2023 Xilinx Summer School -Project

（队员：陈俊杰、戴柳艳）

1.项目简介

在2012年的ImageNet图像识别竞赛中，Hinton率领的团队利用卷积神经网络构建的AlexNet一举夺得了冠军，从此点燃了学术界、工业界等对于深度学习、卷积网络的热情。短短六七年的时间，在图像分类、物体检测、图像分割、人体姿态估计等一系列计算机视觉领域，深度学习都收获了极大的成功。

本项目利用课程所学知识，聚焦于图像物体识别，目标是在FPGA嵌入式开发平台上实现LeNet-5卷积神经网络边缘部署，完成对cifar10数据集中10类物品的识别，并实现相比于GPU和CPU具有更高计算效率和能耗比的设计。

2.项目环境和工具

1.FPGA开发平台：xc7z020clg400（需移植PYNQ框架）

2.Vivado HLS-2019.2

3.Vivado 2019.2

4.PyTorch

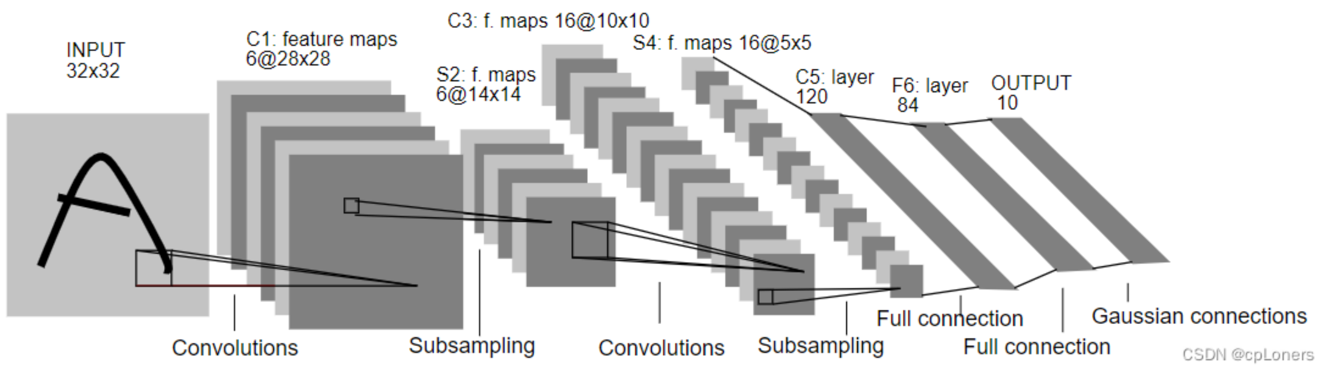
3.项目介绍

1.LeNet-5模型训练

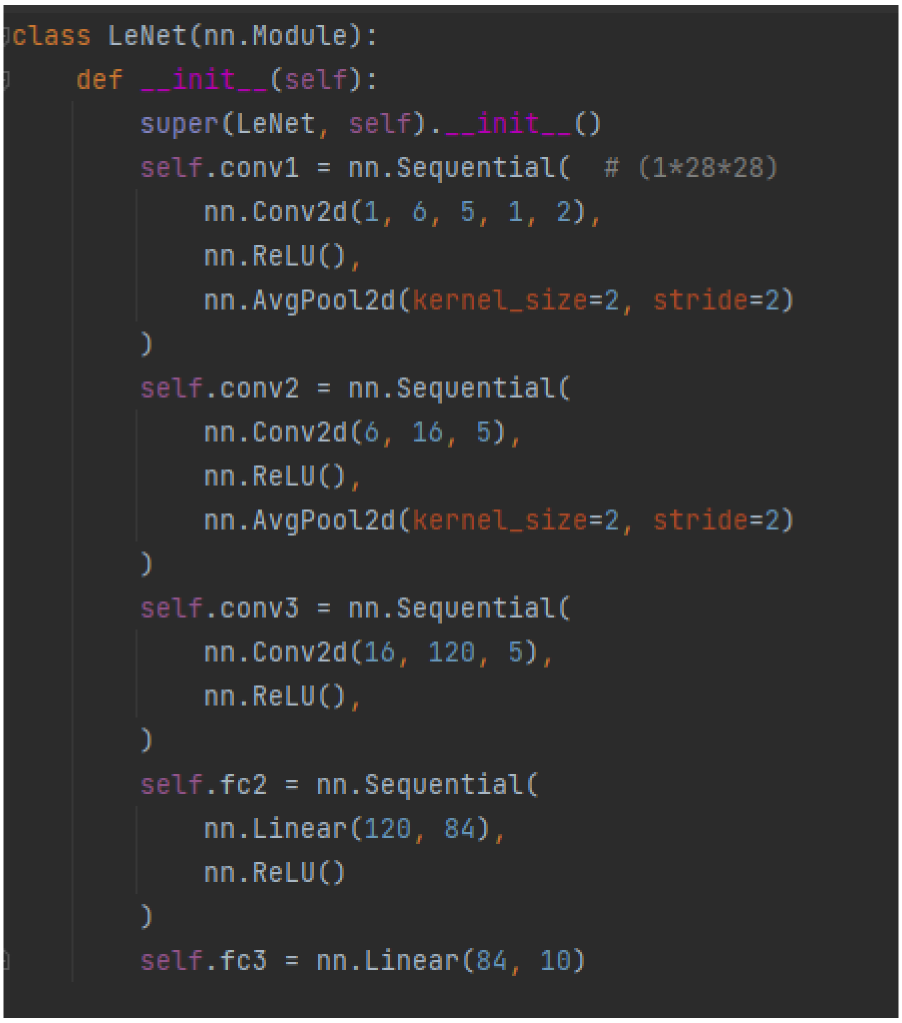
1.模型结构定义

模型训练文件见PyTorch文件夹下的.py文件

首先在PyTorch中搭建LeNet-5模型框架，其模型结构图如下图所示



根据其结构图在PyTorch中定义网络结构，如下：



```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Sequential(  # (1*28*28)
            nn.Conv2d(1, 6, 5, 1, 2),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=2, stride=2)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(6, 16, 5),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=2, stride=2)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(16, 120, 5),
            nn.ReLU(),
        )
        self.fc2 = nn.Sequential(
            nn.Linear(120, 84),
            nn.ReLU()
        )
        self.fc3 = nn.Linear(84, 10)
```

