**目录**

内容目录

[20190422 多标签网络 1](#__RefHeading___Toc524_1139385966)

[20190507 修改数据集&重新运行 3](#__RefHeading___Toc526_1139385966)

[20190508 对比结果txt的预处理 6](#__RefHeading___Toc528_1139385966)

[20190514 对比结果准确率&如何提高 8](#__RefHeading___Toc530_1139385966)

[20190517 双路视频运行BSD网络 10](#__RefHeading___Toc570_1180456630)

**20190422 多标签网络**

1. 学习caffe中的多个分类label的输入问题。默认情况下caffe的训练数据集是单分类器。有几种方法修改为多标签分类器：

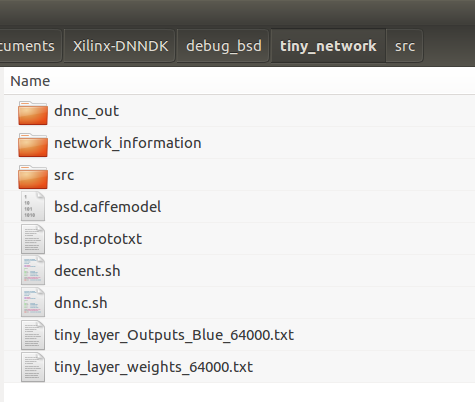
对于DNNDK可以使用如下图作为calibration.txt的样本：



但是，words.txt对应是什么文件？

1. 下面是使用tiny\_network，然后修改了src/main.cpp后，得到的tiny网络的输出：

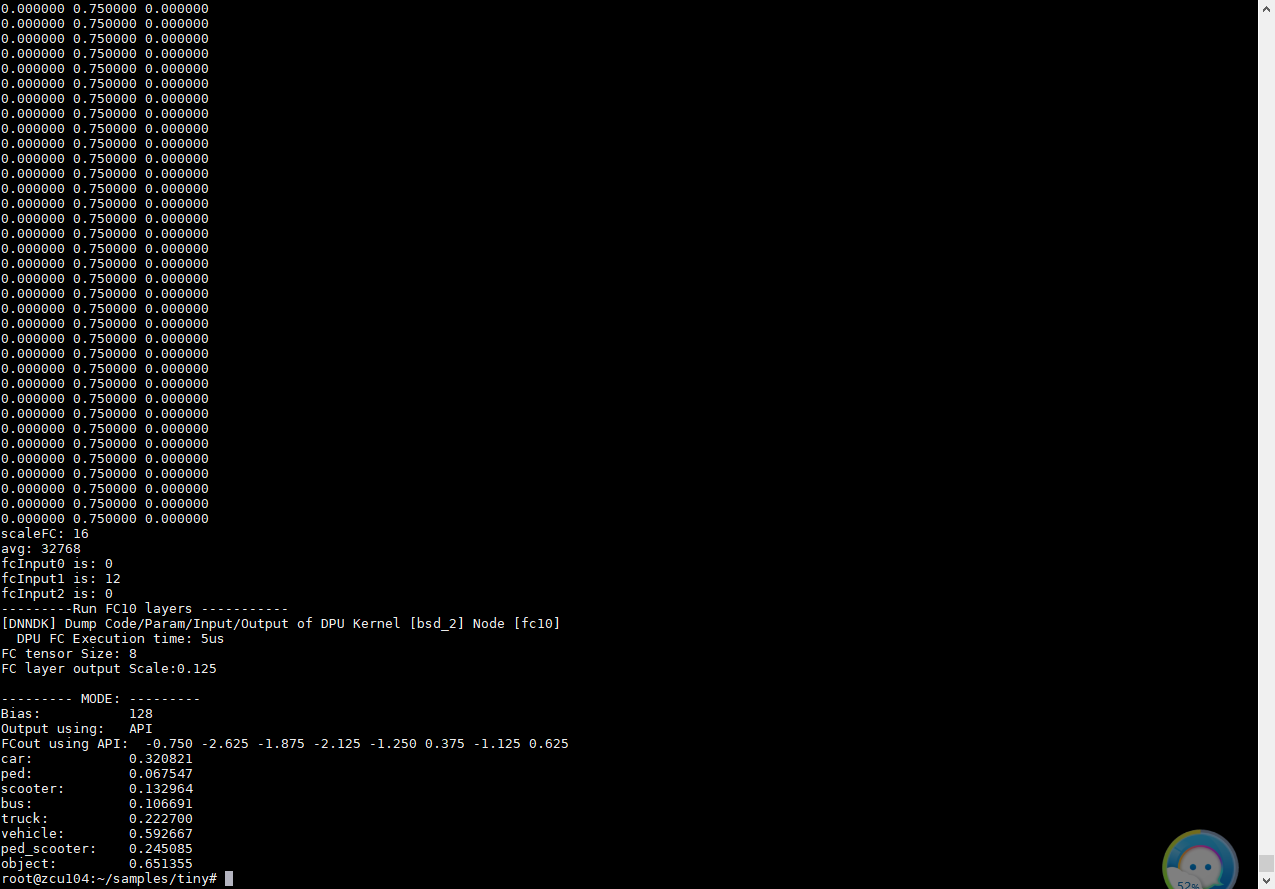
（详细的网络结构文件/参数文件/生成网络信息/main.cc/Makfile/详见debug\_bsd/tiny\_network目录！）



