2019 年 SEU-Xilinx 国际暑期学校团队项目设计文档

(Project Paper Submission Template)

作品名称	基于 Ultra96 的行人检测系统
组员姓名	鲍春 胡心雨 常乐
房间号及桌号	715　1 号

第一部分

小组成员分工

（各成员任务分配）

	姓名	任务分配
组长	鲍春	FPGA 程序开发，系统整体架构设计开发
组员 1	胡心雨	深度学习模型开发、训练；视频处理
组员 2	常乐	嵌入式程序开发，DPU 软件调试

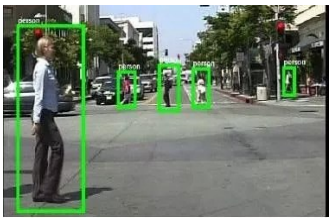
第二部分

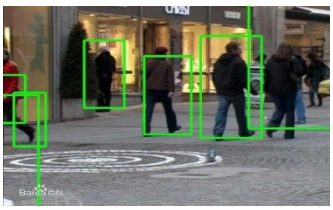
设计概述 /Design Introduction

（请简要描述一下你的设计：1. 功能说明；2. 所用到的设备清单）

1. 系统概述

随着计算机视觉领域的不断发展，特别是近年来深度学习取得了突破性进展，人类正逐步迈入智能时代。行人检测技术在机器人、安防监控、无人驾驶等诸多智能领域都有着重要的应用。另一方面，随着物联网和单片机技术的深入发展，嵌入式系统由于体积小、功耗低等特点得到了很大规模的运用。和传统的行人检测相关算法相比，基于深度学习的方法在相关公开数据集上具有更高的识别准确率，对于具有复杂背景场景以及多目标场景下的识别率有更好的效果。但是现有的行人检测算法的出发点大都建立在科研的基础上，算法运行平台是专门用于科研的高性能计算机，直接将算法应用到实际场景中会面临诸多难题，尤其是在嵌入式端检测的效果很难达到实时性。为了解决嵌入式端应用的问题，减小运算量，实现行人快速检测，本文基于 Ultra96 平台研究 DPU 框架下的实时行人检测系统。

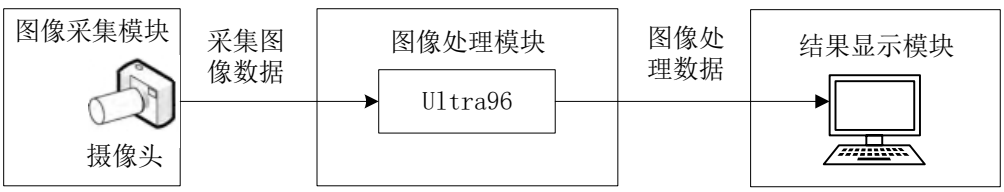




本系统是基于嵌入式系统的行人检测方法设计，其主要内容是将基于 SSD 的行人检测算法按照 DPU 框架的要求搭建在 Ultra96 平台上，从模型压缩的角度出发优化训练模型，改进算法的性能，完成从视频的采集到嵌入端实时处理，并将检测结果传入显示器，实现一个基于 DPU 的行人检测系统。

2. 功能说明

整体系统由三个模块构成，分别为：图像采集模块、图像处理模块和结果显示模块，通过 PS 端的 USB 摄像头将图像数据读入系统，再由硬件平台进行计算加速，最终由 PL 端连接的 HDMI 显示单元进行结果实时显示，实现 USB 摄像头的图像采集以及数据的实时处理和显示输出，具体功能如图所示