2.数据集导入

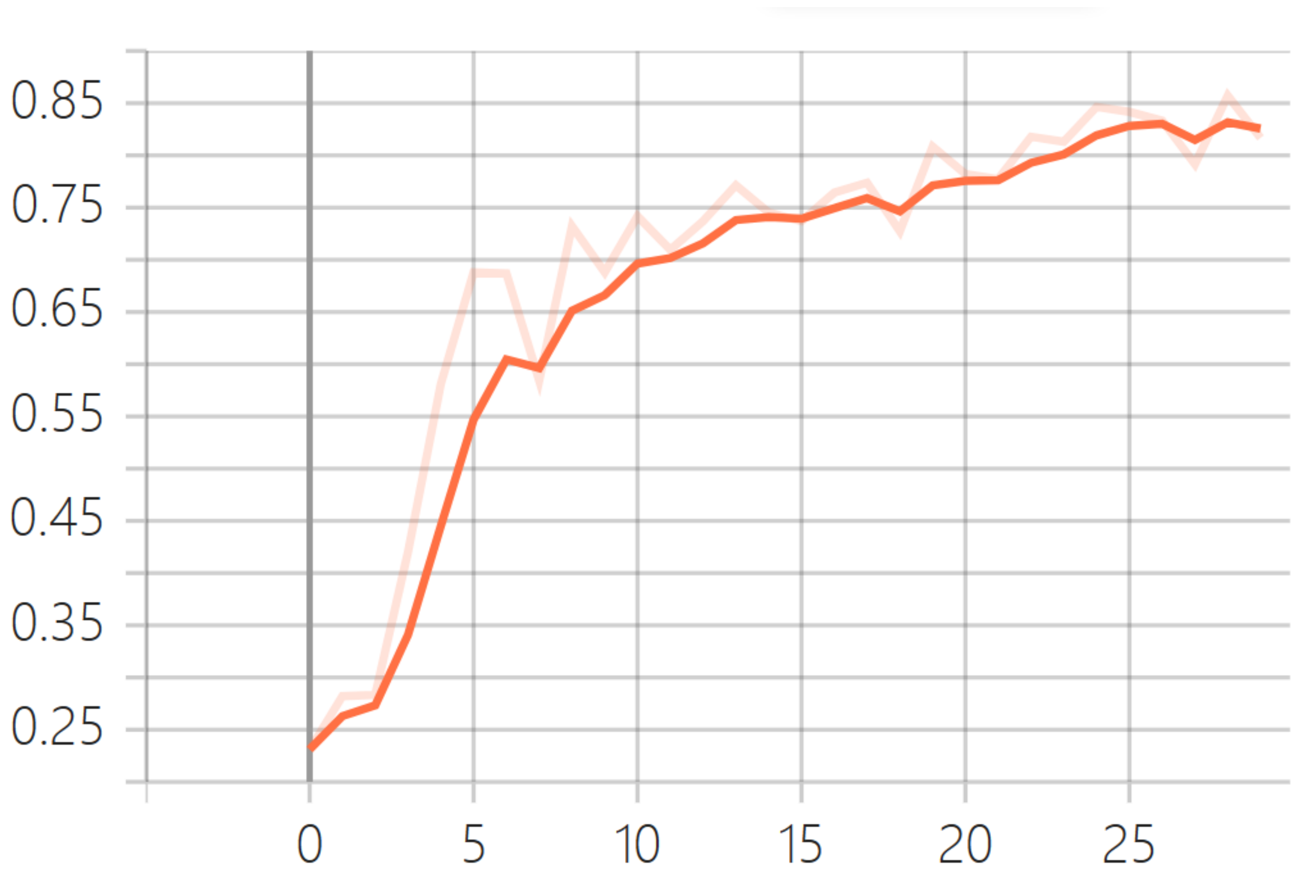
本项目采用了Fashion-MNIST数据集，它包含了10个类别的图像，分别是：t-shirt（T恤），trouser（牛仔裤），pullover（套衫），dress（裙子），coat（外套），sandal（凉鞋），shirt（衬衫），sneaker（运动鞋），bag（包），ankle boot（短靴）。每张图像大小为1x28x28，数据集样张示例如下：

