# 使用Mobilenet-SSD训练自己的数据

## 编译Caffe

首先，下载SSD源码：git clone <https://github.com/weiliu89/caffe/tree/ssd，下载后记得切换到ssd分支下。Caffe的编译安装过程参考官网。>

下载Mobilenet-SSD源码：<https://github.com/chuanqi305/MobileNet-SSD>，放到caffe/examples下。

编译caffe。

## 数据准备

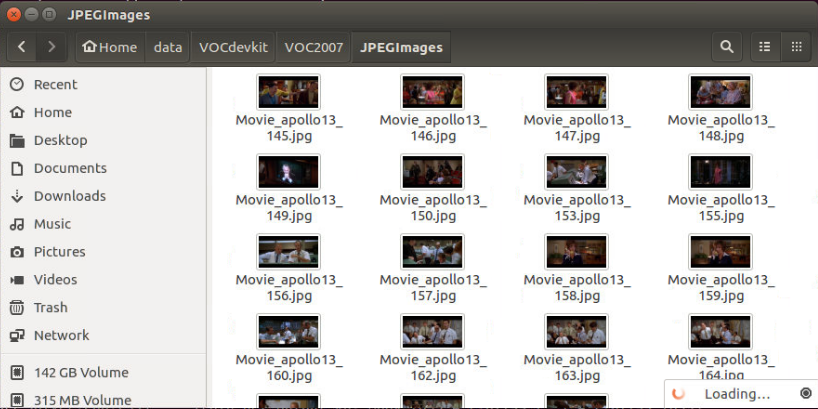
需要将自己的数据格式转成VOC2007的格式，在主目录下新建data文件夹，其下目录结构如下：



