Edge AI Tutorials

YOLOv3 Tutorial: Darknet to Caffe to Xilinx DNNDK

Introduction

YOLOv3 is one of the most famous CNN for Object Detection. It was developed in a ML framework different from Caffe which is named Darknet. To run it on the Xilinx® DNNDK release, you need to convert it into a format compliant with Caffe. For this, you will need a special converter to convert Darknet to Caffe and generate the `yolov3.prototxt` and `yolov3.caffemodel` files as input to the DNNDK.

This Tutorial describes the process of converting the YOLOv3 CNN (originally trained in Darknet with the COCO dataset (80 classes)) before quantizing it with Xilinx DNNDK 2.0.8 release and run on a ZCU102 target board.

The conversion from Darknet to Caffe supports **YOLOv2/tiny**, **YOLOv2**, **YOLOv3/tiny**, and **YOLOv3** basic networks. This conversion relies on a special fork of Caffe (designed by DeePhi), which is placed in the caffe-master folder.

Prerequisites

1. Ubuntu OS 14.04 or 16.04. For more information, see chapter 1 in the DNNDK User Guide UG1327.
2. DNNDK tools and image for evaluation boards (zcu102 used in this example). For more information, see Xilinx AI Developer Hub.
3. Python 2.7 and its virtual environments for Ubuntu OS.
4. The official YOLOv3-608 network model trained with COCO dataset is available here. Download the `yolov3.weights` file (around 248 MB) and place it in the `0_model_darknet` folder.

Project Organization

Assuming the working directory is named `yolo_convertor` in the folder `$ML_DIR`, the project is organized with the following directory structure:

```
yolo_convertor
.
├── caffe-master.tar.gz
├── darknet_origin.tar.gz
├── example_yolov3
```

```
│     ├── 0_convert.sh
│     ├── 0_model_darknet
│     │     ├── yolov3.cfg
│     ├── 0_test_darknet.sh
│     ├── 1_model_caffe
│     │     └── v3.prototxt
│     ├── 1_test_caffe.sh
│     ├── 2_model_for_quantize
│     │     ├── v3_example.prototxt
│     ├── 2_quantize.sh
│     ├── 3_compile.sh
│     ├── 3_model_after_quantize
│     │     └── deploy.prototxt
│     ├── 4_model_elf
│     │     ├── yolo_kernel_graph.jpg
│     │     └── yolo_kernel.info
│     ├── 5_file_for_test
│     │     ├── calib_data.tar
│     │     ├── calib.txt
│     │     ├── coco.data
│     │     ├── coco.names
│     │     ├── image.txt
│     │     └── test.jpg
│     └── results
├── images
├── README.md
├── yolo_convert.py
└── yolov3_deploy.tar.gz.partaaa
└── yolov3_deploy.tar.gz.partaab
```

Preparing the Repository

The tutorial.sh script sets the `$ML_DIR` variable to your working directory (for example `/home/root2/ML/YOLOv3/yolov3_convertor` is shown in all the examples in this tutorial).

```
cd <YOUR_WORKING_DIR>/yolov3_convertor
bash -v tutorial.sh
```

The tutorial.sh script also performs the following actions:

- Uncompresses all the `*.tar.gz` files in the repository, by executing the following commands:

```
$ tar -xvf caffe-master.tar.gz
$ tar -xvf darknet_origin.tar.gz
$ cat yolov3_deploy.tar.gz.part* > yolov3_deploy.tar.gz
$ tar -xvf yolov3_deploy.tar.gz
$ rm yolov3_deploy.tar.gz.part*
$ cd example_yolov3/5_file_for_test
```

```
$ tar -xvf calib_data.tar
$ cd ../../
```

- Runs the following commands from the working directory:

```
$ find . -type f -name "*.txt"   -print0 | xargs -0 dos2unix
$ find . -type f -name "*.data"  -print0 | xargs -0 dos2unix
$ find . -type f -name "*.cfg"   -print0 | xargs -0 dos2unix
$ find . -type f -name "*.names" -print0 | xargs -0 dos2unix
```

You will need to have the dos2unix utility installed in your Linux PC before executing them.

- Sets the path in the coco.data file in the 5_file_for_test folder, according to the following (`PATH_TO` depends on your environment):

```
valid = /PATH_TO/example_yolov3/5_file_for_test/image.txt
names = /PATH_TO/example_yolov3/5_file_for_test/coco.names
```

- Sets the test images path in the image.txt file in the 5_file_for_test folder, according to the following:

```
PATH_TO/example_yolov3/5_file_for_test/xxx.jpg
```

- Sets the Caffe python interface path in the second line of the yolo_convert.py script as in the following (`PATH_TO` depends on your environment):

```
sys.path.append('/PATH_TO/yolo_convertor/caffe-master/python')
```

Processing Flow

Starting from a YOLOv3 CNN trained directly in Darknet with the COCO dataset, in this tutorial you will adopt the following flow:

1. Convert the Darknet model into a Caffe model using the 0_convert.sh script.
2. Test the object detection behavior of either the original Darkenet or the Caffe model with the 0_test_darknet.sh and 1_test_caffe.sh scripts respectively.
3. Quantize the Caffe model generated in the previous step with the DNNDK decent tool by launching the 2_quantize.sh script.
4. Compile the ELF file for the DPU IP core on the ZCU102 target board with the 3_compile.sh script.

5. Build the final application and deploy it on the ZCU102 board. It is archived in the yolov3_deploy.tar.gz file, for your refererence.

Compile Darknet and Caffe

Use the following commands to compile Darknet in the darknet_origin folder:

```
$ cd darknet_origin
$ make -j
$ cd ..
```

Use the following commands to compile Caffe in the caffe-master folder:

> **Note**: Do this from your Python virtual environment.

```
$ cd caffe-master
$ make -j
$ make pycaffe
$ make distribute
$ cd ..
```

You will use this Caffe fork to convert the model from Darkent to Caffe, but you do not need it for training(as the training was already done in the Darknet framework). Therefore, be sure to set the compilation for CPU only in the Makefile.config file , as illustrated in the following lines:

```
# cuDNN acceleration switch (uncomment to build with cuDNN).
#USE_CUDNN := 1
# CPU-only switch (uncomment to build without GPU support).
CPU_ONLY := 1
```

Now, use the following commands to set the `CAFFE_ROOT` environmental variable to the folder of the newly installed Caffe and update the other variables consequently.

```
export CAFFE_ROOT=~/ML/YOLOv3/yolo_converter/caffe-master
export LD_LIBRARY_PATH=$CAFFE_ROOT/distribute/lib:$LD_LIBRARY_PATH
export PYTHONPATH=$CAFFE_ROOT/distribute/python:/usr/local/lib/python2.7/dist-
packages/numpy/core/include/:$PYTHONPATH
```

To test the environment, execute the following command line:

```
$ python -c "import caffe; print caffe.__file__"
```

You will see commands as shown in the following figure:

```
(bvlc1v0_py27) root2@Prec5820Tow:~/ML/YOLOv3/yolo_convertor/caffe-master$
(bvlc1v0_py27) root2@Prec5820Tow:~/ML/YOLOv3/yolo_convertor/caffe-master$ python -c "import caffe; print caffe.__file__"
/home/root2/ML/YOLOv3/yolo_convertor/caffe-master/distribute/python/caffe/__init__.pyc
(bvlc1v0_py27) root2@Prec5820Tow:~/ML/YOLOv3/yolo_convertor/caffe-master$
```

The YOLOv3 Example

This sections details the flow described in 'The Processing Flow' section, using the YOLOv3 CNN as an example. The files are placed in the example_yolov3 folder, which is organized as shown below:

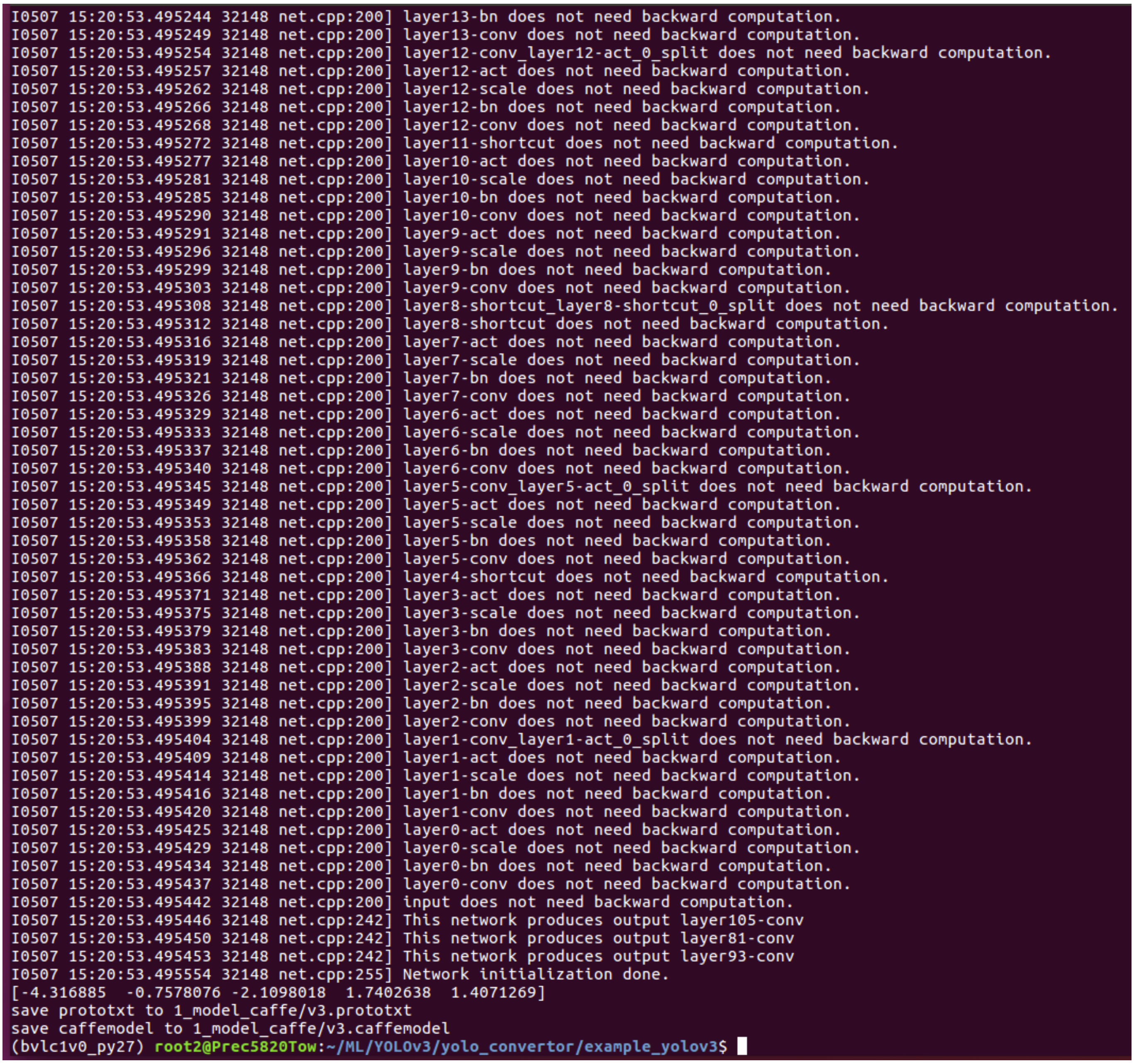
```
example_yolov3/
├── 0_convert.sh
├── 0_model_darknet
│ ├── yolov3.cfg
│ └── yolov3.weights
├── 0_test_darknet.sh
├── 1_model_caffe
│ ├── v3.caffemodel
│ └── v3.prototxt
├── 1_test_caffe.sh
├── 2_model_for_quantize
│ ├── v3.caffemodel
│ ├── v3_example.prototxt
│ └── v3.prototxt
├── 2_quantize.sh
├── 3_compile.sh
├── 3_model_after_quantize
│ ├── deploy.caffemodel
│ └── deploy.prototxt
├── 4_model_elf
│ ├── dpu_yolo.elf
│ ├── yolo_kernel_graph.jpg
│ └── yolo_kernel.info
├── 5_file_for_test
│ ├── calib_data
│ ├── calib.txt
│ ├── coco.data
│ ├── coco.names
│ ├── image.txt
│ ├── test.jpg
│ ├── yolov3_caffe_result.txt
│ └── yolov3_darknet_result.txt
├── detection.jpg
├── results
```

