Lab 6：使用NaiveTPU实现神经网络推理

# 1 实验目标

1. 使用NaiveTPU实现MLP、LeNet神经网络推理

# 2 实验环境

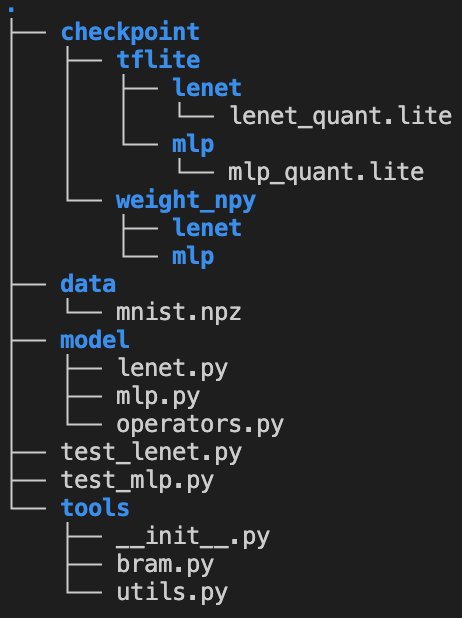
1. Vivado 2019.2 / Vitis 2019.2
2. ZYNQ 7020开发板及其配件
3. ZYNQ上的Linux系统
4. ZYNQ中的Python以及Numpy库
5. 本实验过程中使用到的参考代码与数据文件可在实验附件中获取。
6. **注意：本实验指导书中给出的步骤仅为示意步骤作为参考，每人遇到的情况可能有差异，如果遇到问题可根据实际情况进行探索，或向助教寻求帮助。**

# 3 实验要求

1. 按照5.1中的实验步骤完成实验，并得到正确的测试结果。
2. 回答5.2中提出的问题。（必做）
3. 回答5.3中的附加题，会根据回答结果酌情考虑加分。（选做）
4. 撰写实验报告。实验报告命名：学号+姓名+实验四。（实验报告撰写细节可参考“实验报告撰写格式”）
5. 将实验报告按时交至课程中心作业处。

# 4 实验说明

## 4.1 代码文件目录结构说明



* ./：根目录
  + test\_mlp.py、test\_lenet.py：测试代码入口文件
* checkpoint：存储模型及权重等参数
  + tflite：存储tensorflow lite量化后的模型，可以使用Netron打开查看模型结构、权重、量化参数等信息
  + weight\_npy：存储模型权重信息
* data：存储数据集
* model：模型类实现、神经网络量化推理算子实现
  + mlp.py、lenet.py：模型类实现
  + operators.py：神经网络量化推理算子实现、Matmul实现、量化参数
* tools：存储工具函数和类
  + bram.py：BRAM配置、读写
  + utils.py：工具函数/类（Logger、FLAG、Timer、load\_mnist）

## 4.2 代码介绍

代码调用堆栈如下图所示：



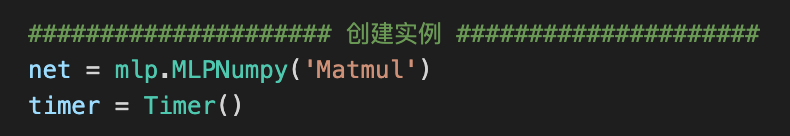
代码运行的测试入口为根目录下的test\_mlp.py或者test\_lenet.py，上图中以test\_lenet.py为例，但是mlp网络同理。接下来对网络测试主要流程进行说明。