3. 系统流程

（1）使用 SSD 网络基于 caffe 框架在训练行人数据集上进行训练

1）预训练模型，SSD 使用 VGG 作为前端网络，训练自己数据时首先加载预训练模型权重，加快模型收敛速度；

2）制作数据集，选用行人数据集 INRIA，并转化成 VOC 格式；

3）将 VOC 转换成 lmdb 文件，通过运行配置文件完成；

4）对 SSD 网络进行训练，选用远程服务器，显卡型号 Titan V，运行环境 Ubuntu16.04；

5）得到已经训练好的模型及原型文件：SSD.caffemodel 和 SSD.prototxt。

（2）Ultra96 板卡加速 SSD 网络实施

1）板卡加载 SD Card 镜像驱动；

2）上位机配置虚拟机环境，并配置深度学习 caffe 环境；

3）将训练好的权重文件（SSD.caffemodel）和原型文件（SSD.prototxt）载入虚拟机中；

4）运行配置文件对模型进行裁剪；

5）将裁剪完的模型进行量化生成.elf 文件；

6）将两个文件交叉编译生成可执行文件；

7）将生成的可执行文件下到目标板中，并运行可执行文件。

4. 硬件设计

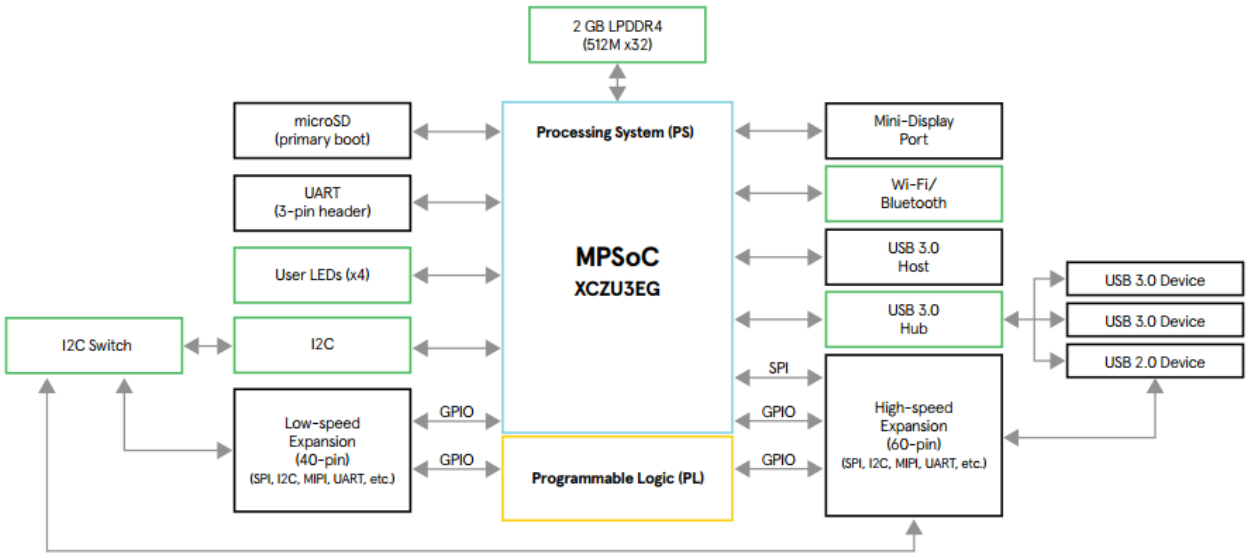
（1）设备清单

Ultra96，USB 摄像头，显示屏，sd 卡，转接线

（2）Ultra96 平台

表 1　Ultra96 资源

主处理器	Xilinx Zynq UltraScale + MPSoC ZU3EG SBVA484
Memory	Micron 2 GB（512M x32）LPDDR4 内存
Data Storage	Delkin 16 GB MicroSD 卡+适配器
Connectivity	Wi-Fi /蓝牙
Display	Mini DisplayPort（MiniDP 或 mDP）
Micro-B	1 个 USB 3.0 型 Micro-B 上游端口
USB	2 个 USB 3.0 A 型，1 个 USB 2.0 HS 夹层下游端口
Other	40 针 96B 低速扩展接头，60 针 96B 高速扩展接头