蓝色图像，网络输入为RGB的signed char类型(**CV\_SC8C3**), R/G/B = -128,-128,127



运行的结果：



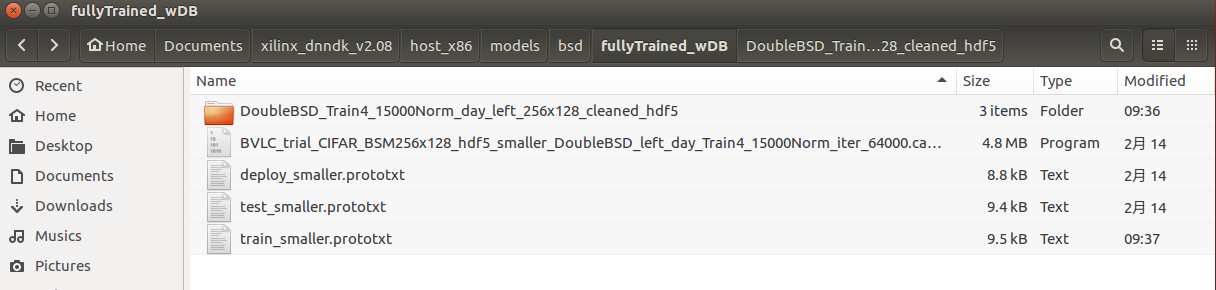
这里生成了中间各层的weights&bias，可以与amandeep生成的weights&bias进行对比。

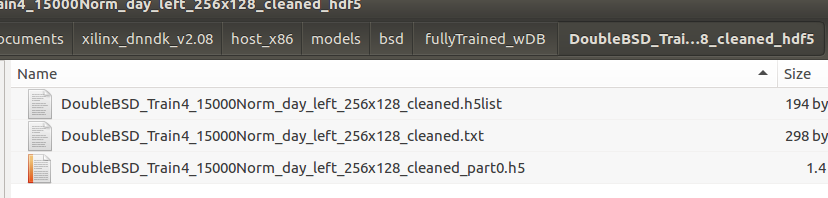
并可以进行各层的计算，看是否有问题。

**20190507 修改数据集&重新运行**

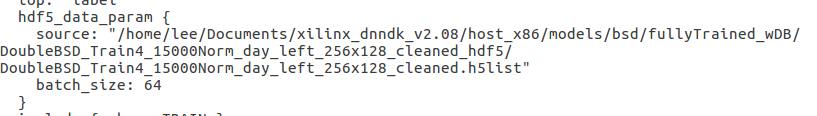
**修改data set得到正确的结果：**

**1.**  Amandeep找到了使用hdf5格式数据作为DNNDK的输入校准源数据。不需要calibration.txt进行校准。使用fully\_Trained\_wDB/ DoubleBSD\_Train4\_15000Norm\_day\_left\_256x128\_cleaned\_hdf5/ 下面的3个文件

