- 100 分标准：建立 VGG19 网络后，给定 VGG19 的网络参数值和输入图像，在完成正确分类的基础上，可以计算得到正确的每层计算量（乘法数量和加法数量），同时分别正确计算每个卷积层的前向传播时间。

3.1.7 实验思考

在实验中请思考如下问题：

1) 在实现深度神经网络基本单元时，如何确保一个层的实现是正确的？

2) 在实现深度神经网络后，如何确保整个网络的实现是正确的？如果是网络中的某个层计算有误，如何快速定位到有错误的层？

3) 如何计算深度神经网络的每层计算量（乘法数量和加法数量）？如何计算整个网络的前向传播时间和网络中每层的前向传播时间？深度神经网络的每层计算量和每层前向传播时间之间有什么联系？

3.2 基于 DLP 平台实现图像分类

3.2.1 实验目的

巩固卷积神经网络的设计原理，能够使用 pycnml 库提供的 Python 接口将 VGG19[3] 网络模型移植到 DLP 上，实现图像分类。具体包括：

1) 使用 pycnml 库实现卷积、ReLU 等基本网络模块。

2) 使用提供的 pycnml 库实现 VGG19 网络。

3) 分析评估 DLP 和 CPU 运行 VGG19 进行图像分类的性能。

实验工作量：约需 1 个小时。

3.2.2 实验环境

硬件环境：DLP。

软件环境：pycnml 库、Python 编译环境及相关的扩展库，包括 Python 2.7.12，Pillow 3.4.2，Scipy 0.18.1，NumPy 1.11.2。

数据集：ImageNet。

3.2.3 实验内容

本实验调用 DLP 平台上的 pycnml 库来搭建 VGG19 网络进行图像分类。模块划分方式与第3.1节实验类似，分为数据加载模块、基本单元模块、网络结构模块和网络推断模块。

3.2.4 实验步骤

3.2.4.1 数据加载模块

数据加载模块实现数据读取和预处理，程序示例如图3.10所示。由于 Python 语言限制，调用 pycnml 库的 Python 接口前需要将数据类型从 numpy.float32 转换为 numpy.float64。

```
# file: vgg19_demo.py
def load_image(self, image_dir):
    # 读取图像数据
    self.image = image_dir
    image_mean = np.array([123.68, 116.779, 103.939])
    print('Loading and preprocessing image from ' + image_dir)
    input_image = scipy.misc.imread(image_dir)
    input_image = scipy.misc.imresize(input_image,[224,224,3])
    input_image = np.array(input_image).astype(np.float32)
    input_image -= image_mean
    input_image = np.reshape(input_image, [1]+list(input_image.shape))
    # input dim [N, channel, height, width]
    input_image = np.transpose(input_image, [0, 3, 1, 2])
    self.input_data = input_image.flatten().astype(np.float)
    # 将图片加载到 DLP 上
    self.net.setInputData(input_data)
```

图 3.10 VGG19 的数据加载模块 DLP 实现示例

3.2.4.2 基本单元模块

VGG19 中包含的卷积层、ReLU 层、最大池化层、全连接层和 softmax 层可以直接调用 pycnml 库来实现对应层的初始化、参数加载、前向传播等操作。pycnml 的使用方式可以参考第2.2.2.2节的示例。

3.2.4.3 网络结构模块

与第2.2.5.3节类似，网络结构模块也使用一个类来定义 VGG19 网络，可以直接使用 pycnml 封装好的基本模块接口来定义。网络结构模块的程序示例如下面程序所示，其中定义了以下成员函数：

- 神经网络初始化：初始化部分创建 pycnml.CnmlNet() 的实例 net。
- 建立神经网络结构：首先加载数据和权重的量化参数，然后调用 net 中创建网络层的接口定义整个神经网络的拓扑结构，并设定每层的超参数。

```
# file: vgg19_demo.py
class VGG19(object):
    def __init__(self):
        # 初始化网络，创建pycnml.CnmlNet() 实例 net
        self.net = pycnml.CnmlNet()
        self.input_quant_params = []
        self.filter_quant_params = []

    def build_model(self,
                    param_path='../data/vgg19_data/imagenet-vgg-verydeep-19.mat',
```

```
                    quant_param_path='../data/vgg19_data/vgg19_quant_param_new.npz'):
        self.param_path = param_path
        # 加载量化参数
        params = np.load(quant_param_path)
        input_params = params['input']
        filter_params = params['filter']
        for i in range(0, len(input_params), 2):
            self.input_quant_params.append(pycnml.QuantParam(int(input_params[i]), float(
    input_params[i+1])))
        for i in range(0, len(filter_params), 2):
            self.filter_quant_params.append(pycnml.QuantParam(int(filter_params[i]), float(
    filter_params[i+1])))
        # TODO: 使用 net 的 createXXXLayer 接口搭建 VGG19 网络
        self.net.setInputShape(1, 3, 224, 224)
        # conv1_1
        self.net.createConvLayer('conv1_1', 64, 3, 1, 1, 1, self.input_quant_params[0])
        # relu1_1
        self.net.createReLuLayer('relu1_1')
        # conv1_2
        self.net.createConvLayer('conv1_2', 64, 3, 1, 1, 1, self.input_quant_params[1])
        # relu1_2
        self.net.createReLuLayer('relu1_2')
        # pool1
        ____________________________
        ____________________________
        ____________________________
        # fc8
        self.net.createMlpLayer('fc8', 1000, self.input_quant_params[18])
        # softmax
        self.net.createSoftmaxLayer('softmax', 1)
```