Step 1: Darknet to Caffe Model Conversion

Use the following commands to launch the Darkent to Caffe conversion process:

```
$ cd example_yolov3
$ bash 0_convert.sh
```

The output is shown in the following figure:



```
I0507 15:20:53.495244 32148 net.cpp:200] layer13-bn does not need backward computation.
I0507 15:20:53.495249 32148 net.cpp:200] layer13-conv does not need backward computation.
I0507 15:20:53.495254 32148 net.cpp:200] layer12-conv_layer12-act_0_split does not need backward computation.
I0507 15:20:53.495257 32148 net.cpp:200] layer12-act does not need backward computation.
I0507 15:20:53.495262 32148 net.cpp:200] layer12-scale does not need backward computation.
I0507 15:20:53.495266 32148 net.cpp:200] layer12-bn does not need backward computation.
I0507 15:20:53.495268 32148 net.cpp:200] layer12-conv does not need backward computation.
I0507 15:20:53.495272 32148 net.cpp:200] layer11-shortcut does not need backward computation.
I0507 15:20:53.495277 32148 net.cpp:200] layer10-act does not need backward computation.
I0507 15:20:53.495281 32148 net.cpp:200] layer10-scale does not need backward computation.
I0507 15:20:53.495285 32148 net.cpp:200] layer10-bn does not need backward computation.
I0507 15:20:53.495290 32148 net.cpp:200] layer10-conv does not need backward computation.
I0507 15:20:53.495291 32148 net.cpp:200] layer9-act does not need backward computation.
I0507 15:20:53.495296 32148 net.cpp:200] layer9-scale does not need backward computation.
I0507 15:20:53.495299 32148 net.cpp:200] layer9-bn does not need backward computation.
I0507 15:20:53.495303 32148 net.cpp:200] layer9-conv does not need backward computation.
I0507 15:20:53.495308 32148 net.cpp:200] layer8-shortcut_layer8-shortcut_0_split does not need backward computation.
I0507 15:20:53.495312 32148 net.cpp:200] layer8-shortcut does not need backward computation.
I0507 15:20:53.495316 32148 net.cpp:200] layer7-act does not need backward computation.
I0507 15:20:53.495319 32148 net.cpp:200] layer7-scale does not need backward computation.
I0507 15:20:53.495321 32148 net.cpp:200] layer7-bn does not need backward computation.
I0507 15:20:53.495326 32148 net.cpp:200] layer7-conv does not need backward computation.
I0507 15:20:53.495329 32148 net.cpp:200] layer6-act does not need backward computation.
I0507 15:20:53.495333 32148 net.cpp:200] layer6-scale does not need backward computation.
I0507 15:20:53.495337 32148 net.cpp:200] layer6-bn does not need backward computation.
I0507 15:20:53.495340 32148 net.cpp:200] layer6-conv does not need backward computation.
I0507 15:20:53.495345 32148 net.cpp:200] layer5-conv_layer5-act_0_split does not need backward computation.
I0507 15:20:53.495349 32148 net.cpp:200] layer5-act does not need backward computation.
I0507 15:20:53.495353 32148 net.cpp:200] layer5-scale does not need backward computation.
I0507 15:20:53.495358 32148 net.cpp:200] layer5-bn does not need backward computation.
I0507 15:20:53.495362 32148 net.cpp:200] layer5-conv does not need backward computation.
I0507 15:20:53.495366 32148 net.cpp:200] layer4-shortcut does not need backward computation.
I0507 15:20:53.495371 32148 net.cpp:200] layer3-act does not need backward computation.
I0507 15:20:53.495375 32148 net.cpp:200] layer3-scale does not need backward computation.
I0507 15:20:53.495379 32148 net.cpp:200] layer3-bn does not need backward computation.
I0507 15:20:53.495383 32148 net.cpp:200] layer3-conv does not need backward computation.
I0507 15:20:53.495388 32148 net.cpp:200] layer2-act does not need backward computation.
I0507 15:20:53.495391 32148 net.cpp:200] layer2-scale does not need backward computation.
I0507 15:20:53.495395 32148 net.cpp:200] layer2-bn does not need backward computation.
I0507 15:20:53.495399 32148 net.cpp:200] layer2-conv does not need backward computation.
I0507 15:20:53.495404 32148 net.cpp:200] layer1-conv_layer1-act_0_split does not need backward computation.
I0507 15:20:53.495409 32148 net.cpp:200] layer1-act does not need backward computation.
I0507 15:20:53.495414 32148 net.cpp:200] layer1-scale does not need backward computation.
I0507 15:20:53.495416 32148 net.cpp:200] layer1-bn does not need backward computation.
I0507 15:20:53.495420 32148 net.cpp:200] layer1-conv does not need backward computation.
I0507 15:20:53.495425 32148 net.cpp:200] layer0-act does not need backward computation.
I0507 15:20:53.495429 32148 net.cpp:200] layer0-scale does not need backward computation.
I0507 15:20:53.495434 32148 net.cpp:200] layer0-bn does not need backward computation.
I0507 15:20:53.495437 32148 net.cpp:200] layer0-conv does not need backward computation.
I0507 15:20:53.495442 32148 net.cpp:200] input does not need backward computation.
I0507 15:20:53.495446 32148 net.cpp:242] This network produces output layer105-conv
I0507 15:20:53.495450 32148 net.cpp:242] This network produces output layer81-conv
I0507 15:20:53.495453 32148 net.cpp:242] This network produces output layer93-conv
I0507 15:20:53.495554 32148 net.cpp:255] Network initialization done.
[-4.316885  -0.7578076 -2.1098018  1.7402638  1.4071269]
save prototxt to 1_model_caffe/v3.prototxt
save caffemodel to 1_model_caffe/v3.caffemodel
(bvlc1v0_py27) root2@Prec5820Tow:~/ML/YOLOv3/yolo_convertor/example_yolov3$
```

The 0_convert.sh script converts the Darknet model(stored in the 0_model_darknet folder), to a Caffe model (stored in the 1_model_caffe folder), using the yolo_convert.py script.

The 0_convert.sh script performs the following actions to generate the two Caffe files, v3.prototxt and v3.caffemodel:

```
$ python ../yolo_convert.py \
       0_model_darknet/yolov3.cfg        #path to Darknet cfg file \
       0_model_darknet/yolov3.weights    #path to Darknet weights file \
       1_model_caffe/v3.prototxt         #path to Caffe prototxt file \
       1_model_caffe/v3.caffemodel       #path to Caffe caffemodel file
```

Step 2: Test the Darknet and Caffe YOLOv3 models

The files that are required to test both the Darknet and Caffe models are placed in the 5_file_for_test folder. Ensure to delete all the `*result*.txt` files in the repository

Test Darknet

Execute the following commands to test Darknet:

```
$ cd example_yolov3
rm results/*
rm 5_file_for_test/yolov3_*_result.txt
$ bash 0_test_darknet.sh
```

