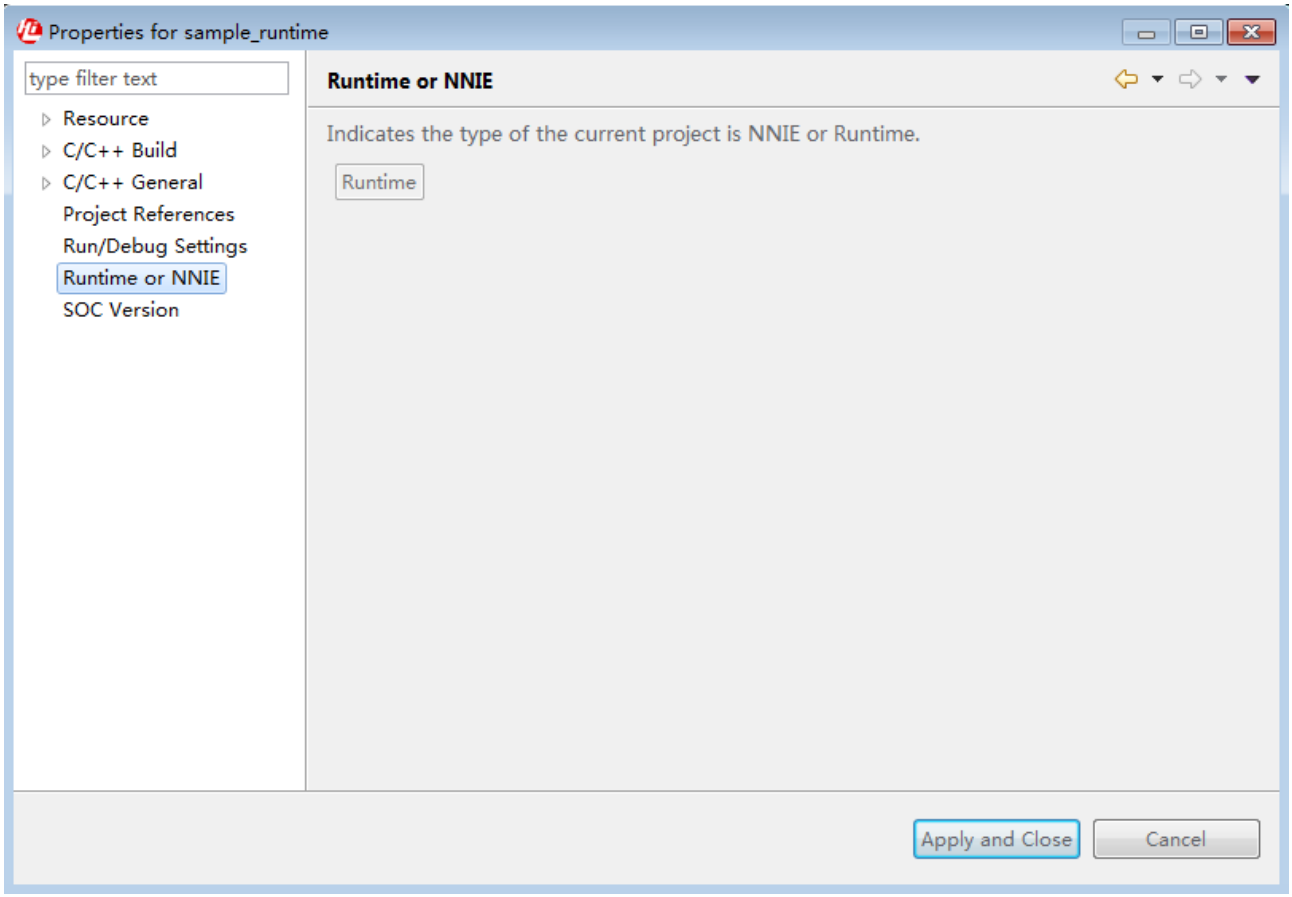
步骤1 选择项目，点击鼠标右键，在弹出的菜单栏选择Properties，弹出对话框如图5-151所示。

图 5-151 属性页



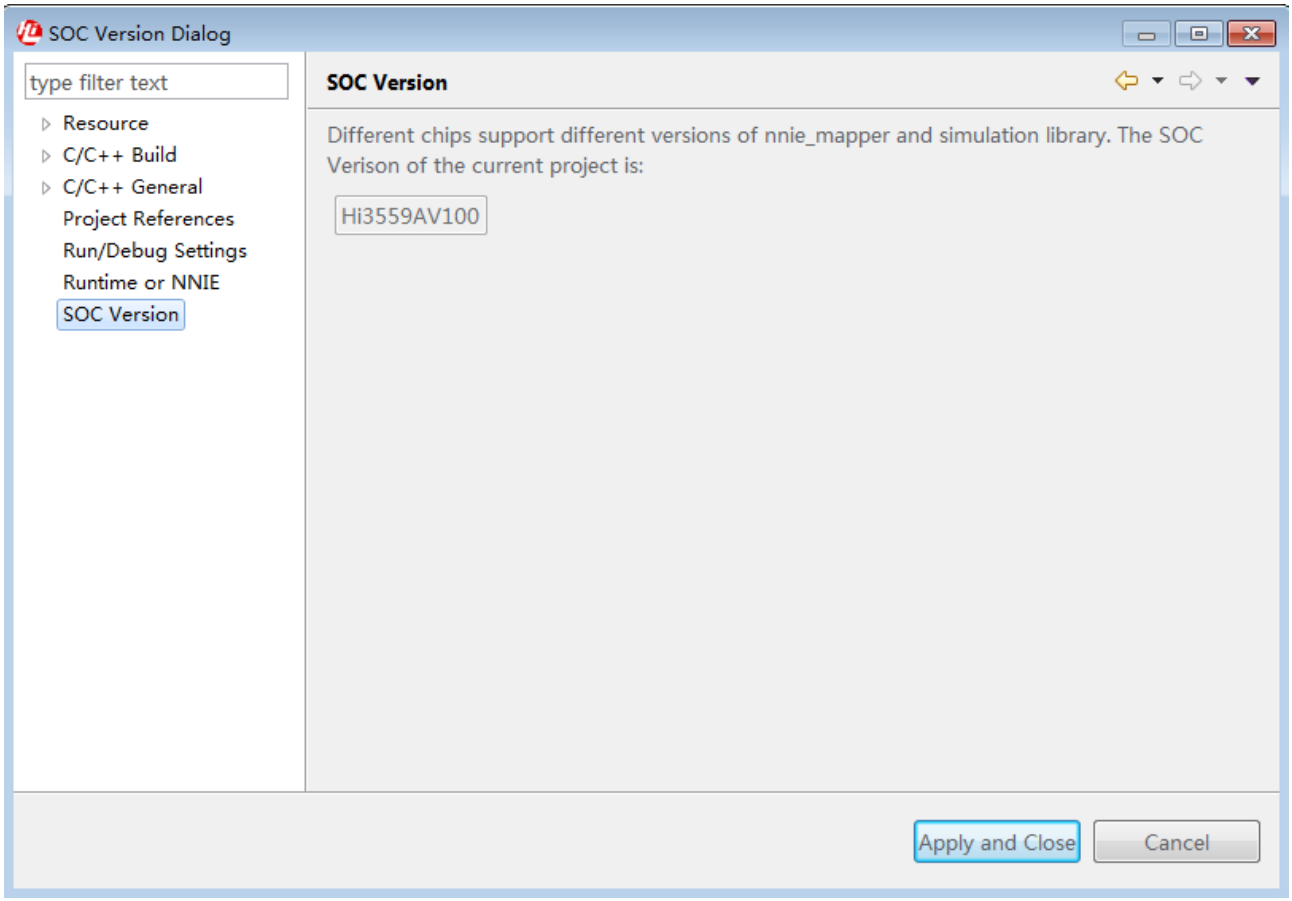
步骤2 选择Runtime or NNIE查看当前项目所属Model，如图5-152：

图 5-152 项目 Model 页



步骤3 选择SOC Version查看当前项目的芯片型号如图5-153。

图 5-153 芯片型号页



----结束

5.10 FAQ

5.10.1 工具启动时一直卡在启动画面“Loading org.eclipse…”无法打开工具的解决办法

问题描述

工具启动时一直卡在启动画面，如图5-154所示，我该怎么办?

图 5-154 工具启动卡主无法打开的现象

解决办法

步骤1 首先，关闭工具；

步骤2 找到当前工具加载的workspace路径，找到workspace下的隐藏文件.matadata文件，删除如下路径的.metadata\.plugins\org.eclipse.e4.workbench\workbench.xmi文件；

步骤3 最后重启工具，如果在以上路径下找不到workbench.xmi，则删除整个.matadata文件夹后再重启工具。

----结束

5.10.2 打印 caffe 中间输出或还原网络时提示 Layer xxx with type xxx is not supported, please refer to chapter 3.1.4 and FAQ of "HiSVP Development Guide.pdf" to extend caffe!

