智能计算系统实验教程

第二章 神经网络实验

中国科学院计算技术研究所
张凯歌 工程师
zhangkaige@ict.ac.cn
手机：18810366140

常用工具

- Windows系统

1. MobaXterm
2. xshell
3. vscode

- Linux系统
- macOS系统

提纲

- 1.基于三层神经网络实现手写数字分类
- 2.基于DLP平台实现手写数字分类

1.基于三层神经网络实现手写数字分类

- 1.1实验目的
- 1.2背景知识
- 1.3实验环境
- 1.4实验内容
- 1.5实验步骤
- 1.6实验评估
- 1.7实验思考

1.1实验目的

- 掌握神经网络的设计原理
- 熟练掌握神经网络的训练和推断方法
- 实现一个三层全连接神经网络模型：训练和推理

1.2背景知识

- 1.神经网络的组成
- **全连接层**：每个输入节点都与前一层的所有输出节点相连
- **ReLU激活函数**：数学形式为：f(x) = max(0, x)
- **Softmax损失函数**：Softmax损失函数常用于多分类问题中。它的作用是将网络的输出转化为概率分布，并计算该概率分布与真实标签之间的差异。
- 2.神经网络训练
- 3.精度评估
- 使用测试集的平均分类正确率来判断分类结果的精度

  Top1,Top5

1.3实验环境

ssh root@paas.extrotec.com -p 31238

硬件环境

```
root@notebook-devenviron-0629-092335--212633-so6ak8-notebook-0:~# cnmon
Fri Aug  2 02:20:54 2024
+-----------------------------------------------------------------------------+
| CNMON v5.10.12                                              Driver v5.10.12 |
+-----------------------------+----------------------+----------------------+
| Card  VF  Name       Firmware |              Bus-Id | Util        Ecc-Error |
| Fan   Temp      Pwr:Usage/Cap |       Memory-Usage | SR-IOV   Compute-Mode |
|=============================+======================+======================|
| 0     /   MLU370-M8    v1.1.4 |         0000:69:00.0 | 0%                 0 |
|  0%   22C         50 W/ 300 W |     0 MiB/ 42396 MiB | N/A          Default |
+-----------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  Card  MI  PID     Command Line                                MLU Memory Usage |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
root@notebook-devenviron-0629-092335--212633-so6ak8-notebook-0:~# lscpu
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
CPU(s):                  128
On-line CPU(s) list:     0-127
Thread(s) per core:      2
Core(s) per socket:      32
Socket(s):               2
NUMA node(s):            8
Vendor ID:               HygonGenuine
CPU family:              24
Model:                   1
Model name:              Hygon C86 7285 32-core Processor
Stepping:                1
CPU MHz:                 1999.980
BogoMIPS:                3999.96
Virtualization:          AMD-V
L1d cache:               32K
L1i cache:               64K
L2 cache:                512K
L3 cache:                8192K
NUMA node0 CPU(s):       0-7,64-71
NUMA node1 CPU(s):       8-15,72-79
NUMA node2 CPU(s):       16-23,80-87
NUMA node3 CPU(s):       24-31,88-95
NUMA node4 CPU(s):       32-39,96-103
NUMA node5 CPU(s):       40-47,104-111
NUMA node6 CPU(s):       48-55,112-119
```

```
root@notebook-devenviron-0629-092335--212633-so6ak8-notebook-0:~# free -h
              total        used        free      shared  buff/cache   available
Mem:           503G         51G         47G        4.9M        405G        451G
Swap:            0B          0B          0B
```