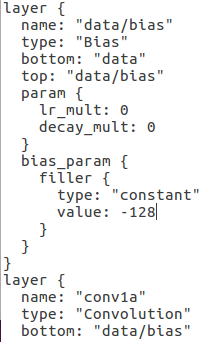




1. 其中的\*\*\*.h5list中需要修改为\*\*\*.h5文件的路径。
2. train\_smaller.prototxt中的数据集源文件需要指定\*\*\*.h5list的地址。如下：

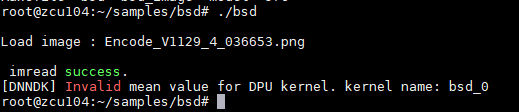


1. DECENT 不认识“Bias”层，需要删除。然后修改conv1a的输入。如下图：



1. 这时decent.sh可以通过，然后需要删除deploy.prototxt中的最后3层(计算loss，来得到学习正确率的)。
2. 最后可以得到elf文件。
3. 从calibration计算的loss可以看到，这个没有修改BIAS版本的错误率loss较高。所以可能到不到正确的结果。(Amandeep又重新进行了训练)

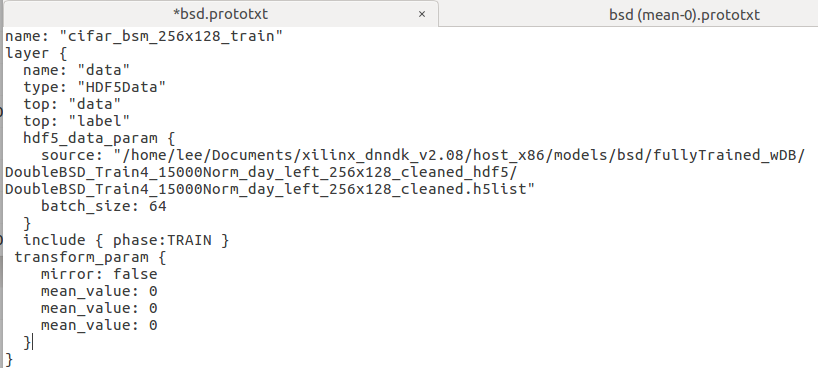
**2.** 运行./bsd出现错误，如下图：



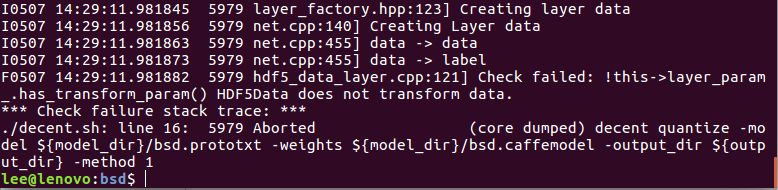
mean value是无效的。

**解决方法:**

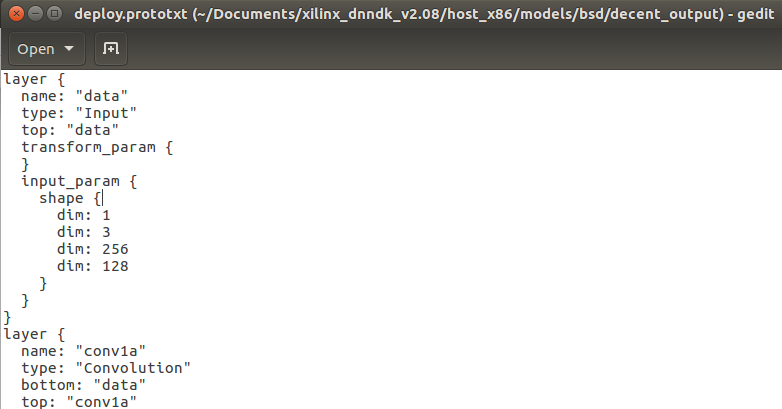
* 在train.prototxt里面添加mean\_value:值，再进行DECENT会报错：