2）硬件框图

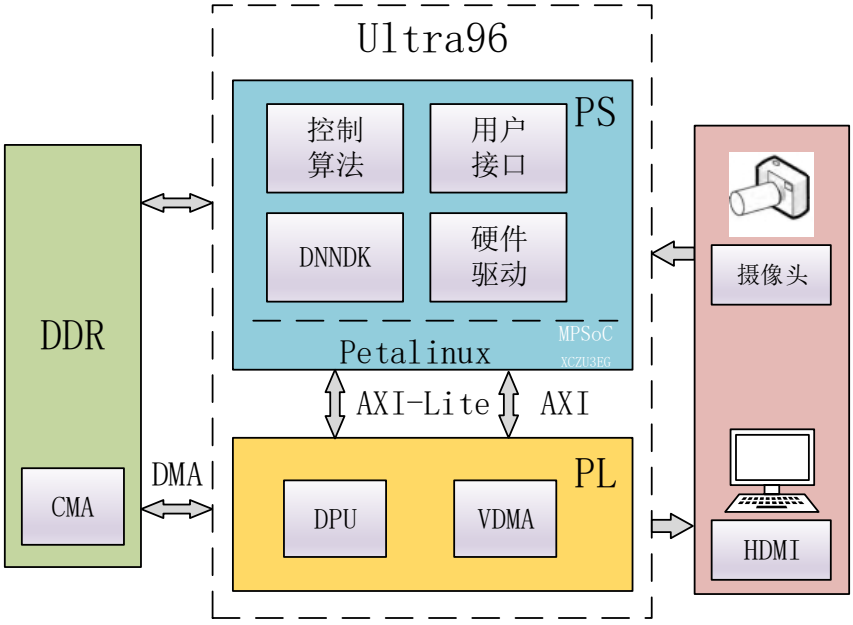
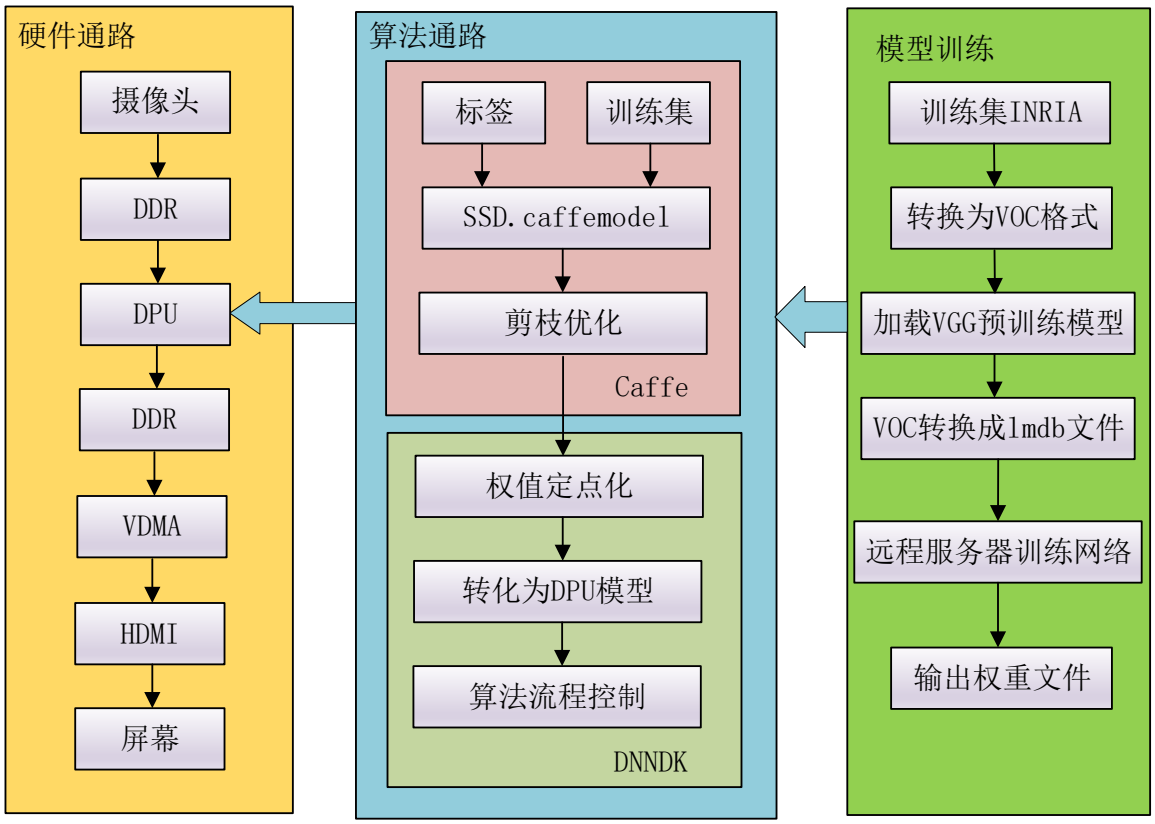