- 下载地址为http://yann.lecun.com/exdb/mnist/
- 数据集MNIST（训练集：60000 测试集：10000），28*28

1.3实验环境

- 软件环境

absl-py	1.4.0
appdirs	1.4.4
astroid	2.15.5
attrs	23.1.0
audioread	3.0.0
boto3	1.18.1
botocore	1.21.1
cachetools	5.3.1
cambricon-dali	0.7.0
certifi	2021.5.30
cffi	1.15.1
charset-normalizer	2.0.3
click	8.1.3
cpplint	1.6.1
cycler	0.11.0
Cython	0.29.23
decorator	5.1.1
dill	0.3.6
distlib	0.3.1
DLLogger	0.1.0
filelock	3.0.12
fonttools	4.38.0
future	0.18.3
google-auth	2.21.0
google-auth-oauthlib	0.4.6
grpcio	1.56.0
idna	3.2
importlib-metadata	4.0.1
isort	5.11.5
jmespath	0.10.0
joblib	1.2.0
kiwisolver	1.4.4

dill	0.3.6
distlib	0.3.1
DLLogger	0.1.0
filelock	3.0.12
fonttools	4.38.0
future	0.18.3
google-auth	2.21.0
google-auth-oauthlib	0.4.6
grpcio	1.56.0
idna	3.2
importlib-metadata	4.0.1
isort	5.11.5
jmespath	0.10.0
joblib	1.2.0
kiwisolver	1.4.4
lazy-object-proxy	1.9.0
librosa	0.8.1
llvmlite	0.39.1
magicmind	1.3.0
Markdown	3.4.3
MarkupSafe	2.1.3
matplotlib	3.5.3
mccabe	0.7.0
ninja	1.11.1
numba	0.56.4
numpy	1.20.1
oauthlib	3.2.2
opencv-python	4.7.0.72
packaging	23.1
pandas	1.3.5
Pillow	7.2.0
pip	22.3.1
platformdirs	3.5.1

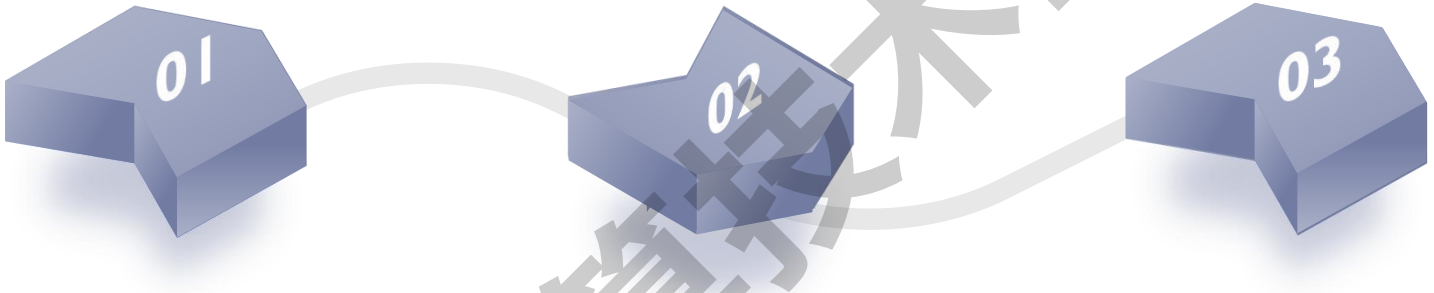
scipy	1.7.3
orn	0.12.2
encepiece	0.1.91
ptools	59.5.0
	1.16.0
dfile	0.12.1
orboard	2.11.2
orboard-data-server	0.6.1
orboard-plugin-wit	1.8.1
adpoolctl	3.1.0
	0.4.12
nizers	0.9.3
i	2.0.1
kit	0.11.8
h	1.6.0
h-mlu	1.14.0-torch1.6
hvision	0.7.0a0+489752
	4.61.2
sformers	3.5.1
d-ast	1.5.4
ng	3.7.4.3
ng-extensions	3.10.0.0
ib3	1.26.6
ualenv	20.4.6
zeug	2.2.3
l	0.38.4
t	1.15.0
	0.1.8
	3.4.1