当前打印caffe中间输出和还原网络的功能提供的一键配置python+caffe的环境中的caffe并不支持**3.1.4.3 扩展层在prototxt中的定义** 提到的DepthwiseConv，RReLU，MatMul等扩展层，而本文档提供的bat脚本的配置，只支持caffe的标准层和PassThrough，Permute，ROIPooling，PSROIPooling，UpSample，Normalize这些扩展层，如果涉及到扩展层，需要自行配置。参考**3.1.4 扩展层规则**章节的描述，具体操作如下：（默认已经系统安装了cmake，并且将cmake可执行文件所在路径添加到了系统环境变量中，默认已经安装了VS，推荐使用VS2015）

步骤1 从**https://github.com/BVLC/caffe/tree/windows下载caffe_windows.zip**包，解压到自定义目录；

步骤2 修改scripts\build_win.cmd文件，配置VS的版本号；

步骤3 修改scripts\build_win.cmd文件，依照自己的情况设置CPU_ONLY，若不用到GPU，则设置为1，否则设置为0，在本例子中，默认为CPU_ONLY为1的场景；

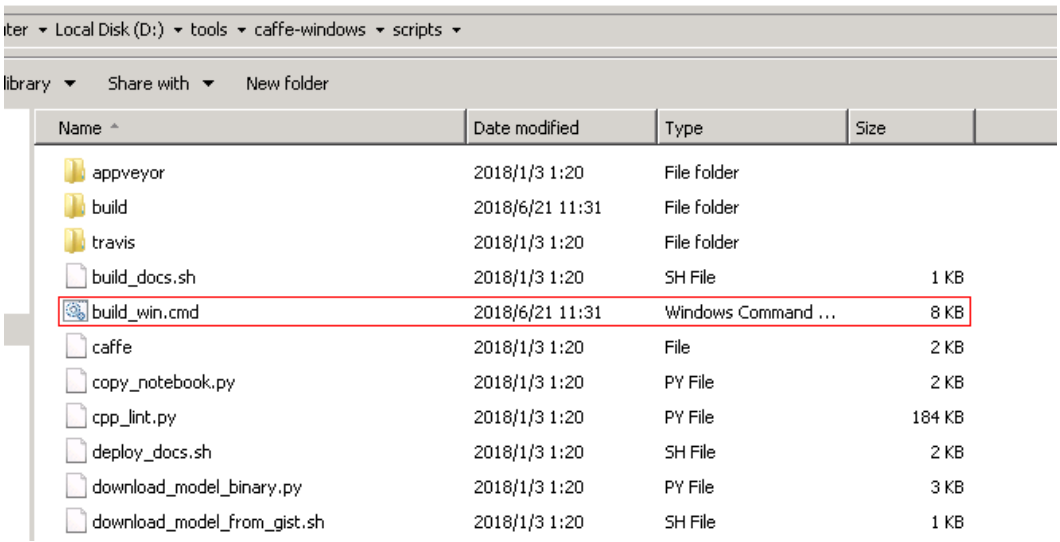
步骤4 配置python的版本号为3，也是与我们安装的python版本号相对应；

图 5-155 修改 build_win.cmd 文件中的部分配置

```
74  if NOT DEFINED WITH_NINJA set WITH_NINJA=0
75  :: Change to 1 to build caffe without CUDA support
76  if NOT DEFINED CPU_ONLY set CPU_ONLY=1
77  :: Change to generate CUDA code for one of the following GPU architectures
78  :: [Fermi  Kepler  Maxwell  Pascal  All]
79  if NOT DEFINED CUDA_ARCH_NAME set CUDA_ARCH_NAME=Auto
80  :: Change to Debug to build Debug. This is only relevant for the Ninja generator the Visu
81  if NOT DEFINED CMAKE_CONFIG set CMAKE_CONFIG=Release
82  :: Set to 1 to use NCCL
83  if NOT DEFINED USE_NCCL set USE_NCCL=0
84  :: Change to 1 to build a caffe.dll
85  if NOT DEFINED CMAKE_BUILD_SHARED_LIBS set CMAKE_BUILD_SHARED_LIBS=0
86  :: Change to 3 if using python 3.5 (only 2.7 and 3.5 are supported)
87  if NOT DEFINED PYTHON_VERSION set PYTHON_VERSION=3
```

步骤5 点击运行build_win.cmd文件；

图 5-156 运行 build_win.cmd 文件



步骤6 首次运行，会有如下打印，并且会出现卡顿，此时会发现，在C:\Users\Username(用户名，视用户本身用户名的情况而定)\.caffe\dependencies\download文件夹下会出现一个大小为0的libraries_v140_x64_py35_1.1.0.tar.bz2文件，需要手动从https://github.com/willyd/caffe-builder/releases下载**libraries_v140_x64_py35_1.1.0.tar.bz2**并替换到该位置；

图 5-157 运行打印，下载卡顿

```
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler using: Visual Studio 14 2015 Win64
-- Check for working CXX compiler using: Visual Studio 14 2015 Win64 -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found PythonInterp: D:/python/python.exe (found suitable version "3.5.4", min
imum required is "2.7")
-- Downloading prebuilt dependencies to C:/Users/Administrator/.caffe/dependenci
es/download/libraries_v140_x64_py35_1.1.0.tar.bz2
```

步骤7 修改caffe-windows\scripts下的download_prebuilt_dependencies.py文件，注释掉如下内容以及\caffe-windows\cmake \WindowsDownloadPrebuiltDependencies.cmake；

图 5-158 download_prebuilt_dependencies.py

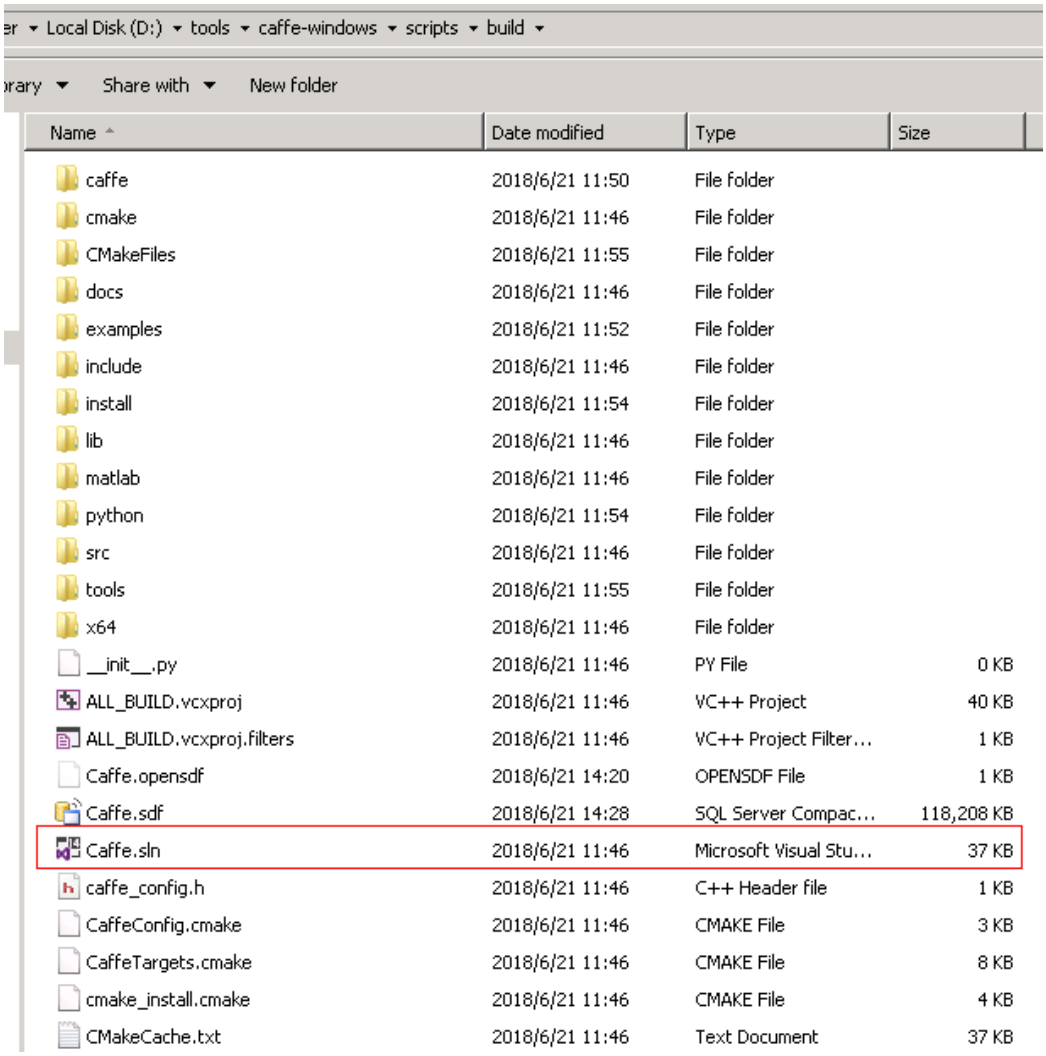
```
45      dep_filename = os.path.split(url)[1]
46      # Download binaries
47      #print("Downloading dependencies ({}). Please wait...".format(dep_filename))
48      #urllib.request.urlretrieve(url, dep_filename, reporthook)
49      #if not model_checks_out(dep_filename, sha1):
50      #    print('ERROR: dependencies did not download correctly! Run this again.')
51      #    sys.exit(1)
52      #print("\nDone.")
```

图 5-159 WindowsDownloadPrebuiltDependencies.cmake

```
55      # download and extract the file if it does not exist or if does not match the sha1
56      get_filename_component(_download_filename ${DEPENDENCIES_URL} NAME)
57      set(_download_path ${CAFFE_DEPENDENCIES_DOWNLOAD_DIR}/${_download_filename})
58      #set(_download_file 1)
59      #if(EXISTS ${_download_path})
60      #    file(SHA1 ${_download_path} _file_sha)
61      #    if("${_file_sha}" STREQUAL "${DEPENDENCIES_SHA}")
62      #        set(_download_file 0)
63      #    else()
64      #        set(_download_file 1)
65      #        message(STATUS "Removing file because sha1 does not match.")
66      #        file(REMOVE ${_download_path})
67      #    endif()
68      #endif()
69      #if(_download_file)
70      #    message(STATUS "Downloading prebuilt dependencies to ${_download_path}")
71      #    file(DOWNLOAD "${DEPENDENCIES_URL}"
72      #                  "${_download_path}"
73      #                  EXPECTED_HASH SHA1=${DEPENDENCIES_SHA}
74      #                  SHOW_PROGRESS
75      #                  )
76      #    if(EXISTS ${CAFFE_DEPENDENCIES_DIR}/libraries)
77      #        file(REMOVE_RECURSE ${CAFFE_DEPENDENCIES_DIR}/libraries)
78      #    endif()
79      #endif()
```