图 2　系统硬件架构

3．软件设计

本系统软件主要分为三个部分，分别为：模型训练、算法通路、硬件通路，系统软件架构如图所示。在模型训练部分，我们选择 INRIA 数据集，对数据集进行预处理，使其适配 caffe 框架，在服务器中，训练 SSD 模型。算法通路主要部署在 Ultra96 的 PS 端，在 PS 端，我们利用 DNNDK 工具进行 SSD 模型的剪枝及定点处理。硬件通路主要部署在 Ultra96 的 PL 端，通过 Vivado 的 Block Design，我们将 DPU IP 添加进硬件工程，将带有 DPU IP 的工程生成硬件电路载入板卡，此外，外设的驱动电路也会同时部署，PL 端资源的调度以及图像采集任务都由 PS 端的 Petalinux 完成。



第三部分

详细设计 /Detailed Design

（请详细说明你作品要实现的所有功能以及如何组建系统以实现该功能，还包括为实现该功能需要用到的所有参数和所有操作的详细说明，必要的地方多用图表形式表述）

如因文档篇幅限制无法详述，可提供附件。

详细设计见 github 上的文档 readme

链接：https://github.com/EdwardBao1006/SSD_detection_by_DPU_on_Ultra_96.git

第四部分

完成情况及性能参数 /Final Design & Performance Parameters

（作品完成情况，附照片/视频链接）

实现了基于 SSD 的行人检测，检测速度能达到 47 帧每秒，能够实现摄像头的实时检测
详细完成情况见视频

第五部分

项目总结 /Conclusions

（项目中所用到的知识点，项目收获及心得）

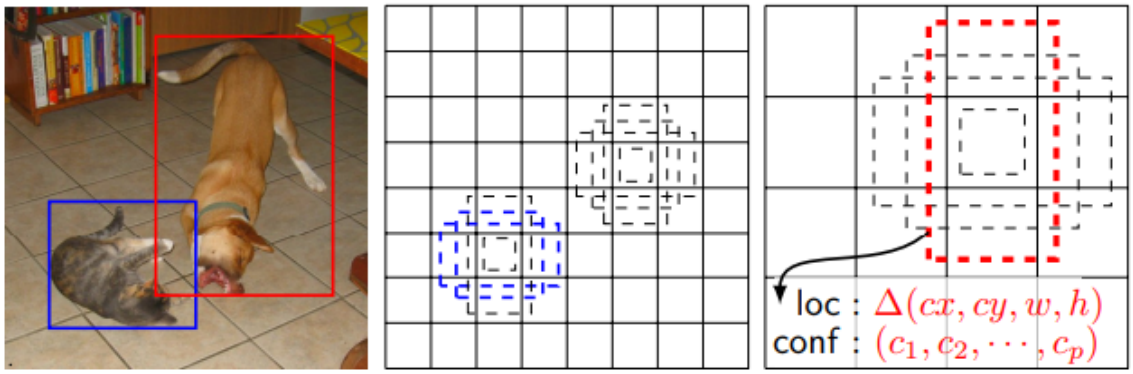
1. 关于 SSD 网络的认识

SSD 全称 ：Single Shot MultiBox Detector。在 R-CNN 系列模型里。Region Proposal 和分类是分作两块来进行的。SSD 则将其统一成一个步骤来使得模型更加简单并且速度更快。YOLO 与 SSD 可以一步到位完成检测。相比 YOLO，SSD 采用 CNN 来直接进行检测，而不是像 Yolo 那样在全连接层之后做检测。它跟 Faster R-CNN 主要有两点不一样，1，对于锚框，不再首先判断它是不是含有感兴趣物体，再将正类锚框放入真正物体分类。SSD 里我们直接使用一个 C+1 类分类器来判断它对应的是哪类物体，还是只是背景。我们不再有额外的回归器对边框再进一步预测，而是直接使用单个回归器来预测真实边框。2，SSD 不只是对卷积神经网络输出的特征做预测，它会进一步将特征通过卷积和池化层变小来做预测。这样达到多尺度预测的效果。