1.4实验内容

- 设计三层神经网络

1.5实验步骤

补全 layer_1.py、mnist_mlu_cpu.py 代码



环境申请

```
# 登录云平台
ssh root@xxx.xxx.xxx.xxx -p xxxxx
# 进入 code_chap_2_3_student 目录
cd /opt/code_chap_2_3/code_chap_2_3_student
```

代码实现

```
# 进入实验目录
cd exp_2_1_mnist_mlp
# 补全 layers_1.py, mnist_mlp_cpu.py
vim stu_upload/layers_1.py
vim stu_upload/mnist_mlp_cpu.py
```

运行实验

```
# 运行完整实验
python main_exp_2_1.py
```

代码示例 2.1　MNIST 数据集文件的读取和预处理

```python
# file: mnist_mlp_cpu.py
def load_mnist(self, file_dir, is_images = 'True'):
    bin_file = open(file_dir, 'rb')
    bin_data = bin_file.read()
    bin_file.close()
    if is_images: # 读取图像数据
        fmt_header = '>iiii'
        magic, num_images, num_rows, num_cols = struct.unpack_from(fmt_header, bin_data, 0)
    else: # 读取标记数据
        fmt_header = '>ii'
        magic, num_images = struct.unpack_from(fmt_header, bin_data, 0)
        num_rows, num_cols = 1, 1
    data_size = num_images * num_rows * num_cols
    mat_data = struct.unpack_from('>' + str(data_size) + 'B', bin_data, struct.calcsize(fmt_header))
    mat_data = np.reshape(mat_data, [num_images, num_rows * num_cols])
    return mat_data
```

代码示例 2.2　MNIST 子数据集的读取和预处理

```python
# file: mnist_mlp_cpu.py
def load_data(self):
    # TODO: 调用函数 load_mnist 读取和预处理 MNIST 中训练数据和测试数据的图像和标记
    train_images = self.load_mnist(os.path.join(MNIST_DIR, TRAIN_DATA), True)
    train_labels = _____________
    test_images = _____________
    test_labels = _____________
    self.train_data = np.append(train_images, train_labels, axis=1)
    self.test_data = np.append(test_images, test_labels, axis=1)
```

1.6实验评估

- 本实验的评估标准设定如下：
- • 60 分标准：给定全连接层、ReLU 层、Softmax 损失层的前向传播的输入矩阵、参数值、反向传播的输入，可以得到正确的前向传播的输出矩阵、反向传播的输出和参数梯度。
- • 80 分标准：实现正确的三层神经网络，并进行训练和推断，使最后训练得到的模型在 MNIST 测试数据集上的平均分类正确率高于 92%。
- • 90 分标准：实现正确的三层神经网络，并进行训练和推断，通过调整与训练相关的超参数，使最后训练得到的模型在 MNIST 测试数据集上的平均分类正确率高于 95%。
- • 100 分标准：在三层神经网络基础上设计自己的神经网络结构，并进行训练和推断，使最后训练得到的模型在 MNIST 测试数据集上的平均分类正确率高于 98%

1.7实验思考

- 1) 在实现神经网络基本单元时，如何确保一个网络层的实现是正确的?
- 2) 在实现神经网络后，如何在不改变网络结构的条件下提高精度?
- 3) 如何通过修改网络结构提高精度？可以从哪些方面修改网络结构?

2.基于DLP平台实现手写数字分类

- 2.1实验目的
- 2.2背景知识
- 2.3实验环境
- 2.4实验内容
- 2.5实验步骤
- 2.6实验评估
- 2.7实验思考

2.1实验目的

一. 熟悉深度学习处理器DLP平台的使用；

二. 将2.1节的神经网络推断移植到DLP平台

01 利用 pycnnl 库中的 Python 接口搭建手写数字分类的三层神经网络。

02 熟悉在 DLP 上运行神经网络的流程，为在后续章节详细学习 DLP 高性能库以及智能编程语言打下基础

03 与第2.1节的实验进行比较，了解 DLP 相对于 CPU 的优势和劣势。

2.2背景知识

- Python 接口的深度学习编程库 pycnnl

pycnnl

深度学习编程库 pycnnl 通过调用 DLP 上的 CNNL 库中的高性能算子实现了全连接层、卷积层、池化层、ReLU 激活层、Softmax 损失层等常用的网络层的基本功能，并提供了常用网络层的 Python 接口