You will see the following:

```
   58 res   55                 38 x  38 x 512   ->   38 x  38 x 512
   59 conv    256  1 x 1 / 1   38 x  38 x 512   ->   38 x  38 x 256  0.379 BFLOPs
   60 conv    512  3 x 3 / 1   38 x  38 x 256   ->   38 x  38 x 512  3.407 BFLOPs
   61 res   58                 38 x  38 x 512   ->   38 x  38 x 512
   62 conv   1024  3 x 3 / 2   38 x  38 x 512   ->   19 x  19 x1024  3.407 BFLOPs
   63 conv    512  1 x 1 / 1   19 x  19 x1024   ->   19 x  19 x 512  0.379 BFLOPs
   64 conv   1024  3 x 3 / 1   19 x  19 x 512   ->   19 x  19 x1024  3.407 BFLOPs
   65 res   62                 19 x  19 x1024   ->   19 x  19 x1024
   66 conv    512  1 x 1 / 1   19 x  19 x1024   ->   19 x  19 x 512  0.379 BFLOPs
   67 conv   1024  3 x 3 / 1   19 x  19 x 512   ->   19 x  19 x1024  3.407 BFLOPs
   68 res   65                 19 x  19 x1024   ->   19 x  19 x1024
   69 conv    512  1 x 1 / 1   19 x  19 x1024   ->   19 x  19 x 512  0.379 BFLOPs
   70 conv   1024  3 x 3 / 1   19 x  19 x 512   ->   19 x  19 x1024  3.407 BFLOPs
   71 res   68                 19 x  19 x1024   ->   19 x  19 x1024
   72 conv    512  1 x 1 / 1   19 x  19 x1024   ->   19 x  19 x 512  0.379 BFLOPs
   73 conv   1024  3 x 3 / 1   19 x  19 x 512   ->   19 x  19 x1024  3.407 BFLOPs
   74 res   71                 19 x  19 x1024   ->   19 x  19 x1024
   75 conv    512  1 x 1 / 1   19 x  19 x1024   ->   19 x  19 x 512  0.379 BFLOPs
   76 conv   1024  3 x 3 / 1   19 x  19 x 512   ->   19 x  19 x1024  3.407 BFLOPs
   77 conv    512  1 x 1 / 1   19 x  19 x1024   ->   19 x  19 x 512  0.379 BFLOPs
   78 conv   1024  3 x 3 / 1   19 x  19 x 512   ->   19 x  19 x1024  3.407 BFLOPs
   79 conv    512  1 x 1 / 1   19 x  19 x1024   ->   19 x  19 x 512  0.379 BFLOPs
   80 conv   1024  3 x 3 / 1   19 x  19 x 512   ->   19 x  19 x1024  3.407 BFLOPs
   81 conv    255  1 x 1 / 1   19 x  19 x1024   ->   19 x  19 x 255  0.189 BFLOPs
   82 detection
   83 route  79
   84 conv    256  1 x 1 / 1   19 x  19 x 512   ->   19 x  19 x 256  0.095 BFLOPs
   85 upsample            2x   19 x  19 x 256   ->   38 x  38 x 256
   86 route  85 61
   87 conv    256  1 x 1 / 1   38 x  38 x 768   ->   38 x  38 x 256  0.568 BFLOPs
   88 conv    512  3 x 3 / 1   38 x  38 x 256   ->   38 x  38 x 512  3.407 BFLOPs
   89 conv    256  1 x 1 / 1   38 x  38 x 512   ->   38 x  38 x 256  0.379 BFLOPs
   90 conv    512  3 x 3 / 1   38 x  38 x 256   ->   38 x  38 x 512  3.407 BFLOPs
   91 conv    256  1 x 1 / 1   38 x  38 x 512   ->   38 x  38 x 256  0.379 BFLOPs
   92 conv    512  3 x 3 / 1   38 x  38 x 256   ->   38 x  38 x 512  3.407 BFLOPs
   93 conv    255  1 x 1 / 1   38 x  38 x 512   ->   38 x  38 x 255  0.377 BFLOPs
   94 detection
   95 route  91
   96 conv    128  1 x 1 / 1   38 x  38 x 256   ->   38 x  38 x 128  0.095 BFLOPs
   97 upsample            2x   38 x  38 x 128   ->   76 x  76 x 128
   98 route  97 36
   99 conv    128  1 x 1 / 1   76 x  76 x 384   ->   76 x  76 x 128  0.568 BFLOPs
  100 conv    256  3 x 3 / 1   76 x  76 x 128   ->   76 x  76 x 256  3.407 BFLOPs
  101 conv    128  1 x 1 / 1   76 x  76 x 256   ->   76 x  76 x 128  0.379 BFLOPs
  102 conv    256  3 x 3 / 1   76 x  76 x 128   ->   76 x  76 x 256  3.407 BFLOPs
  103 conv    128  1 x 1 / 1   76 x  76 x 256   ->   76 x  76 x 128  0.379 BFLOPs
  104 conv    256  3 x 3 / 1   76 x  76 x 128   ->   76 x  76 x 256  3.407 BFLOPs
  105 conv    255  1 x 1 / 1   76 x  76 x 256   ->   76 x  76 x 255  0.754 BFLOPs
  106 detection
Loading weights from 0_model_darknet/yolov3.weights...Done!
Learning Rate: 0.001, Momentum: 0.9, Decay: 0.0005
eval: Using default 'voc'
1
Cannot load image ""
2
Total Detection Time: 0.120223 Seconds
cat results/yolov3_results_* >> 5_file_for_test/yolov3_darknet_result.txt
(bvlc1v0_py27) root2@Prec5820Tow:~/ML/YOLOv3/yolo_convertor/example_yolov3$
```

The 0_test_darknet.sh script uses the standard Darknet framework to get the detection result of the network, which is saved as yolov3_darknet_result.txt. The folder and file name in the script can be modified accordingly.

> **Note:** The confidence threshold will determine the number of bounding boxes that will be the output of the final detection result. If the threshold needs to be changed in testing the Darknet model, modify line 419 of the detector.c file, clean and re-make Darknet, and run `0_test_darknet.sh` again.

Test Caffe

Execute the following commands to test Caffe:

```
$ cd example_yolov3
$ bash 1_test_caffe.sh
```

You will see the following:

```
I0507 16:08:47.387763 33963 net.cpp:200] layer10-act does not need backward computation.
I0507 16:08:47.387766 33963 net.cpp:200] layer10-scale does not need backward computation.
I0507 16:08:47.387768 33963 net.cpp:200] layer10-bn does not need backward computation.
I0507 16:08:47.387773 33963 net.cpp:200] layer10-conv does not need backward computation.
I0507 16:08:47.387774 33963 net.cpp:200] layer9-act does not need backward computation.
I0507 16:08:47.387779 33963 net.cpp:200] layer9-scale does not need backward computation.
I0507 16:08:47.387784 33963 net.cpp:200] layer9-bn does not need backward computation.
I0507 16:08:47.387786 33963 net.cpp:200] layer9-conv does not need backward computation.
I0507 16:08:47.387790 33963 net.cpp:200] layer8-shortcut_layer8-shortcut_0_split does not need backward computation.
I0507 16:08:47.387795 33963 net.cpp:200] layer8-shortcut does not need backward computation.
I0507 16:08:47.387800 33963 net.cpp:200] layer7-act does not need backward computation.
I0507 16:08:47.387806 33963 net.cpp:200] layer7-scale does not need backward computation.
I0507 16:08:47.387810 33963 net.cpp:200] layer7-bn does not need backward computation.
I0507 16:08:47.387814 33963 net.cpp:200] layer7-conv does not need backward computation.
I0507 16:08:47.387817 33963 net.cpp:200] layer6-act does not need backward computation.
I0507 16:08:47.387821 33963 net.cpp:200] layer6-scale does not need backward computation.
I0507 16:08:47.387825 33963 net.cpp:200] layer6-bn does not need backward computation.
I0507 16:08:47.387830 33963 net.cpp:200] layer6-conv does not need backward computation.
I0507 16:08:47.387833 33963 net.cpp:200] layer5-conv_layer5-act_0_split does not need backward computation.
I0507 16:08:47.387837 33963 net.cpp:200] layer5-act does not need backward computation.
I0507 16:08:47.387841 33963 net.cpp:200] layer5-scale does not need backward computation.
I0507 16:08:47.387845 33963 net.cpp:200] layer5-bn does not need backward computation.
I0507 16:08:47.387850 33963 net.cpp:200] layer5-conv does not need backward computation.
I0507 16:08:47.387853 33963 net.cpp:200] layer4-shortcut does not need backward computation.
I0507 16:08:47.387857 33963 net.cpp:200] layer3-act does not need backward computation.
I0507 16:08:47.387861 33963 net.cpp:200] layer3-scale does not need backward computation.
I0507 16:08:47.387866 33963 net.cpp:200] layer3-bn does not need backward computation.
I0507 16:08:47.387869 33963 net.cpp:200] layer3-conv does not need backward computation.
I0507 16:08:47.387874 33963 net.cpp:200] layer2-act does not need backward computation.
I0507 16:08:47.387878 33963 net.cpp:200] layer2-scale does not need backward computation.
I0507 16:08:47.387881 33963 net.cpp:200] layer2-bn does not need backward computation.
I0507 16:08:47.387886 33963 net.cpp:200] layer2-conv does not need backward computation.
I0507 16:08:47.387889 33963 net.cpp:200] layer1-conv_layer1-act_0_split does not need backward computation.
I0507 16:08:47.387894 33963 net.cpp:200] layer1-act does not need backward computation.
I0507 16:08:47.387897 33963 net.cpp:200] layer1-scale does not need backward computation.
I0507 16:08:47.387900 33963 net.cpp:200] layer1-bn does not need backward computation.
I0507 16:08:47.387902 33963 net.cpp:200] layer1-conv does not need backward computation.
I0507 16:08:47.387907 33963 net.cpp:200] layer0-act does not need backward computation.
I0507 16:08:47.387910 33963 net.cpp:200] layer0-scale does not need backward computation.
I0507 16:08:47.387914 33963 net.cpp:200] layer0-bn does not need backward computation.
I0507 16:08:47.387918 33963 net.cpp:200] layer0-conv does not need backward computation.
I0507 16:08:47.387923 33963 net.cpp:200] input does not need backward computation.
I0507 16:08:47.387926 33963 net.cpp:242] This network produces output layer105-conv
I0507 16:08:47.387930 33963 net.cpp:242] This network produces output layer81-conv
I0507 16:08:47.387934 33963 net.cpp:242] This network produces output layer93-conv
I0507 16:08:47.388046 33963 net.cpp:255] Network initialization done.
I0507 16:08:47.514207 33963 upgrade_proto.cpp:79] Attempting to upgrade batch norm layers using deprecated params: 1_model_caffe/v3.caffemodel
I0507 16:08:47.514286 33963 upgrade_proto.cpp:82] Successfully upgraded batch norm layers using deprecated params.
I0507 16:08:54.271584 33963 yolov3_detect.cpp:257] /home/root2/ML/YOLOv3/yolo_convertor/example_yolov3/5_file_for_test/test.jpg 0 0.996799 42.0844 16.3985 353 485.351
I0507 16:08:54.271641 33963 yolov3_detect.cpp:257] /home/root2/ML/YOLOv3/yolo_convertor/example_yolov3/5_file_for_test/test.jpg 16 0.245967 48.6832 237.693 193.577 373.885
I0507 16:08:54.271667 33963 yolov3_detect.cpp:257] /home/root2/ML/YOLOv3/yolo_convertor/example_yolov3/5_file_for_test/test.jpg 16 0.0978759 1 108.419 353 490.407
I0507 16:08:54.271692 33963 yolov3_detect.cpp:257] /home/root2/ML/YOLOv3/yolo_convertor/example_yolov3/5_file_for_test/test.jpg 24 0.00668065 1 176.519 353 484.48
I0507 16:08:54.271733 33963 yolov3_detect.cpp:257] /home/root2/ML/YOLOv3/yolo_convertor/example_yolov3/5_file_for_test/test.jpg 56 0.105866 1 108.419 353 490.407
I0507 16:08:54.271771 33963 yolov3_detect.cpp:257] /home/root2/ML/YOLOv3/yolo_convertor/example_yolov3/5_file_for_test/test.jpg 56 0.00619625 14.7675 25.3903 350.165 324.561
I0507 16:08:54.271801 33963 yolov3_detect.cpp:257] /home/root2/ML/YOLOv3/yolo_convertor/example_yolov3/5_file_for_test/test.jpg 57 0.208174 4.07681 33.4884 353 500
I0507 16:08:54.271831 33963 yolov3_detect.cpp:257] /home/root2/ML/YOLOv3/yolo_convertor/example_yolov3/5_file_for_test/test.jpg 59 0.012626 4.07681 33.4884 353 500
```

The 1_test_caffe.sh script uses the functions added to the standard Caffe framework by DeePhi to get the detection result of the network after the conversion. The final detection result is saved in the 5_file_for_test folder as yolov3_caffe_result.txt. The path names, `classes`, and `anchorCnt` parameters in the script can be modified accordingly.