数据集示例

数据集已经过处理，导出成jpg图像文件形式，见dataset文件夹。

3.模型训练

模型具体训练参数定义见PyTorch文件夹下的train.py文件，在经过80次迭代后，模型收敛，其训练过程中的精度曲线如下。

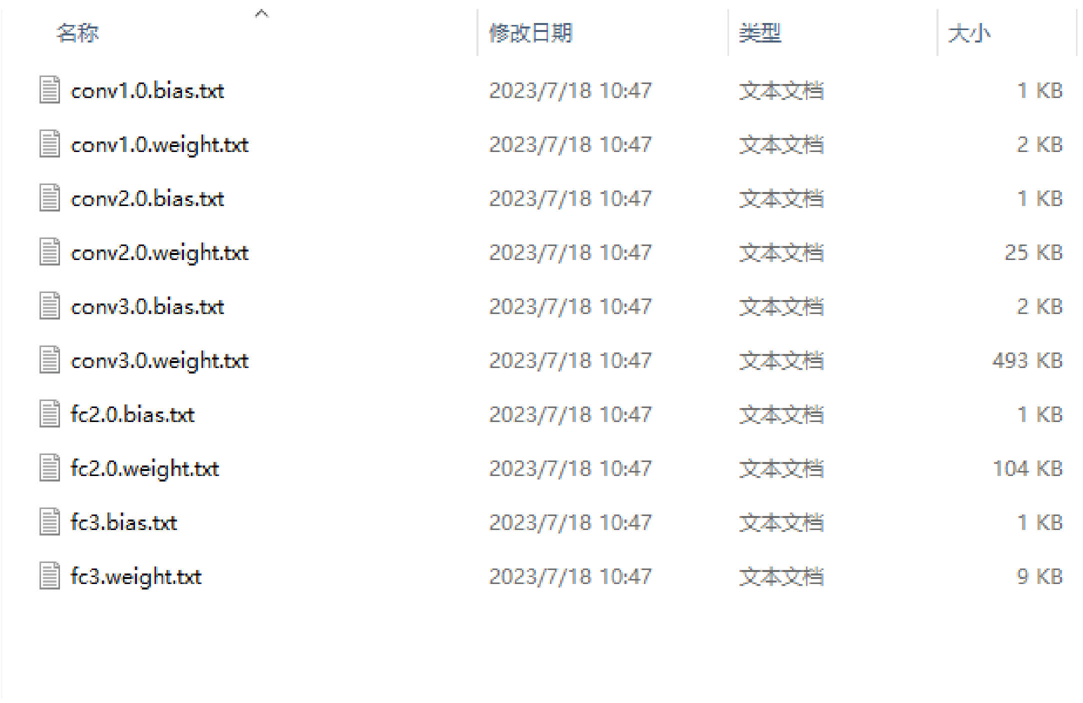


训练完成后，模型在测试集中的识别准确率为84.4%，能够达到基本识别性能要求。

80轮迭代后的模型准确率

4. 模型参数导出

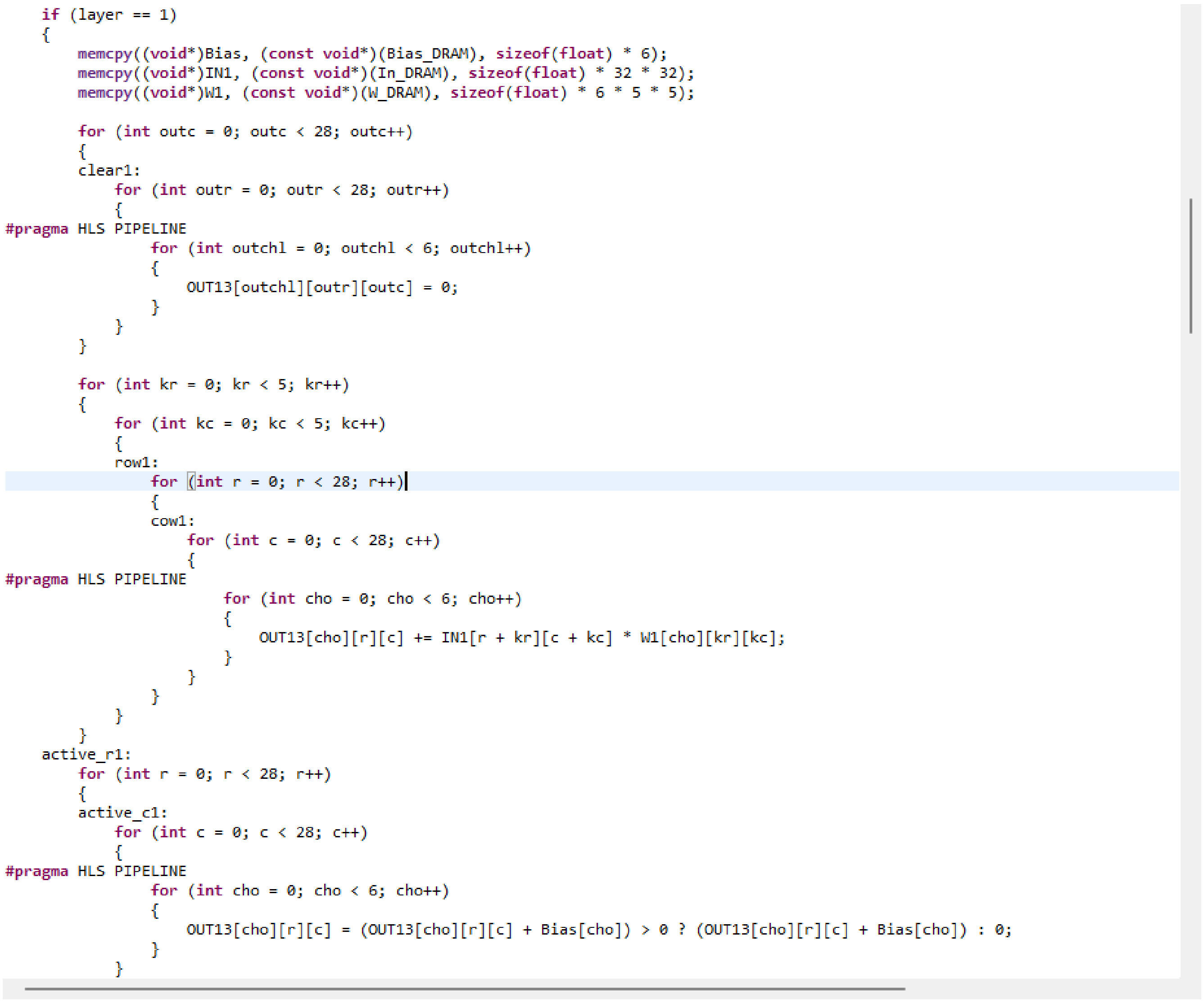
此部分代码见PyTorch文件夹下的extract.py文件，经过数据类型转化、数据展平等处理后，分别将每层的权重参数和偏置参数保存成.txt文件，具体参数文件见parameter文件夹，导出后模型参数如下所示：



名称	修改日期	类型	大小
conv1.0.bias.txt	2023/7/18 10:47	文本文档	1 KB
conv1.0.weight.txt	2023/7/18 10:47	文本文档	2 KB
conv2.0.bias.txt	2023/7/18 10:47	文本文档	1 KB
conv2.0.weight.txt	2023/7/18 10:47	文本文档	25 KB
conv3.0.bias.txt	2023/7/18 10:47	文本文档	2 KB
conv3.0.weight.txt	2023/7/18 10:47	文本文档	493 KB
fc2.0.bias.txt	2023/7/18 10:47	文本文档	1 KB
fc2.0.weight.txt	2023/7/18 10:47	文本文档	104 KB
fc3.bias.txt	2023/7/18 10:47	文本文档	1 KB
fc3.weight.txt	2023/7/18 10:47	文本文档	9 KB