我们要将各文件夹放入相应的数据文件。

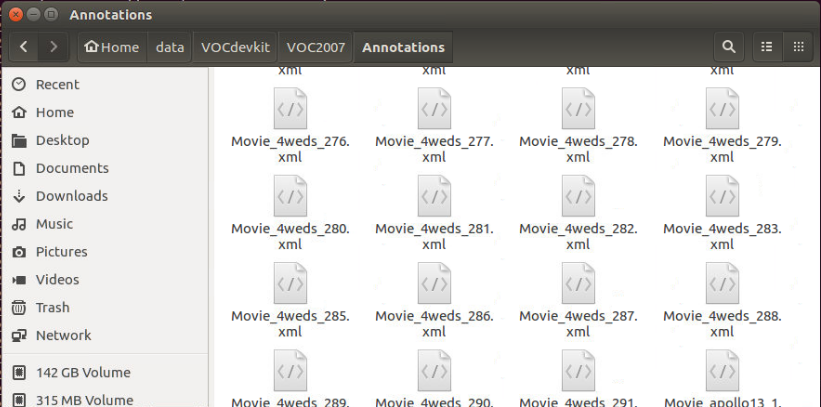
### JPEGImages文件夹

这里放置训练的图像文件，格式为jpg文件。



### Annotations文件夹

这里放置图像文件对应的标柱文件。



标注文件为xml格式，文件名一对应的图像文件名相同，格式类似如下：



由于我下载的数据集不是VOC格式的，而是matlab的mat文件格式，需要转换。为了方便以后数据转换方便，我先将数据转成如下的文本文件：



第一行为图像的高、宽、通道数量，下面每一行分别为目标的类型编号（从1开始）目标左上角坐标及右下角坐标，放置在SRC\_LABEL目录下。

我通过如下的python代码转成xml格式。

|  |
| --- |
| import os, sys  import numpy as np  class Object:  def \_\_init\_\_(self):  self.name = ''  self.minX = 0  self.minY = 0  self.maxX = 0  self.maxY = 0  class Annotation:  def \_\_init\_\_(self):  self.filename = ''  self.width = 0  self.height = 0  self.depth = 0  self.objs = []  self.classmap = ['background', 'hand']  def read(self, filepath):  if os.path.isfile(filepath):  self.filename = os.path.basename(filepath).split(".")[0] + ".jpg"  with open(filepath, 'r') as file:  file\_info = file.readline().split(' ')  self.height = int(file\_info[0])  self.width = int(file\_info[1])  self.depth = int(file\_info[2])  while(True):  obj\_info = file.readline().split(' ')  if not obj\_info or len(obj\_info) != 5:  break  obj = Object()  class\_idx = int(obj\_info[0])  obj.name = self.classmap[class\_idx]  obj.minX = int(obj\_info[1])  obj.minY = int(obj\_info[2])  obj.maxX = int(obj\_info[3])  obj.maxY = int(obj\_info[4])  self.objs.append(obj)  root\_dir = './VOCdevkit'  annotations\_dir = root\_dir + '/Annotations'  imagesets\_dir = root\_dir + '/ImageSets'  jpegimages\_dir = root\_dir + '/JPEGImages'  src\_dir = root\_dir + '/SRC\_LABEL'  dct = ['hand']  label\_list = os.listdir(src\_dir)  file\_count = 0  for i in range(0, len(label\_list)):  path = os.path.join(src\_dir, label\_list[i])  if os.path.isfile(path):  #read label  ano = Annotation()  ano.read(filepath = path)  image\_name = label\_list[i].split(".")[0]  annotaiton\_name = annotations\_dir + "/" + image\_name + ".xml"  with open(annotaiton\_name, 'w') as annotation:  annotation.write("<annotation>\n")  #folder  annotation.write("<folder>")  annotation.write("VOC2007")  annotation.write("</folder>\n")  #filename  annotation.write("<filename>")  annotation.write(ano.filename)  annotation.write("</filename>\n")  #source  annotation.write("<source>\n")  annotation.write("<database>")  annotation.write("Unknown")  annotation.write("</database>\n")  annotation.write("</source>\n")  #size  annotation.write("<size>\n")  annotation.write("<width>")  annotation.write(str(ano.width))  annotation.write("</width>\n")  annotation.write("<height>")  annotation.write(str(ano.height))  annotation.write("</height>\n")  annotation.write("<depth>")  annotation.write(str(ano.depth))  annotation.write("</depth>\n")  annotation.write("</size>\n")  #segmented  annotation.write("<segmented>")  annotation.write(str(0))  annotation.write("</segmented>\n")  #object  for i in range(len(ano.objs)):  annotation.write("<object>")  annotation.write("<name>")  annotation.write(ano.objs[i].name)  annotation.write("</name>\n")  annotation.write("<pose>")  annotation.write("Unspecified")  annotation.write("</pose>\n")  annotation.write("<truncated>")  annotation.write(str(0))  annotation.write("</truncated>\n")  annotation.write("<difficult>")  annotation.write(str(0))  annotation.write("</difficult>\n")  annotation.write("<bndbox>\n")  annotation.write("<xmin>")  annotation.write(str(ano.objs[i].minX))  annotation.write("</xmin>\n")  annotation.write("<ymin>")  annotation.write(str(ano.objs[i].minY))  annotation.write("</ymin>\n")  annotation.write("<xmax>")  annotation.write(str(ano.objs[i].maxX))  annotation.write("</xmax>\n")  annotation.write("<ymax>")  annotation.write(str(ano.objs[i].maxY))  annotation.write("</ymax>\n")  annotation.write("</bndbox>\n")  annotation.write("</object>\n")  annotation.write("</annotation>")  print file\_count, "file finish!"  file\_count = file\_count + 1 |