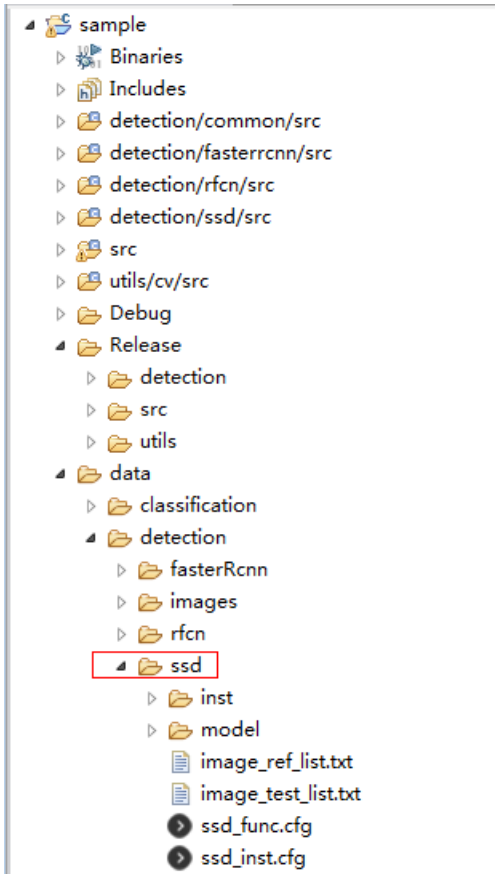
步骤8 运行build_win.cmd，就会生成VS工程文件，整个过程大概要用7分钟，此时的工程文件就是编译后未扩展的caffe1.0工程；

图 5-160 Caffe1.0 基础工程



步骤9 参照3.1.4 扩展层规则章节的描述，客户需要手动对caffe.proto文件进行修改，注册层信息。以检测网的ssd网络为例：

图 5-161 检测网的 ssd 网络



在nnie_ssd_deploy.prototxt文件，会发现有Normalize和Permute这两个拓展层。

图 5-162 nnie_ssd_deploy.prototxt 文件的 Normalize 层

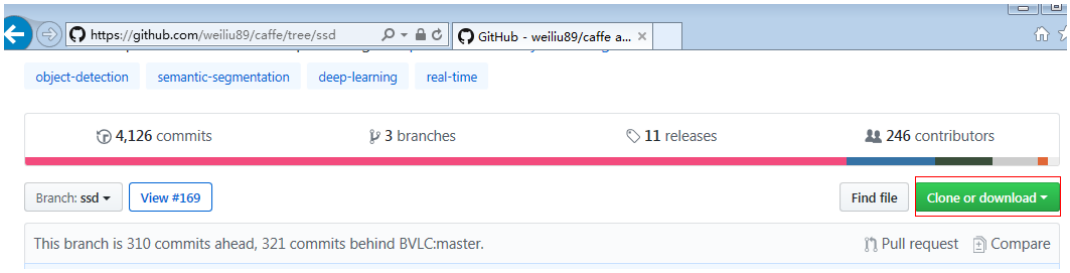
```
layer {
  name: "conv4_3_norm"
  type: "Normalize"
  bottom: "conv4_3"
  top: "conv4_3_norm"
  norm_param {
    across_spatial: false
    scale_filler {
      type: "constant"
      value: 20
    }
    channel_shared: false
  }
}
```


图 5-163 nnie_ssd_deploy.prototxt 文件的 Permute 层

```
856  layer {
857    name: "conv4_3_norm_mbox_loc_perm"
858    type: "Permute"
859    bottom: "conv4_3_norm_mbox_loc"
860    top: "conv4_3_norm_mbox_loc_perm"
861    permute_param {
862      order: 0
863      order: 2
864      order: 3
865      order: 1
866    }
867  }
```

步骤10 从**3.1.4.4 扩展层的参考设计**提供的Normalize层和Permute层的开源工程地址**https://github.com/weiliu89/caffe/tree/ssd**下载相关代码；

图 5-164 下载 caffe-ssd.zip 包



步骤11 修改caffe-windows\src\caffe\proto\caffe.proto文件，在LayerParameter中增加相关拓展层的定义。可以参考caffe-ssd\src\caffe\proto\caffe.proto文件的写法；

图 5-165 增加 NormalizeParameter 和 PermuteParameter 的定义（1）

caffe.proto | common.hpp | upsample_layer.cpp | roi_pooling_layers.cpp

```
403    optional SliceParameter slice_param = 126;
404    optional TanHParameter tanh_param = 127;
405    optional ThresholdParameter threshold_param = 128;
406    optional TileParameter tile_param = 138;
407    optional WindowDataParameter window_data_param = 129;
408
409    optional ROIPoolingParameter roi_pooling_param = 100000;
410    optional NormalizeParameter norm_param = 100001;
411    //optional RSROIPoolingParameter psroi_pooling_param = 100002;
412    optional UpsampleParameter upsample_param = 100003;
413    optional PermuteParameter permute_param = 100004;
```


图 5-166 增加 NormalizeParameter 和 PermuteParameter 的定义（2）

caffe.proto | common.hpp | upsample_layer.cpp | roi_pooling_layers.cpp

```
1430
1431  // Message that stores parameters used by NormalizeLayer
1432  message NormalizeParameter {
1433    optional bool across_spatial = 1 [default = true];
1434    // Initial value of scale. Default is 1.0 for all
1435    optional FillerParameter scale_filler = 2;
1436    // Whether or not scale parameters are shared across channels.
1437    optional bool channel_shared = 3 [default = true];
1438    // Epsilon for not dividing by zero while normalizing variance
1439    optional float eps = 4 [default = 1e-10];
1440  }
1441
1442  message PermuteParameter {
1443    // The new orders of the axes of data. Notice it should be with
1444    // in the same range as the input data, and it starts from 0.
1445    // Do not provide repeated order.
1446    repeated uint32 order = 1;
1447  }
```

步骤12 将NormalizeParamter和PermuteParamter的实现代码从caffe-ssd\src\caffe\layers拷贝到caffe-windows\src\caffe\layers文件夹下，包括头文件和源文件。

步骤13 在工程文件中，caffe->Source->Layers右键->Add->Existing Item加入到工程，注册层信息；

图 5-167 NormalizeParameter 和 PermuteParameter 层的代码文件路径

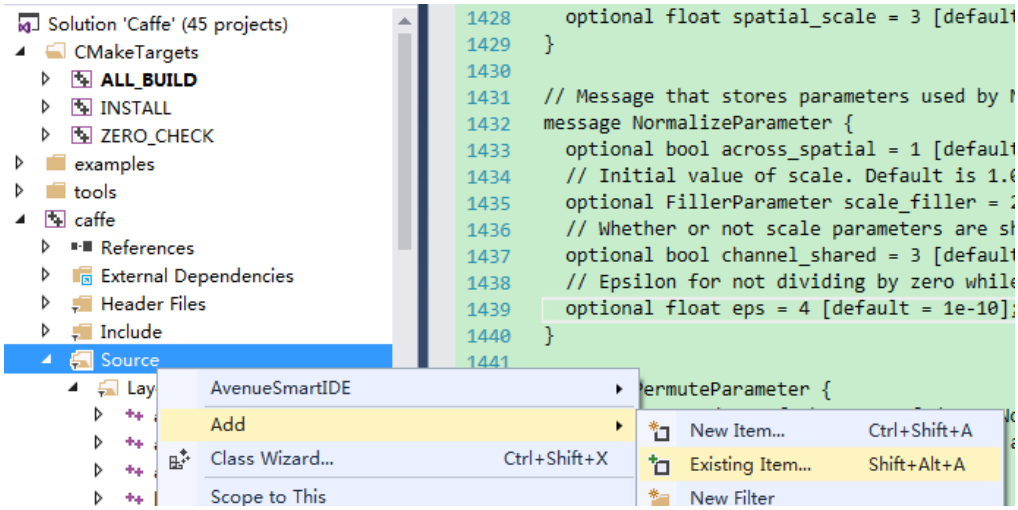
Local Disk (D:) ▸ test ▸ caffe-ssd ▸ src ▸ caffe ▸ layers

w folder

Name	Date modified	Type	Size
neuron_layer.cpp	2018/1/2 11:07	C++ Source file	1 KB
normalize_layer.cpp	2018/1/2 11:07	C++ Source file	10 KB
normalize_layer.cu	2018/1/2 11:07	CU File	8 KB
parameter_layer.cpp	2018/1/2 11:07	C++ Source file	1 KB
permute_layer.cpp	2018/1/2 11:07	C++ Source file	5 KB
permute_layer.cu	2018/1/2 11:07	CU File	3 KB