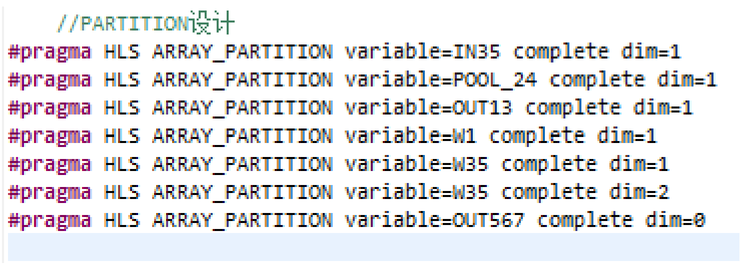
2. 基于VIvado HLS的LeNet硬件加速器IP构建

由于模型足够小，为方便起见其硬件加速器的设计采用模型每层都设计对应的硬件加速电路，而不是可重构的加速器设计，如下图为模型第一层卷积加速器设计，其在每层卷积中实现了5*5卷积窗口的单层pipeline设计。



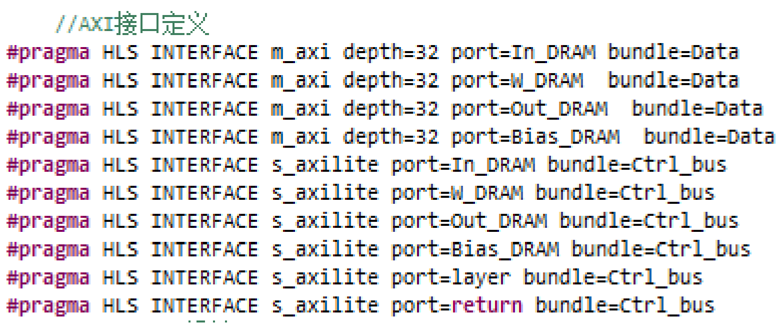
```
    if (layer == 1)
    {
        memcpy((void*)Bias, (const void*)(Bias_DRAM), sizeof(float) * 6);
        memcpy((void*)IN1, (const void*)(In_DRAM), sizeof(float) * 32 * 32);
        memcpy((void*)W1, (const void*)(W_DRAM), sizeof(float) * 6 * 5 * 5);

        for (int outc = 0; outc < 28; outc++)
        {
        clear1:
            for (int outr = 0; outr < 28; outr++)
            {
#pragma HLS PIPELINE
                for (int outchl = 0; outchl < 6; outchl++)
                {
                    OUT13[outchl][outr][outc] = 0;
                }
            }
        }

        for (int kr = 0; kr < 5; kr++)
        {
            for (int kc = 0; kc < 5; kc++)
            {
            row1:
                for (int r = 0; r < 28; r++)
                {
                cow1:
                    for (int c = 0; c < 28; c++)
                    {
#pragma HLS PIPELINE
                        for (int cho = 0; cho < 6; cho++)
                        {
                            OUT13[cho][r][c] += IN1[r + kr][c + kc] * W1[cho][kr][kc];
                        }
                    }
                }
            }
        }
    active_r1:
        for (int r = 0; r < 28; r++)
        {
        active_c1:
            for (int c = 0; c < 28; c++)
            {
#pragma HLS PIPELINE
                for (int cho = 0; cho < 6; cho++)
                {
                    OUT13[cho][r][c] = (OUT13[cho][r][c] + Bias[cho]) > 0 ? (OUT13[cho][r][c] + Bias[cho]) : 0;
                }
            }
```

为数据并行访问速度，采用partition设计，具体partition对象如下：



```
    //PARTITION设计
#pragma HLS ARRAY_PARTITION variable=IN35 complete dim=1
#pragma HLS ARRAY_PARTITION variable=POOL_24 complete dim=1
#pragma HLS ARRAY_PARTITION variable=OUT13 complete dim=1
#pragma HLS ARRAY_PARTITION variable=W1 complete dim=1
#pragma HLS ARRAY_PARTITION variable=W35 complete dim=1
#pragma HLS ARRAY_PARTITION variable=W35 complete dim=2
#pragma HLS ARRAY_PARTITION variable=OUT567 complete dim=0
```

网络加速器的输入输出结构定义如下，对于输入图像和feature map、卷积结果输出和卷积核权重和前置值输入等大数据量的传输接口，采用axi_full总线，而其余参数配置接口则采用axi_lite总线。



```
    //AXI接口定义
#pragma HLS INTERFACE m_axi depth=32 port=In_DRAM bundle=Data
#pragma HLS INTERFACE m_axi depth=32 port=W_DRAM  bundle=Data
#pragma HLS INTERFACE m_axi depth=32 port=Out_DRAM  bundle=Data
#pragma HLS INTERFACE m_axi depth=32 port=Bias_DRAM  bundle=Data
#pragma HLS INTERFACE s_axilite port=In_DRAM bundle=Ctrl_bus
#pragma HLS INTERFACE s_axilite port=W_DRAM bundle=Ctrl_bus
#pragma HLS INTERFACE s_axilite port=Out_DRAM bundle=Ctrl_bus
#pragma HLS INTERFACE s_axilite port=Bias_DRAM bundle=Ctrl_bus
#pragma HLS INTERFACE s_axilite port=layer bundle=Ctrl_bus
#pragma HLS INTERFACE s_axilite port=return bundle=Ctrl_bus
```