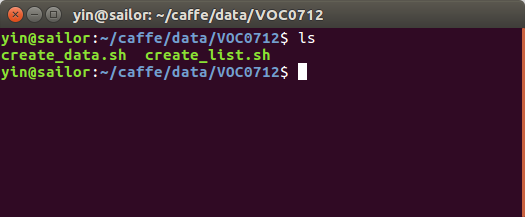
### ImageSets文件夹

ImageSets下有一个Main文件夹，放置训练、测试的图像文件名，可以由以下python代码生成：

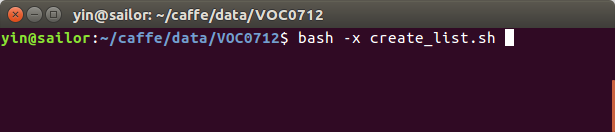
|  |
| --- |
| import os  import random    train\_percent = 0.70  xmlfilepath = 'Annotations'  txtsavepath = 'ImageSets\Main'  total\_xml = os.listdir(xmlfilepath)    num=len(total\_xml)  list=range(num)  tr=int(num\*train\_percent)  train=random.sample(list,tr)  test = random.sample(list, int(num \* 0.1))    ftrain = open('ImageSets/Main/train.txt', 'w')  fval = open('ImageSets/Main/val.txt', 'w')  ftrainval = open('ImageSets/Main/trainval.txt', 'w')  ftest = open('ImageSets/Main/test.txt', 'w')    for i in list:  name=total\_xml[i][:-4]+'\n'  if i in train:  ftrain.write(name)  else:  fval.write(name)  ftrainval.write(name)  if i in test:  ftest.write(name)      ftrain.close()  fval.close()  print ("finished") |

### **生成lmdb数据**

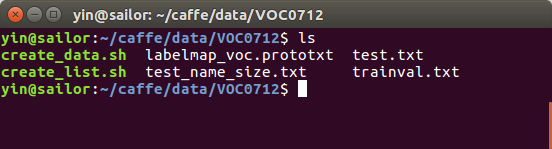
在caffe/data/VOC0712目录下有以下文件：



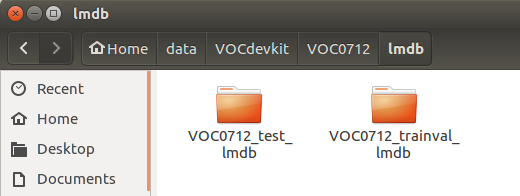
运行create\_list.sh：bash -x create\_list.sh:



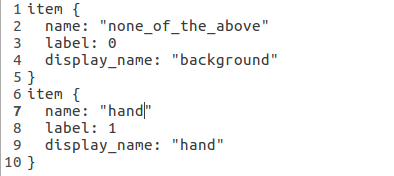
生成以下文件：



执行bash -x create\_data.sh，生成lmdb文件：



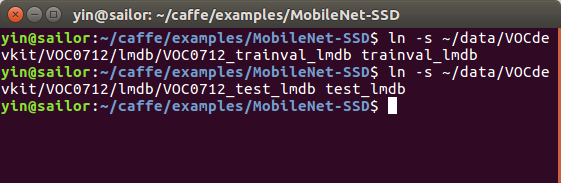
此外，在caffe/data/VOC0712目录下，制作一个labelmap\_voc.prototxt文件，内容如下：



### 训练

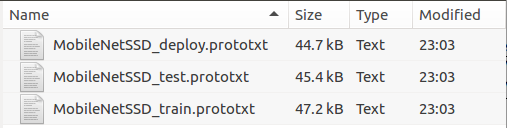
切换到/caffe/examples/MobileNet-SSD目录下，执行以下命令生成软链接：

|  |
| --- |
| ln -s ~/data/VOCdevkit/VOC0712/lmdb/VOC0712\_trainval\_lmdb trainval\_lmdb  ln -s ~/data/VOCdevkit/VOC0712/lmdb/VOC0712\_test\_lmdb test\_lmdb |