The following is the 1_test_caffe.sh script:

```
$ ../caffe-master/build/examples/yolo/yolov3_detect.bin \
                                     1_model_caffe/v3.prototxt         #path to
prototxt \
                                     1_model_caffe/v3.caffemodel       #path to
caffemodel \
                                     5_file_for_test/image.txt         #image.txt
specifies images for test \
                                     -confidence_threshold 0.005       #threshold
for confidence \
                                     -classes 80                       #class num
of network \
                                     -anchorCnt 3                      #anchor num
of network \
```

```
                                                      -out_file
5_file_for_test/yolov3_caffe_result.txt
```

If you plan to use the Darknet network model with different anchor parameters, modify line 134 of the `yolov3_detect.cpp` file according to anchor parameters in the `cfg` file. In the official YOLOv3 model, the anchor parameters of the Darknet model are the following:

```
anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,
```

Therefore, the corresponding values in `yolov3_detect.cpp` are:

```
float biases[18]= {10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198,
373,326};
```

> **Note:** To test the Caffe networks converted from **YOLOv2/tiny YOLOv2** basic networks, use `yolov2_detect.bin` placed in the Caffe build examples folder (`caffe-master/build/examples/yolo/`), and modify the related parameters in `yolov2_detect.cpp`.

> **Note:** To test the Caffe networks converted from **YOLOv3/tiny YOLOv3** basic networks, use `yolov3_detect.bin` placed in the Caffe build examples folder (`caffe-master/build/examples/yolo/`) and modify the related parameters in `yolov3_detect.cpp`.

Detection Result Comparison

The Darknet detection results for the input image `test.jpg` extracted from the yolov3_darknet_result.txt file (you might get the same results but in a different order), are as follows:

```
test 0.006225 14.738144 25.408401  350.169128 324
test 0.006644 1.000000  176.401611 353.000000 484
test 0.012753 4.063614  33.500977  353.000000 500
test 0.097317 1.000000  108.329865 353.000000 490
test 0.105670 1.000000  108.329865 353.000000 490
test 0.209107 4.063614  33.500977  353.000000 500
test 0.247878 48.710381 237.655457 193.563782 373
test 0.996792 42.086151 16.412552  353.000000 485
```

The Caffe detection results for the example image extracted from the yolov3_caffe_result.txt file (you might get the same results but in a different order), are as follows:

```
test.jpg 0.00619627 14.7674 25.3904 350.165 324
test.jpg 0.00668068 1       176.519 353     484
test.jpg 0.0126261  4.07681 33.4886 353     500
test.jpg 0.0978761  1       108.419 353     490
test.jpg 0.105866   1       108.419 353     490
```

```
test.jpg 0.208174   4.07681 33.4886 353     500
test.jpg 0.245967   48.6832 237.693 193.577 373
test.jpg 0.996799   42.0844 16.3986 353     485
```

The difference of confidences and bonding box coordinates between Darknet and Caffe model is negligible.

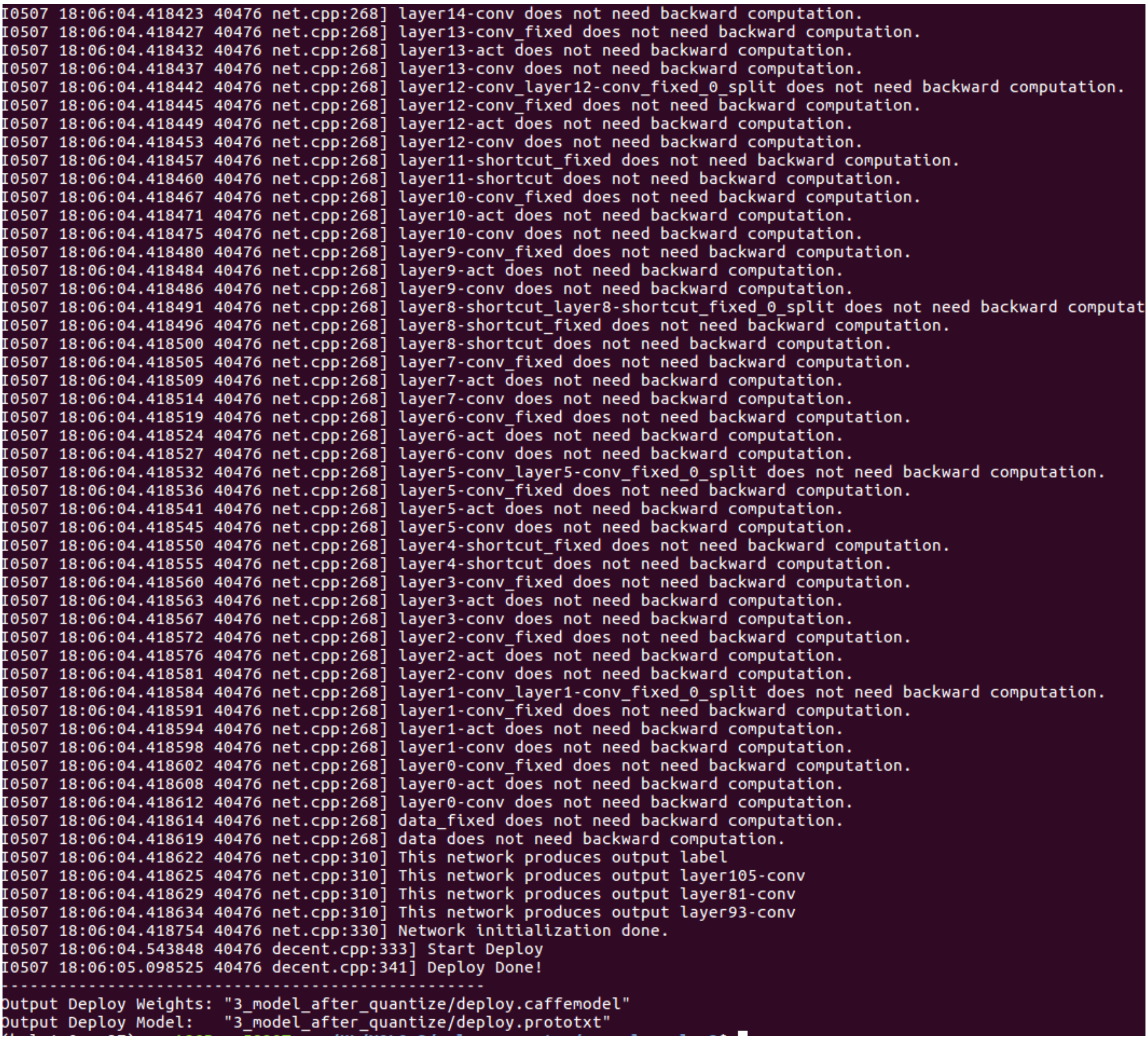
Step 3: Quantize the Caffe Model

Use the following commands to quantize the Caffe model:

```
$ cd example_yolov3
$ cp 1_model_caffe/v3.caffemodel  ./2_model_for_quantize/
$ bash 2_quantize.sh
```