表 2.2 pycnnl 接口说明

接口	功能描述	参数/返回值
setInputShape: pycnnl.CnnlNet().setInputShape(dim_1, dim_2, dim_3, dim_4)	设定网络第一层输入数据的形状	dim_1 (int)：维度 1 dim_2 (int)：维度 2 dim_3 (int)：维度 3 dim_4 (int)：维度 4
createConvLayer: pycnnl.CnnlNet().createConvLayer(layer_name, input_shape, output_channel, kernel_size, stride, dilation, pad)	创建卷积层	layer_name: 该层的名称 input_shape(list)：输入数据的形状,[N,input channel,input height,input width] output_channel (int)：输出 channel 的大小 kernel_size (int)：卷积核的大小 stride (int)：卷积步长 dilation (int)：膨胀系数 pad (int)：填充大小
createMlpLayer: pycnnl.CnnlNet().createMlpLayer(layer_name, input_shape, weight_shape, output_shape)	创建全连接层	layer_name: 该层的名称 input_shape(list)：输入数据的形状,[N,input height,input weight,input channel] weight_shape (list)：输入数据的形状，[N,weight height,input channel,output channel] output_shape (list)：输入数据的形状，[N,output weight,output width,output channel]
createReLuLayer: pycnnl.CnnlNet().createReLuLayer(layer_name)	创建 ReLU 激活函数层	layer_name: 该层的名称
createSoftmaxLayer: pycnnl.CnnlNet().createSoftmaxLayer(layer_name, input_shape, axis=1)	创建 Softmax 损失层	layer_name: 该层的名称 input_shape：输入数据的形状：batch_size, 1, output_classes axis (int)：进行 Softmax 计算的维度
createPoolingLayer: pycnnl.CnnlNet().createPoolingLayer(layer_name, input_shape, kernel_size, stride)	创建最大池化层	layer_name: 该层的名称 input_shape：输入数据的形状 kernel_size (int)：pool 窗口的大小 stride (int)：窗口滑动步长
loadParams: pycnnl.CnnlNet().loadParams(layer_id, filter_data, bias_data)	为指定的层加载参数	layer_id (int)：需要加载权重的层的 id，CnnlNet 中将创建的层存储在一个数组中，id 即为当前层在该数组中的下标，比如第一个层的 id 为 0，第二个层的 id 则为 1 filter_data (list)：权重数据，必须是一维数组 bias_data(list)：参数偏置
setInputData: pycnnl.CnnlNet().setInputData(input_data)	加载输入数据	input_data (list)：输入数据，必须是一维数组，数据布局为 NHWC
forward: pycnnl.CnnlNet().forward()	进行前向传播计算	
getOutputData: pycnnl.CnnlNet().getOutputData()	获取网络的计算结果	返回值 output_data：网络最后一层的计算结果
size: pycnnl.CnnlNet().size()	获取当前神经网络中层的数量	返回值 layers_num (int)：当前网络中层的数量

pycnnl编程接口

pycnnl 提供的编程接口可以用于在 DLP 上加速神经网络算法。pycnnl 用 Python 封装了一个 C++ 类 CnnlNet，该类的成员函数定义了神经网络中层的创建、网络前向传播、参数加载等操作。

具体解释参考教程24页

```
# 实例化 CnnlNet
net = pycnnl.CnnlNet()
# 设定神经网络输入维度
net.setInputShape(1, 3, 224, 224)
# conv1_1
# 创建卷积层
input_shape1 = pycnnl.IntVector(4)
input_shape1[0] = 1
input_shape1[1] = 3
input_shape1[2] = 224
input_shape1[3] = 224
net.createConvLayer('conv1_1', input_shape1, 64, 3, 1, 1, 1)
# relu1_1
net.createReLuLayer('relu1_1')
```