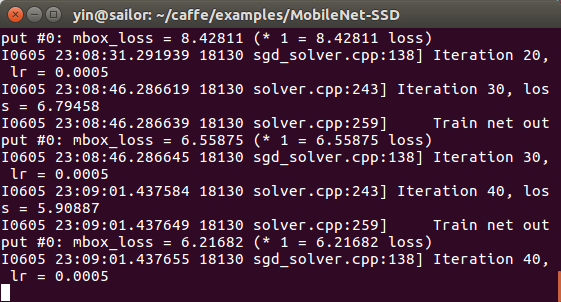


运行./gen\_model.sh 2命令，生成模型文件，其中2表示两类，这里是一个人手检测任务，只有1类，加上背景，共两类。

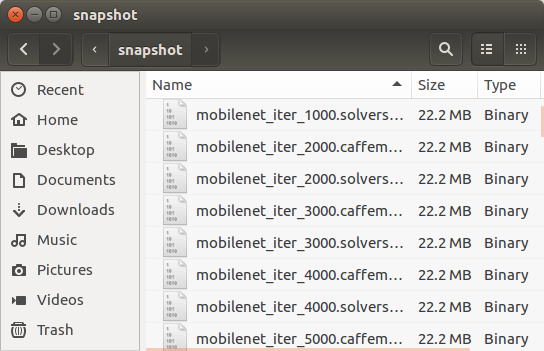
此时在example目录下生成以下文件：



运行sudo sh train.sh进行训练：



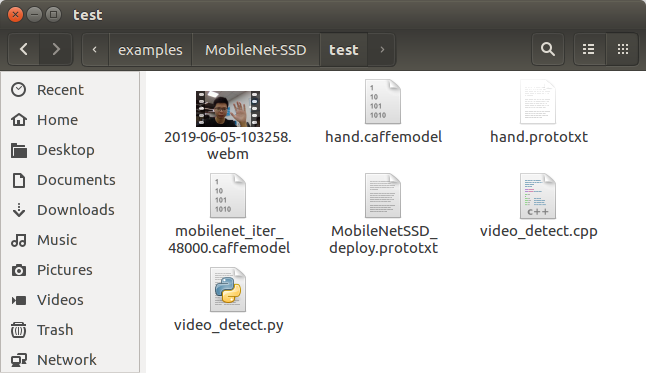
训练过程中会在snapshot目录下生成中间模型文件：



我在MobileNet-SSD目录下创建一个test目录，将文件MobileNetSSD\_deploy.prototxt和mobilenet\_iter\_48000.caffemodel放在改目录中，执行：

|  |
| --- |
| python merge\_bn.py --model test/MobileNetSSD\_deploy.prototxt --weights test/mobilenet\_iter\_48000.caffemodel |

在MobileNet-SSD中生成no\_bn.caffemodel和no\_bn.prototxt文件，我重命名为hand.caffemodel和hand.prototxt，同样放到test目录中。