You will see the following:

```
I0507 18:06:02.136471 40476 quantize.cu:225] calib start
I0507 18:06:02.146513 40476 quantize.cu:225] calib start
I0507 18:06:02.160681 40476 quantize.cu:225] calib start
I0507 18:06:02.163228 40476 decent.cpp:223] Calibration iter: 90/100 ,loss: 0
I0507 18:06:02.286618 40476 quantize.cu:225] calib start
I0507 18:06:02.296618 40476 quantize.cu:225] calib start
I0507 18:06:02.311795 40476 quantize.cu:225] calib start
I0507 18:06:02.315058 40476 decent.cpp:223] Calibration iter: 91/100 ,loss: 0
I0507 18:06:02.435952 40476 quantize.cu:225] calib start
I0507 18:06:02.446030 40476 quantize.cu:225] calib start
I0507 18:06:02.460167 40476 quantize.cu:225] calib start
I0507 18:06:02.462708 40476 decent.cpp:223] Calibration iter: 92/100 ,loss: 0
I0507 18:06:02.584262 40476 quantize.cu:225] calib start
I0507 18:06:02.594298 40476 quantize.cu:225] calib start
I0507 18:06:02.608292 40476 quantize.cu:225] calib start
I0507 18:06:02.610978 40476 decent.cpp:223] Calibration iter: 93/100 ,loss: 0
I0507 18:06:02.733706 40476 quantize.cu:225] calib start
I0507 18:06:02.743721 40476 quantize.cu:225] calib start
I0507 18:06:02.757822 40476 quantize.cu:225] calib start
I0507 18:06:02.760499 40476 decent.cpp:223] Calibration iter: 94/100 ,loss: 0
I0507 18:06:02.883291 40476 quantize.cu:225] calib start
I0507 18:06:02.893353 40476 quantize.cu:225] calib start
I0507 18:06:02.907459 40476 quantize.cu:225] calib start
I0507 18:06:02.910037 40476 decent.cpp:223] Calibration iter: 95/100 ,loss: 0
I0507 18:06:03.031909 40476 quantize.cu:225] calib start
I0507 18:06:03.042520 40476 quantize.cu:225] calib start
I0507 18:06:03.057039 40476 quantize.cu:225] calib start
I0507 18:06:03.059604 40476 decent.cpp:223] Calibration iter: 96/100 ,loss: 0
I0507 18:06:03.181257 40476 quantize.cu:225] calib start
I0507 18:06:03.191325 40476 quantize.cu:225] calib start
I0507 18:06:03.205544 40476 quantize.cu:225] calib start
I0507 18:06:03.208112 40476 decent.cpp:223] Calibration iter: 97/100 ,loss: 0
I0507 18:06:03.331638 40476 quantize.cu:225] calib start
I0507 18:06:03.341691 40476 quantize.cu:225] calib start
I0507 18:06:03.355825 40476 quantize.cu:225] calib start
I0507 18:06:03.358393 40476 decent.cpp:223] Calibration iter: 98/100 ,loss: 0
I0507 18:06:03.483387 40476 quantize.cu:225] calib start
I0507 18:06:03.493489 40476 quantize.cu:225] calib start
I0507 18:06:03.507766 40476 quantize.cu:225] calib start
I0507 18:06:03.510339 40476 decent.cpp:223] Calibration iter: 99/100 ,loss: 0
I0507 18:06:03.631850 40476 quantize.cu:225] calib start
I0507 18:06:03.641919 40476 quantize.cu:225] calib start
I0507 18:06:03.655931 40476 quantize.cu:225] calib start
I0507 18:06:03.658504 40476 decent.cpp:223] Calibration iter: 100/100 ,loss: 0
I0507 18:06:03.658516 40476 decent.cpp:228] Calibration Done!
I0507 18:06:04.037096 40476 net.cpp:98] Initializing net from parameters:
state {
  phase: TRAIN
}
layer {
  name: "data"
  type: "ImageData"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
```



```
I0507 18:06:04.418423 40476 net.cpp:268] layer14-conv does not need backward computation.
I0507 18:06:04.418427 40476 net.cpp:268] layer13-conv_fixed does not need backward computation.
I0507 18:06:04.418432 40476 net.cpp:268] layer13-act does not need backward computation.
I0507 18:06:04.418437 40476 net.cpp:268] layer13-conv does not need backward computation.
I0507 18:06:04.418442 40476 net.cpp:268] layer12-conv_layer12-conv_fixed_0_split does not need backward computation.
I0507 18:06:04.418445 40476 net.cpp:268] layer12-conv_fixed does not need backward computation.
I0507 18:06:04.418449 40476 net.cpp:268] layer12-act does not need backward computation.
I0507 18:06:04.418453 40476 net.cpp:268] layer12-conv does not need backward computation.
I0507 18:06:04.418457 40476 net.cpp:268] layer11-shortcut_fixed does not need backward computation.
I0507 18:06:04.418460 40476 net.cpp:268] layer11-shortcut does not need backward computation.
I0507 18:06:04.418467 40476 net.cpp:268] layer10-conv_fixed does not need backward computation.
I0507 18:06:04.418471 40476 net.cpp:268] layer10-act does not need backward computation.
I0507 18:06:04.418475 40476 net.cpp:268] layer10-conv does not need backward computation.
I0507 18:06:04.418480 40476 net.cpp:268] layer9-conv_fixed does not need backward computation.
I0507 18:06:04.418484 40476 net.cpp:268] layer9-act does not need backward computation.
I0507 18:06:04.418486 40476 net.cpp:268] layer9-conv does not need backward computation.
I0507 18:06:04.418491 40476 net.cpp:268] layer8-shortcut_layer8-shortcut_fixed_0_split does not need backward computat
I0507 18:06:04.418496 40476 net.cpp:268] layer8-shortcut_fixed does not need backward computation.
I0507 18:06:04.418500 40476 net.cpp:268] layer8-shortcut does not need backward computation.
I0507 18:06:04.418505 40476 net.cpp:268] layer7-conv_fixed does not need backward computation.
I0507 18:06:04.418509 40476 net.cpp:268] layer7-act does not need backward computation.
I0507 18:06:04.418514 40476 net.cpp:268] layer7-conv does not need backward computation.
I0507 18:06:04.418519 40476 net.cpp:268] layer6-conv_fixed does not need backward computation.
I0507 18:06:04.418524 40476 net.cpp:268] layer6-act does not need backward computation.
I0507 18:06:04.418527 40476 net.cpp:268] layer6-conv does not need backward computation.
I0507 18:06:04.418532 40476 net.cpp:268] layer5-conv_layer5-conv_fixed_0_split does not need backward computation.
I0507 18:06:04.418536 40476 net.cpp:268] layer5-conv_fixed does not need backward computation.
I0507 18:06:04.418541 40476 net.cpp:268] layer5-act does not need backward computation.
I0507 18:06:04.418545 40476 net.cpp:268] layer5-conv does not need backward computation.
I0507 18:06:04.418550 40476 net.cpp:268] layer4-shortcut_fixed does not need backward computation.
I0507 18:06:04.418555 40476 net.cpp:268] layer4-shortcut does not need backward computation.
I0507 18:06:04.418560 40476 net.cpp:268] layer3-conv_fixed does not need backward computation.
I0507 18:06:04.418563 40476 net.cpp:268] layer3-act does not need backward computation.
I0507 18:06:04.418567 40476 net.cpp:268] layer3-conv does not need backward computation.
I0507 18:06:04.418572 40476 net.cpp:268] layer2-conv_fixed does not need backward computation.
I0507 18:06:04.418576 40476 net.cpp:268] layer2-act does not need backward computation.
I0507 18:06:04.418581 40476 net.cpp:268] layer2-conv does not need backward computation.
I0507 18:06:04.418584 40476 net.cpp:268] layer1-conv_layer1-conv_fixed_0_split does not need backward computation.
I0507 18:06:04.418591 40476 net.cpp:268] layer1-conv_fixed does not need backward computation.
I0507 18:06:04.418594 40476 net.cpp:268] layer1-act does not need backward computation.
I0507 18:06:04.418598 40476 net.cpp:268] layer1-conv does not need backward computation.
I0507 18:06:04.418602 40476 net.cpp:268] layer0-conv_fixed does not need backward computation.
I0507 18:06:04.418608 40476 net.cpp:268] layer0-act does not need backward computation.
I0507 18:06:04.418612 40476 net.cpp:268] layer0-conv does not need backward computation.
I0507 18:06:04.418614 40476 net.cpp:268] data_fixed does not need backward computation.
I0507 18:06:04.418619 40476 net.cpp:268] data does not need backward computation.
I0507 18:06:04.418622 40476 net.cpp:310] This network produces output label
I0507 18:06:04.418625 40476 net.cpp:310] This network produces output layer105-conv
I0507 18:06:04.418629 40476 net.cpp:310] This network produces output layer81-conv
I0507 18:06:04.418634 40476 net.cpp:310] This network produces output layer93-conv
I0507 18:06:04.418754 40476 net.cpp:330] Network initialization done.
I0507 18:06:04.543848 40476 decent.cpp:333] Start Deploy
I0507 18:06:05.098525 40476 decent.cpp:341] Deploy Done!
--------------------------------------------------
Output Deploy Weights: "3_model_after_quantize/deploy.caffemodel"
Output Deploy Model:   "3_model_after_quantize/deploy.prototxt"
```

> **Note:** It is normal to see Loss values of 0 in the calibration phase, because loss layers are not included in the converted Caffe model.

In order to quantize the converted Caffe network, copy the v3.prototxt and v3.caffemodel files from the 1_model_caffe folder to the 2_model_for_quantize folder. Then, modify the `v3.prototxt` file by:

- Commenting out the first five lines
- Adding an **ImageData** layer with the calibration images for the train phase as shown in the following fragment:

```
name: "Darkent2Caffe"
#####Comment following five lines generated by converter#####
#input: "data"
#input_dim: 1
#input_dim: 3
#input_dim: 608
#input_dim: 608
#####Change input data layer to ImageDate and modify root_folder/source before run
DECENT#####
```

```
layer {
name: "data"
type: "ImageData"
top: "data"
top: "label"
include {
  phase: TRAIN
}
transform_param {
  mirror: false
  yolo_height:608    #change height according to Darknet model
  yolo_width:608     #change width  according to Darknet model
}
image_data_param {
  source:"/PATH_TO/5_file_for_test/calib.txt"         #change path accordingly
  root_folder:"/PATH_TO/5_file_for_test/calib_data/"  #change path accordingly
  batch_size: 1
  shuffle: false
}
}
##### No changes after below layers#####
```

For your convenience, the `v3_example.prototxt` file illustrates the changes you should do on the original `v3.prototxt` file.

The format of `calib.txt` used in the calibration phase of DNNDK `decent` is as follows:

```
#image_name                          fake_label_number
COCO_train2014_000000000009.jpg 1
COCO_train2014_000000000025.jpg 1
COCO_train2014_000000000030.jpg 1
COCO_train2014_000000000034.jpg 1
COCO_train2014_000000000036.jpg 1
```

> **Note:** The label number is not used in the calibration process.

The `5_file_for_test/calib_data` folder contains some images from the COCO dataset, to be used for the calibration process. The `2_quantize.sh` script performs the following actions:

```
#Assuming "decent" tool is already in the PATH
$ decent quantize -model 2_model_for_quantize/v3.prototxt        #path to prototxt
\
                -weights 2_model_for_quantize/v3.caffemodel    #path to caffemodel
\
                -gpu 0 \
                -sigmoided_layers layer81-conv,layer93-conv,layer105-conv \
                -output_dir 3_model_after_quantize \
                -method 1
```

In the YOLOv3 network, the `conv` layer before the `yolo` layer (that is, the output layer in Caffe model) will be quantized with the `-sigmoided_layers` flag for better accuracy. For more information, use the `decent –help` command or see the DNNDK User Guide UG1327.