3.2.4.4 网络推断模块

DLP 实现的 VGG19 的网络推断模块程序示例如下面程序所示。同样划分为参数加载、前向传播、推断函数主体等操作，这些操作使用 VGG19 神经网络类的成员函数来定义：

- 神经网络参数的加载：VGG19 网络参数包括卷积层和全连接层的权重和偏置。首先读取量化过的 VGG19 预训练模型文件，然后循环遍历 net 中的所有层，如果当前层是卷积或全连接层，则将对应的权重、偏置以及量化参数加载到层中。将模型文件读入内存之后，也需要做两方面的处理：一方面，训练得到的模型中权重维度为 $H \times W \times C_{in} \times C_{out}$，而 DLP 处理网络层时权重的维度为 $C_{out} \times C_{in} \times H \times W$，因此需要对读取的权重做一次维度交换，使其与 DLP 中权重的维度一致；另一方面，需要手动将 numpy 数据类型转为 np.float64 类型。

- 神经网络的前向传播：将经过预处理的图像输入，net.forward 函数会自动遍历调用 net 中的每一层的前向传播函数，并返回最后一层的结果。

- 神经网络推断函数主体：与第3.1.5.4节的 CPU 实现类似，给定一张经过预处理的图像数据，执行网络的前向传播函数即可得到 VGG19 预测的 1000 个类别的分类概率，然后选取概率最高的类别作为网络最终预测的分类类别。

```
# file: vgg19_demo.py
def load_model(self): # 加载神经网络参数
    print('Loading parameters from file ' + self.param_path)
    params = scipy.io.loadmat(self.param_path)
    self.image_mean = params['normalization'][0][0][0]
    self.image_mean = np.mean(self.image_mean, axis=(0, 1))
    count = 0
    for idx in range(self.net.size()):
        if 'conv' in self.net.getLayerName(idx):
            weight, bias = params['layers'][0][idx][0][0][0][0]
            # matconvnet: weights dim [height, width, in_channel, out_channel]
            # ours: weights dim [out_channel, in_channel, height, width]
            weight = np.transpose(weight,[3,2,0,1]).flatten().astype(np.float)
            bias = bias.reshape(-1).astype(np.float)
            self.net.loadParams(idx, weight, bias, self.filter_quant_params[count])
            count += 1
        if 'fc' in self.net.getLayerName(idx):
            weight, bias = params['layers'][0][idx-1][0][0][0][0]
            weight = weight.reshape([weight.shape[0]*weight.shape[1]*weight.shape[2], weight.shape
    [3]])
            weight = np.transpose(weight, [1, 0]).flatten().astype(np.float)
            bias = bias.reshape(-1).astype(np.float)
            self.net.loadParams(idx, weight, bias, self.filter_quant_params[count])
            count += 1

def forward(self): # 神经网络的前向传播
    return self.net.forward()

def get_top5(self, label):
    # 打印推理的时间
    start = time.time()
    self.forward()
    end = time.time()
    print('inference time: %f'%(end - start))
    result = self.net.getOutputData()
    # 打印 top1/5 结果
    top1 = False
    top5 = False
    print('------ Top 5 of ' + self.image + ' ------')
    prob = sorted(list(result), reverse=True)[:6]
    if result.index(prob[0]) == label:
        top1 = True
    for i in range(5):
        top = prob[i]
        idx = result.index(top)
        if idx == label:
            top5 = True
        print('%f - '%top + self.labels[idx].strip())
    return top1,top5

def evaluate(self, file_list): # 推断函数主体
    top1_num = 0
    top5_num = 0
    total_num = 0
    # 读取标签
    self.labels = []
    with open('synset_words.txt', 'r') as f:
        self.labels = f.readlines()
    # 记录推断所有图片的总时间
    start = time.time()
    with open(file_list, 'r') as f:
        file_list = f.readlines()
```

```
        total_num = len(file_list)
        for line in file_list:
            image = line.split()[0].strip()
            label = int(line.split()[1].strip())
            self.load_image(image)
            top1,top5 = self.get_top5(label) # 获取推断结果
            if top1 :
                top1_num += 1
            if top5 :
                top5_num += 1
    end = time.time()
    print('Global accuracy : ')
    print('accuracy1: %f (%d/%d) '%(float(top1_num)/float(total_num), top1_num, total_num))
    print('accuracy5: %f (%d/%d) '%(float(top5_num)/float(total_num), top5_num, total_num))
    print('Total execution time: %f'%(end - start))
```