其余层具体设计代码见conv.cpp文件，经过C synthsis后，其综合信息如下图所示。

2c3f13620c6509b552ad6efb5f655f8

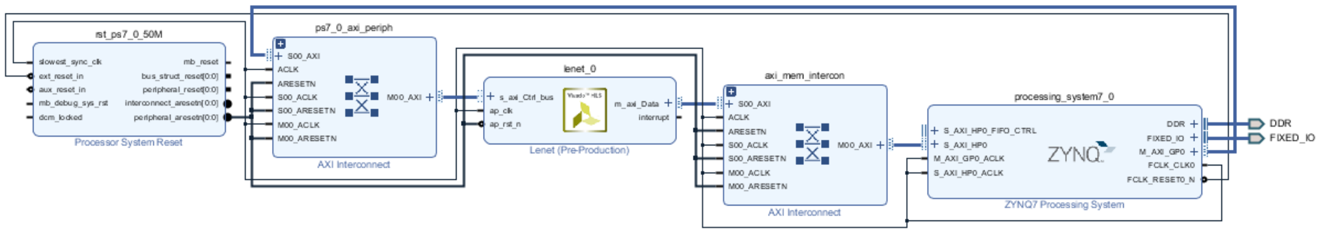
其预估资源消耗如下：

加速器资源消耗

最后以IP核的形式导出该加速器设计，以便后续使用，ip核文件见xilinx_com_hls_lenet_1_0.zip。

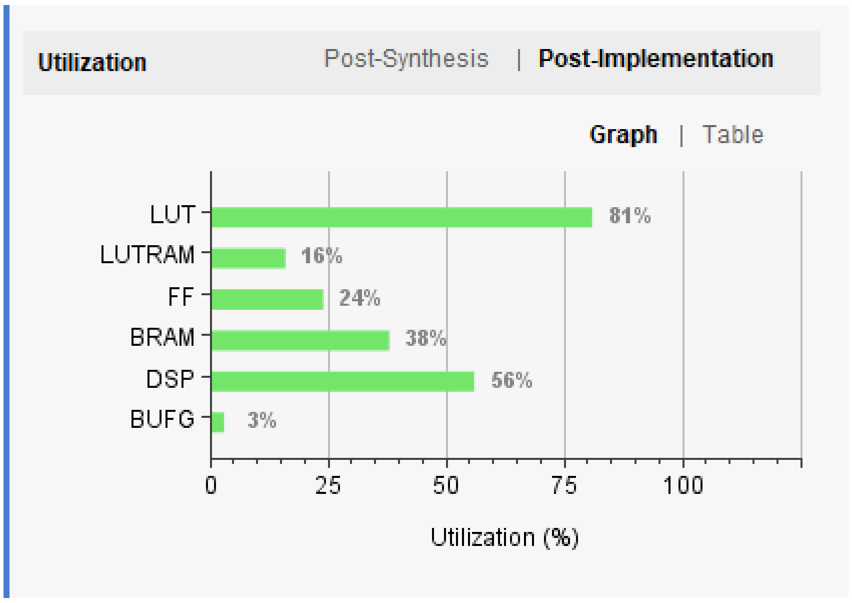
3. 基于Vivado的系统设计

在Vivado中导入上一步中所设计的LeNet网络加速器，并加入zynq处理器IP，BD连接如下

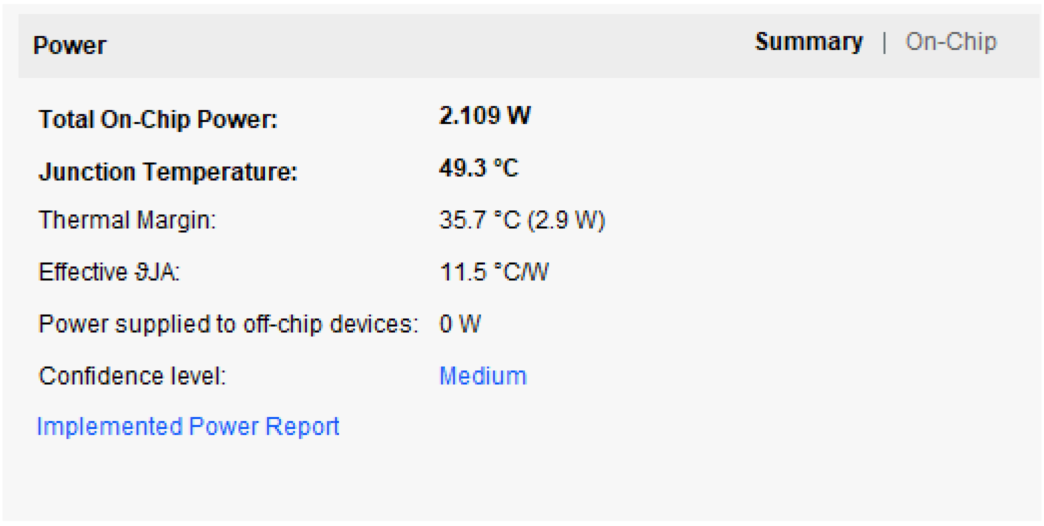


最后，经过synthesis,implementation,generate bitstream后，导出.bit和.hwh文件

综合后，系统资源消耗如下



power表现如下：



Total On-Chip Power:	**2.109 W**
Junction Temperature:	**49.3 °C**
Thermal Margin:	35.7 °C (2.9 W)
Effective ϑJA:	11.5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Implemented Power Report