2.3实验环境

- 硬件环境：DLP
- 软件环境：pycnnl 库，Python 编译环境及相关的扩展库，包括 Python 3.6.12、Pillow 7.2.0、Scipy 1.2.0、NumPy 1.19.5、CNNL 高性能算子库，CNRT 运行时库
- 数据集：MNIST 手写数字库。
- 模型文件：模型权重参数文件

2.3实验环境

硬件环境 01

智能处理卡DLP

模型文件 04

模型权重参数文件

02 软件环境

pycnnl 库，Python 编译环境及相关的扩展库，包括 Python 3.6.12、Pillow 7.2.0、Scipy 1.2.0、NumPy 1.19.5、CNNL 高性能算子库，CNRT 运行时库

03 数据集

MNIST 手写数字库

2.4实验内容

pycnnl 搭建一个三层全连接神经网络

- **数据加载模块**：读取测试数据并进行预处理
- **基本单元模块**：不同网络层的定义，以及前向传播计算等基本功能。
- **网络结构模块**：利用基本单元模块搭建完整的网络。
- **网络推断模块**：使用已有的神经网络模型，对测试数据进行预测。

2.5实验步骤

实例化三层神经网络对应的类

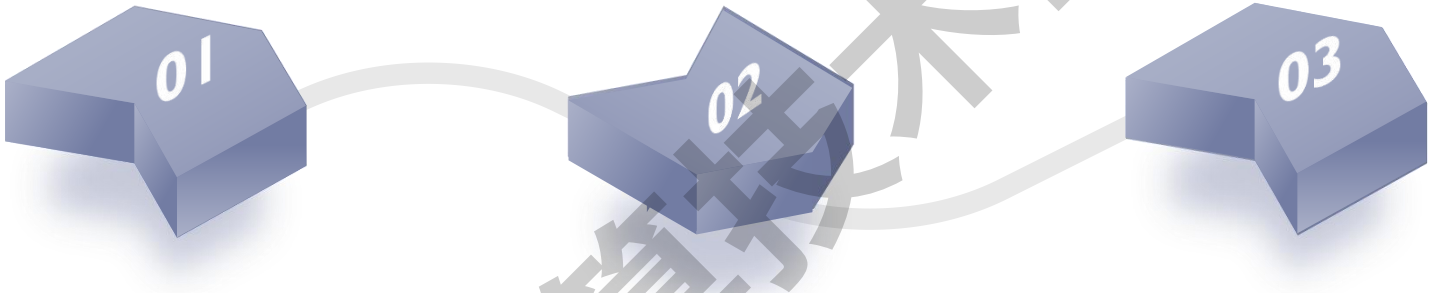
调用 load_data 函数进行数据的加载和预处理

调用 evaluate 函数执行网络推断模块得到预测结果，并测试模型的精度

调用网络结构模块 build_model 建立神经网络，指定神经网络的超参数

调用 load_model 函数从文件中读取训练好的模型参数

补全 layers_1.py，mnist_mlp_cpu.py，mnist_mlp_demo.py 文件。



环境申请

```
# 登录云平台
ssh root@xxx.xxx.xxx.xxx -p xxxxx
# 进入 code_chap_2_3_student 目录
cd /opt/code_chap_2_3/code_chap_2_3_student
# 进入pycnnl/cnnl_python目录下
cd pycnnl/cnnl_python
# 安装swig
pip install swig
# 运行build_pycnnl.sh脚本文件，生成扩展模块
./build_pycnnl.sh
```

代码实现

```
# 进入实验目录
cd exp_2_2_mnist_mlp_dlp
# 补全实验代码
vim stu_upload/layers_1.py
vim stu_upload/mnist_mlp_cpu.py
vim stu_upload/mnist_mlp_demo.py
```