### 4.2.1 test\_lenet.py

该代码为测试神经网络的主要入口代码，首先加载数据集：



创建网络、时间统计器的实例，网络实例用于执行推理的时候调用，时间统计器统计器用于统计神经网络推理时间。



执行神经网络推理并计算推理延时和准确率。

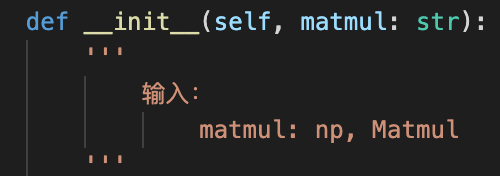


1. 从测试集中取出输入图片和标签
2. 神经网络执行推理
3. 判断此次推理的正确性

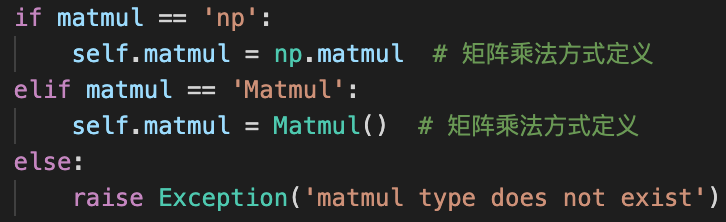
### 4.2.2 model/lenet.py

该文件中LeNetNumpy类用于实现LeNet量化推理函数，其中\_\_init\_\_方法执行一些初始化操作，forward方法执行网络前向传播操作。

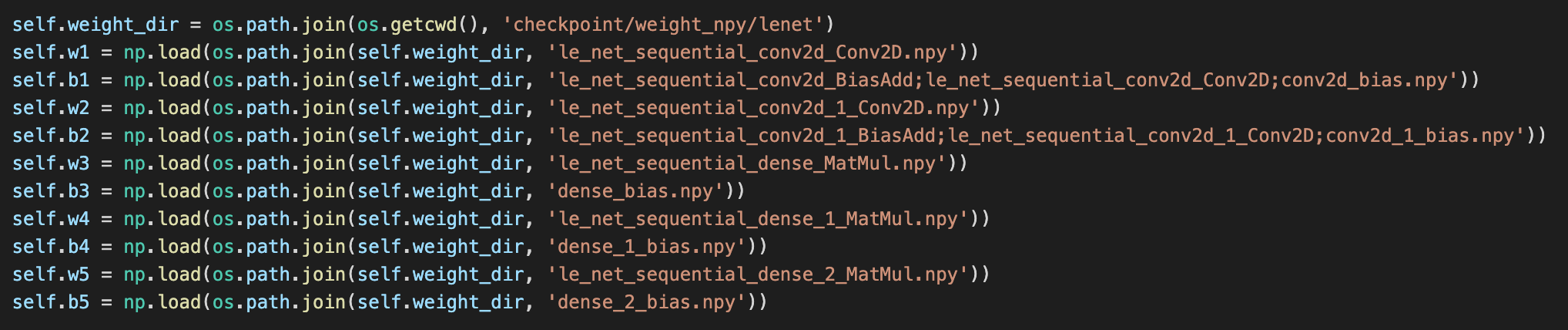
\_\_init\_\_方法中拥有一个输入参数，matmul，该参数为字符串，用于决定矩阵乘法的方式，传入np使用np.matmul方式；传入Matmul则使用同学们Lab4、Lab5中自己实现的Matmul接口。（请注意，需要同学们用己的Matmul类实现替换model/operators.py文件中的Matmul类）



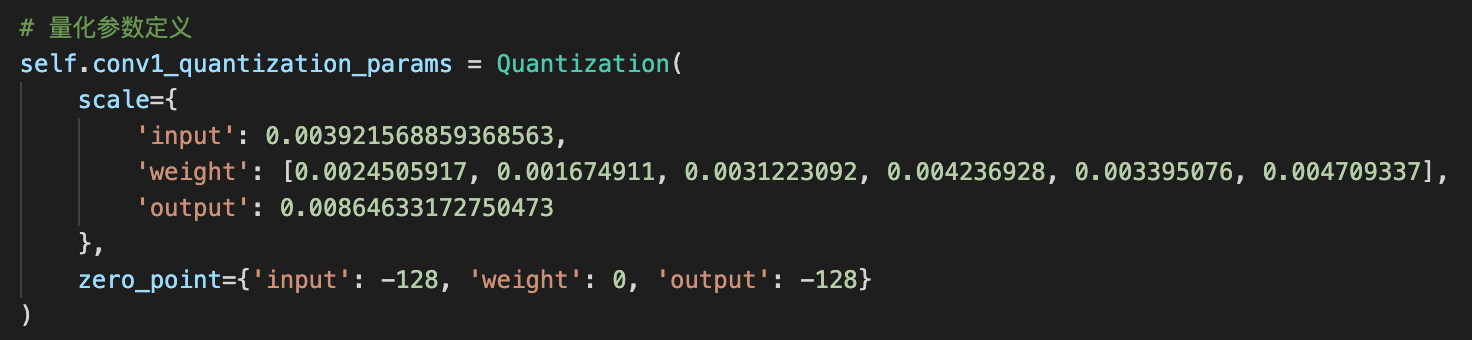
根据输入字符串选择矩阵乘法方式：



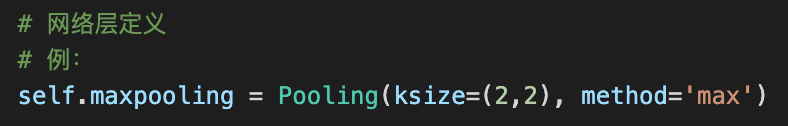
读取神经网络参数，包括各层的weight和bias。



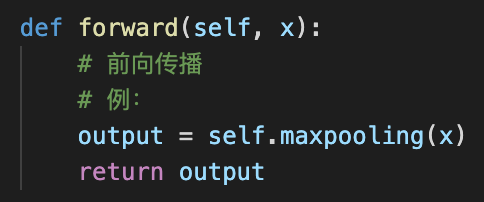
量化参数定义，在本课程中，只有全连接层（Dense）和卷积层（Conv2D）需要传入量化参数*（想了解量化推理过程的同学可以查看PPT）*：