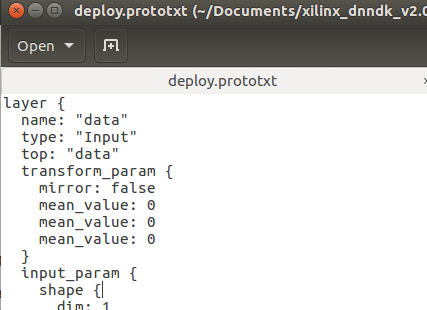
提示：



* 需要在DECENT生成的deploy.prototxt中进行修改：



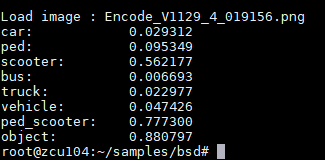
添加transform\_param{} 里面的mean\_value：0。

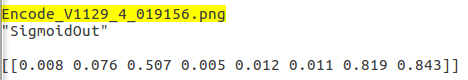


然后即可执行DNNC.SH运行得到elf文件。

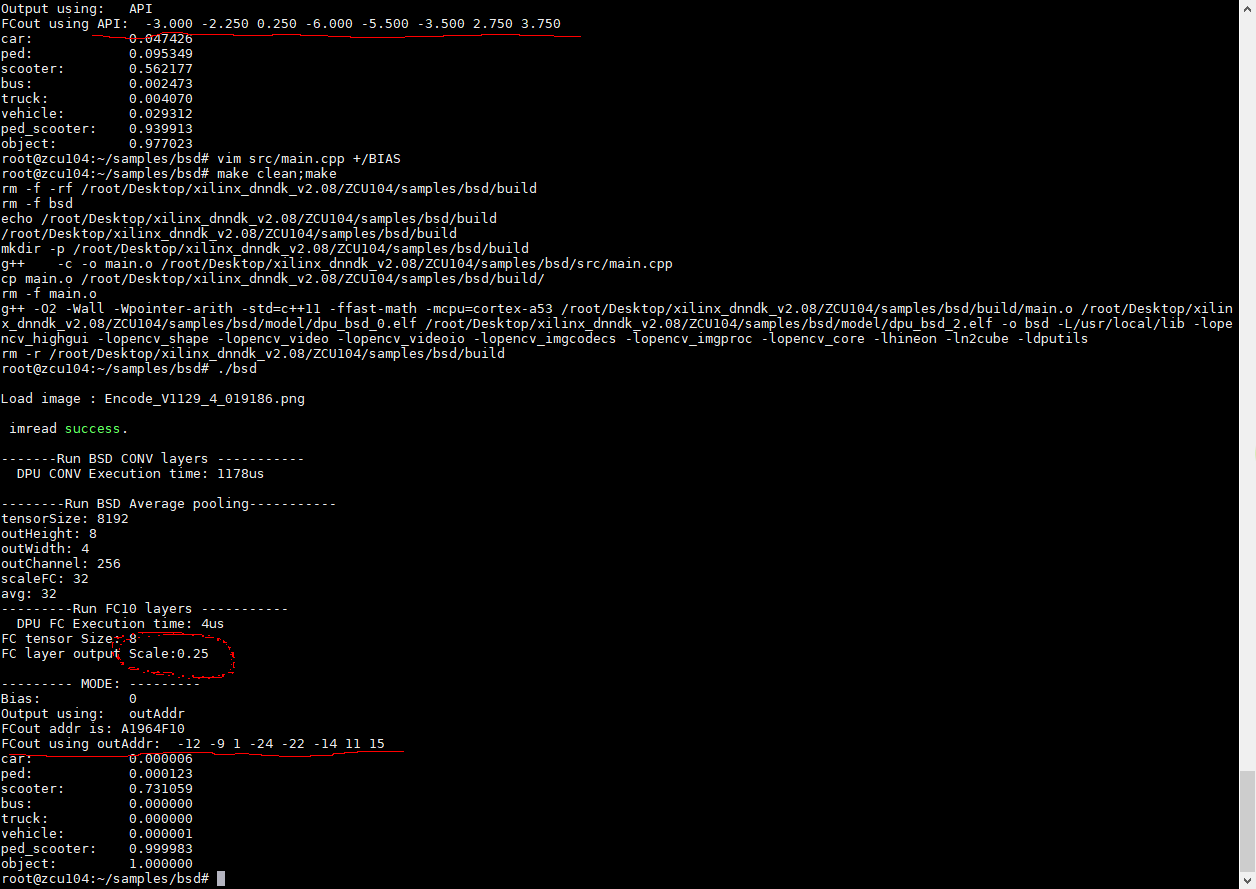
得到结果：原始图片和ZCU104&PC端执行结果如下：







**3.** 对比了使用API和outAddr来获取fc层输出，可以从上面看到，API和outAddr的值是相差一个scaleFC因子1/4的倍数。如下图：



**4. 某些图片和PC端的计算结果还是有比较大的出入的。**

**20190508 对比结果txt的预处理**

**对比结果预处理程序**

**1. 文件源代码**

/home/lee/Documents/Project/c\_excise/compare\_BSD\_output：

#include <stdio.h>

#include <string.h>

#define IMG\_CNT 738

int main(void)

{

FILE\* fzcu=NULL;

FILE\* fpc=NULL;

FILE\* fout=NULL;

fzcu = fopen("bsd\_Outputs.txt", "r+");

if(fzcu == NULL)

{

printf("open fzcu error!\n");

return 0;

}

fpc = fopen("layerOutputs\_imgs\_noBiasNetwork\_copy.txt", "r+");

if(fpc == NULL)

{

printf("open layer error!\n");

return 0;

}

fout = fopen("results.txt", "w+");

if(fout == NULL)

{

printf("open fout error!\n");

return 0;

}

int i=0, j=0; //loop param

char fpcName[100] = {0}; //array