Step 4: Compile the Quantized Model

Use the following commands to compile the ELF file:

```
$ cd example_yolov3
$ cp 3_model_after_quantize/ref_deploy.prototxt
3_model_after_quantize/deploy.prototxt
$ bash 3_compile.sh
```

You will see the following:

```
(bvlc1v0_py27) root2@Prec5820Tow:~/ML/YOLOv3/yolo_convertor/example_yolov3$ bash -v 3_compile.sh
#Assume the dnnc-dpu1.3.0 is installed in /usr/local/bin

dnnc-dpu1.3.0 --prototxt=3_model_after_quantize/deploy.prototxt \
              --caffemodel=3_model_after_quantize/deploy.caffemodel \
              --dpu=4096FA \
              --cpu_arch=arm64 --output_dir=4_model_elf \
              --net_name=yolo --mode=normal --save_kernel

DNNC Kernel Information

1. Overview
kernel numbers  : 1
kernel topology : yolo_kernel_graph.jpg

2. Kernel Description in Detail
kernel id       : 0
kernel name     : yolo
type            : DPUKernel
nodes           : NA
input node(s)   : layer0_conv(0)
output node(s)  : layer81_conv(0) layer93_conv(0) layer105_conv(0)

(bvlc1v0_py27) root2@Prec5820Tow:~/ML/YOLOv3/yolo_convertor/example_yolov3$
(bvlc1v0_py27) root2@Prec5820Tow:~/ML/YOLOv3/yolo_convertor/example_yolov3$
(bvlc1v0_py27) root2@Prec5820Tow:~/ML/YOLOv3/yolo_convertor/example_yolov3$
```

Modify the `deploy.prototxt` (generated in the previous step) in the `3_model_after_quantize` folder as follows:

```
layer {
name: "data"
type: "Input"
top: "data"
#####Comment following five lines #####
#transform_param {
# mirror: false
# yolo_height: 608
# yolo_width: 608
# }
#####Nothing change to below layers#####
input_param {
shape {yolov3_deploy.tar.gz
dim: 1
```

```
dim: 3
dim: 608
dim: 608
}
}
}
```

> **Note:** The ref_deploy.prototxt file already contains all the above described changes. So, just copy it and rename as `deploy.prototxt`.

The `3_compile.sh` script uses `dnnc` to compile and generate the ELF file for the target ZCU102 board as follows:

```
#Assume the dnnc-dpu1.3.0 is installed in your $PATH

$ dnnc-dpu1.3.0 --prototxt=3_model_after_quantize/deploy.prototxt \
              --caffemodel=3_model_after_quantize/deploy.caffemodel \
              --dpu=4096FA \
              --cpu_arch=arm64 \
              --output_dir=4_model_elf \
              --net_name=yolo \
              --mode=normal \
              --save_kernel
```

For more information, use the `dnnc-dpu1.3.0 -help` command or see the DNNDK User Guide UG1327. For other platforms, specify `--dpu` and `--cpu_arch` accordingly.

Step 5: Deploy YOLOv3 on the ZCU102 Board

Use the following steps to deploy YOLOv3 on the ZCU102 board:

1. Copy the `dpu_yolo.elf` file (generated by `dnnc`) from the 4_model_elf folder, to the `model` folder. Then, use the following commands to archive the `yolov3_deploy` folder.

```
$ cd yolo_convertor
$ cp example_yolov3/4_model_elf/dpu_yolo.elf yolov3_deploy/model/
$ tar -cvf yolov3_deploy.tar ./yolov3_deploy
$ gzip -v  yolov3_deploy.tar
```

2. Assuming that the ZCU102 board is turned on and it is connected to a 4K external monitor via Display Port, and the usual communication is setup between *target* board and *host* Linux PC (that is, the microUSB-to-USB cable and an Ethernet Point2Point cable), open PuTTy with the following command:

```
$ sudo putty /dev/ttyUSB0 -serial -sercfg 115200,8,n,1,N
```

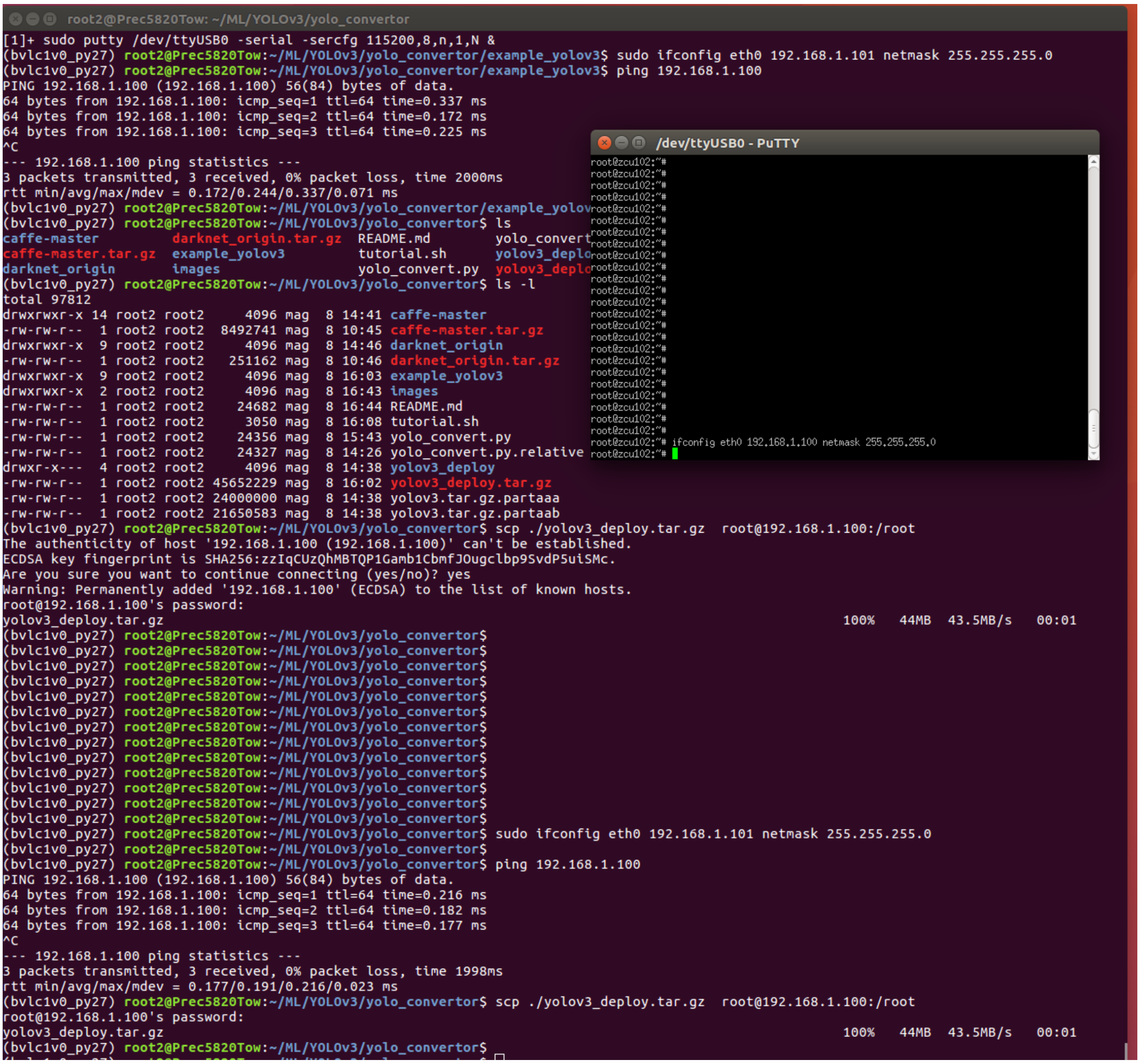
3. Once the UART communication is established, execute the following command:

```
$ ifconfig eth0 192.168.1.100 netmask 255.255.255.0
```

4. From the *host* PC execute the following command (depending on your PC, select either `eth0` or `eth1`. In this case `eth1` is selected)

```
$ sudo ifconfig eth1 192.168.1.101 netmask 255.255.255.0
```

5. Establish the UART communication between the *host* and the *target*. What you have done so far, after booting the board, was to use the serial port to set the IP address of the board and ensure that the *host* PC and *target* board are in the same subnet. The following screenshot illustrates this procedure:



6. From your *host* PC copy the `yolov3_deploy.tar.gz` file to the ZCU102 board using the `ssh/scp` commands:

```
$ scp yolov3_coco80_zcu102.tar root@192.168.1.100:/root
```

7. Open the archive on the board and type the following command:

```
tar -xvf yolov3_deploy.tar.gz
```

After extraction, the files in the package are as follows:

```
yolov3_deploy
├── coco_test.jpg
├── test.avi
├── Makefile
├── model
│ └── dpu_yolo.elf
└── src
├── main.cc
└── utils.h
```