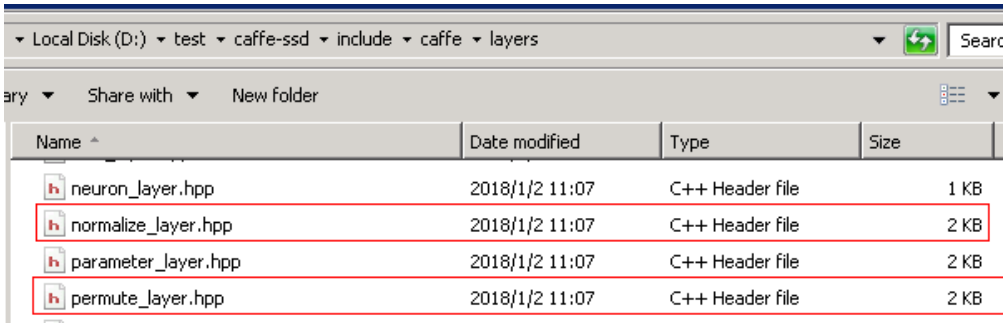
cu结尾的是cuda实现，如果要支持cuda实现，需要选择.cu后缀的文件。

图 5-168 添加代码 permute_layer.cpp 和 normalize_layer.cpp 到工程文件



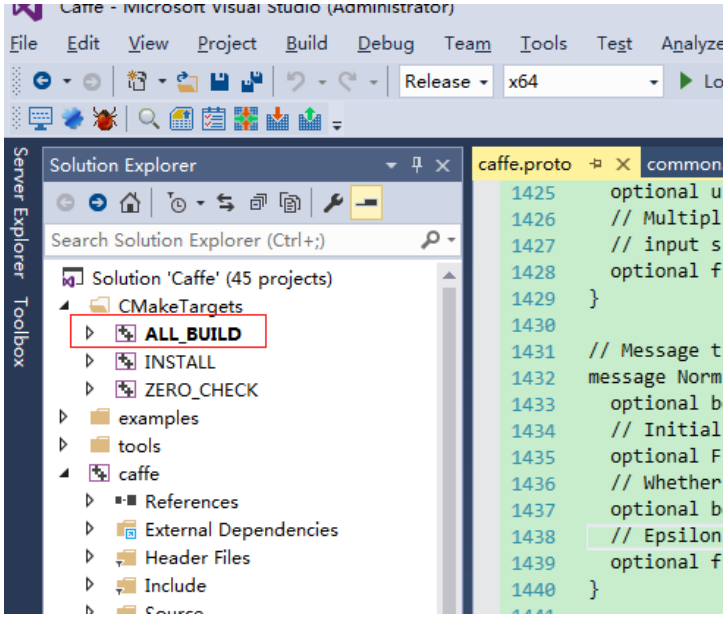
步骤14 在工程文件中，caffe->Header Files->Layers右键->Add->Existing Item加入到工程。

图 5-169 添加代码 permute_layer.hpp 和 normalize_layer.hpp 到工程文件



步骤15 重新编译工程；

图 5-170 Release 模式编译



步骤16 将caffe-windows\scripts\build\lib\Release下的_caffe.pyd替换到python35\Lib\site-packages\caffe\python\caffe目录下；

步骤17 将caffe_windows\caffe-windows\scripts\build\include\caffe\proto下的caffe_pb2.py替换到python35\Lib\site-packages\caffe\python\caffe\proto目录下；（这里拷贝的两个文件是必要文件，用户可依据情况，将最新编译出的相关文件选择性的拷贝替换进去）

步骤18 在RuyiStudio-xx.xx.xx\Resources\pythonScript文件夹下，扩展CNN_convert_to_Caffemodel.py和CNN_convert_bin_and_print_featuremap.py这两个脚本中的supported_layers，将新增的层增加到supported_layers的定义中，如图5-171所示：

图 5-171 在代码中增加扩展层

```
supported_layers=[
"Convolution", "Deconvolution", "Pooling", "InnerProduct", "LRN", "BatchNorm", "Scale", "Bias", "Eltwise",
"Parameter", "Reduction", "Proposal",  "Custom", "Input", "Dropout", "Normalize", "Permute"]
```

步骤19 参照“5.5.5 输出caffe的中间结果”章节，设置prototxt，caffemodel以及图片文件，点击ok键，成功得到caffe的中间输出。如果客户不依赖工具，有自己的一套python脚本，可直接参考上一步骤中提到的CNN_convert_bin_and_print_featuremap.py中的print_CNNfeaturemap函数，运用于自己的脚本中。

图 5-172 打印 caffe 中间结果

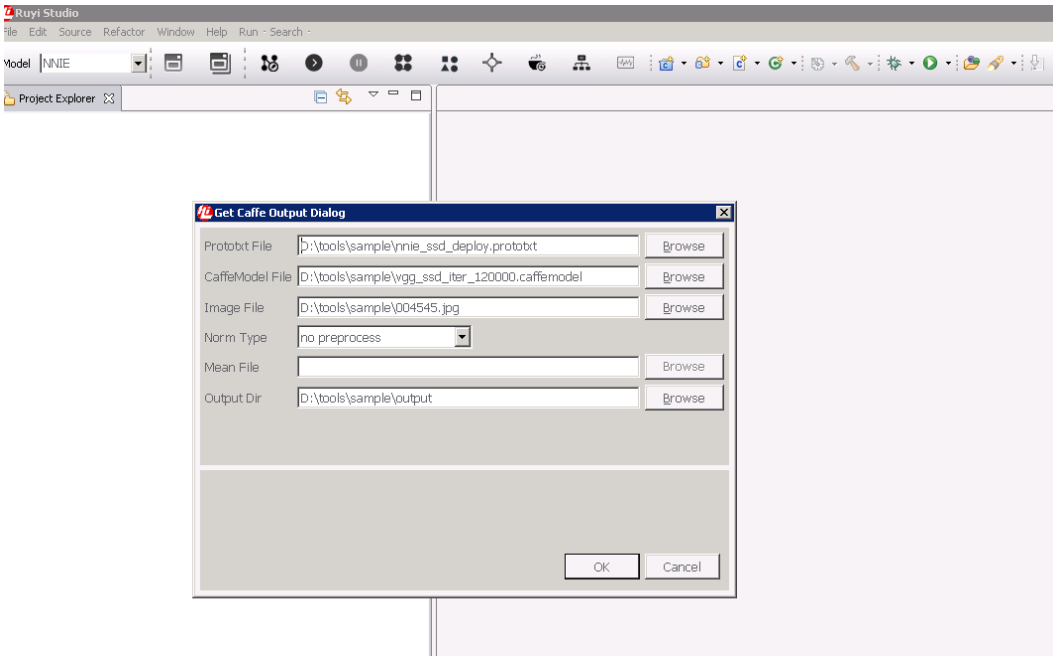
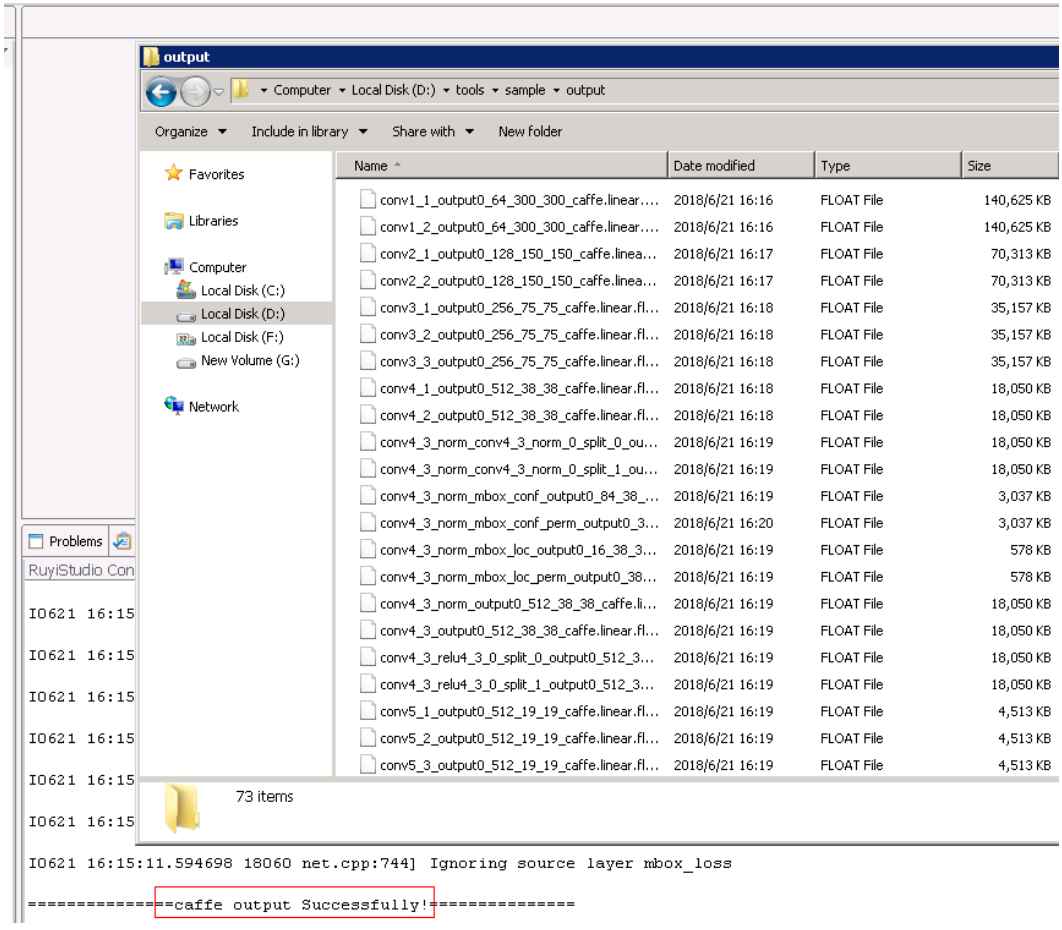


图 5-173 成功得到 caffe 的中间输出



----结束

5.10.3 工具在运行仿真库代码时，当仿真库对应的 exe 发生异常崩溃时，所有 printf 信息在控制台上打印不出来的解决办法

问题描述

工具在运行仿真库代码时，当仿真库对应的exe发生异常崩溃时，所有printf信息在控制台上打印不出来，我该怎么办?