char fzcuName[100] = {0};//array

char\* pcname = fpcName; //pointer

char\* pzcuname = fzcuName; //pointer

char stringcmp = 0; //used to compare the image name string

//read image names:

for(i=0; i<IMG\_CNT; ++i)

{

fgets(pcname, 40, fpc);

printf("%s", pcname);

fseek(fzcu, 0, SEEK\_SET);//move FILE\* to the beginning.

for(j=0; j<IMG\_CNT; ++j)

{

//fseek(fzcu, 0, SEEK\_SET);//move the FILE\* to the head of file.

fgets(pzcuname, 40, (FILE\*)fzcu);//travese all image names.

if( !strcmp(pcname, pzcuname) )

{

fputs(pzcuname, (FILE\*)fout);//write name tofout.

fgets(pzcuname, 60, (FILE\*)fzcu);//read ZCU data.

fputs(pzcuname, (FILE\*)fout);//write ZCU data to fout.

fgets(pcname, 60, (FILE\*)fpc);//read PC data

fputs(pcname, (FILE\*)fout); //write PC data to fout.

printf("zcu line:%d \n", j);

break;

}

else

{

fgets(pzcuname, 60, (FILE\*)fzcu);//read ZCU data for next name string.

}

}

printf("\n");

if(j >= IMG\_CNT)

{

printf("Didn't find %s !\n", pcname);

}

//fgets(pcname, 100, (FILE\*)fpc);//read PC data for read next name string.