8. Compile the code in the `yolov3_deploy` folder on ZCU102 with the `make` command, as follows:

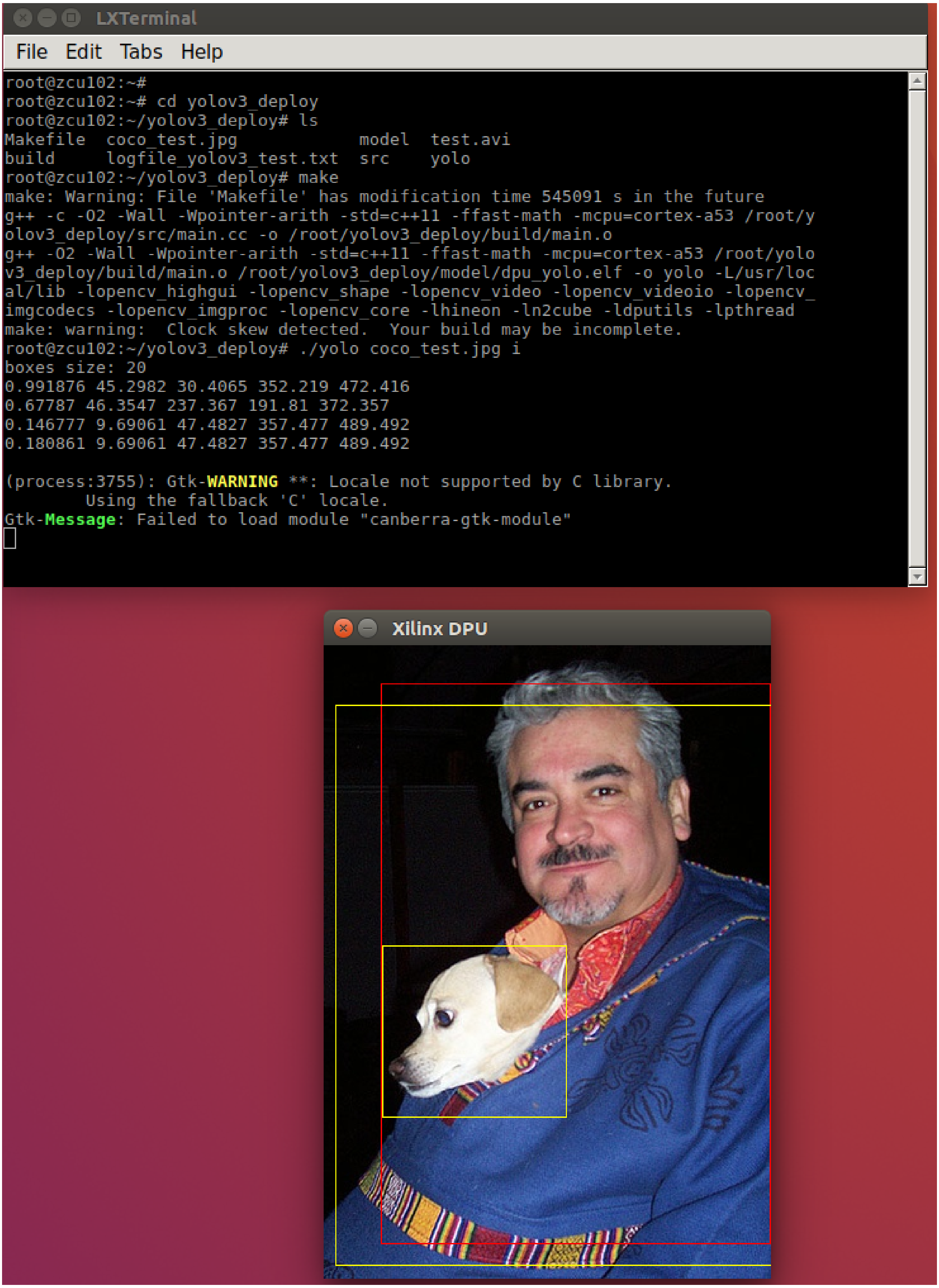
```
make -j
```

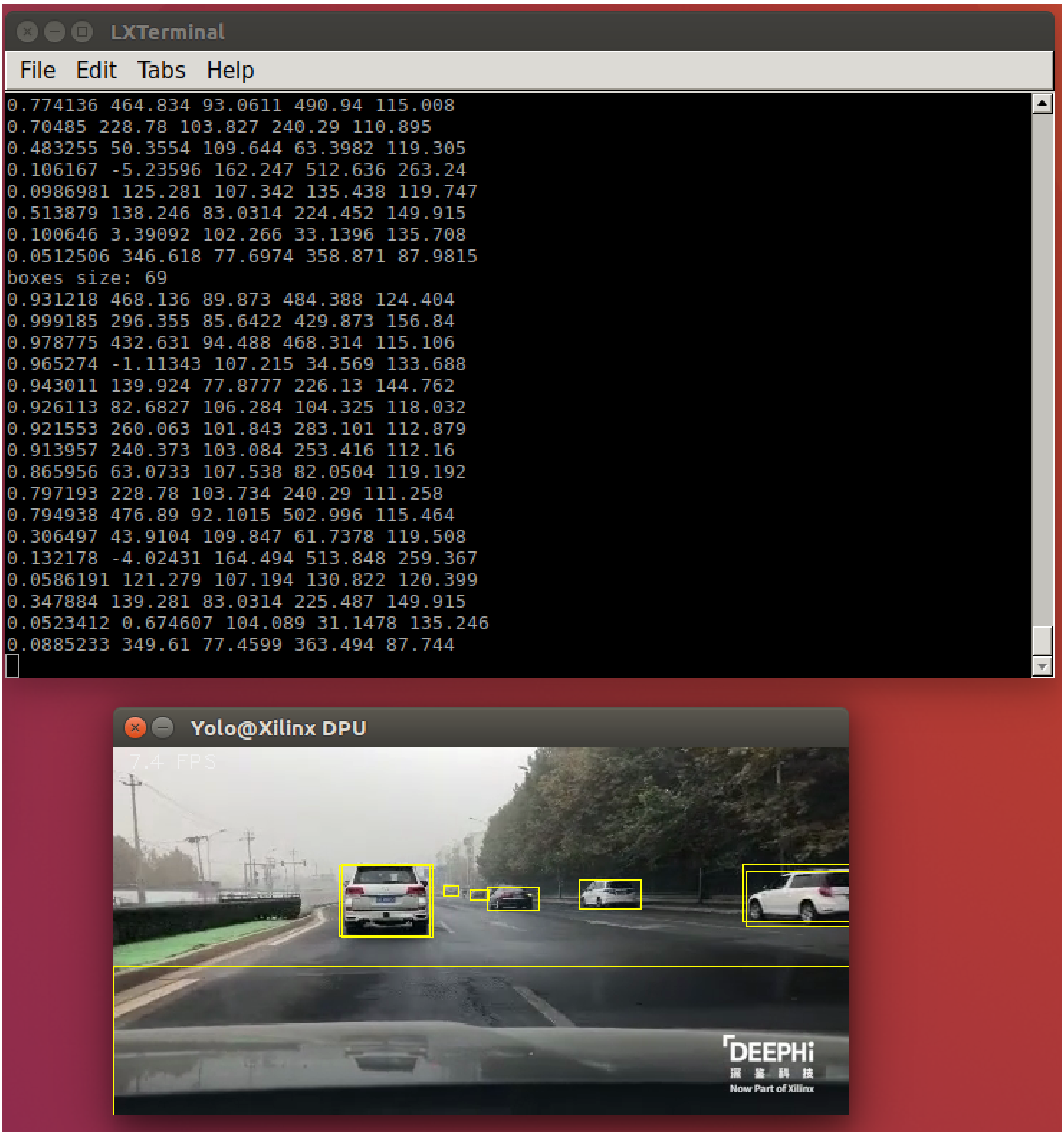
9. Use the following command to run the executable `yolo`:

`#Test image ./yolo coco_test.jpg i`

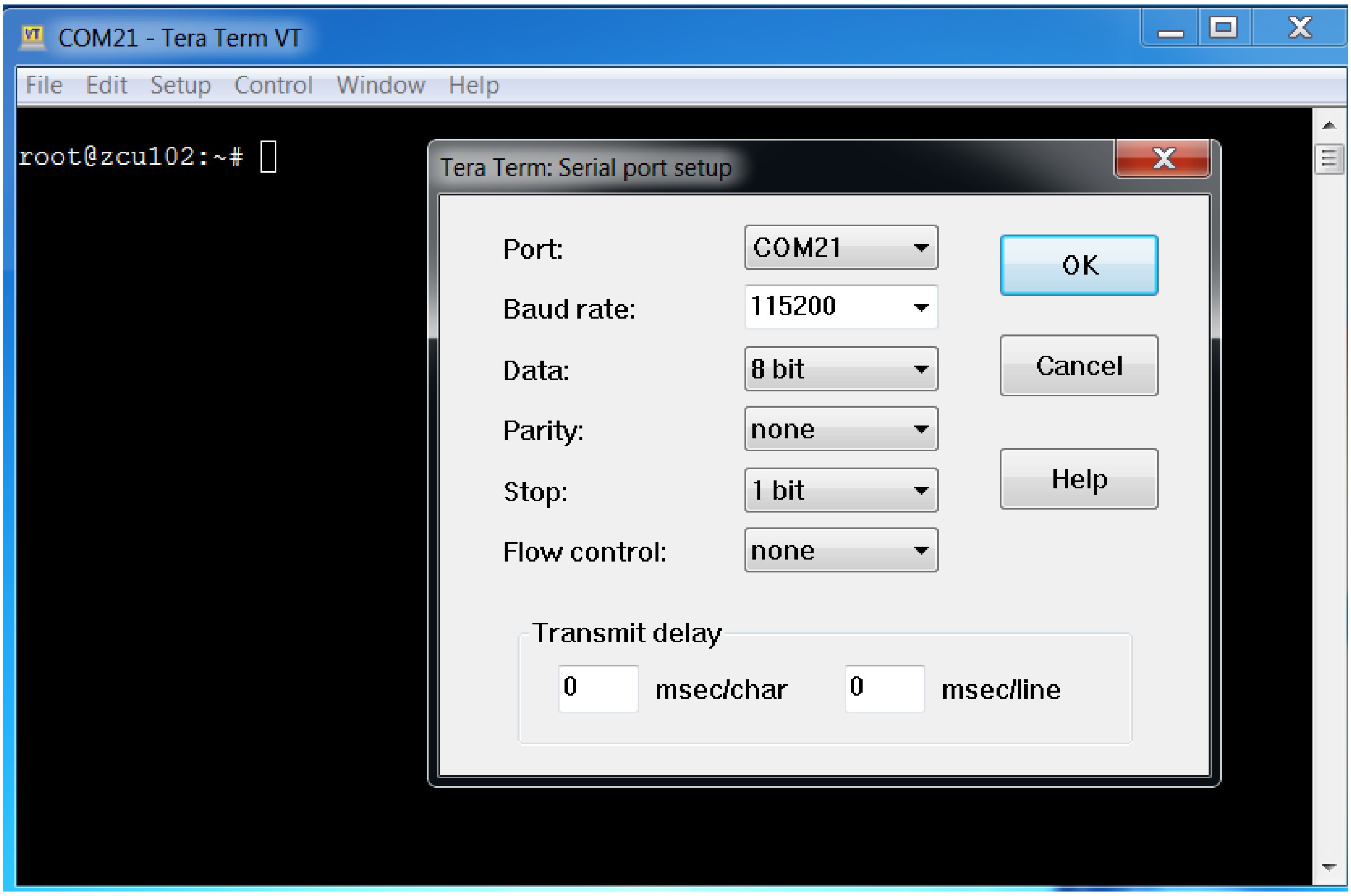
`#Test video ./yolo test.video v`

You will see the following:

