# 使用TensorRT加速SSD网络

这里使用的是TensorRT-5.1.5.0版本，虽然在官方给的sample里有sampleSSD示例程序，但要跑通，还是要一番摸索，下面的代码也是在sampleSSD修改而来，记录跑通的过程，以备以后参考。

## 模型准备

下载模型文件，地址：

<https://drive.google.com/open?id=0BzKzrI_SkD1_WVVTSmQxU0dVRzA>

百度网盘地址：

链接: https://pan.baidu.com/s/1gu1TaSyl3RXRatw3W7sAzA 提取码: eq5r

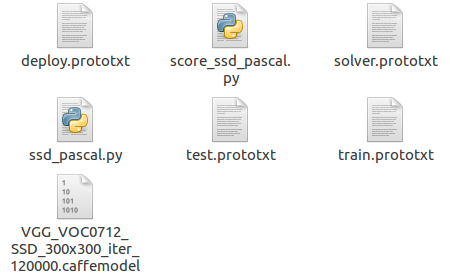


Figure 1 模型文件

我们需要使用这里的deploy.prototxt和caffemodel文件。这里需要对deploy.prototxt做一些修改：

1. 将所有的Flatten层修改成Reshape层，并设置reshape\_param参数，一共有13处，注意不要少了括号，例如：

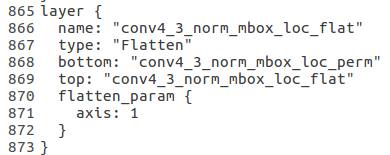


Figure 2 修改前

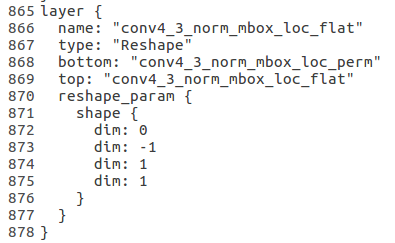
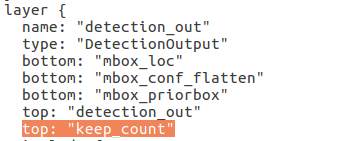


Figure 3 修改后

1. 在最后的detection\_out层增加一个top:”keep\_count”：



## 代码

这里程序流程与其它TensorRT程序流程类似，但在数据预处理时要注意，作者并没有把图像数据归一化到[-1,1]区间，而是直接将图像数据与均值相减。另外由于OpenCV的图像格式是[BGRBGR...]这样逐一排列的，而网络的数据是按[BBBBBB...][GGGGGG...][RRRRR...]这样按通道排列的，因此需要做一下转换：

|  |
| --- |
| for(int r = 0; r < kINPUT\_H;r++)  {  for(int c = 0; c < kINPUT\_W;c++)  {  for(int t = 0;t < kINPUT\_C;t++)  {  data[t \* kINPUT\_H \* kINPUT\_W + r \* kINPUT\_W + c] = image\_f.ptr<Vec3f>(r)[c][0] - mean[t];  }  }  } |

### 2.1 common.h

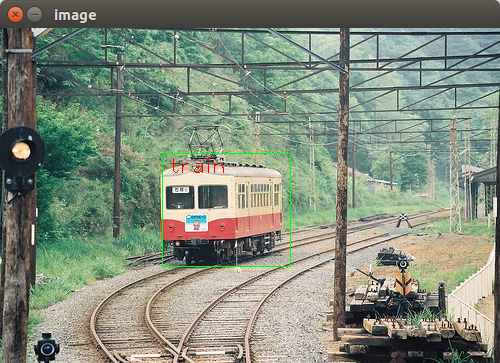
|  |
| --- |
| #ifndef FBC\_TENSORRT\_TEST\_COMMON\_HPP\_  #define FBC\_TENSORRT\_TEST\_COMMON\_HPP\_  #include <cuda\_runtime.h>  #include <device\_launch\_parameters.h>  #include <NvInfer.h>  #include <iostream>  #define CHECK(status) \  do \  { \  auto ret = (status); \  if (ret != 0) \  { \  std::cout << "Cuda failure: " << ret << std::endl; \  abort(); \  } \  } while (0)  class Logger : public nvinfer1::ILogger  {      void log(Severity severity, const char\* msg) override      {          if (severity != Severity::kINFO)              std::cout << msg << std::endl;      }  };  #endif |