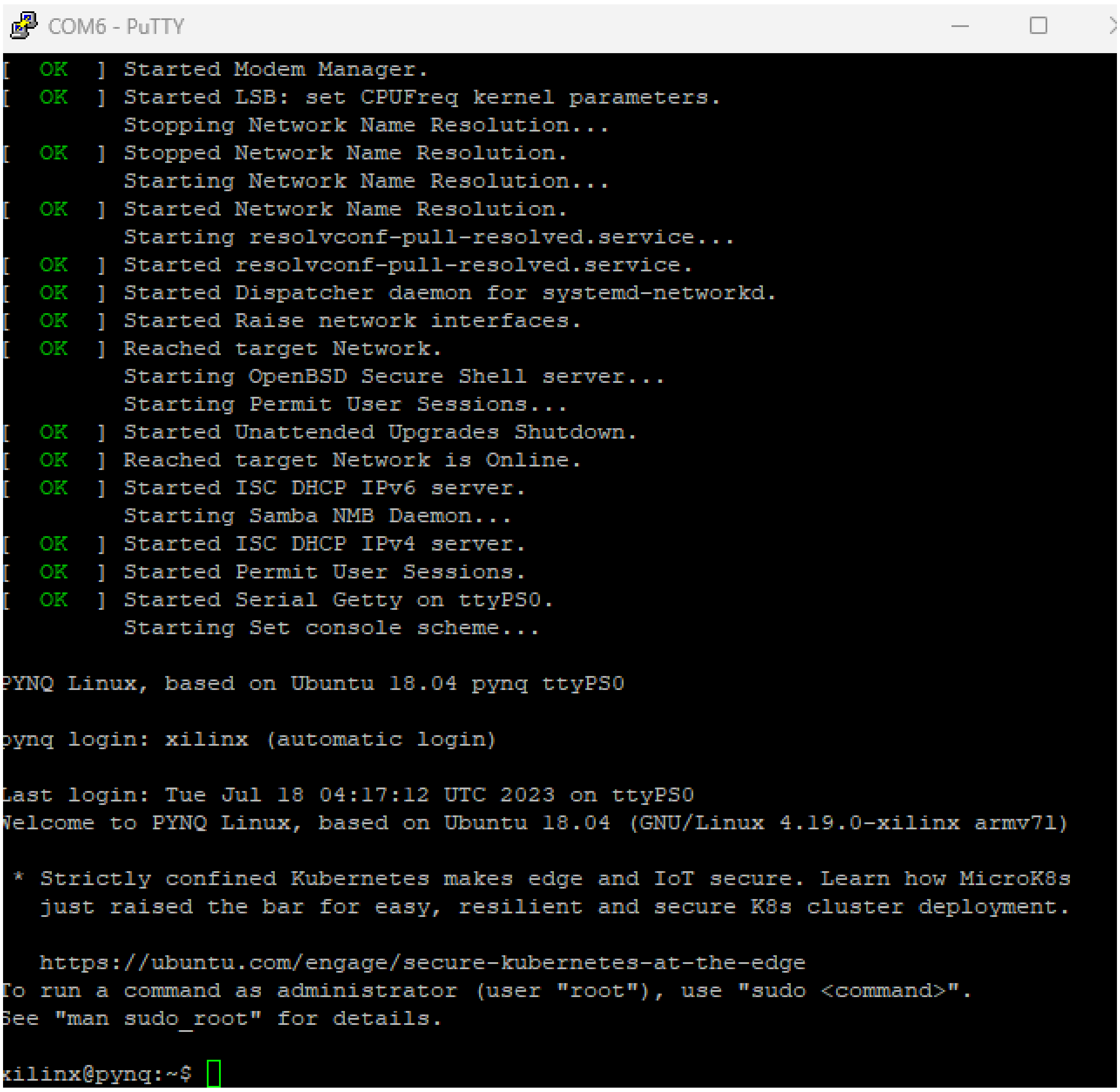
4. 构建PYNQ设计

1. PYNQ框架移植完成后，插入SD卡，设置板卡的启动方式为SD卡启动，插入网线并与主机或路由器相连接，实物连接示意如下。

1875bd9839496dcae88b61251f63dfe

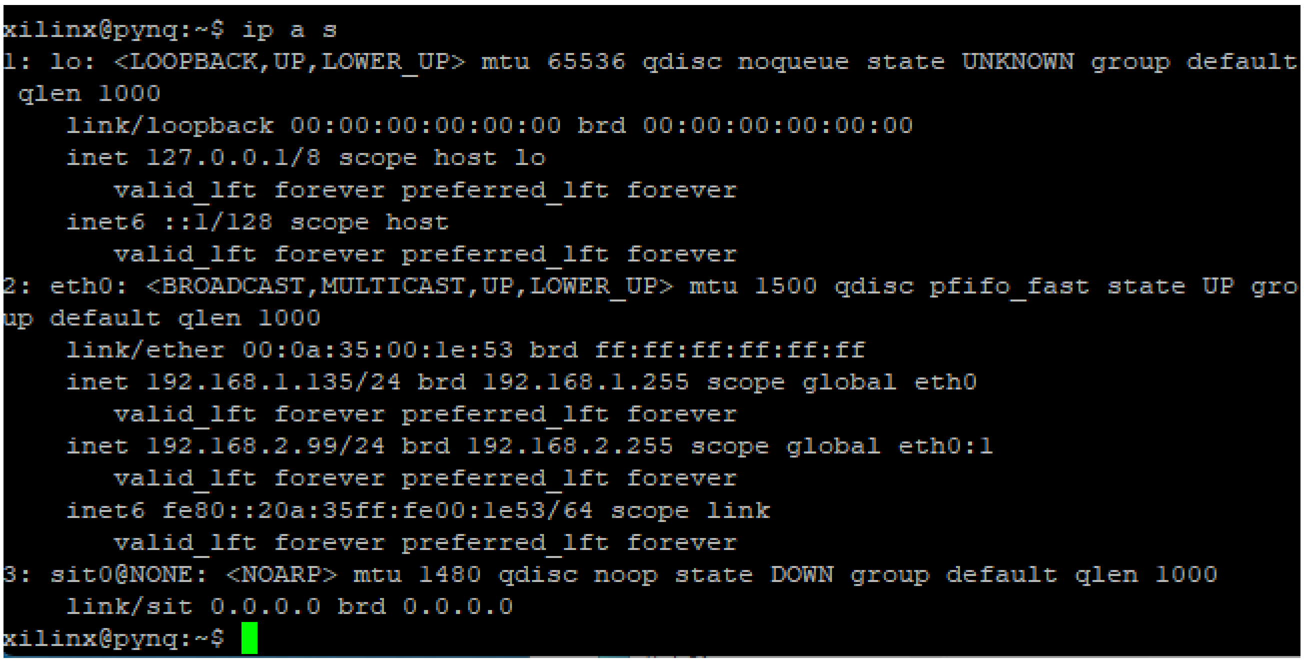
2. 启动pynq

将板卡串口与主机相连接，打开串口终端Putty， 若PYNQ成功移植，在上电后终端后会打印出系统信息，如下，pynq已经启动。


COM6 - PuTTY

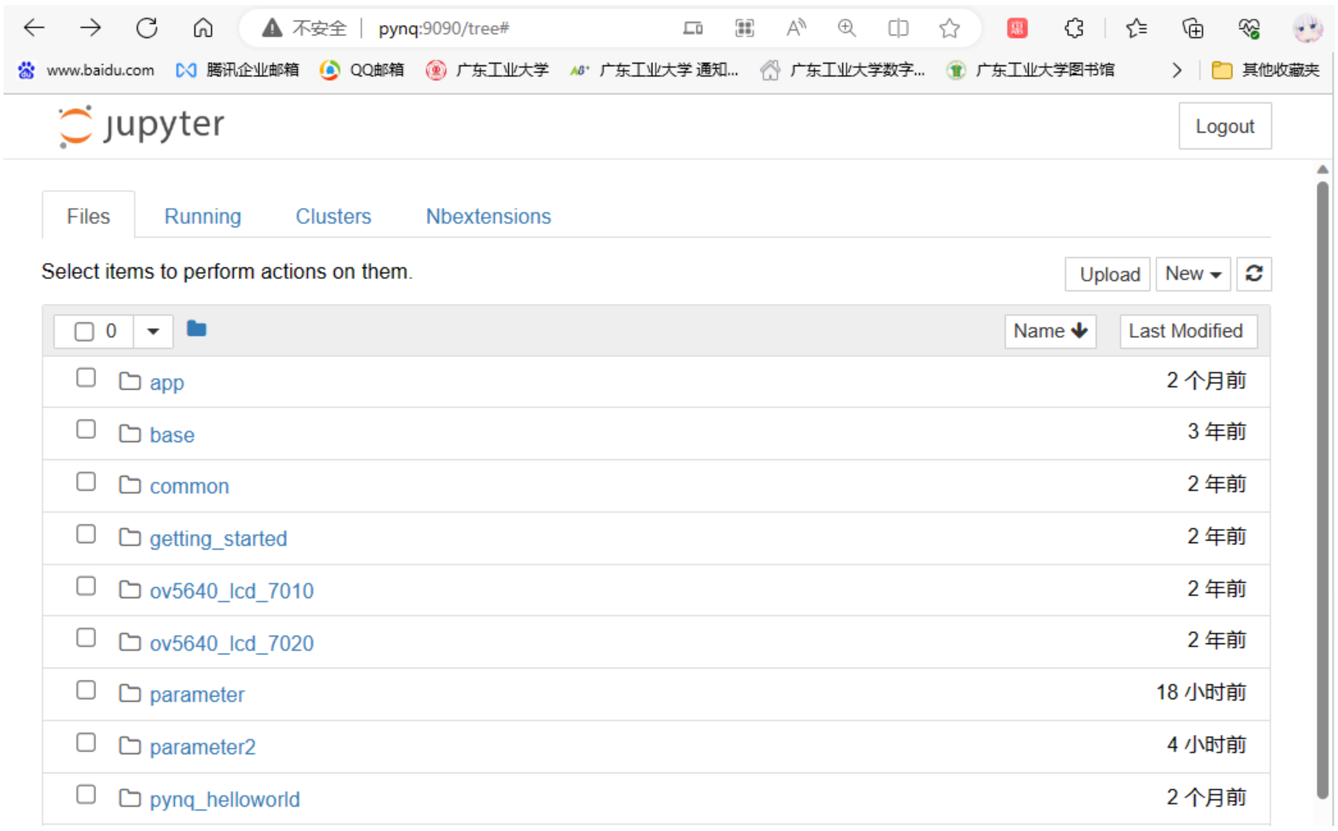
```
[  OK  ] Started Modem Manager.
[  OK  ] Started LSB: set CPUFreq kernel parameters.
         Stopping Network Name Resolution...
[  OK  ] Stopped Network Name Resolution.
         Starting Network Name Resolution...
[  OK  ] Started Network Name Resolution.
         Starting resolvconf-pull-resolved.service...
[  OK  ] Started resolvconf-pull-resolved.service.
[  OK  ] Started Dispatcher daemon for systemd-networkd.
[  OK  ] Started Raise network interfaces.
[  OK  ] Reached target Network.
         Starting OpenBSD Secure Shell server...
         Starting Permit User Sessions...
[  OK  ] Started Unattended Upgrades Shutdown.
[  OK  ] Reached target Network is Online.
[  OK  ] Started ISC DHCP IPv6 server.
         Starting Samba NMB Daemon...
[  OK  ] Started ISC DHCP IPv4 server.
[  OK  ] Started Permit User Sessions.
[  OK  ] Started Serial Getty on ttyPS0.
         Starting Set console scheme...

PYNQ Linux, based on Ubuntu 18.04 pynq ttyPS0

pynq login: xilinx (automatic login)

Last login: Tue Jul 18 04:17:12 UTC 2023 on ttyPS0
Welcome to PYNQ Linux, based on Ubuntu 18.04 (GNU/Linux 4.19.0-xilinx armv7l)

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

xilinx@pynq:~$ 
```

在终端中输入指令ip a s, 可查看该pynq设计的ip地址信息，如下：