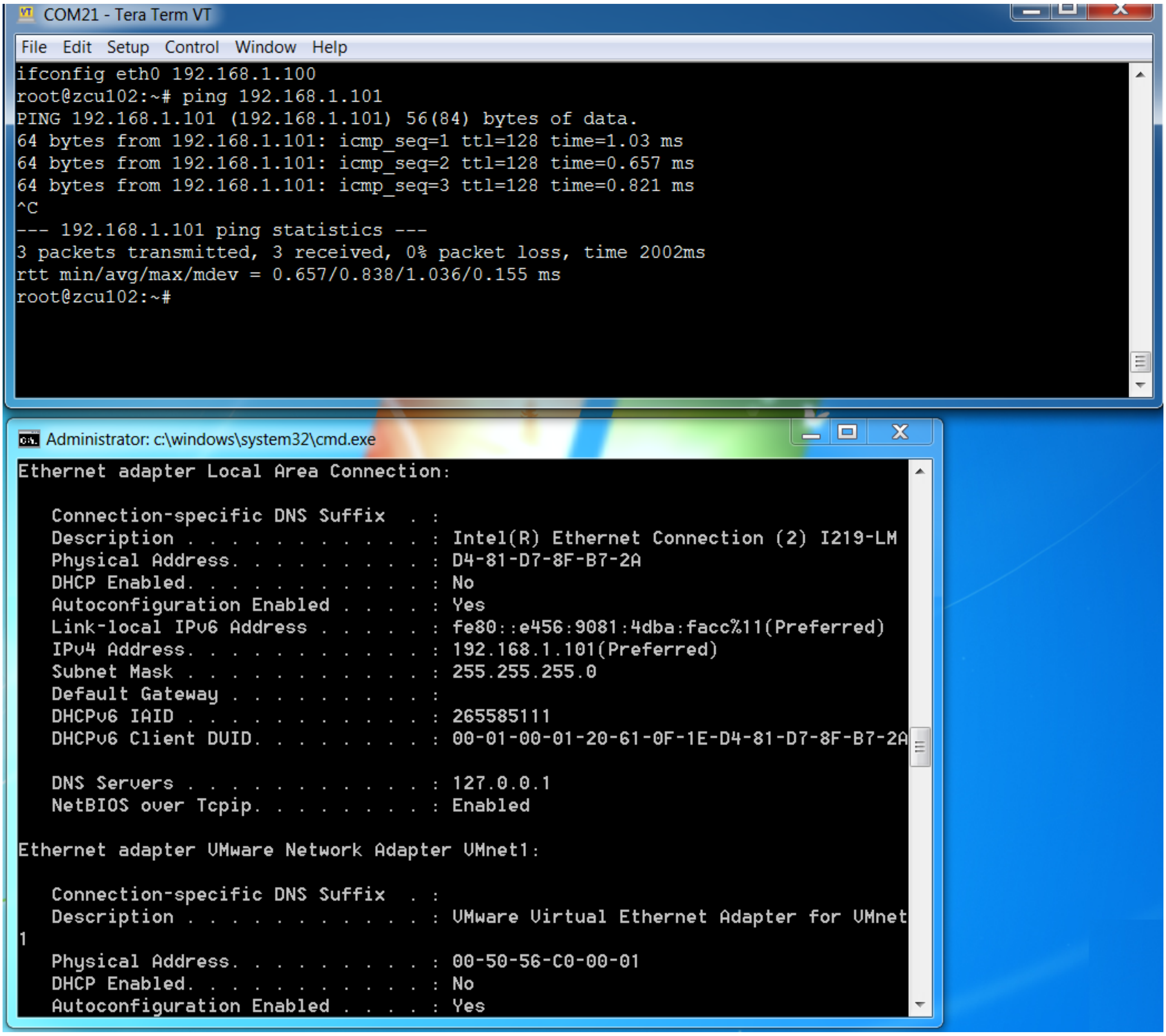


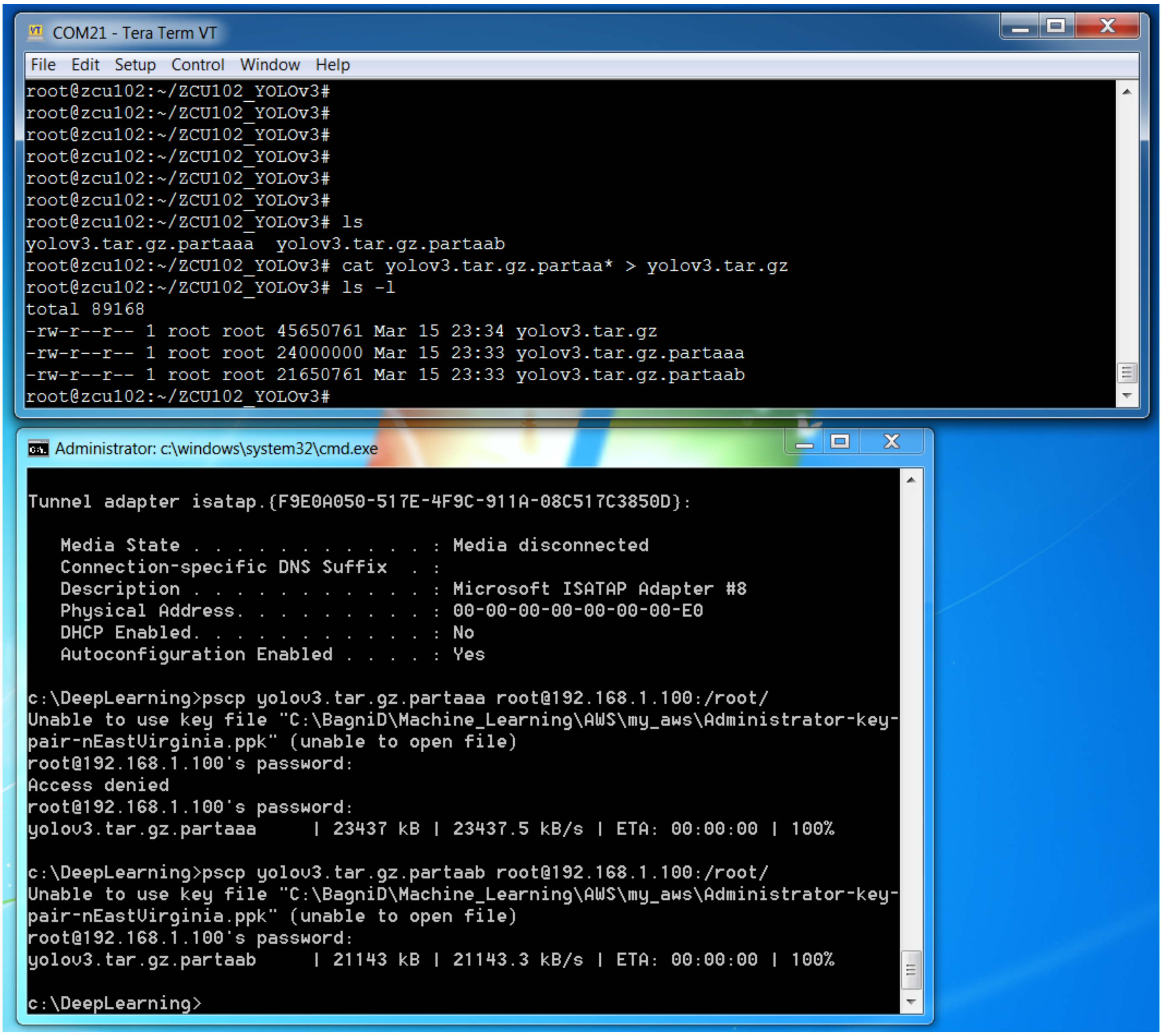

LXTerminal
File Edit Tabs Help
0.774136 464.834 93.0611 490.94 115.008
0.70485 228.78 103.827 240.29 110.895
0.483255 50.3554 109.644 63.3982 119.305
0.106167 -5.23596 162.247 512.636 263.24
0.0986981 125.281 107.342 135.438 119.747
0.513879 138.246 83.0314 224.452 149.915
0.100646 3.39092 102.266 33.1396 135.708
0.0512506 346.618 77.6974 358.871 87.9815
boxes size: 69
0.931218 468.136 89.873 484.388 124.404
0.999185 296.355 85.6422 429.873 156.84
0.978775 432.631 94.488 468.314 115.106
0.965274 -1.11343 107.215 34.569 133.688
0.943011 139.924 77.8777 226.13 144.762
0.926113 82.6827 106.284 104.325 118.032
0.921553 260.063 101.843 283.101 112.879
0.913957 240.373 103.084 253.416 112.16
0.865956 63.0733 107.538 82.0504 119.192
0.797193 228.78 103.734 240.29 111.258
0.794938 476.89 92.1015 502.996 115.464
0.306497 43.9104 109.847 61.7378 119.508
0.132178 -4.02431 164.494 513.848 259.367
0.0586191 121.279 107.194 130.822 120.399
0.347884 139.281 83.0314 225.487 149.915
0.0523412 0.674607 104.089 31.1478 135.246
0.0885233 349.61 77.4599 363.494 87.744
Yolo@Xilinx DPU
DEEPHi
Now Part of Xilinx

If you are using a Windows OS PC, you can use Teraterm with the same settings as Putty, as shown below:



In that case you proceed accordingly to set the IP addresses and use `pscp.exe` utility instead of `scp`, as shown in the following screenshot:


COM21 - Tera Term VT
File Edit Setup Control Window Help
ifconfig eth0 192.168.1.100
root@zcu102:~# ping 192.168.1.101
PING 192.168.1.101 (192.168.1.101) 56(84) bytes of data.
64 bytes from 192.168.1.101: icmp_seq=1 ttl=128 time=1.03 ms
64 bytes from 192.168.1.101: icmp_seq=2 ttl=128 time=0.657 ms
64 bytes from 192.168.1.101: icmp_seq=3 ttl=128 time=0.821 ms
^C
--- 192.168.1.101 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.657/0.838/1.036/0.155 ms
root@zcu102:~#
Administrator: c:\windows\system32\cmd.exe
Ethernet adapter Local Area Connection:
Connection-specific DNS Suffix . :
Description . . . . : Intel(R) Ethernet Connection (2) I219-LM
Physical Address. . . . . : D4-81-D7-8F-B7-2A
DHCP Enabled. . . . . : No
Autoconfiguration Enabled . . . . : Yes
Link-local IPv6 Address . . . . . : fe80::e456:9081:4dba:facc%11(Preferred)
IPv4 Address. . . . . : 192.168.1.101(Preferred)
Subnet Mask . . . . : 255.255.255.0
Default Gateway . . . . :
DHCPv6 IAID . . . . : 265585111
DHCPv6 Client DUID. . . . . : 00-01-00-01-20-61-0F-1E-D4-81-D7-8F-B7-2A
DNS Servers . . . . : 127.0.0.1
NetBIOS over Tcpip. . . . . : Enabled
Ethernet adapter VMware Network Adapter VMnet1:
Connection-specific DNS Suffix . :
Description . . . . : VMware Virtual Ethernet Adapter for VMnet
1
Physical Address. . . . . : 00-50-56-C0-00-01
DHCP Enabled. . . . . : No
Autoconfiguration Enabled . . . . : Yes



> **Note:** For more information on setting up the ZCU102 communication, preparing to boot the SD card, connecting serial port/ethernet cables, and setting up the 4K display, see the DNNDK User Guide UG1327.