### 2.2 ssd\_test.cpp

|  |
| --- |
| #include <cassert>  #include <cmath>  #include <cstring>  #include <cuda\_runtime\_api.h>  #include <unordered\_map>  #include "NvCaffeParser.h"  #include "NvInferPlugin.h"  #include "common.h"  #include <vector>  #include <opencv2/opencv.hpp>  using namespace nvinfer1;  using namespace nvcaffeparser1;  using namespace plugin;  using namespace std;  using namespace cv;  // Network details  const char\* gNetworkName = "ssd"; // Network name  static const int kINPUT\_C = 3; // Input image channels  static const int kINPUT\_H = 300; // Input image height  static const int kINPUT\_W = 300; // Input image width  static const int kOUTPUT\_CLS\_SIZE = 21; // Number of classes  static const int kKEEP\_TOPK = 200; // Number of total bboxes to be kept per image after NMS step. It is same as detection\_output\_param.keep\_top\_k in prototxt file  const std::string gCLASSES[kOUTPUT\_CLS\_SIZE] = {"background", "aeroplane", "bicycle", "bird", "boat",  "bottle", "bus", "car", "cat", "chair",  "cow", "diningtable", "dog", "horse", "motorbike",  "person", "pottedplant", "sheep", "sofa", "train",  "tvmonitor"}; // List of class labels  static const char\* kINPUT\_BLOB\_NAME = "data"; // Input blob name  static const char\* kOUTPUT\_BLOB\_NAME0 = "detection\_out"; // Output blob name  static const char\* kOUTPUT\_BLOB\_NAME1 = "keep\_count"; // Output blob name  Logger logger;  // Visualization  const float kVISUAL\_THRESHOLD = 0.6f;  void caffeToTRTModel(const std::string& deployFile, // Name for caffe prototxt  const std::string& modelFile, // Name for model  const std::vector<std::string>& outputs, // Network outputs  unsigned int maxBatchSize, // Batch size - NB must be at least as large as the batch we want to run with)  IHostMemory\*\* trtModelStream) // Output stream for the TensorRT model  {  // Create the builder  IBuilder\* builder = createInferBuilder(logger);  assert(builder != nullptr);  // Parse the caffe model to populate the network, then set the outputs  INetworkDefinition\* network = builder->createNetwork();  ICaffeParser\* parser = createCaffeParser();  const IBlobNameToTensor\* blobNameToTensor = parser->parse(deployFile.c\_str(),  modelFile.c\_str(),  \*network,  nvinfer1::DataType::kHALF);  // Specify which tensors are outputs  for (auto& s : outputs)  network->markOutput(\*blobNameToTensor->find(s.c\_str()));  // Build the engine  builder->setMaxBatchSize(maxBatchSize);  builder->setMaxWorkspaceSize(36 << 20);  builder->setFp16Mode(true);  builder->setInt8Mode(false);  builder->allowGPUFallback(true);  ICudaEngine\* engine;  engine = builder->buildCudaEngine(\*network);  assert(engine);  network->destroy();  parser->destroy();  (\*trtModelStream) = engine->serialize();  engine->destroy();  builder->destroy();  }  void doInference(IExecutionContext& context, float\* inputData, float\* detectionOut, int\* keepCount, int batchSize)  {  const ICudaEngine& engine = context.getEngine();  // input and output buffer pointers that we pass to the engine - the engine requires exactly IEngine::getNbBindings(),  // of these, but in this case we know that there is exactly 1 input and 2 output.  assert(engine.getNbBindings() == 3);  void\* buffers[3];  // In order to bind the buffers, we need to know the names of the input and output tensors.  // note that indices are guaranteed to be less than IEngine::getNbBindings()  int inputIndex = engine.getBindingIndex(kINPUT\_BLOB\_NAME),  outputIndex0 = engine.getBindingIndex(kOUTPUT\_BLOB\_NAME0),  outputIndex1 = engine.getBindingIndex(kOUTPUT\_BLOB\_NAME1);  // Create GPU buffers and a stream  CHECK(cudaMalloc(&buffers[inputIndex], batchSize \* kINPUT\_C \* kINPUT\_H \* kINPUT\_W \* sizeof(float))); // Data  CHECK(cudaMalloc(&buffers[outputIndex0], batchSize \* kKEEP\_TOPK \* 7 \* sizeof(float))); // Detection\_out  CHECK(cudaMalloc(&buffers[outputIndex1], batchSize \* sizeof(int))); // KeepCount (BBoxs left for each batch)  cudaStream\_t stream;  CHECK(cudaStreamCreate(&stream));  // DMA the input to the GPU, execute the batch asynchronously, and DMA it back:  CHECK(cudaMemcpyAsync(buffers[inputIndex], inputData, batchSize \* kINPUT\_C \* kINPUT\_H \* kINPUT\_W \* sizeof(float), cudaMemcpyHostToDevice, stream));  context.enqueue(batchSize, buffers, stream, nullptr);  CHECK(cudaMemcpyAsync(detectionOut, buffers[outputIndex0], batchSize \* kKEEP\_TOPK \* 7 \* sizeof(float), cudaMemcpyDeviceToHost, stream));  CHECK(cudaMemcpyAsync(keepCount, buffers[outputIndex1], batchSize \* sizeof(int), cudaMemcpyDeviceToHost, stream));  cudaStreamSynchronize(stream);  // Release the stream and the buffers  cudaStreamDestroy(stream);  CHECK(cudaFree(buffers[inputIndex]));  CHECK(cudaFree(buffers[outputIndex0]));  CHECK(cudaFree(buffers[outputIndex1]));  }  int main(int argc, char\*\* argv)  {  initLibNvInferPlugins(&logger, "");  IHostMemory\* trtModelStream{nullptr};  // Create a TensorRT model from the caffe model and serialize it to a stream  const int N = 1; // Batch size  caffeToTRTModel("../model/ssd.prototxt",  "../model/VGG\_VOC0712\_SSD\_300x300\_iter\_120000.caffemodel",  std::vector<std::string>{kOUTPUT\_BLOB\_NAME0, kOUTPUT\_BLOB\_NAME1},  N, &trtModelStream);  std::vector<cv::String> imageList;  cv::glob("../images/\*.jpg", imageList);  printf("get %d images\n", imageList.size());  IRuntime\* runtime = createInferRuntime(logger);  ICudaEngine\* engine = runtime->deserializeCudaEngine(trtModelStream->data(), trtModelStream->size(), nullptr);  assert(engine != nullptr);  IExecutionContext\* context = engine->createExecutionContext();  assert(context != nullptr);  trtModelStream->destroy();  float\* detectionOut = new float[N \* kKEEP\_TOPK \* 7];  int\* keepCount = new int[N];  float mean[3] = {104, 117, 123};  for(int idx = 0;idx<imageList.size();idx++)  {  cv::Mat image = cv::imread(imageList[idx]);  cv::Mat image\_f;  image.convertTo(image\_f, CV\_32FC3);  cv::resize(image\_f, image\_f, cv::Size(kINPUT\_W, kINPUT\_H));  float \*data = new float[kINPUT\_C \* kINPUT\_H \* kINPUT\_W];  for(int r = 0; r < kINPUT\_H;r++)  {  for(int c = 0; c < kINPUT\_W;c++)  {  for(int t = 0;t < kINPUT\_C;t++)  {  data[t \* kINPUT\_H \* kINPUT\_W + r \* kINPUT\_W + c] = image\_f.ptr<Vec3f>(r)[c][0] - mean[t];  }  }  }  doInference(\*context, data, detectionOut, keepCount, N);  for (int p = 0; p < N; ++p)  {  for (int i = 0; i < keepCount[p]; ++i)  {  float\* det = detectionOut + (p \* kKEEP\_TOPK + i) \* 7;  if (det[2] < kVISUAL\_THRESHOLD)  continue;  float xmin = det[3] \* image.cols;  float ymin = det[4] \* image.rows;  float xmax = det[5] \* image.cols;  float ymax = det[6] \* image.rows;  std::string className = gCLASSES[(int) det[1]];  cv::rectangle(image, cv::Rect(cv::Point(xmin, ymin), cv::Point(xmax, ymax)), cv::Scalar(0, 255, 0), 1);  cv::putText(image, className, cv::Point(xmin + 5, ymin + 20), 1, 1.5, cv::Scalar(0, 0, 255));  }  }  delete [] data;  data = nullptr;  cv::imshow("image", image);  cv::waitKey(0);  }  // Destroy the engine  context->destroy();  engine->destroy();  runtime->destroy();  delete[] detectionOut;  delete[] keepCount;  shutdownProtobufLibrary();  } |