定义网络所需要的算子，此处以Pooling为例：

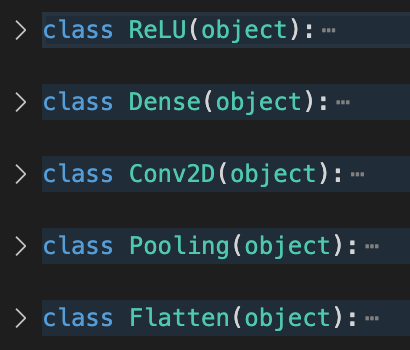


forward()函数中使用定义好的算子执行网络前向传播过程，即网络推理过程。



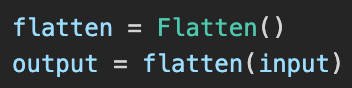
### 4.2.3 model/operators.py

该文件中包含神经网络量化推理算子的具体实现、Matmul矩阵乘法类、Quantization量化参数收集处理类。其中，量化推理算子包含：

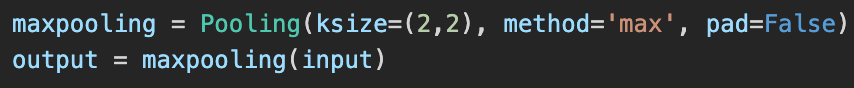


其中，需要注意，ReLU算子不需要使用，ReLU过程已经蕴含在了各个量化推理过程中。

Flatten()算子使用方法：



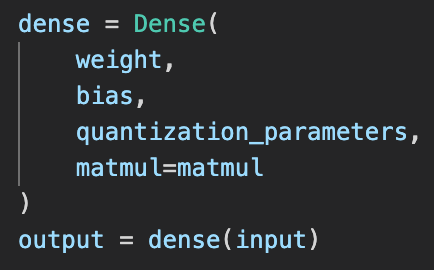
Pooling算子使用方法：



其中，各参数含义如下：

* ksize：滤波器大小
* method：max 或者 mean，即选择最大池化或者平均池化
* pad：是否进行padding操作，即使得输入和输出大小保持一致

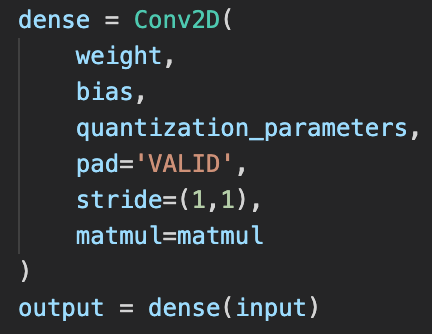
Dense算子使用方法：



其中，各参数含义如下：

* weight：权重
* bias：偏移值
* quantization\_parameters：量化参数，Quantization类型
* matmul：矩阵乘法接口，np.matmul或者Matmul()

Conv2D算子使用方法：



其中，各参数含义如下：

* weight：权重
* bias：偏移值
* quantization\_parameters：量化参数，Quantization类型
* pad：VALID模式或者SAME模式

### 4.2.4 tools/bram.py

bram.py中实现了BramConfig以及BRAM读写类，该部分中在Lab4中已经介绍过了，此处不再复述。此处说明一下代码变化的部分。

1. BRAM类中read接口变化：



read接口中offset支持两种输入模式，除了BramConfig中配置的offset的key值外，可以传如类型为int的值，表示将名称为block\_name的BRAM块内的偏移量作为起始地址开始读数据。（仅作为接口扩展，非必须使用项）

1. read接口中读取方式的修改



原方法使用read\_bytes单字节读取，耗时长，因此在神经网络测试中修改为read方法，可实现4字节同时读取。

*（注：该修改存在一定的问题，若使用pl\_simulate.py脚本进行测试会出现读取结果不正确的问题，因此若同学们希望使用pl\_simulate.py需要将该部分修改为read\_bytes的方法）*