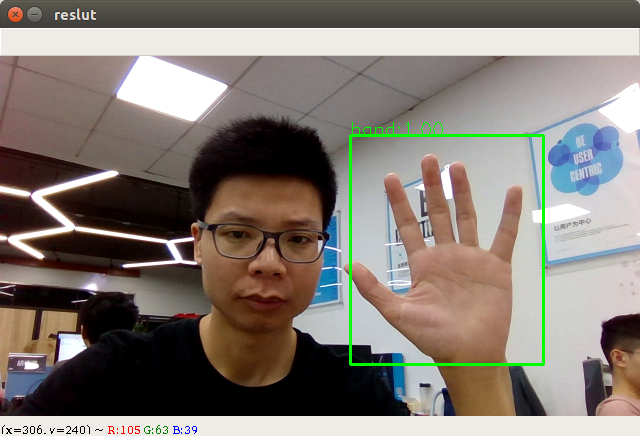


### 测试

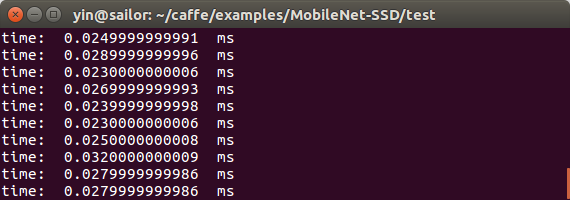
这里使用Caffe的Python和C++相关函数进行测试：

### 4.1 python版本

|  |
| --- |
| import numpy as np  import sys,os  import cv2  import time  caffe\_root = '/home/hy17003/caffe/'  sys.path.insert(0, caffe\_root + 'python')  import caffe  net\_file= 'hand.prototxt'  caffe\_model='hand.caffemodel'  video\_path = '2019-06-05-103258.webm'  if not os.path.exists(caffe\_model):  print("MobileNetSSD\_deploy.affemodel does not exist,")  print("use merge\_bn.py to generate it.")  exit()  caffe.set\_mode\_gpu()  caffe.set\_device(0)  net = caffe.Net(net\_file,caffe\_model,caffe.TEST)  CLASSES = ('background','hand')  def preprocess(src):  img = cv2.resize(src, (300,300))  img = img - 127.5  img = img \* 0.007843  return img  def postprocess(img, out):  h = img.shape[0]  w = img.shape[1]  box = out['detection\_out'][0,0,:,3:7] \* np.array([w, h, w, h])  cls = out['detection\_out'][0,0,:,1]  conf = out['detection\_out'][0,0,:,2]  return (box.astype(np.int32), conf, cls)  def detect(origimg):  img = preprocess(origimg)  img = img.astype(np.float32)  img = img.transpose((2, 0, 1))  net.blobs['data'].data[...] = img  out = net.forward()  t0 = time.clock()  box, conf, cls = postprocess(origimg, out)  t1 = time.clock()  print "time: ", (t1 - t0) \* 1000, " ms"  for i in range(len(box)):  if conf[i] > 0.4:  p1 = (box[i][0], box[i][1])  p2 = (box[i][2], box[i][3])  cv2.rectangle(origimg, p1, p2, (0,255,0), 2)  p3 = (max(p1[0], 15), max(p1[1], 15))  title = "%s:%.2f" % (CLASSES[int(cls[i])], conf[i])  cv2.putText(origimg, title, p3, cv2.FONT\_ITALIC, 0.6, (0, 255, 0), 1)  return origimg  cap = cv2.VideoCapture(video\_path)  success, frame = cap.read()  while success:  height, width = frame.shape[0], frame.shape[1]  small\_size = (int(width\*0.5), int(height\*0.5))  frame = cv2.resize(frame, small\_size)  result = detect(frame)  cv2.imshow("reslut", result)  cv2.waitKey(1)  success, frame = cap.read() |