3.2.4.5 实验完整流程

完成以上所有模块后，就可以调用上述模块中的函数，在 DLP 上运行 VGG19 网络实现给定图像的分类。完整的流程程序示例如图3.11所示。与第3.1节的 CPU 实现类似，首先实例化 VGG19 网络的类，其次通过网络结构模块建立网络结构，设置每层的超参数，然后读取模型文件为每层加载参数，最后输入待分类的图像并调用推断模块获得网络的预测结果。

```
# file: vgg19_demo.py
if __name__ == '__main__':
    vgg = VGG19()
    vgg.build_model()
    vgg.load_model()
    vgg.evaluate('file_list')
```

图 3.11 VGG19 进行图像分类的完整流程 DLP 实现示例

3.2.5 实验考核

本实验仍然选择图3.9所示猫咪的图像进行分类测试，该猫咪图像的真实类别为 tabby cat，对应 ImageNet 数据集类别编号的 281。若实验结果将该图像的类别编号判断为 281，则可认为判断正确。性能评判标准为预测猫咪类别时 VGG19 网络 forward 函数运行的时间。

本实验的评分标准设定如下：

- 100 分标准：使用 pycnml 搭建 VGG19 网络，给定 VGG19 的网络参数值和输入图像，可以得到正确的 Softmax 层输出结果和正确的图像分类结果。

3.2.6 实验思考

在实验中请思考如下问题：

1) 阅读 pycnml/src/net.cpp 中 forward 函数的实现，比较 DLP 在计算哪些层时比 CPU 要快，为什么？

2) 观察 forward 函数的实现，在 VGG19 网络的一次完整推断过程中，DLP 每执行完一层都需要和 CPU 交互一次，这种交互是否有必要？有什么办法可以避免这种交互吗？

3.3 非实时图像风格迁移

3.3.1 实验目的

掌握深度学习的训练方法，能够使用 Python 语言基于 VGG19 网络模型实现非实时图像风格迁移[6]。具体包括：

1) 加深对卷积神经网络的理解，利用 VGG19 模型进行图像特征提取。

2) 使用 Python 语言实现风格迁移中相关风格和内容损失函数的计算，加深对非实时风格迁移的理解。

3) 使用 Python 语言实现非实时风格迁移中迭代求解风格化图像的完整流程，为后续实现实时风格迁移并建立更复杂的综合实验奠定基础。

实验进程：20%。

实验工作量：约 300 行代码，约需 3 个小时。

3.3.2 背景介绍

图像风格迁移根据给定的目标风格图像和目标内容图像求解风格迁移图像，使风格迁移图像在风格上与目标风格图像一致，在内容上与目标内容图像一致。图像风格迁移分为非实时风格迁移与实时风格迁移。非实时风格迁移仅对当前给定的目标内容图像进行风格化，实现较为简单，但需要对每张输入的内容图像做训练。而实时风格迁移训练一个模型，对任意内容图像均可以生成风格化图像，实现相对复杂。

风格迁移通常用 VGG 模型（如 VGG19）提取图像的特征，然后计算风格迁移图像与目标风格（内容）图像的风格（内容）损失作为风格（内容）损失函数。在非实时风格迁移中，计算风格（内容）损失并进行反向传播，获得风格迁移图像的梯度，更新风格迁移图像。通过多次迭代，不断减小风格迁移图像与目标风格（内容）图像的风格（内容）损失，最终获得风格化后的图像。在非实时风格迁移中，通常用加入噪声的目标内容图像作为初始的风格迁移图像。完整的非实时风格迁移过程见图3.12。下面详细介绍非实时风格迁移中使用的内容损失函数和风格损失函数。

内容损失函数

内容损失层用于计算风格迁移图像的第 l 层特征图与目标内容图像的第 l 层特征图的内容损失。假设风格迁移图像的第 l 层特征图为 $\boldsymbol{X}^l$，是维度为 $N \times C \times H \times W$ 的四维矩阵，N、C、H、W 分别代表输入特征图的样本个数（在本实验中 $N = 1$）、通道数、高和宽。假设目标内容图像的第 l 层特征图为 $\boldsymbol{Y}^l$，维度同样为 $N \times C \times H \times W$，则目标内容图像的该层特征图可视为是内容损失层的标记。内容损失 L 用 $\boldsymbol{X}^l$ 和 $\boldsymbol{Y}^l$ 之间的欧式距离表示，计算公