}

return 0;

}

生成需要对比的结果文件：

Encode\_V1129\_6\_017610.png

[[0.004 0.005 0.003 0.003 0.002 0.003 0.005 0.005]]

[[0.005 0.008 0.002 0.005 0.002 0.005 0.006 0.011]]

Encode\_V1129\_5\_025891.png

[[0.060 0.119 0.119 0.009 0.007 0.119 0.818 0.905]]

[[0.002 0.522 0.020 0.007 0.010 0.021 0.938 0.957]]

Encode\_V1129\_22\_004613.png

[[0.047 0.014 0.881 0.060 0.119 0.940 0.905 1.000]]

[[0.053 0.099 0.682 0.516 0.002 0.767 0.843 0.999]]

Encode\_V1129\_15\_006230.png

[[0.953 0.047 0.622 0.001 0.047 0.977 0.679 0.999]]

[[1.000 0.049 0.860 0.001 0.004 1.000 0.961 1.000]]

Encode\_V1129\_21\_011905.png

[[0.982 0.002 0.005 0.076 0.001 0.998 0.004 0.998]]

[[1.000 0.017 0.009 0.015 0.001 0.999 0.029 0.999]]

Encode\_V1129\_6\_036049.png

[[0.001 0.000 0.095 0.009 0.005 0.007 0.047 0.119]]

[[0.002 0.001 0.073 0.005 0.002 0.003 0.061 0.066]]

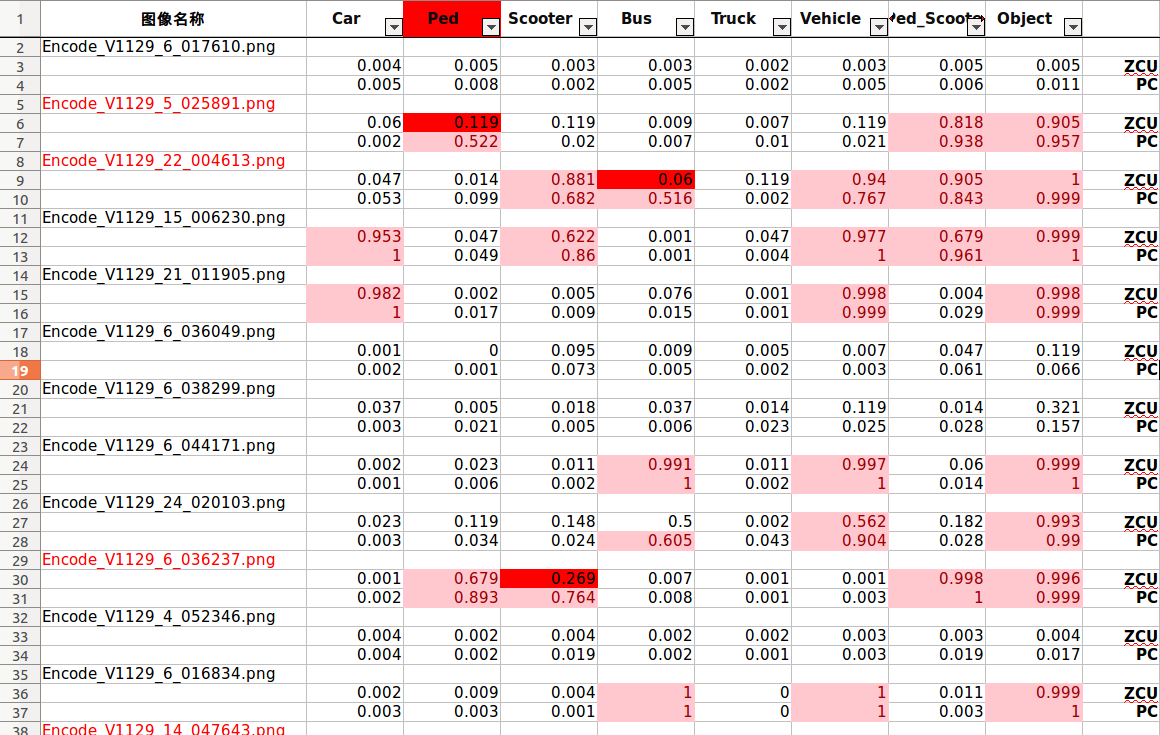
Encode\_V1129\_6\_038299.png

[[0.037 0.005 0.018 0.037 0.014 0.119 0.014 0.321]]