```
xilinx@pynq:~$ ip a s
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
 qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP gro
up default qlen 1000
    link/ether 00:0a:35:00:1e:53 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.135/24 brd 192.168.1.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet 192.168.2.99/24 brd 192.168.2.255 scope global eth0:1
       valid_lft forever preferred_lft forever
    inet6 fe80::20a:35ff:fe00:1e53/64 scope link
       valid_lft forever preferred_lft forever
3: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
xilinx@pynq:~$
```

在浏览器中输入该ip地址，即可打开jupyter页面



Select items to perform actions on them.

Name	Last Modified
app	2 个月前
base	3 年前
common	2 年前
getting_started	2 年前
ov5640_lcd_7010	2 年前
ov5640_lcd_7020	2 年前
parameter	18 小时前
parameter2	4 小时前
pynq_helloworld	2 个月前

3. 上传构建系统文件

将jupyter文件夹下的parameter文件夹，lenet.bit,lenet.hwh,lenet.ipynb文件以及测试图像上传至jupyter中。

4. 系统运行

在jupyter中打开lenet.ipynb文件，开始运行

首先导入一些必要的包，并根据lenet.hwh文件中的地址指示定义lenet加速器IP的输入输出信号地址

```
from pynq import Overlay
from pynq import Xlnk
import numpy as np
import cv2
from PIL import Image as PIL_Image
from PIL import ImageEnhance
from PIL import ImageOps
from scipy import misc

overlay = Overlay('my_lenet.bit')
top = overlay.top_fun_0

def lenet(pic_in, w_in, out, bias, layer):
    top.write(0x10,pic_in)#pic_in
    top.write(0x18,w_in)#w
    top.write(0x20,out)#out
    top.write(0x28,bias)#bias
    top.write(0x30,layer)#layer
    top.write(0x00,0x01)#写入1  启动ip
    while(top.read(0)!=0x04):#等待计算完成
        a=1
    return 0
```