运行实验

```
# 运行完整实验
python main_exp_2_2.py
```

2.6实验评估

- 本实验中，精度评判标准与第2.1 节实验一样，使用测试集的平均分类正确率判断分类
- 结果的精度。性能评判标准为设置 batch size 为 10000 时，进行一次 forward 的时间。本实验的评分标准设定如下：
- • 60 分标准：完善本节实验代码，用 pycnnl 搭建出的三层神经网络能够在 DLP 上进行
- 推断，并且在测试集上的平均分类正确率高于 90%。
- • 80 分标准：修改网络的规模，使用第2.1 节实验的代码重新训练模型，使训练得到的
- 模型在 DLP 上运行的推断（forward）耗时为 CPU 推断耗时的 1/50 或更低，并且在测试集
- 上的平均分类正确率高于 95%
- • 100 分标准：基于三层神经网络重新设计自己的神经网络结构，使用第2.1 节实验的代码重新训练模型，使训练得到的模型在 DLP 上运行的推断耗时为 CPU 推断耗时的 1/200或更低，并且在测试集上的平均分类正确率高于 98%。

2.7实验思考

- 在实验中请思考如下问题：
- 1) DLP 在运行神经网络推断时，相对于 CPU 有什么优势和劣势？
- 2) 在什么样的神经网络结构下，DLP 能够最大化发挥它的性能优势？

智能计算系统实验教程

第三章 深度学习应用实验

中国科学院计算技术研究所
张凯歌 工程师
zhangkaige@ict.ac.cn
手机：18810366140

提纲

- 1.基于VGG19实现图像分类
- 2.基于DLP实验平台实现图像分类
- 3.非实时图像风格迁移

1.基于VGG19实现图像分类

- 1.1实验目的
- 1.2背景介绍
- 1.3实验环境
- 1.4实验内容
- 1.5实验步骤
- 1.6实验评估
- 1.7实验思考

1.1 实验目的

1. 掌握卷积神经网络的设计原理。
2. 掌握卷积神经网络的使用方法。
3. Python 实现 VGG19，输入图像分类。

1.2 背景介绍

VGG19网络中增加了卷积层、最大池化层

名字	类型	卷积核/池化核	步长	边界扩充	输入通道数	输出通道数	输出特征图的高和宽
conv1_1	卷积层	3	1	1	3	64	224
conv1_2	卷积层	3	1	1	64	64	224
pool1	最大池化层	2	2	-	64	64	112
conv2_1	卷积层	3	1	1	64	128	112
conv2_2	卷积层	3	1	1	128	128	112
pool2	最大池化层	2	2	-	128	128	56
conv3_1	卷积层	3	1	1	128	256	56
conv3_2	卷积层	3	1	1	256	256	56
conv3_3	卷积层	3	1	1	256	256	56
conv3_4	卷积层	3	1	1	256	256	56
pool3	最大池化层	2	2	-	256	256	28
conv4_1	卷积层	3	1	1	256	512	28
conv4_2	卷积层	3	1	1	512	512	28
conv4_3	卷积层	3	1	1	512	512	28
conv4_4	卷积层	3	1	1	512	512	28
pool4	最大池化层	2	2	-	512	512	14
conv5_1	卷积层	3	1	1	512	512	14
conv5_2	卷积层	3	1	1	512	512	14
conv5_3	卷积层	3	1	1	512	512	14
conv5_4	卷积层	3	1	1	512	512	14
pool5	最大池化层	2	2	-	512	512	7
fc6	全连接层	-	-	-	25088	4096	-
fc7	全连接层	-	-	-	4096	4096	-
fc8	全连接层	-	-	-	4096	1000	-
Softmax	损失层	-	-	-	-	-	-