SSD(Single Shot MultiBox Detector)用回归方法作检测，把定位和分类放在一个网络里面。SSD 的网络是在 VGG16 上作修改，把 VGG16 的全连接层换成卷积层。添加的每个卷积层都输出一个特征图，并以此作为预测的一个输入，从而得到多尺度的特征图来进行回归。低层的特征图包含了更多的信息，有利于保留细节，回传训练误差，提高了检测的精度。SSD 同时借鉴了 YOLO 和 Faster-RCNN 两种方法的优点，效果也超越了两者，mAP 达到了 75%,速度也得到了提高。

（1）采用多尺度特征图用于检测

多尺度采用大小不同的特征图，CNN 网络一般前面的特征图比较大，后面会逐渐采用 stride=2 的卷积或者 pool 来降低特征图大小，一个比较大的特征图和一个比较小的特征图，它们都用来做检测。这样做的好处是比较大的特征图来用来检测相对较小的目标，而小的特征图负责检测大目标，如图下图所示，8x8 的特征图可以划分更多的单元，但是其每个单元的先验框尺度比较小。



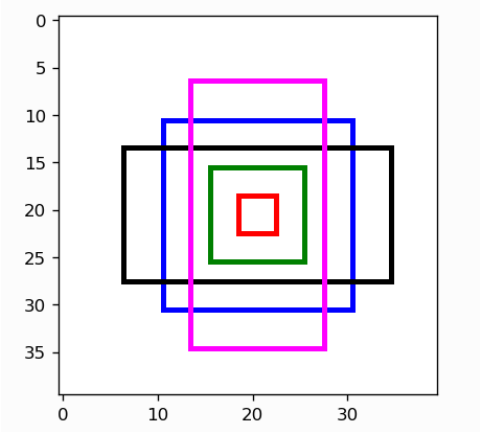
(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

（2）采用卷积进行检测

与 Yolo 最后采用全连接层不同，SSD 直接采用卷积对不同的特征图来进行提取检测结果。对于形状为 m×n×p 的特征图，只需要采用 3×3×p 这样比较小的卷积核得到检测值。

（3）设置先验框

在 Yolo 中，每个单元预测多个边界框，但是其都是相对这个单元本身（正方块），但是真实目标的形状是多变的，Yolo 需要在训练过程中自适应目标的形状。而 SSD 借鉴了 Faster R-CNN 中 anchor 的理念，每个单元设置尺度或者长宽比不同的先验框，预测的边界框（bounding boxes）是以这些先验框为基准的，在一定程度上减少训练难度。一般情况下，每个单元会设置多个先验框，其尺度和长宽比存在差异，如下图所示，可以看到每个单元使用了 5 个不同的先验框，图片中不同物体采用最适合它们形状的先验框来进行训练。

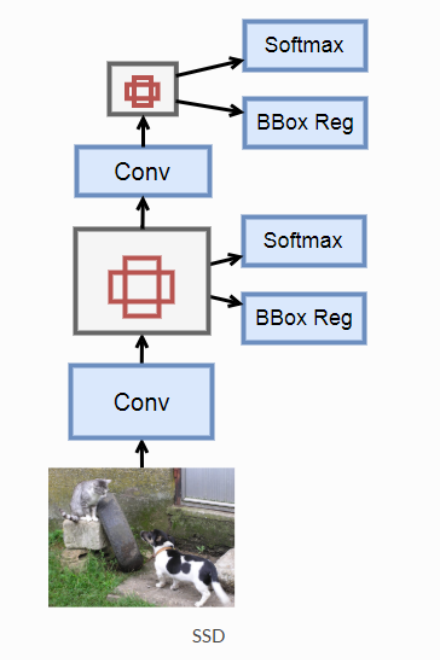


（4）预测物体类别

对每一个锚框我们需要预测它是不是包含了我们感兴趣的物体，还是只是背景。使用一个 3×3 的卷积层来做预测，加上 pad=1 使用它的输出和输入一样。同时输出的通道数是 num_anchors*(num_classes+1)，每个通道对应一个锚框对某个类的置信度。假设输出是 Y，那么对应输入中第 n 个样本的第(i, j)像素的置信值是在 Y[n, :, i, j]里。具体来说，对于以(i, j)为中心的第 a 个锚框。通道 a*(num_class+1) 是其只包含背景的分数，通道 a*(num_class+1)+1+b 是其包含第 b 个物体的分数。

（5）预测边界框

因为真实的边界框可以是任意形状，我们需要预测如何从一个锚框变换成真正的边界框。这个变换可以由一个长为 4 的向量来描述。同上一样，我们用一个有 num_anchors * 4 通道的卷积。假设输出是 Y，那么对应输入中第 n 个样本的第 (i, j) 像素为中心的锚框的转换在 Y[n, :, i, j]里。具体来说，对于第 a 个锚框，它的变换在 a*4 到 a*4+3 通道里。



SSD

2. 关于 DPU 的认识

（1）resnet 测试

（2）人脸检测

实现摄像头的实时检测

（3）ssd 测试

在 Ultra 板卡上实现了 SSD，

ssd 网络在 voc 数据集的训练下生成文件的检测结果，其检测帧数最高能达到 45 帧，完全能做到实时检测的效果。其检测结果图如图所示。其检测到的区域用红框框选出。

3. 项目收获

（1）sd 卡的空间问题

在制作 SD 卡镜像时，大量 sd 卡的空间没有被释放，导致不能正常将 SDK 包和一些动态链接库导入板子中。通过查找资料解决了空间不足的问题。

解决方法：

1）格式化 SD 卡，格式为 FAT32 格式

2）使用 win32 Disk Imager 将镜像烧录到 SD 卡中(注意是点击 Write 开始烧录)

3）镜像烧录成功后提示：“Write Succesful”

4）使用 DiskGenius 将空闲分区激活

5）建立新分区

6）将 SD 卡放进读卡器中，加电启动电脑，并用 ROOT 登录；用 df －hl 查看空间情况，用 fdisk －l 查看 SD 卡空间

删除分区，并且重新建立分区

使用 resize2fs 调整分区大小

允许 df -hl 是否空间增加已经成功呢，Size 由原来的 950MB 变成了 7.4G

```
root@ultra96:/usr/share/XILINX_AI_SDK/samples/ssd# df -hl
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        13G  4.3G  7.9G  35% /
devtmpfs        736M     0  736M   0% /dev
tmpfs           992M     0  992M   0% /dev/shm
tmpfs           992M  9.5M  983M   1% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           992M     0  992M   0% /sys/fs/cgroup
tmpfs           199M  4.0K  199M   1% /run/user/0
root@ultra96:/usr/share/XILINX_AI_SDK/samples/ssd#
```

（2）caffe 环境的搭建

在 lab2 提供的虚拟机镜像中使用本身带有的 caffe，总是报错，no modulecalled_caffe，在添加路径后，能使用 caffe，但是由于本身的 caffe 缺少文件，在实行 create_list 时总是报 module' object has no attribute 'LabelMap'这个错误也尝试了许多办法，也不知道哪里报错了，例如添加环境变量或在运行程序中，制定使用 python 和 caffe 的路径，也试过重新编译也没有成功，后来想到可能是本身 caffe 环境的问题，于是重新下载了 caffe，在编译的时候报 gpu、cudn 等错误，想到可能这个 caffe 配置可能是 gpu 的，

于是修改 makefile 将有关 gpu 的配置去掉，然后再进行编译，可以成功加载 caffe 的库，但是还是在运行中报错，发现在环境变量中没有 caffe，进行配置，再 source 生效环境变量，发现还是报 no module called caffe 经过仔细检查发现在这个镜像文件中还有一个 python2.7，由于有进行更改环境变量 pythonpath 重新指向 caffe 中的 python，然后 caffe 环境不再报错，caffe 可以运行名为后面生成 lmdb 文件和 ssd 的训练做准备。

第六部分

源代码/Source code

（作品源码 Github 链接，github 操作步骤见 *Github 项目上传简单教程*）

具体代码见 github

链接：https://github.com/EdwardBao1006/SSD_detection_by_DPU_on_Ultra_96.git

第七部分（技术总结/心得/笔记等分享）

（可分享的技术文档附件或已发表的文章链接）

1. 有关 ML SSD PASCAL Caffe Tutorial (UG1340)一些自己的理解

目的：在 zcu102 上用 dpu 部署 ssd 网络并进行测试

该教程主要分为六个部分：

1）安装 caffe 环境

2）准备数据集和数据库

3）训练 SSD 网络

4）评估 SSD 网络

5）量化和编译网络

6）在开放板上运行

（1）安装 caffe 环境（注意事项）

1）想要使用 SSD 首先需要 caffe 环境，这里我们需要一个特定的 caffe 环境，这个 caffe 是专门用来训练 SSD 的，请将$ CAFFE_ROOT 环境变量设置到~/ .bashrc 文件

2）export CAFFE_ROOT = / caffe 的绝对路径 / caffe-ssd

3）教程中介绍到如果使用 Windows，请安装 Win32DiskImager 并将映像刷新到 16GB 或更大的 SD 卡，真的 root 分区下容量不够的问题可以参考上面增大内存的教程。

4）可以将 x86 主机工具复制到 linux x86 主机以量化/编译模型。

5）在编译 caffe 的时候需要注意是 CPU 还是 GPU

6）在 make -j2 时候，需要输入命令查看虚拟机或者电脑的配置，j 后的数字应该与#cat /proc/cpuinfo |grep “cores” |uniq 得到结果保持一致

（2）准备数据集和数据库

1）http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval_11-May-2012.tar
http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-2007.tar
http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtest_06-Nov-2007.tar
这个三个地址是训练时 ssd 网络有到的数据，当然有 VOC 的数据可以直接放在 $CAFFE_ROOT/data/VOC0712 的下边，然后进行解压，请注意路径问题，最好按照教程来摆放数据，否则你就需要改许多的路径，这里建议还是按照教程中存放
2）接下来的步骤会依赖于 part1 中已经搭建好的 caffe

```
cd $CAFFE_ROOT/data/VOC0712
source create_list.sh
source create_data.sh
cp -r examples/* ../../examples
```

其中这里面重要的是 VOC 数据在哪里存放，生成的 lmdb 文件有存放在哪里，建议保持教程里所对应对的路径，在训练时会省掉一些关于路径的错误。

（3）训练 ssd 模型

1）ssd_pascal.py 的功能说明

此 python 脚本将使用所有必需参数为 SSD 网络创建 solver.prototxt，deploy.prototxt，test.prototxt 和 train.prototxt 文件。这些都将在$ CAFFE_ROOT/models / VGGNet / VOC0712 / SSD_300x300 下创建。查看原型文件并注意它们都假设将 LDMB 数据库放在$ CAFFE_ROOT / examples / VOC0712 / VOC0712_test_lmdb 文件夹中，同时会进行训练，输出快照将在以下目录下逐步创建：$ CAFFE_ROOT / models / VGGNet / VOC0712 / SSD_300x300，但此时您需要通过在终端中输入 CTRL + C 来尽早停止训练过程。基本原理是您可以暂停培训过程，对原型文件进行一些手动修改，以便与 DPU 和 DNNDK 工具兼容。
2）ssd_pascal.py 有关 gpu 和 cpu 的问题
该 py 文件是通过调用 gpu 来完成训练的，但是在实验环境下，还是用 cpu 进行训练，这是就需要删除一些有关 gpu 的部分
在 cpu 训练中请将 ssd_pascal.py 中的部分代码注释

```
#gpus = "0,1,2,3"
#gpulist = gpus.split(",")
#num_gpus = len(gpulist)
# Divide the mini-batch to different GPUs.
batch_size = 32
accum_batch_size = 32
iter_size = accum_batch_size / batch_size
solver_mode = P.Solver.CPU
device_id = 0
batch_size_per_device = batch_size
#if num_gpus > 0:
#    batch_size_per_device = int(math.ceil(float(batch_size) / num_gpus))
#    iter_size = int(math.ceil(float(accum_batch_size) / (batch_size_per_device * num_gpus)))
#    solver_mode = P.Solver.GPU
#    device_id = int(gpulist[0])
```