### 2.3 CMakeLists.txt

|  |
| --- |
| cmake\_minimum\_required(VERSION 2.8)  PROJECT(ssd\_test)  SET(CMAKE\_CXX\_STANDARD 11)  FIND\_PACKAGE(OpenCV REQUIRED)  FIND\_PACKAGE(CUDA REQUIRED)  INCLUDE\_DIRECTORIES(./include  /home/yin/TensorRT-5.1.5.0/include  ${CUDA\_INCLUDE\_DIRS}  )  SET(TENSORRT\_LIB\_PATH "/home/yin/TensorRT-5.1.5.0/lib")  FILE(GLOB TENSORRT\_LIBS "${TENSORRT\_LIB\_PATH}/\*.so")  AUX\_SOURCE\_DIRECTORY(src SRC\_LIST)  ADD\_EXECUTABLE(${PROJECT\_NAME} ${SRC\_LIST})  TARGET\_LINK\_LIBRARIES(${PROJECT\_NAME} ${OpenCV\_LIBS} ${TENSORRT\_LIBS} ${CUDA\_LIBRARIES}) |



## 部署其它SSD网络

下面以pelee-ssd为例，对以上程序作少部分修改即可对其它ssd网络进行加速。

1. 输入图像大小

|  |
| --- |
| static const int kINPUT\_H = 300; // Input image height  static const int kINPUT\_W = 300; // Input image width |

1. 模型文件，同样要对prototxt进行相应修改

|  |
| --- |
| caffeToTRTModel("../model/ssd.prototxt",  "../model/VGG\_VOC0712\_SSD\_300x300\_iter\_120000.caffemodel",  std::vector<std::string>{kOUTPUT\_BLOB\_NAME0, kOUTPUT\_BLOB\_NAME1},  N, &trtModelStream); |

1. 预处理

与VGG不同，这里的输入是归一化到[-1,1]的，应作适当调整：

|  |
| --- |
| float mean[3] = {127.5, 127.5, 127.5};  for(int r = 0; r < kINPUT\_H;r++)  {  for(int c = 0; c < kINPUT\_W;c++)  {  for(int t = 0;t < kINPUT\_C;t++)  {  data[t \* kINPUT\_H \* kINPUT\_W + r \* kINPUT\_W + c] = (image\_f.ptr<Vec3f>(r)[c][0] - mean[t]) \* 0.007843;  }  }  } |

1. 主要代码

|  |
| --- |
| #include <cassert>  #include <cmath>  #include <cstring>  #include <cuda\_runtime\_api.h>  #include <unordered\_map>  #include "NvCaffeParser.h"  #include "NvInferPlugin.h"  #include "common.h"  #include <vector>  #include <opencv2/opencv.hpp>  using namespace nvinfer1;  using namespace nvcaffeparser1;  using namespace plugin;  using namespace std;  using namespace cv;  // Network details  const char\* gNetworkName = "ssd"; // Network name  static const int kINPUT\_C = 3; // Input image channels  static const int kINPUT\_H = 300; // Input image height  static const int kINPUT\_W = 300; // Input image width  // static const int kINPUT\_H = 304; // Input image height  // static const int kINPUT\_W = 304; // Input image width  static const int kOUTPUT\_CLS\_SIZE = 21; // Number of classes  static const int kKEEP\_TOPK = 200; // Number of total bboxes to be kept per image after NMS step. It is same as detection\_output\_param.keep\_top\_k in prototxt file  const std::string gCLASSES[kOUTPUT\_CLS\_SIZE] = {"background", "aeroplane", "bicycle", "bird", "boat",  "bottle", "bus", "car", "cat", "chair",  "cow", "diningtable", "dog", "horse", "motorbike",  "person", "pottedplant", "sheep", "sofa", "train",  "tvmonitor"}; // List of class labels  static const char\* kINPUT\_BLOB\_NAME = "data"; // Input blob name  static const char\* kOUTPUT\_BLOB\_NAME0 = "detection\_out"; // Output blob name  static const char\* kOUTPUT\_BLOB\_NAME1 = "keep\_count"; // Output blob name  Logger logger;  // Visualization  const float kVISUAL\_THRESHOLD = 0.6f;  void caffeToTRTModel(const std::string& deployFile, // Name for caffe prototxt  const std::string& modelFile, // Name for model  const std::vector<std::string>& outputs, // Network outputs  unsigned int maxBatchSize, // Batch size - NB must be at least as large as the batch we want to run with)  IHostMemory\*\* trtModelStream) // Output stream for the TensorRT model  {  // Create the builder  IBuilder\* builder = createInferBuilder(logger);  assert(builder != nullptr);  // Parse the caffe model to populate the network, then set the outputs  INetworkDefinition\* network = builder->createNetwork();  ICaffeParser\* parser = createCaffeParser();  const IBlobNameToTensor\* blobNameToTensor = parser->parse(deployFile.c\_str(),  modelFile.c\_str(),  \*network,  nvinfer1::DataType::kHALF);  // Specify which tensors are outputs  for (auto& s : outputs)  network->markOutput(\*blobNameToTensor->find(s.c\_str()));  // Build the engine  builder->setMaxBatchSize(maxBatchSize);  builder->setMaxWorkspaceSize(36 << 20);  builder->setFp16Mode(true);  builder->setInt8Mode(false);  builder->allowGPUFallback(true);  ICudaEngine\* engine;  engine = builder->buildCudaEngine(\*network);  assert(engine);  network->destroy();  parser->destroy();  (\*trtModelStream) = engine->serialize();  engine->destroy();  builder->destroy();  }  void doInference(IExecutionContext& context, float\* inputData, float\* detectionOut, int\* keepCount, int batchSize)  {  const ICudaEngine& engine = context.getEngine();  // input and output buffer pointers that we pass to the engine - the engine requires exactly IEngine::getNbBindings(),  // of these, but in this case we know that there is exactly 1 input and 2 output.  assert(engine.getNbBindings() == 3);  void\* buffers[3];  // In order to bind the buffers, we need to know the names of the input and output tensors.  // note that indices are guaranteed to be less than IEngine::getNbBindings()  int inputIndex = engine.getBindingIndex(kINPUT\_BLOB\_NAME),  outputIndex0 = engine.getBindingIndex(kOUTPUT\_BLOB\_NAME0),  outputIndex1 = engine.getBindingIndex(kOUTPUT\_BLOB\_NAME1);  // Create GPU buffers and a stream  CHECK(cudaMalloc(&buffers[inputIndex], batchSize \* kINPUT\_C \* kINPUT\_H \* kINPUT\_W \* sizeof(float))); // Data  CHECK(cudaMalloc(&buffers[outputIndex0], batchSize \* kKEEP\_TOPK \* 7 \* sizeof(float))); // Detection\_out  CHECK(cudaMalloc(&buffers[outputIndex1], batchSize \* sizeof(int))); // KeepCount (BBoxs left for each batch)  cudaStream\_t stream;  CHECK(cudaStreamCreate(&stream));  // DMA the input to the GPU, execute the batch asynchronously, and DMA it back:  CHECK(cudaMemcpyAsync(buffers[inputIndex], inputData, batchSize \* kINPUT\_C \* kINPUT\_H \* kINPUT\_W \* sizeof(float), cudaMemcpyHostToDevice, stream));  context.enqueue(batchSize, buffers, stream, nullptr);  CHECK(cudaMemcpyAsync(detectionOut, buffers[outputIndex0], batchSize \* kKEEP\_TOPK \* 7 \* sizeof(float), cudaMemcpyDeviceToHost, stream));  CHECK(cudaMemcpyAsync(keepCount, buffers[outputIndex1], batchSize \* sizeof(int), cudaMemcpyDeviceToHost, stream));  cudaStreamSynchronize(stream);  // Release the stream and the buffers  cudaStreamDestroy(stream);  CHECK(cudaFree(buffers[inputIndex]));  CHECK(cudaFree(buffers[outputIndex0]));  CHECK(cudaFree(buffers[outputIndex1]));  }  int main(int argc, char\*\* argv)  {  initLibNvInferPlugins(&logger, "");  IHostMemory\* trtModelStream{nullptr};  // Create a TensorRT model from the caffe model and serialize it to a stream  const int N = 1; // Batch size  // caffeToTRTModel("../model/pelee.prototxt",  // "../model/pelee\_SSD\_304x304\_iter\_120000.caffemodel",  // std::vector<std::string>{kOUTPUT\_BLOB\_NAME0, kOUTPUT\_BLOB\_NAME1},  // N, &trtModelStream);  caffeToTRTModel("../model/ssd.prototxt",  "../model/VGG\_VOC0712\_SSD\_300x300\_iter\_120000.caffemodel",  std::vector<std::string>{kOUTPUT\_BLOB\_NAME0, kOUTPUT\_BLOB\_NAME1},  N, &trtModelStream);  std::vector<cv::String> imageList;  cv::glob("../images/\*.jpg", imageList);  printf("get %d images\n", imageList.size());  IRuntime\* runtime = createInferRuntime(logger);  ICudaEngine\* engine = runtime->deserializeCudaEngine(trtModelStream->data(), trtModelStream->size(), nullptr);  assert(engine != nullptr);  IExecutionContext\* context = engine->createExecutionContext();  assert(context != nullptr);  trtModelStream->destroy();  float\* detectionOut = new float[N \* kKEEP\_TOPK \* 7];  int\* keepCount = new int[N];  // float mean[3] = {104, 117, 123};  for(int idx = 0;idx<imageList.size();idx++)  {  cv::Mat image = cv::imread(imageList[idx]);  cv::Mat image\_f;  image.convertTo(image\_f, CV\_32FC3);  cv::resize(image\_f, image\_f, cv::Size(kINPUT\_W, kINPUT\_H));  float \*data = new float[kINPUT\_C \* kINPUT\_H \* kINPUT\_W];  float mean[3] = {127.5, 127.5, 127.5};  for(int r = 0; r < kINPUT\_H;r++)  {  for(int c = 0; c < kINPUT\_W;c++)  {  for(int t = 0;t < kINPUT\_C;t++)  {  //data[t \* kINPUT\_H \* kINPUT\_W + r \* kINPUT\_W + c] = image\_f.ptr<Vec3f>(r)[c][0] - mean[t]);  data[t \* kINPUT\_H \* kINPUT\_W + r \* kINPUT\_W + c] = (image\_f.ptr<Vec3f>(r)[c][0] - mean[t]) \* 0.007843;  }  }  }  doInference(\*context, data, detectionOut, keepCount, N);  for (int p = 0; p < N; ++p)  {  for (int i = 0; i < keepCount[p]; ++i)  {  float\* det = detectionOut + (p \* kKEEP\_TOPK + i) \* 7;  if (det[2] < kVISUAL\_THRESHOLD)  continue;  float xmin = det[3] \* image.cols;  float ymin = det[4] \* image.rows;  float xmax = det[5] \* image.cols;  float ymax = det[6] \* image.rows;  std::string className = gCLASSES[(int) det[1]];  cv::rectangle(image, cv::Rect(cv::Point(xmin, ymin), cv::Point(xmax, ymax)), cv::Scalar(0, 255, 0), 1);  cv::putText(image, className, cv::Point(xmin + 5, ymin + 20), 1, 1.5, cv::Scalar(0, 0, 255));  }  }  delete [] data;  data = nullptr;  cv::imshow("image", image);  cv::waitKey(0);  }  // Destroy the engine  context->destroy();  engine->destroy();  runtime->destroy();  delete[] detectionOut;  delete[] keepCount;  shutdownProtobufLibrary();  } |