参数数量 = $K_H * K_W * C_{in} * C_{out}$

1.3 实验环境

硬件环境	CPU
软件环境	Python 编译环境及相关的扩展库，包括 Python 3.6.12、Pillow 7.2.0、Scipy1.2.0、NumPy 1.19.5（本实验不需使用 Pytorch 等深度学习框架）
数据集	ImageNet[4] 图像数据集，该数据集包括约 128 万训练图像和 5 万张测试图像，共有 1000 个不同的类别。本实验使用官方基于 ImageNet 数据集训练好的模型参数，不需要使用 ImageNet 数据集进行 VGG19 模型的训练。

1.4 实验内容

1.建立VGG19的网络结构

2.利用VGG19的官方模型参数对给定图像进行分类

3.ImageNet数据集中1000 个类别概率

本实验仅包括数据加载模块、基本单元模块、网络结构模块、网络推断模块，不包括网络训练模块。

1.5实验步骤

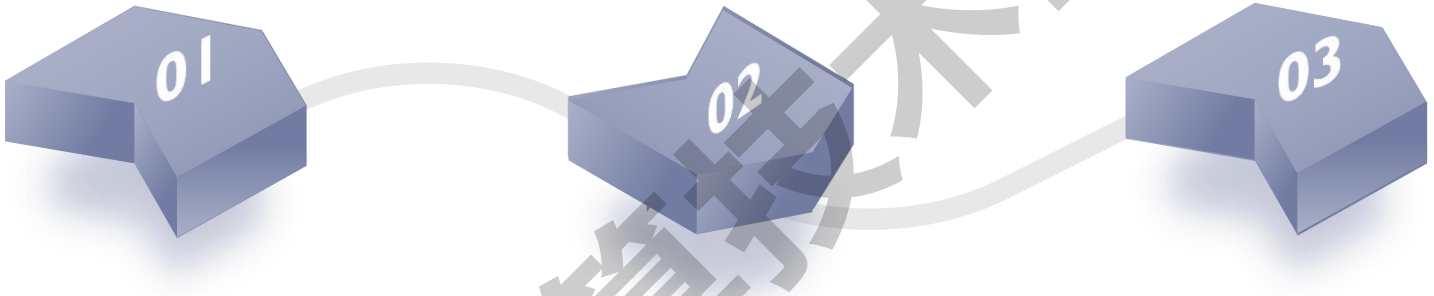
实例化VGG19神经网络对应的类

加载图像并进行预处理

建立VGG19网络结构，参数初始化

调用推理模块进行推理

补全layer_1.py、layer_2.py、vgg_cpu.py 代码



环境申请

```
# 登录云平台
ssh root@xxx.xxx.xxx.xxx -p xxxxx
# 进入 code_chap_2_3_student 目录
cd /opt/code_chap_2_3/code_chap_2_3_student
```

代码实现

```
# 进入实验目录
cd exp_3_1_vgg
# 补全 layers_1.py, layers_2.py, vgg_cpu.py
vim stu_upload/layers_1.py
vim stu_upload/layers_2.py
vim stu_upload/vgg_cpu.py
```

运行实验

```
# 运行完整实验
python main_exp_3_1.py
```

1.6 实验评估

- 本实验的评估标准设定如下：
- • 60 分标准：给定卷积层和池化层的前向传播输入矩和参数值，可以得到正确的前向传播输出矩阵。
- • 80 分标准：建立 VGG19 网络后，给定 VGG19 的网络参数值和输入图像，可以得到正确的 pool5 层输出结果。
- • 100 分标准：建立 VGG19 网络后，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。

1.7 实验思考

- 1) 在实现深度神经网络基本单元时，如何确保一个网络层的实现是正确的?
- 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中每层的计算量（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? 深度神经网络中每层的计算量和每层的前向传播时间之间有什么关系?