根据lenet网络模块每层的输入输出数据结构，定义一段连续的数据地址

```
xlnk = Xlnk()
input_pic = xlnk.cma_array(shape=(32*32,), dtype=np.float32)
POOL24_DRAM = xlnk.cma_array(shape=(16*14*14,), dtype=np.float32)
C5_DRAM = xlnk.cma_array(shape=(120,), dtype=np.float32)
C6_DRAM = xlnk.cma_array(shape=(84,), dtype=np.float32)
C7_DRAM = xlnk.cma_array(shape=(10,), dtype=np.float32)
W_CONV1 = xlnk.cma_array(shape=(6*5*5,), dtype=np.float32)
W_CONV3 = xlnk.cma_array(shape=(16*6*5*5,), dtype=np.float32)
W_CONV5 = xlnk.cma_array(shape=(120*16*5*5,), dtype=np.float32)
b_conv1 = xlnk.cma_array(shape=(6,), dtype=np.float32)
b_conv3 = xlnk.cma_array(shape=(16,), dtype=np.float32)
b_conv5 = xlnk.cma_array(shape=(120,), dtype=np.float32)
WFC6 = xlnk.cma_array(shape=(10080,), dtype=np.float32)
WFC7 = xlnk.cma_array(shape=(840,), dtype=np.float32)
b_fc6 = xlnk.cma_array(shape=(84,), dtype=np.float32)
b_fc7 = xlnk.cma_array(shape=(10,), dtype=np.float32)
```

将模型的参数保存至本地

```
W_CONV1_buff = np.loadtxt('parameter2/conv1.0.weight.txt')
W_CONV3_buff = np.loadtxt('parameter2/conv2.0.weight.txt')
W_CONV5_buff = np.loadtxt('parameter2/conv3.0.weight.txt')

b_conv1_buff = np.loadtxt('parameter2/conv1.0.bias.txt')
b_conv3_buff = np.loadtxt('parameter2/conv2.0.bias.txt')
b_conv5_buff = np.loadtxt('parameter2/conv3.0.bias.txt')

WFC6_buff = np.loadtxt('parameter2/fc2.0.weight.txt')
WFC7_buff = np.loadtxt('parameter2/fc3.weight.txt')

b_fc6_buff = np.loadtxt('parameter2/fc2.0.bias.txt')
b_fc7_buff = np.loadtxt('parameter2/fc3.bias.txt')
```

载入测试图像

```
img_load = PIL_Image.open('vali_data2/0.jpg').convert("L")
img_load = img_load.resize((28, 28),PIL_Image.ANTIALIAS)

img_numpy = np.asarray(img_load)

img_numpy= np.pad(img_numpy, ((2, 2), (2, 2)), 'constant', constant_values=(255, 255))
input_pic.physical_address
for i in range(32):
    for j in range(32):
        input_pic[i*32+j] = 1 - float(img_numpy[i][j]) / 255.0;
```

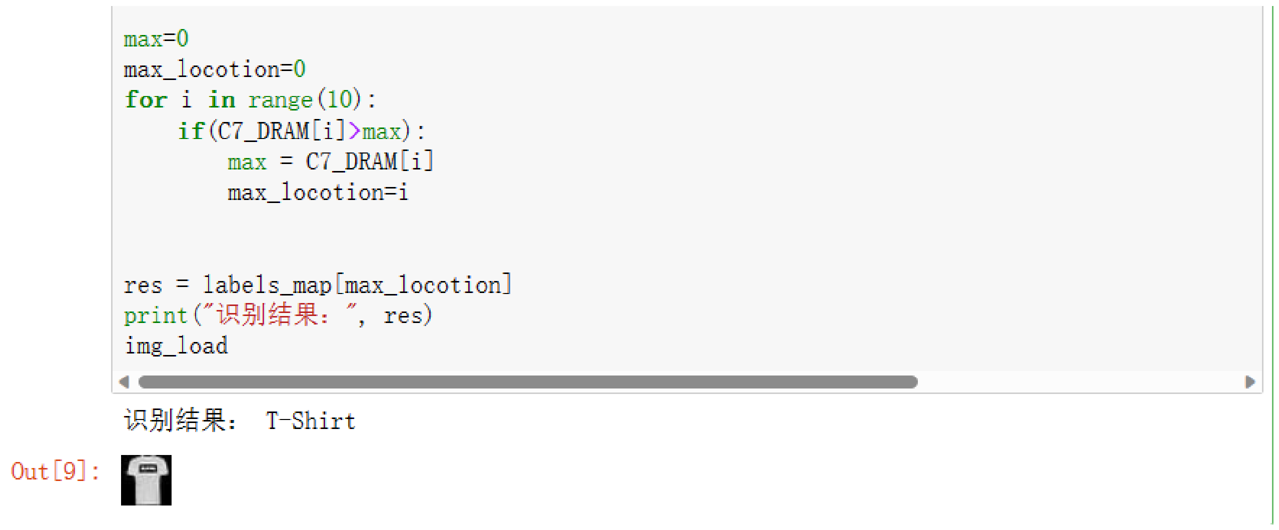
图像输入，逐层计算

```
lenet( input_pic.physical_address, W_CONV1.physical_address, POOL24_DRAM.physical_address,
b_conv1.physical_address, 1)
lenet( POOL24_DRAM.physical_address, W_CONV3.physical_address,
POOL24_DRAM.physical_address, b_conv3.physical_address, 2)
lenet( POOL24_DRAM.physical_address, W_CONV5.physical_address, C5_DRAM.physical_address,
b_conv5.physical_address, 3)
lenet( C5_DRAM.physical_address, WFC6.physical_address, C6_DRAM.physical_address,
b_fc6.physical_address, 4)
lenet( C6_DRAM.physical_address, WFC7.physical_address, C7_DRAM.physical_address,
b_fc7.physical_address, 5)
```

最后输出识别结果，并输出导入图像进行验证

```
max=0
max_locotion=0
for i in range(10):
    if(C7_DRAM[i]>max):
        max = C7_DRAM[i]
        max_locotion=i

res = labels_map[max_locotion]
print("识别结果:", res)
img_load
```

输出结果如下:



```
max=0
max_locotion=0
for i in range(10):
    if(C7_DRAM[i]>max):
        max = C7_DRAM[i]
        max_locotion=i


res = labels_map[max_locotion]
print("识别结果：", res)
img_load
```

```
识别结果： T-Shirt
```

Out[9]: