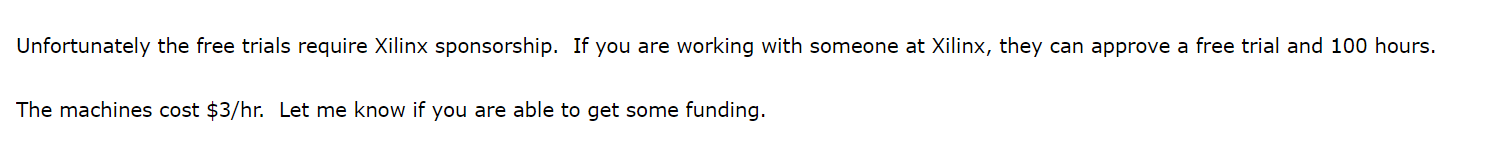
基于Alveo平台的眼底网络实现

尝试申请Alveo云资源：NimBix提供100小时的云资源免费使用时间，利用Nimbix主页contact us功能得到的回复如下



所以申请失败。考虑到我们自己拥有ZCU102平台，所以打算使用该平台部署Vitis AI眼底血管分割应用。

Centos 7环境安装配置：docker 和 vitis ai

Docker安装：

1. 添加docker repository

sudo yum install -y yum-utils

sudo yum-config-manager \

--add-repo \

<https://download.docker.com/linux/centos/docker-ce.repo>

2,安装最新的docker

sudo yum install docker-ce docker-ce-cli containerd.io

3,启动docker

sudo systemctl start docker

4，为非root添加启动docker的权限

<https://docs.docker.com/engine/install/linux-postinstall/>

clone vitis ai 仓库:

git clone https://github.com/Xilinx/Vitis-AI

基于CPU的vitis ai docker下载和安装：(早上安装下载，晚上时间段速度极慢)  
 docker pull xilinx/vitis-ai-cpu:latest (本实验使用)

docker pull xilinx/vitis-ai:tools-1.0.0-cpu

前端模型量化和编译：

首先进入vitis ai的docker环境：

Cd vitis\_ai

./docker\_run.sh xilinx/vitis-ai-cpu:latest

量化Float model->deploy model

文件准备：

float.caffemodel 存放经过caffe训练过得模型权重

float.prototxt描述网络模型结构

原始的眼底图片和分割后的血管图片 用于量化过程中的训练，减少量化误差

conda activate vitis-ai-caffe

vai\_q\_caffe quantize -model float.prototxt -weights float.caffemodel

编译：

由于dpu并不支持sigmoid层， 所以这里先把deploy.prototxt中的sigmoid层去掉，该层利用arm CPU处理。

Cd quantize\_results

vai\_c\_caffe --prototxt ./deploy.prototxt --caffemodel ./deploy.caffemodel --arch /opt/vitis\_ai/compiler/arch/DPUCZDX8G/ZCU102/arch.json --output\_dir model --net\_name bstdsn

值得注意的是，最新版的vitis ai docker中zcu102.json文件目录和文件名和ug1414指导手册给出的不同。

至此，网络模型的量化和编译完成。

ZCU102模型部署

主机端安装交叉编译器：

下载 [sdk-2020.2.0.0.sh](https://www.xilinx.com/bin/public/openDownload?filename=sdk-2020.2.0.0.sh)

./sdk-2020.2.0.0.sh

环境变量设置

source sdk\_directory/environment-setup-aarch64-xilinx-linux

接下来仿照resnet50模型的main.cc文件编写眼底模型的main.cc文件，并进行编译

bash -x build.sh

生成可执行文件eye

ZCU102配置

下载镜像文件：<https://www.xilinx.com/bin/public/openDownload?filename=xilinx-zcu102-dpu-v2020.2-v1.3.1.img.gz>

利用Win32DiskImager烧写软件制作SD卡镜像，注意SD卡不要使用>=32GB，否则正常无法启动。

将vitis ai编译后的xmodel文件、交叉编译的可执行文件和待分割的图像拷贝利用FTP服务拷贝到zcu102的linux系统中。

通过ssh连接到zcu102的linux系统中，并执行：

./eye eye.xmodel

由于分割结果还需要进行后处理，才能展示出较好的显示效果，这里采用python脚本完成dpu输出结果的后处理。结果如下：