3）train_test.prototxt 是通过 train.prototxt 进行修改得到的，不建议使用本身自带的 train_test.prototxt，还是应该按照教程进行 train_test.protxt 文件的生成
4）mbox_conf_reshape，mbox_conf_softmax 和 mbox_conf_flatten 图层，以确保这些图层仅用于测试。 需要添加的条目

5）将 fc6 改为 fc_682，并进行命名，将 num_output 改为 682

6）我认为在迭代步数中权重保存是根据“solver.prototxt”中指定的参数确定的，并注意默认情况下第一个保存的 caffemodule 将在 80K 迭代后发生。

（4）评估 SSD 网络

1）这些脚本将在 120K 迭代训练模型上进行。如果您没有此模型，只需打开脚本（score.sh 和 webcam.sh）并将模型名称更改为您可用的模型。重要信息：在运行脚本之前，需要将评估文件夹复制到 $ CAFFE_ROOT 目录下，运行两个不同的脚本：一个通过连续从 Web 摄像头抓取帧，另一个使用测试数据集对网络进行评分。本教程中使用在带有一块 GTX 1080ti GPU 卡的 Xeon 机器，网络摄像头版本大约为 30FPS。

2） 在运行脚本文件时注意路径问题

（5）在开放板上运行

1）sudo ./install.sh ZCU102 这条命令，由于我们是在 Utr96 上运行的更改为初始化+板子名称

2）这个 float.prototxt 应该与你的 train_test.prototxt 几乎相同，但有一些修改：

已修改输入层以指定校准数据的路径以及启用 DNNDK 中的自动测试功能。以相对方式指定了以下路径，以便在复制到$ CAFFE_ROOT 时指向正确的位置：

第 55 行： “../examples / VOC0712 / VOC0712_test_lmdb”

第 62 行：label_map_file：“../ data / VOC0712 / labelel_voc.prototxt”

第 1706 行：output_directory：“./ test_results”

第 1709 行：label_map_file：“../ data / VOC0712 / labelel_voc.prototxt”

第 1710 行：name_size_file：“../ data / VOC0712 / test_name_size.txt”

3）在 dnnc.sh 进行编译需要注意

```
dnnc --prototxt = $ {model_dir} /deploy.prototxt \
        --caffemodel = $ {model_dir} /deploy.caffemodel \
        --output_dir = $ {output_dir} \
        --net_name = $ {net} \
        --dpu = 4096FA \
        --cpu_arch = arm64
```

Arm64 是针对板子 ps 端的数据总线是多少位的

Dpu 的型号要与开发板相互对应

3） 将 elf 文件传入目标板与 main 函数联合编译生成可执行文件

Makefile：用于在 ZCU102 上构建应用程序，并将软件应用程序.elf 与已编译的 SSD 模型链接.elf。

run.sh 文件：用于通过在 ZCU102 上运行源 run.sh 来在目标上执行应用程序。

stop.sh 文件：用于通过在 ZCU102 上运行源 stop.sh 来停止目标上的执行。

model 目录：这是您需要复制已编译的模型可执行文件并确保它名为 dpu_ssd.elf 的目录。

src 目录：此目录包含将执行以读取视频文件，调整其大小并调用 DPU API 以处理图像，然后在 displayport 监视器上显示，我们这里使用的 Utr96。