问题原因

因printf方法对应的实现原理是将要打印的数据线刷新到缓存流中等待用户只需fflush方法才会刷新到控制台中显示，故当程序出现崩溃时，用户未添加fflush是无法正常打印之前需要打印的内容的。

解决办法

在代码的main方法中最前面添加如下两句setbuf(stderr, NULL); setbuf(stdout, NULL)，如图5-174，目的是设置输出缓存BUFF为NULL即可使得printf方法做到实时输出。

图 5-174 main 方法中设置缓存 BUFFF 为 NULL

```
main.cpp
#include <stdio.h>
#include "SvpSampleWk.h"

int main(int argc, char* argv[])
{
    /*set stderr &stdout buffer to NULL to flush print info immediately*/
    setbuf(stderr, NULL);
    setbuf(stdout, NULL);

    /*Classificacion*/
    SvpSampleCnnClfLenet();
    SvpSampleCnnClfAlexnet();
    SvpSampleCnnClfVgg16();
    SvpSampleCnnClfGooglenet();
    SvpSampleCnnClfResnet50();
    SvpSampleCnnClfSqueezenet();

    /*Detection*/
    SvpSampleRoiDetFasterRCNNAlexnet();
    SvpSampleRoiDetFasterRCNNVGG16();
    SvpSampleRoiDetFasterRCNNResnet18();
    SvpSampleRoiDetFasterRCNNResnet34();
    SvpSampleRoiDetFasterRCNNPvanet();
    SvpSampleRoiDetFasterRCNNDoubleRoi();
    SvpSampleRoiDetRFCNResnet50();
    SvpSampleCnnDetYoloV1();
    SvpSampleCnnDetYoloV2();
    SvpSampleCnnDetSSD();

    /*Segmentation*/
    SvpSampleCnnFcnSegnet();

    /*LSTM*/
    SvpSampleRecurrentLSTMFC();
    SvpSampleRecurrentLSTMRelu();

    //printf("press any key to exit ... \n");
    //getchar();

    return 0;
}
```

5.10.4 在创建项目时，出现中文时，导致项目修改文件失败，编译和运行失败的问题

问题描述

在创建项目的时候，出现中文时，导致项目修改文件失败，编译和运行失败我该怎么办?

问题原因

因为原生的CDT是不支持中文的，所以当出现中文时，就可能会出现上面描述的问题。

解决办法

为了保证创建的工程能够正确编译和运行，以及文件的修改，在创建工程的时候，不要出现中文，以及中文路径。

5.10.5 如何在 Ruyi 工具中修改 Runtime 预置的插件库名称

问题描述

修改插件库名称后，在Ruyi工具中加载新插件库失败。

问题原因

修改插件库名称后，未重启Ruyi工具，导致插件库未能重新加载。

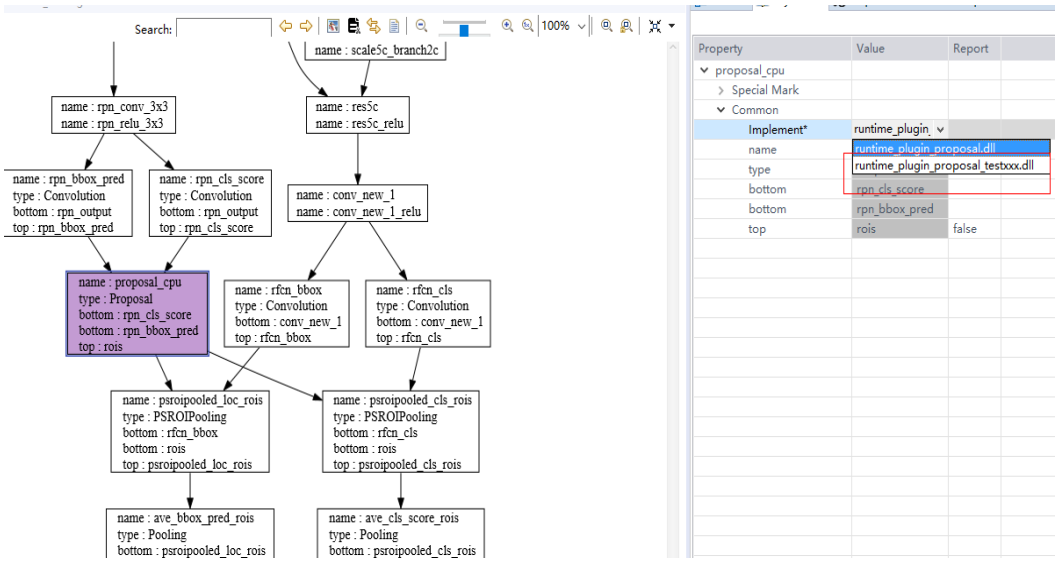
解决办法

步骤1 进入Ruyi工具插件库目录，在RuyiStudio-x.x.xx\Resources\runtime_plugin下修改插件库名称。

步骤2 修改完插件库名称后，重新启动Ruyi工具。

步骤3 进入Ruyi工具中，打开cfg文件，点击mark，选择对应插件库。

图 5-175 Ruyi 工具选择插件库



----结束

5.10.6 创建 C/C++工程编译后运行失败，工具提示需要在 sim_out 路径下才能运行

问题描述

Ruyi工具创建C/C++工程编译后运行失败，工具提示需要在sim_out 路径下才能运行，但默认没有sim_out这个目录，我该怎么办?

问题原因

工具跑c/c++的功能仿和指令仿时，为了与mapper区分执行路径，在开源软件中修改了代码，默认在sim_out目录下运行；因此，当创建了插件库或者任意c/c++工程时，都会先去workspace下找sim_out文件夹，需要按照Run->Run Configuration->Aguments->删除/sim_out的操作执行，详细参考解决办法描述。