[[0.003 0.021 0.005 0.006 0.023 0.025 0.028 0.157]]

**20190514 对比结果准确率&如何提高**

/home/lee/Documents/Project/c\_excise/compare\_BSD\_output/CompareBSDResults.xlsx

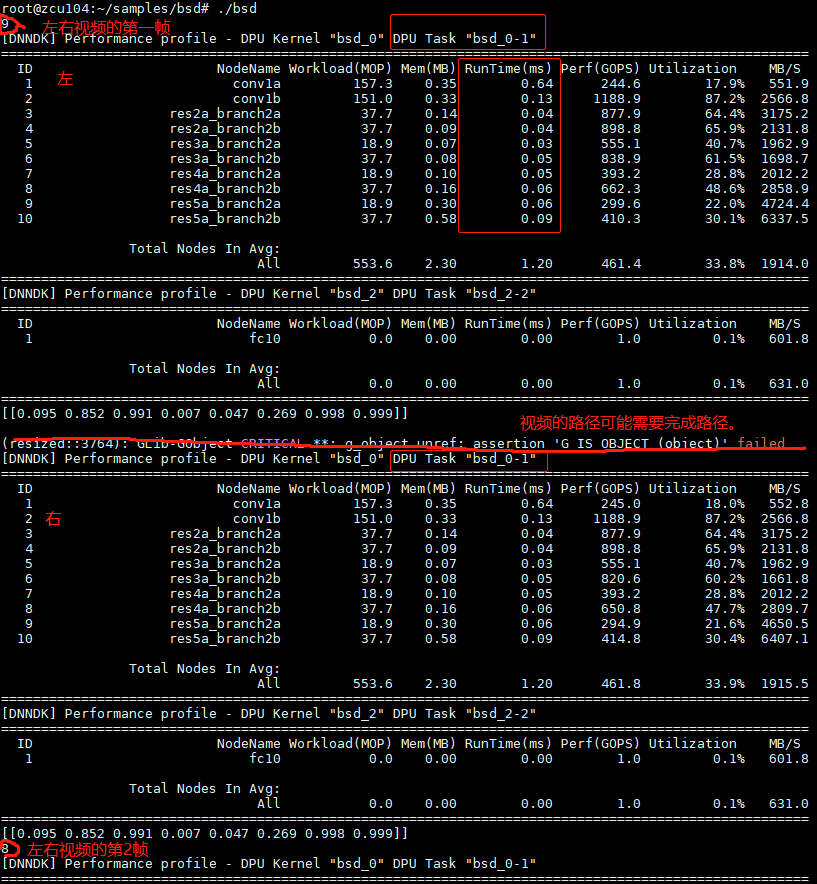


不同结果使用红色标出，测试的738张图片中，有158张不同结果，其他的结果基本近似相等。以PC结果无标准，那么158/738大概是80%的正确率。

1). **如果提高准确率问题：**

* 优化网络结构
* 使用更全路况的代表图片进行calibration，因为使用不全的话会影响网络判断，偏向于训练的数据集，更正确。

2). 网络运行的时间&帧率



3). 还需要验证使用摄像头采集的真实道路录像，运行BSD网络的结果。

验证是否达到一定的准确率。

**20190517 双路视频运行BSD网络**

学习opencv的videoCapture类的知识，以及相关的成员函数。

**20190610 读入1280x720的mp4视频**

从amandeep那里copy了Encode\_V1129\_167\_1.mp4，然后使用评估BSD的程序，读取视频，然后显示。出现了十分卡顿的现象。

测试了capture.read和imshow的执行时间，大概时间参考E:\1.Soterea\Project\@Proj03.XLINX\_ZCU104。

的《OpenCV读取TDA2x录像\_读取视频非常卡顿.docx》，如果删除了cvWaitKey（）那么执行的速度就很快了，不知道是什么原因？？

使用另外一个视频，测试还是很卡顿。