时间0.02ms左右：



### 4.2 C++版本

重新编译caffe后，会在caffe/build/example/test目录中生成video\_detect.bin文件。

|  |
| --- |
| // This is a demo code for using a SSD model to do detection.  #include <caffe/caffe.hpp>  #include <opencv2/core/core.hpp>  #include <opencv2/highgui/highgui.hpp>  #include <opencv2/imgproc/imgproc.hpp>  #include <algorithm>  #include <iomanip>  #include <iosfwd>  #include <memory>  #include <string>  #include <utility>  #include <vector>  #include <sys/time.h>  #ifdef USE\_OPENCV  using namespace caffe; // NOLINT(build/namespaces)  double getTime()  {  struct timeval tv;  gettimeofday(&tv,0);  return (double)(tv.tv\_sec \* 1000 + double(tv.tv\_usec) / 1000);  }  class Detector {  public:  Detector(const string& model\_file,  const string& weights\_file,  const string& mean\_file,  const string& mean\_value);  std::vector<vector<float> > Detect(const cv::Mat& img);  private:  void SetMean(const string& mean\_file, const string& mean\_value);  void WrapInputLayer(std::vector<cv::Mat>\* input\_channels);  void Preprocess(const cv::Mat& img,  std::vector<cv::Mat>\* input\_channels);  private:  shared\_ptr<Net<float> > net\_;  cv::Size input\_geometry\_;  int num\_channels\_;  cv::Mat mean\_;  };  Detector::Detector(const string& model\_file,  const string& weights\_file,  const string& mean\_file,  const string& mean\_value) {  Caffe::set\_mode(Caffe::GPU);  /\* Load the network. \*/  net\_.reset(new Net<float>(model\_file, TEST));  net\_->CopyTrainedLayersFrom(weights\_file);  CHECK\_EQ(net\_->num\_inputs(), 1) << "Network should have exactly one input.";  CHECK\_EQ(net\_->num\_outputs(), 1) << "Network should have exactly one output.";  Blob<float>\* input\_layer = net\_->input\_blobs()[0];  num\_channels\_ = input\_layer->channels();  CHECK(num\_channels\_ == 3 || num\_channels\_ == 1)  << "Input layer should have 1 or 3 channels.";  input\_geometry\_ = cv::Size(input\_layer->width(), input\_layer->height());  /\* Load the binaryproto mean file. \*/  SetMean(mean\_file, mean\_value);  }  std::vector<vector<float> > Detector::Detect(const cv::Mat& img) {  Blob<float>\* input\_layer = net\_->input\_blobs()[0];  input\_layer->Reshape(1, num\_channels\_,  input\_geometry\_.height, input\_geometry\_.width);  /\* Forward dimension change to all layers. \*/  net\_->Reshape();  std::vector<cv::Mat> input\_channels;  WrapInputLayer(&input\_channels);  Preprocess(img, &input\_channels);  net\_->Forward();  /\* Copy the output layer to a std::vector \*/  Blob<float>\* result\_blob = net\_->output\_blobs()[0];  const float\* result = result\_blob->cpu\_data();  const int num\_det = result\_blob->height();  vector<vector<float> > detections;  for (int k = 0; k < num\_det; ++k) {  if (result[0] == -1) {  // Skip invalid detection.  result += 7;  continue;  }  vector<float> detection(result, result + 7);  detections.push\_back(detection);  result += 7;  }  return detections;  }  /\* Load the mean file in binaryproto format. \*/  void Detector::SetMean(const string& mean\_file, const string& mean\_value) {  cv::Scalar channel\_mean;  if (!mean\_file.empty()) {  CHECK(mean\_value.empty()) <<  "Cannot specify mean\_file and mean\_value at the same time";  BlobProto blob\_proto;  ReadProtoFromBinaryFileOrDie(mean\_file.c\_str(), &blob\_proto);  /\* Convert from BlobProto to Blob<float> \*/  Blob<float> mean\_blob;  mean\_blob.FromProto(blob\_proto);  CHECK\_EQ(mean\_blob.channels(), num\_channels\_)  << "Number of channels of mean file doesn't match input layer.";  /\* The format of the mean file is planar 32-bit float BGR or grayscale. \*/  std::vector<cv::Mat> channels;  float\* data = mean\_blob.mutable\_cpu\_data();  for (int i = 0; i < num\_channels\_; ++i) {  /\* Extract an individual channel. \*/  cv::Mat channel(mean\_blob.height(), mean\_blob.width(), CV\_32FC1, data);  channels.push\_back(channel);  data += mean\_blob.height() \* mean\_blob.width();  }  /\* Merge the separate channels into a single image. \*/  cv::Mat mean;  cv::merge(channels, mean);  /\* Compute the global mean pixel value and create a mean image  \* filled with this value. \*/  channel\_mean = cv::mean(mean);  mean\_ = cv::Mat(input\_geometry\_, mean.type(), channel\_mean);  }  if (!mean\_value.empty()) {  CHECK(mean\_file.empty()) <<  "Cannot specify mean\_file and mean\_value at the same time";  stringstream ss(mean\_value);  vector<float> values;  string item;  while (getline(ss, item, ',')) {  float value = std::atof(item.c\_str());  values.push\_back(value);  }  CHECK(values.size() == 1 || values.size() == num\_channels\_) <<  "Specify either 1 mean\_value or as many as channels: " << num\_channels\_;  std::vector<cv::Mat> channels;  for (int i = 0; i < num\_channels\_; ++i) {  /\* Extract an individual channel. \*/  cv::Mat channel(input\_geometry\_.height, input\_geometry\_.width, CV\_32FC1,  cv::Scalar(values[i]));  