解决办法

步骤1 新创建一个C/C++工程，如图5-176；在此工程下添加一个demo.cpp文件，如图5-177、如图5-178。

图 5-176 新创建一个 C/C++工程

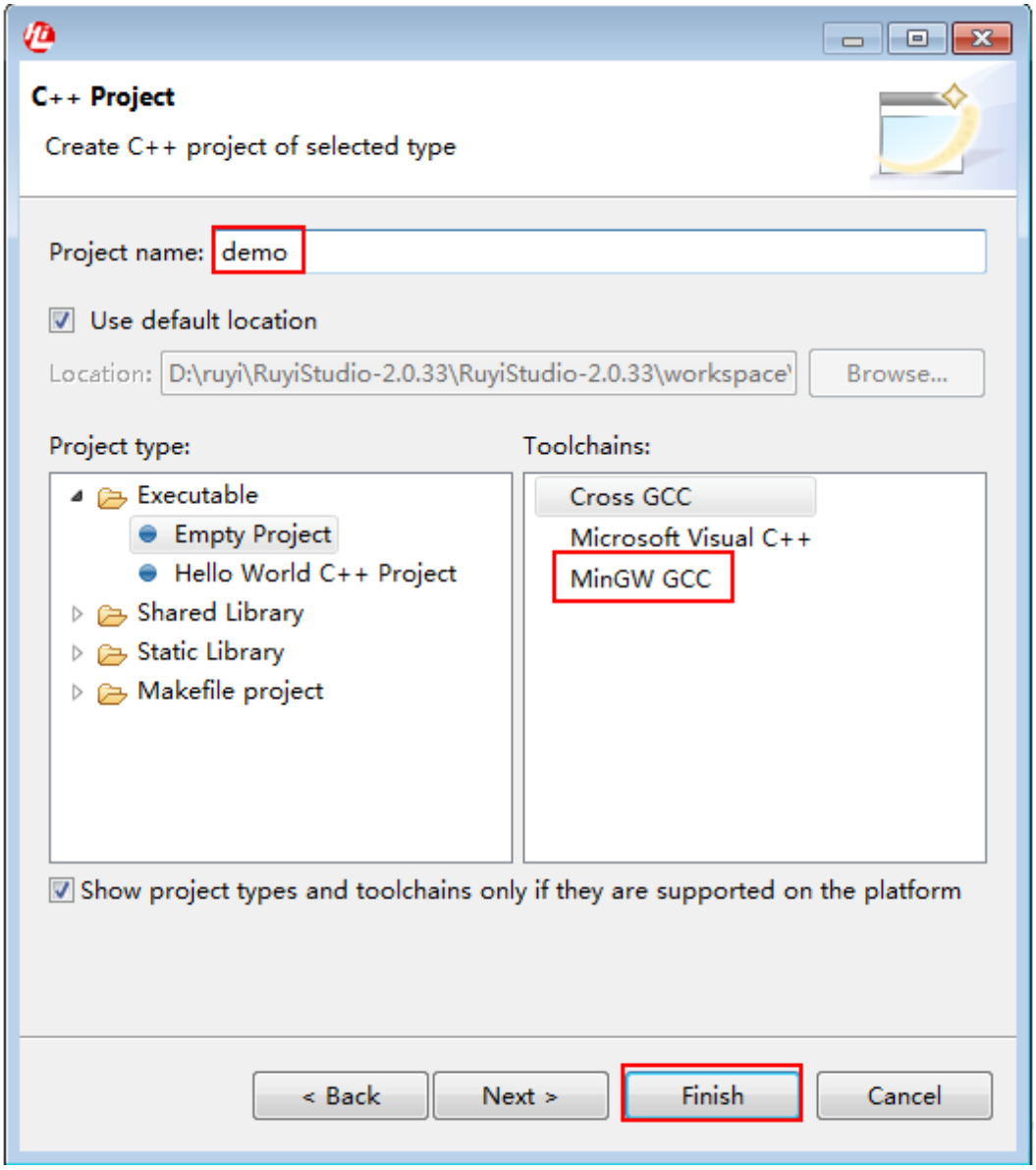


图 5-177 添加 demo.cpp 文件

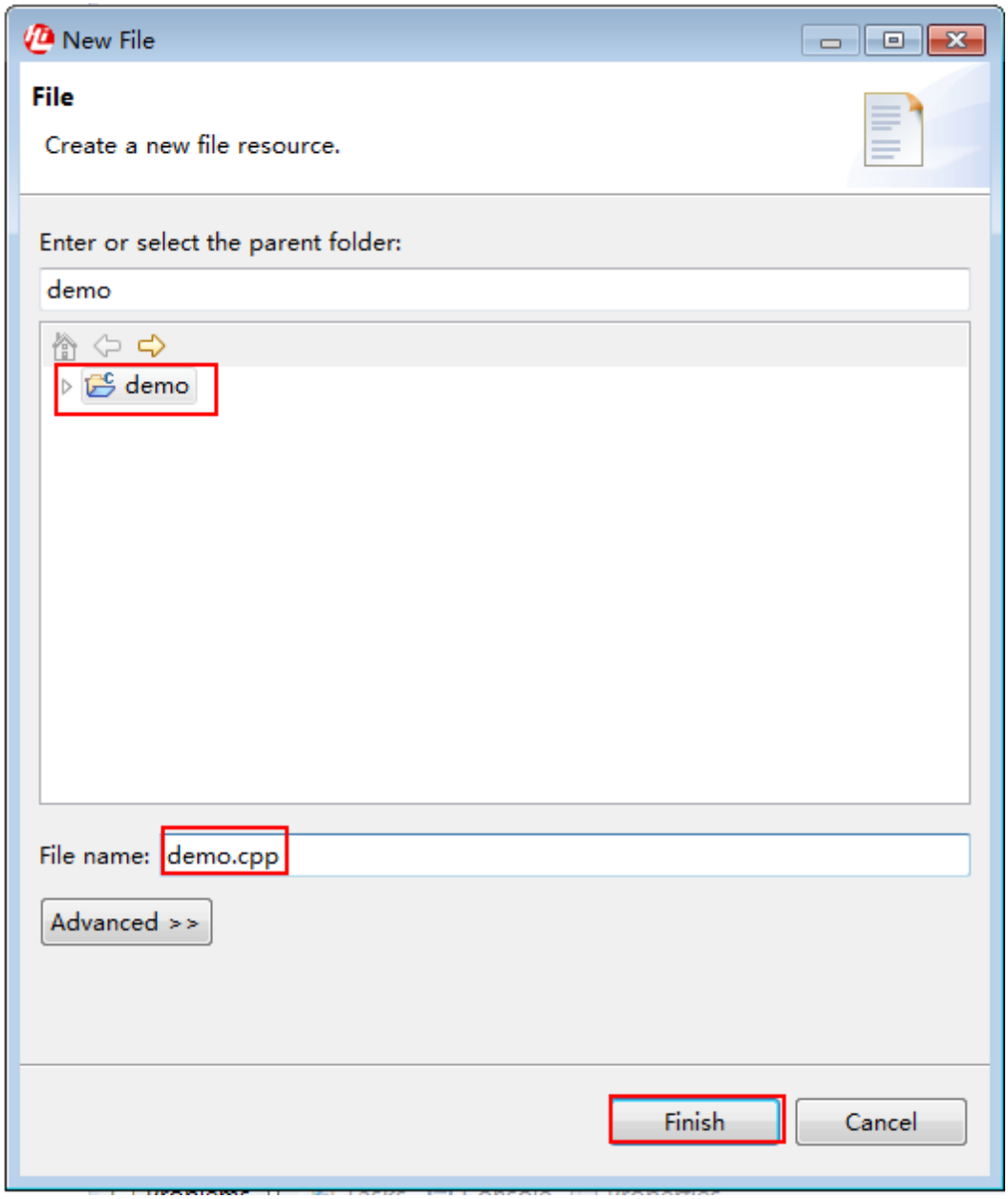
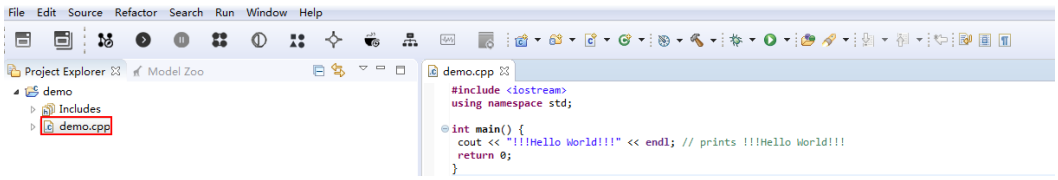


图 5-178 Demo.cpp 文件



步骤2 编译demo.cpp文件，如图5-179；点击Run运行该文件，Ruyi工具提示参数引用路径不正确，如图5-180。

图 5-179 编译 demo.cpp 文件

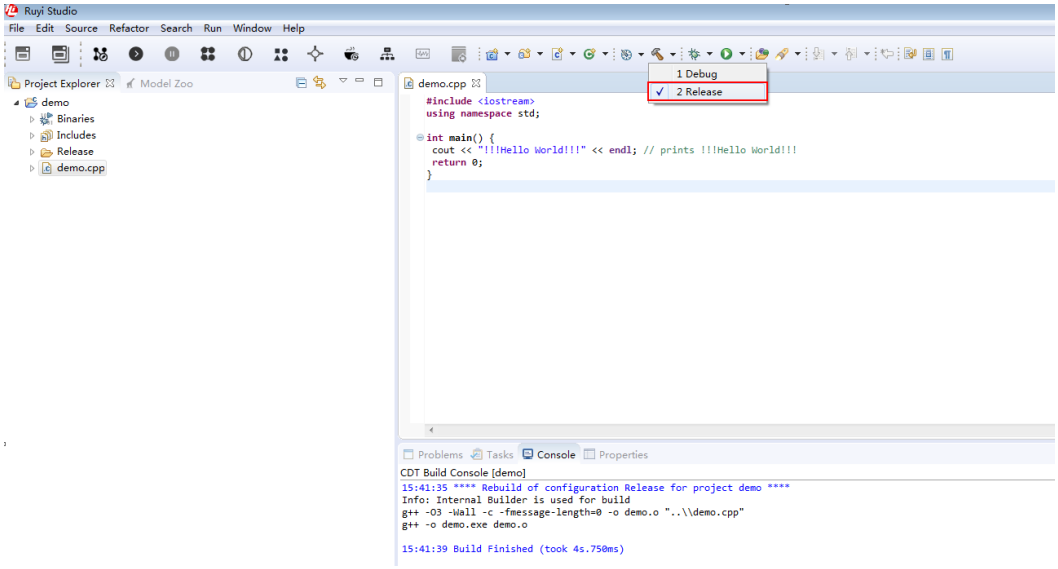
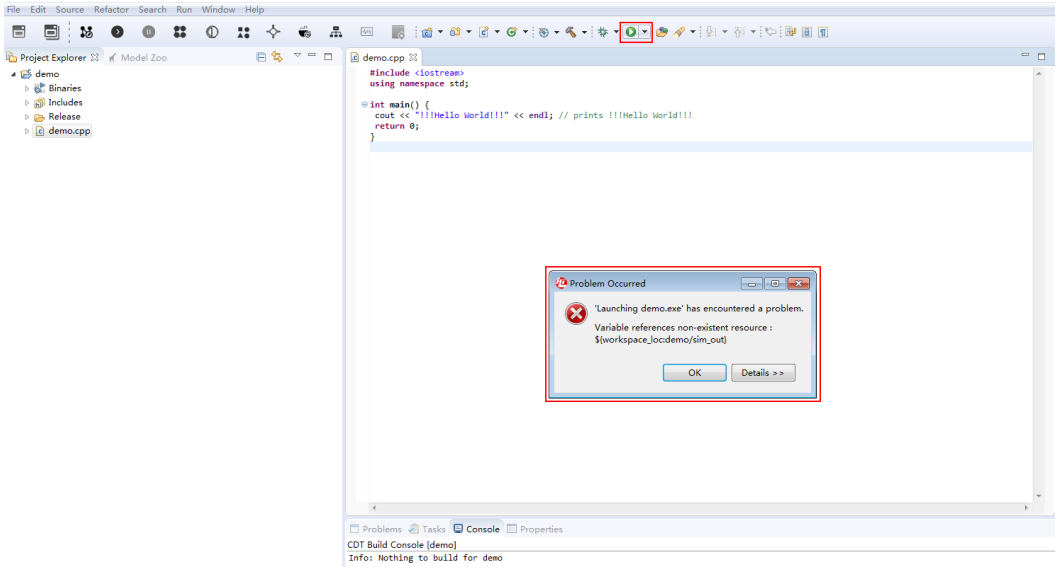
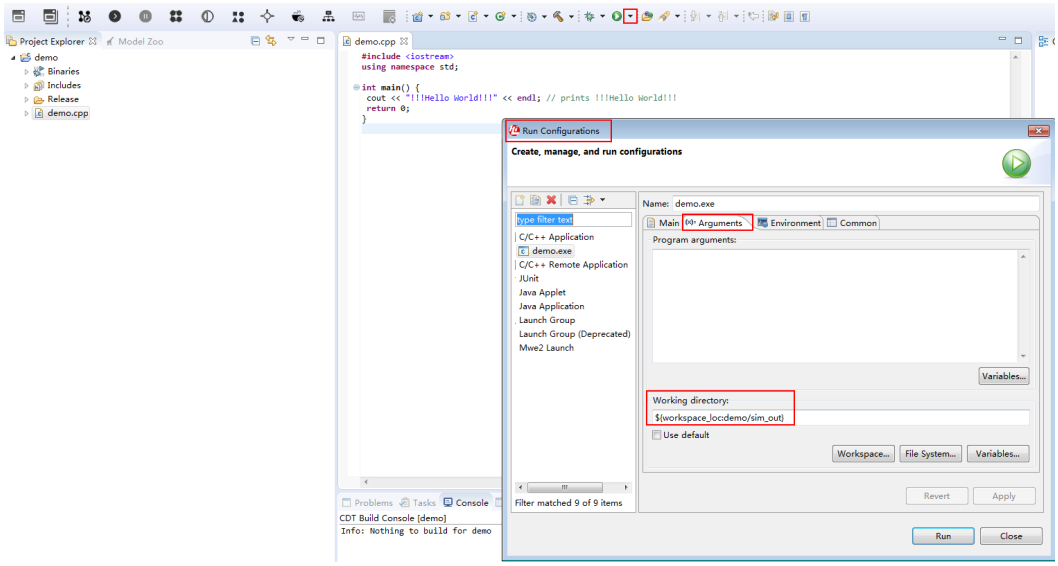