1. BRAM类中write接口变化



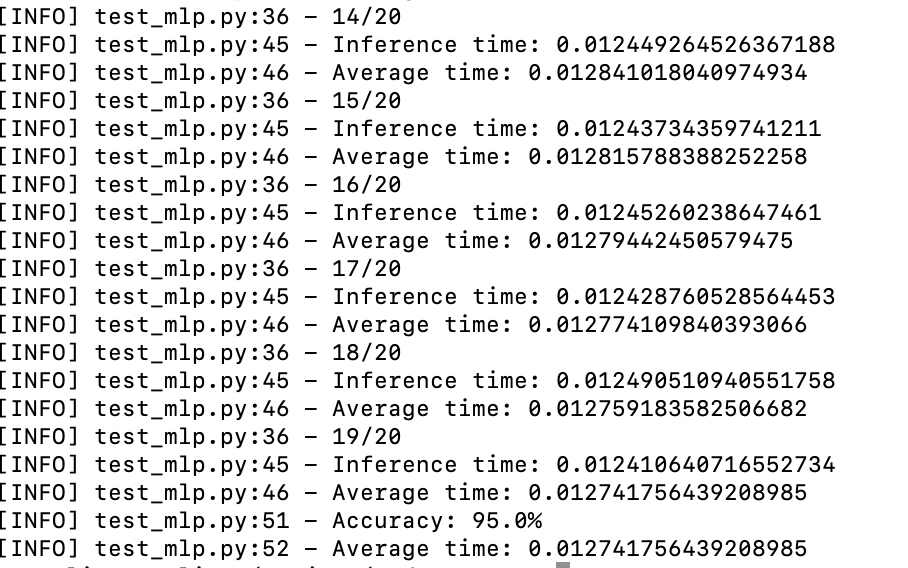
和1)中变化一致，提供灵活的块内起始地址偏移方式，此处不再赘述。

## 4.3 测试结果

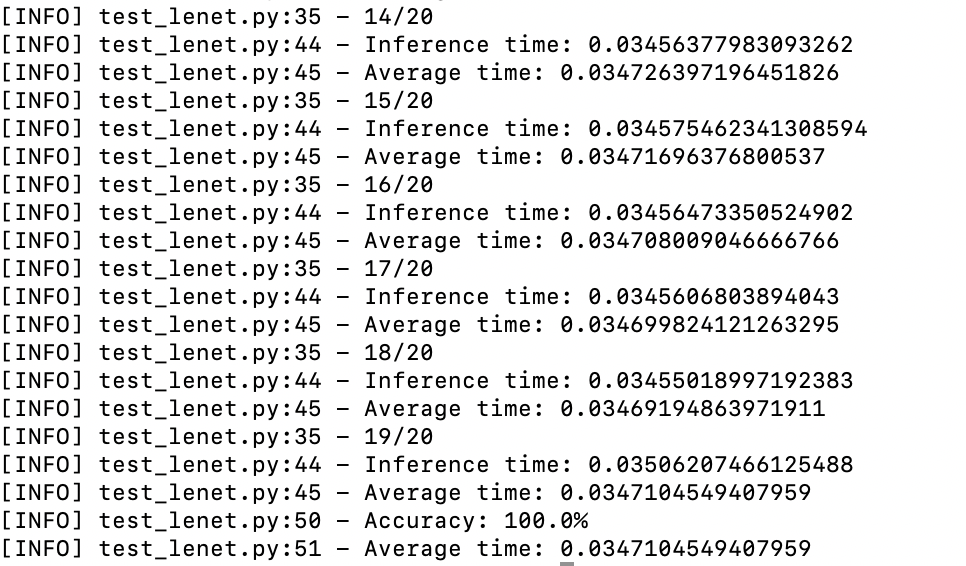
测试结果中存在以下几个字段：

* 0/20：表示在测试20张图片中编号为0的图片（编号范围为0 ~ 19）
* Inference time：单次神经网络前向推理的时间
* Average time：平均每次神经网络前向推理的时间
* Accuracy：识别准确率

MLP测试结果：



LeNet测试结果：



# 5 实验内容

## 5.1 实验步骤

FPGA侧：

1. 烧写Lab5中的bitstream至开发板

ARM侧：

1. 替换model/operators.py中的Matmul类为自己的Matmul实现（该实现在Lab4、Lab5中均有用到）
2. 补全model/mlp.py以及model/lenet.py中网络定义以及前向推理部分（注意，ReLU激活函数不需要实现，直接跳过即可，因为在量化过程中已经被融合）
3. 运行test\_mlp.py以及test\_lenet.py，完成神经网络测试

## 5.2 问题

1. 对比以下两个方案的执行时间，对比执行时间：
   1. 在ARM上使用Numpy运行矩阵乘法，执行神经网络
   2. 使用FPGA运行矩阵乘法，执行神经网络

请回答方案a和b哪一个执行时间更快？为什么？（详细阐述分析的思路、过程）

## 5.3 附加题

1. 能否优化使用FPGA运行神经网络的时间？若能，请提出具体的方案，最好可以实现并验证效果
2. 当ARM侧采用跳地址存储的方式实现BRAM写入时运行时间如何？给出对比结果并分析
3. 参考Lab5中的问题4，当总线数据位宽为64bit或者128bit时，神经网络推理运行时间如何？给出对比结果并加以分析