channels.push\_back(channel);  }  cv::merge(channels, mean\_);  }  }  /\* Wrap the input layer of the network in separate cv::Mat objects  \* (one per channel). This way we save one memcpy operation and we  \* don't need to rely on cudaMemcpy2D. The last preprocessing  \* operation will write the separate channels directly to the input  \* layer. \*/  void Detector::WrapInputLayer(std::vector<cv::Mat>\* input\_channels) {  Blob<float>\* input\_layer = net\_->input\_blobs()[0];  int width = input\_layer->width();  int height = input\_layer->height();  float\* input\_data = input\_layer->mutable\_cpu\_data();  for (int i = 0; i < input\_layer->channels(); ++i) {  cv::Mat channel(height, width, CV\_32FC1, input\_data);  input\_channels->push\_back(channel);  input\_data += width \* height;  }  }  void Detector::Preprocess(const cv::Mat& img,  std::vector<cv::Mat>\* input\_channels) {  /\* Convert the input image to the input image format of the network. \*/  cv::Mat sample;  if (img.channels() == 3 && num\_channels\_ == 1)  cv::cvtColor(img, sample, cv::COLOR\_BGR2GRAY);  else if (img.channels() == 4 && num\_channels\_ == 1)  cv::cvtColor(img, sample, cv::COLOR\_BGRA2GRAY);  else if (img.channels() == 4 && num\_channels\_ == 3)  cv::cvtColor(img, sample, cv::COLOR\_BGRA2BGR);  else if (img.channels() == 1 && num\_channels\_ == 3)  cv::cvtColor(img, sample, cv::COLOR\_GRAY2BGR);  else  sample = img;  cv::Mat sample\_resized;  if (sample.size() != input\_geometry\_)  cv::resize(sample, sample\_resized, input\_geometry\_);  else  sample\_resized = sample;  cv::Mat sample\_float;  if (num\_channels\_ == 3)  sample\_resized.convertTo(sample\_float, CV\_32FC3);  else  sample\_resized.convertTo(sample\_float, CV\_32FC1);  cv::Mat sample\_normalized;  cv::subtract(sample\_float, mean\_, sample\_normalized);  sample\_normalized = sample\_normalized \* 0.07843;  /\* This operation will write the separate BGR planes directly to the  \* input layer of the network because it is wrapped by the cv::Mat  \* objects in input\_channels. \*/  cv::split(sample\_normalized, \*input\_channels);  CHECK(reinterpret\_cast<float\*>(input\_channels->at(0).data)  == net\_->input\_blobs()[0]->cpu\_data())  << "Input channels are not wrapping the input layer of the network.";  }  DEFINE\_string(mean\_file, "",  "The mean file used to subtract from the input image.");  DEFINE\_string(mean\_value, "104,117,123",  "If specified, can be one value or can be same as image channels"  " - would subtract from the corresponding channel). Separated by ','."  "Either mean\_file or mean\_value should be provided, not both.");  DEFINE\_string(file\_type, "image",  "The file type in the list\_file. Currently support image and video.");  DEFINE\_string(out\_file, "",  "If provided, store the detection results in the out\_file.");  DEFINE\_double(confidence\_threshold, 0.01,  "Only store detections with score higher than the threshold.");  int main(int argc, char\*\* argv) {  if(argc < 4)  {  printf("usage: ssd\_detect modelTxt modelBin video");  return -1;  }    const string& model\_file = string(argv[1]);  const string& weights\_file = string(argv[2]);  const string& mean\_file = "";  //const string& mean\_value = "104,117,123";  const string& mean\_value = "127.5,127.5,127.5";  std::string file = string(argv[3]);  const float confidence\_threshold = 0.3;  // Initialize the network.  Detector detector(model\_file, weights\_file, mean\_file, mean\_value);  cv::VideoCapture cap(file);  if (!cap.isOpened())  {  LOG(FATAL) << "Failed to open video: " << file;  }  cv::Mat img;  int frame\_count = 0;  while (true)  {  bool success = cap.read(img);  if (!success)  {  LOG(INFO) << "Process " << frame\_count << " frames from " << file;  break;  }  CHECK(!img.empty()) << "Error when read frame";  cv::resize(img, img, cv::Size(img.cols / 2, img.rows / 2));  double t1 = getTime();  std::vector<vector<float> > detections = detector.Detect(img);  double t2 = getTime();  printf("detect time: %f ms\n", t2 - t1);  /\* Print the detection results. \*/  for (int i = 0; i < detections.size(); ++i)  {  const vector<float>& d = detections[i];  // Detection format: [image\_id, label, score, xmin, ymin, xmax, ymax].  CHECK\_EQ(d.size(), 7);  const float score = d[2];  if (score >= confidence\_threshold)  {  float xmin = static\_cast<int>(d[3] \* img.cols);  float ymin = static\_cast<int>(d[4] \* img.rows);  float xmax = static\_cast<int>(d[5] \* img.cols);  float ymax = static\_cast<int>(d[6] \* img.rows);  cv::rectangle(img, cv::Rect(cv::Point(xmin, ymin), cv::Point(xmax, ymax)), cv::Scalar(0, 255, 0), 1);  }  }  cv::imshow("img", img);  cv::waitKey(1);  ++frame\_count;  }  if (cap.isOpened())  {  cap.release();  }  return 0;  }  #else  int main(int argc, char\*\* argv) {  LOG(FATAL) << "This example requires OpenCV; compile with USE\_OPENCV.";  }  #endif // USE\_OPENCV |