图 5-180 Ruyi 工具提示参数引用路径不正确



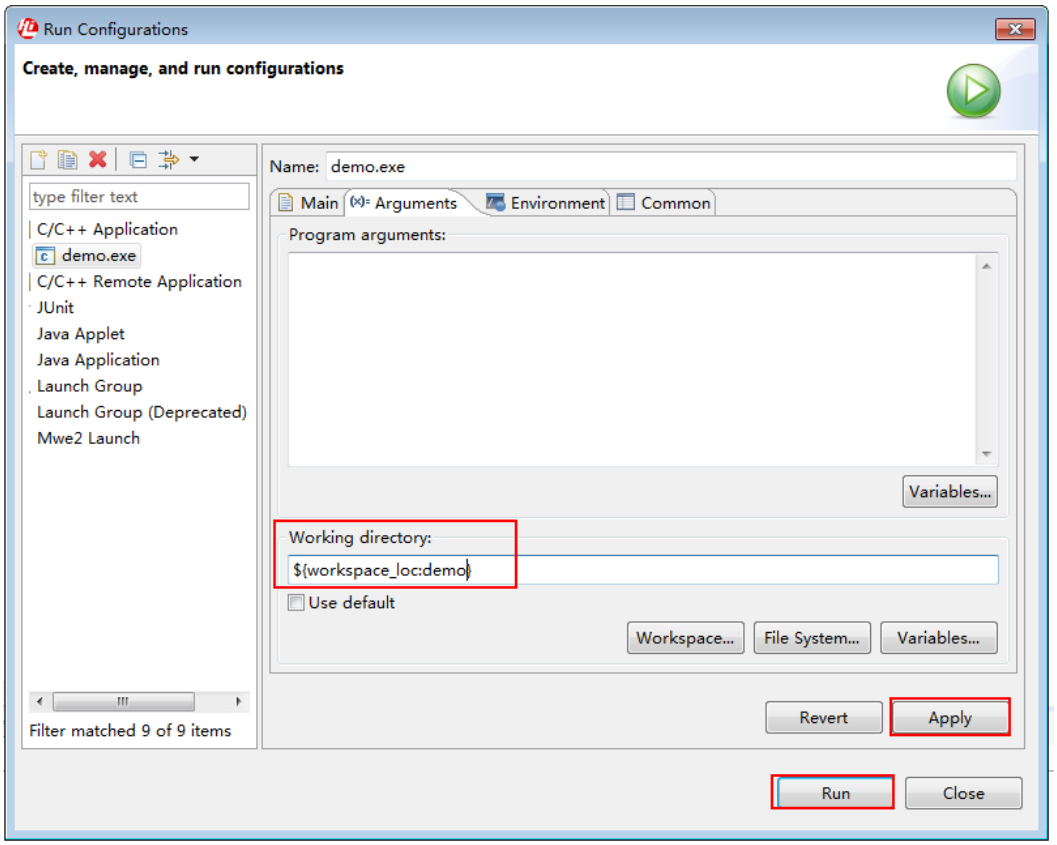
步骤3 打开Run配置对话框，Run-> Run Configurations，选择Aguments，如图5-181。

图 5-181 Run Configurations 下选择 Aguments



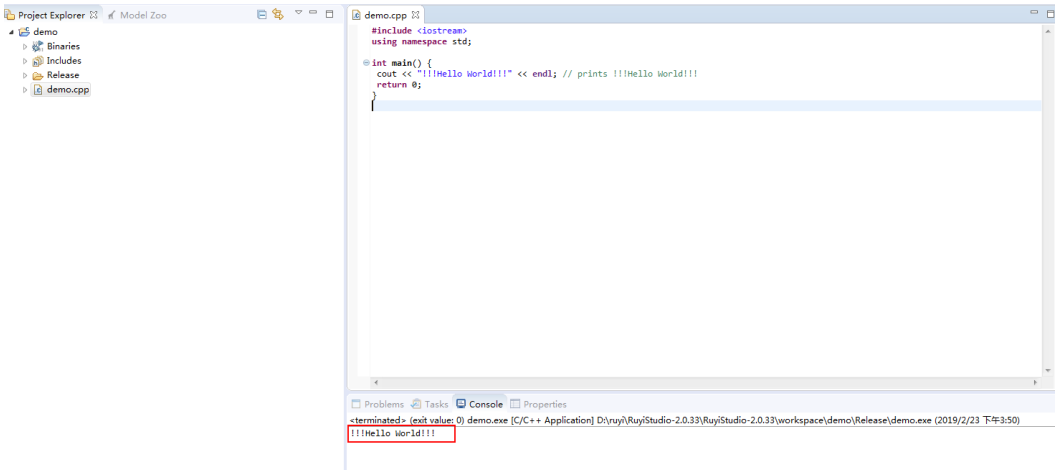
步骤4 删除Working directory中${workspace_loc:temp3/sim_out}的/sim_out，如图5-182，点击apply->run。

图 5-182 删除/sim_out



步骤5 Demo运行成功，控制台正常输出打印信息，如图5-183。

图 5-183 控制台输出结果



----结束

6 Ruyicmd 工具使用指南

Ruyicmd工具为linux版本，命名为ruyicmd-vx.x.x.tar.gz。

Ruyicmd工具功能包括，调用nnie_mapper或runtime_mapper编译NNIE模型文件，向量相似度对比，仿真性能数据查看，Prototxt预处理，获取Caffe中间层数据，对prototxt添加量化层功能。

6.1 Ruyicmd 安装

6.1.1 安装 mapper 运行依赖的库文件

请参考**3.4 Linux版NNIE mapper安装**章节。

6.1.2 安装 caffe 环境（需要使用获取 caffe 中间层结果功能时才需要提前安装）

在Ruyicmd工具同目录下会有一个ruyi_env_setup_linux-vx.x.x的文件夹，可以进入该文件夹参考readme执行脚本一键安装。

6.1.3 Ruyicmd 工具依赖的 JRE

工具已集成了JRE8u211版本的JRE，故无需安装可直接使用。

6.2 Ruyicmd 功能介绍

6.2.1 Ruyicmd 编译 NNIE 模型文件功能

6.2.1.1 Nnie_mapper 场景说明

- 支持指定nnie_mapper生成NNIE模型文件；
- 支持对NNIE不识别的Custom层添加shape信息；
- 支持对需要上报的中间层对应的top处添加_report标识指定上报；

6.2.1.2 Runtime_mapper 场景说明

- 支持指定runtime_mapper生成NNIE模型文件；
- 支持对NNIE不识别的Custom层添加shape信息和指定用户自定义插件库；
- 支持对NNIE不识别的Proposal层指定用户自定义插件库；
- 支持对NNIE支持的层指定用户自定义插件库，并将其层名添加_cpu标记；
- 支持对需要上报的中间层对应的top处添加_report标识指定上报；
- 支持单输入单输出Custom层；
- 支持多输入多输出Custom层（单输入多输出 + 多输入单输出）；
- 支持多个Custom层（分为多个使用一个自定义插件库，多个Custom层使用不同自定义插件库）。

6.2.1.3 命令说明

编译 NNIE 模型文件命令说明

功能说明：支持使用nnie_mapper或runtime_mapper来制作NNIE模型文件。

命令示例如下：

Usage:

```
./ruyicmd.sh <-mw | --makewk> <cfg> [mapper_type] [chip] [gpu] [execute_path]
[custom_info] [output_layers]
```

Example:

```
./ruyicmd.sh --makewk cfg=./../data/classification/alexnet/runtime_alexnet_no_group_func.cfg
mapper_type=1 chip=0 gpu=0 execute_path=./sample_runtime/
custom_info="layername1:libname1:[n,c,h,w];layername2:libname2:[n,c,h,w]
[n,c,h,w];layname3:libname3"
output_layers="layername1:topname1,topname2;layername2:topname1;layername3:topname1"
```

参数说明：cfg: cfg配置文件的路径，该参数为并且必须为第一个参数。

mapper_type: 选择编译wk文件使用的mapper类型

0 nnie_mapper

1 runtime_mapper (默认)

chip: 选择编译的wk文件需要使用的芯片平台

0 hi3559av100 (默认)

1 hi3559cv100

2 hi3519av100

3 hi3516dv300

4 hi3516cv500

5 hi3559v200

6 hi3516av300

7 hi3531dv200

8 hi3562v100

9 hi3566v100

10 hi3535av100

11 hi3521dv200

12 hi3520dv500

13 hi3569v100

14 hi3568v100

gpu: 当运行环境安装好cuda环境时可以选择使用GPU版本的mapper

0 unuse gpu (默认)

1 use gpu

execute_path: 设置mapper运行路径，注意此路径就是cfg中的相对路径的当前路径。

默认路径为：./../mapper_output

custom_info: 可选参数，设置Custom层或Proposal层对应层的自定义算子及设置Custom层对应的shape信息。

注意在此处输入的shape信息个数需要与当前层top个数对应，shape信息个数与top个数保持一致，若prototxt中已有shape信息但此处也输入了则会用当前输入更新shape。

例子1：设置runtime_mapper需要的custom层对应的自定义算子及shape信息

```
custom_info="custom1:plugin_test2:[1,3,227,227];custom2:plugin_test1:[1,3,227,227]
[1,3,227,227]"
```

例子2：设置runtime_mapper需要的Proposal层对应的自定义算子

```
custom_info="proposal:runtime_plugin_proposal"
```

例子3：设置nnie_mapper需要的Custom层对应的shape信息

```
custom_info="layername1:[n,c,h,w];layername2:[n,c,h,w][n,c,h,w];layername3:[n,c,h,w]"
```

output_layers: 可选参数，设置某些层为中间上报层，工具会自动在这些层的对应top上添加_report标记.

```
output_layers="layername1:topname1,topname2;layername2:topname1;layername3:topname1"
```

须知

1. 运行makewk命令时，会自动将当前cfg中的prototxt进行如下检查，如需修改则会创建一个marked后的prototxt并替换cfg中对应prototxt值。

修改如下：

首先，将NNIE或Runtime不支持的层标记为custom或proposal；

然后，删除custom中的其他参数；

2. 如果custom_info和output_layers的值中如果包含"；"或空格则需要用""将等号右边的值包含。

6.2.1.4 Ruyicmd 中 runtime_mapper 使用的自定义插件库说明：

在makewk命令中的custom_info参数需要指定自定义插件库名称，而自定义插件库可以参考**4.3.5 客户自定义插件**来生成。

6.2.2 Ruyicmd 向量比较文件夹功能

6.2.2.1 功能说明

支持不同功能配置下每一层的向量统计量文件夹中，文件名匹配的文件进行比较。

6.2.2.2 命令说明

命令示例如下：

Usage:

```
./ruyicmd.sh <-cd | --comparedir>  <left dir>    <right dir>   <parseDotFile>
```

Example:

```
./ruyicmd.sh <-cd | --comparedir>  ../func_layer_output_linear   ../func_layer_output_linear    ../
cnn_net_tree.dot
```

参数说明：

left dir：左侧的比较文件夹

right dir：右侧的比较文件夹

parseDotFile: 输入dot文件用于解析层名

6.2.3 Ruyicmd 向量比较单个文件功能

6.2.3.1 功能说明

支持不同功能配置下单个层向量的统计量文件两两比较，输出详细比较信息。

6.2.3.2 命令说明

命令示例如下：

Usage:

```
./ruyicmd.sh <-cf | --comparefile> <left file>    <right file>   <toFloat>
```

Example:

```
./ruyicmd.sh <-cf | -- comparefile> ../caffe_output/conv1_output0_96_55_55_caffe.linear.float  ./
func_layer_output_linear/seg0_layer0_output0_func.linear.hex   true   .
```

参数说明：

left file：左侧的比较文件

right file：右侧的比较文件

toFloat：是否将比较内容转换为float类型

true：转换

false：不转换

6.2.4 Ruyicmd showPerf 功能

6.2.4.1 功能说明

支持将仿真性能文件根据dot文件解析层名后打印性能指标支持的cycle数、带宽查看。

6.2.4.2 命令说明

命令示例如下：

Usage:

```
./ruyicmd.sh <-sp | --showperf>  <parseDotFile>   <performance data dir>
```

Example:

```
./ruyicmd.sh <-sp | --showperf>  ../cnn_net_tree.dot   ../cycs_result_dir
```

参数说明：

parseDotFile：输入dot文件用于解析层名

performance data dir：性能数据所在文件夹

6.2.5 Ruyicmd Preprocess 预处理功能

6.2.5.1 功能说明：

功能说明：支持对prototxt源文件进行预处理检查，修改当前NNIE不支持的语法，增强易用性，有如下几点修改：

- 将prototxt中NNIE不支持的层改为custom和proposal
- 将改为custom或proposal的层清空其他参数
- 将prototxt中满足NNIE支持的uninplace层改为inplace
- 将prototxt中inplace层中有custom或proposal的层提示报错
- 检查并修改inputshape信息
- 将layers修改为layer
- 将type信息从整型表达修改为字符串表达

6.2.5.2 命令说明

命令示例如下：

Usage:

```
./ruyicmd.sh <-p | --preprocess>   <net type>     <src prototxt file>  <caffemodel file>
<dest prototxt file >          [isInplace]
```

Example:


```
./ruyicmd.sh <-p | --preprocess>   0              bvlc_alexnet_deploy.prototxt
bvlc_alexnet.caffemodel     bvlc_alexnet_deploy_output.prototxt     true
```

net type：网络类型

0 CNN

1 ROI/PSROI

2 Recurrent

isInplace：设置是否需要将prototxt转换为inplace，为提升后续mapper执行效率，默认为true

true

false

6.2.6 Ruyicmd GetCaffeOutput 获取 caffe 中间层功能

6.2.6.1 功能说明

支持获取caffe中间层数据。

6.2.6.2 命令说明

命令示例如下：支持获取caffe输出的统计量。

```
./ruyicmd.sh <-gc | --getCaffeoutput>  <iniFile>  <prototxtFile>    <caffemodelFile>
<destFileDir>
```

Example:

```
./ruyicmd.sh <-gc | --getCaffeoutput>  alexnet.ini bvlc_alexnet_deploy.prototxt
bvlc_alexnet.caffemodel   ../caffe_output
```

alexnet.ini参考例子：

[data_scale] 0.0078125

[RGB_order] BGR

[image_file] /home/images/1.jpg

[mean_file] /home/model/channel_mean.txt

[norm_type] 5

其中norm_type可选的范围为:

0 no preprocess

1 mean file

2 channel mean_value

3 data_scale

4 mean file with data_scale

5 channel mean_value with data_scale

6.2.7 Ruyicmd AddQuant 添加量化层功能

6.2.7.1 功能说明

支持对prototxt添加量化层，用于客户在caffe环境下进行重训练。

6.2.7.2 命令说明

命令示例如下：

Usage:

```
./ruyicmd.sh <-aq | --addquant> <srcPrototxtFile>  <destPrototxt File>
```

Example:

./ruyicmd.sh <-aq | --addquant> bvlc_alexnet_deploy.prototxt bvlc_alexnet_deploy_output.prototxt

6.3 Ruyicmd 配置文件说明

在工具目录下的config文件中有config.ini，说明如下：

配置config文件夹上的config.ini(配置文件中#表示该行注释)，配置项有：

[SOC Version Setting] 工具支持的芯片名称

Supported = Hi3559AV100,Hi3559CV100,Hi3519AV100,Hi3516CV500,Hi3516DV300

[NNIE Type Setting] 芯片对应的NNIE版本

NNIE1.1 = Hi3559AV100,Hi3559CV100

NNIE1.2 = Hi3519AV100,Hi3516CV500,Hi3516DV300

[Vector Comparison Setting] 向量比较算法配置

Supported Algorithms 1.support 0.not support

CosineSimilarity=1

MaxAbsoluteError=1

RelativeEuclideanDistance=1

StandardDeviation=1

6.4 Ruyicmd 目录文件说明

图 6-1 Ruyicmd 目录结构

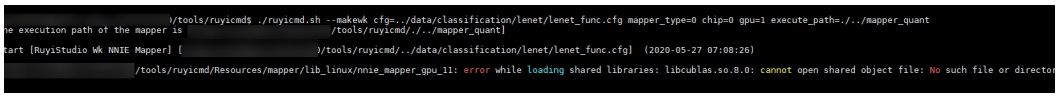
6.5 FAQ

6.5.1 CUDA 10.1 环境下编译知识库错误问题

问题描述

CUDA 10.1环境下编译知识库错误问题，提示libcublas.so.8.0: cannot open shared object file: No such file ordirectory. 如图6-2所示。

图 6-2 提示 libcublas.so.8.0 找不到的现象



解决办法

步骤1 cd /root/ruyicmd/Resources/mapper/lib_linux，/root/ruyicmd为ruyicmd解压路径；

步骤2 备份需要替换的mapper，如nnie_mapper_gpu_11;

步骤3 cd ./ ubuntu18.04_cuda10.1，再将需要替换的mapper拷贝到lib_linux目录下；

步骤4 重新make wk。

----结束

7 网络安全注意事项

7.1 NNIE 模型文件

使用Ruyicmd、RuyiStudio或直接使用nnie_mapper编译模型生成的NNIE模型文件(*.wk)，用于板端运行或PC端仿真时，需要用户确保完整性，防止文件被恶意篡改导致板端运行或PC端仿真异常。

7.2 仿真库

仿真库(功能仿真和指令仿真)提供的接口参数（如：内存指针等）、内存大小和内容需要由调用者确保正确，防止非法参数导致执行异常。

仿真库的hi_mpi_svp_nnie_forward接口可以通过用户输入的NNIE模型文件实现特征信息提取等功能，会涉及到敏感数据的处理，用户在使用该接口时需要保证输入和输出数据的安全，遵循当地法律法规。