2.基于DLP平台实现图像分类

- 2.1实验目的
- 2.2实验环境
- 2.3实验内容
- 2.4实验步骤
- 2.5实验评估
- 2.6实验思考

2.1 实验目的

1. 使用 pycnnl 库实现卷积、ReLU 等基本网络模块
2. 使用提供的 pycnnl 库实现 VGG19 网络。
3. 分析并比较 DLP 和 CPU 平台上运行 VGG19 进行图像分类的性能。

2.2 实验环境

硬件环境	DLP
软件环境	pycnnl 库、Python 编译环境及相关的扩展库，包括 Python 3.6.12、Pillow 7.2.0、Scipy1.2.0、NumPy 1.19.5（本实验不需使用 Pytorch 等深度学习框架）
数据集	ImageNet2012

2.3 实验内容

pycnnl 搭建
VGG19
神经网络

- **数据加载模块**：读取测试数据并进行预处理
- **基本单元模块**：不同网络层的定义，以及前向传播计算等基本功能。
- **网络结构模块**：利用基本单元模块搭建完整的网络。
- **网络推断模块**：使用VGG19神经网络模型，对测试数据进行预测。

2.4 实验步骤

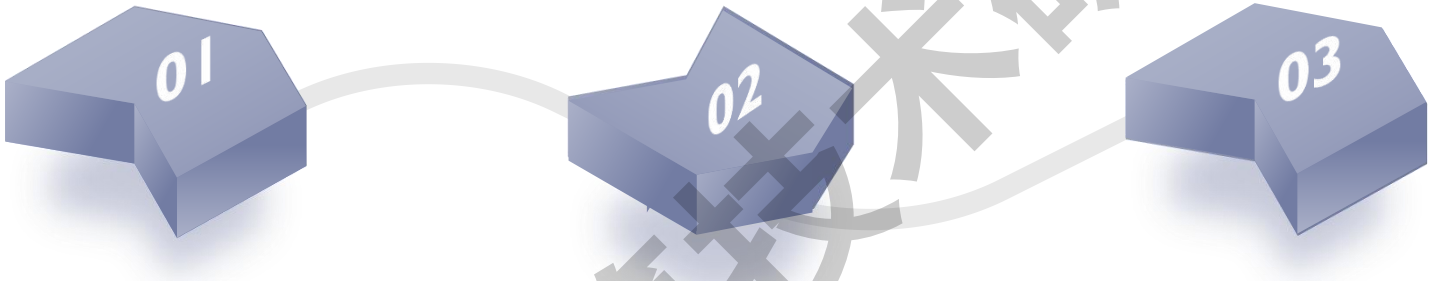
实例化VGG19神经网络对应的类

加载图像并进行预处理

建立VGG19网络结构，参数初始化

调用推理模块进行推理

补全vgg19_demo.py 代码



环境申请　　代码实现　　运行实验

```
# 登录云平台
ssh root@xxx.xxx.xxx.xxx -p xxxxx
# 进入 code_chap_2_3_student 目录
cd /opt/code_chap_2_3/code_chap_2_3_student
# 进入pycnnl/cnnl_python目录下
cd pycnnl/cnnl_python
# 运行build_pycnnl.sh脚本文件，生成扩展模块
./build_pycnnl
```

```
# 进入实验目录
cd exp_3_1_vgg
# 补全 vgg19_demo.py
vim stu_upload/vgg19_demo.py
```

```
# 运行完整实验
python main_exp_3_2.py
```

2.5 实验评估

- 本实验的评分标准设定如下：
- • 100 分标准：使用 pycnnl 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。

2.6 实验思考

- 在实验中请思考如下问题：
- 1) 阅读 pycnnl/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.非实时图像风格迁移

- 3.1实验目的
- 3.2背景知识
- 3.3实验环境
- 3.4实验内容
- 3.5实验步骤
- 3.6实验评估
- 3.7实验思考

3.1 实验目的

1. 利用VGG19 模型进行图像特征提取
2. 实现风格迁移中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. 实现非实时风格迁移完整流程，为后续实现实时风格迁移以及更杂的综合实验奠定基础

3.2 背景介绍

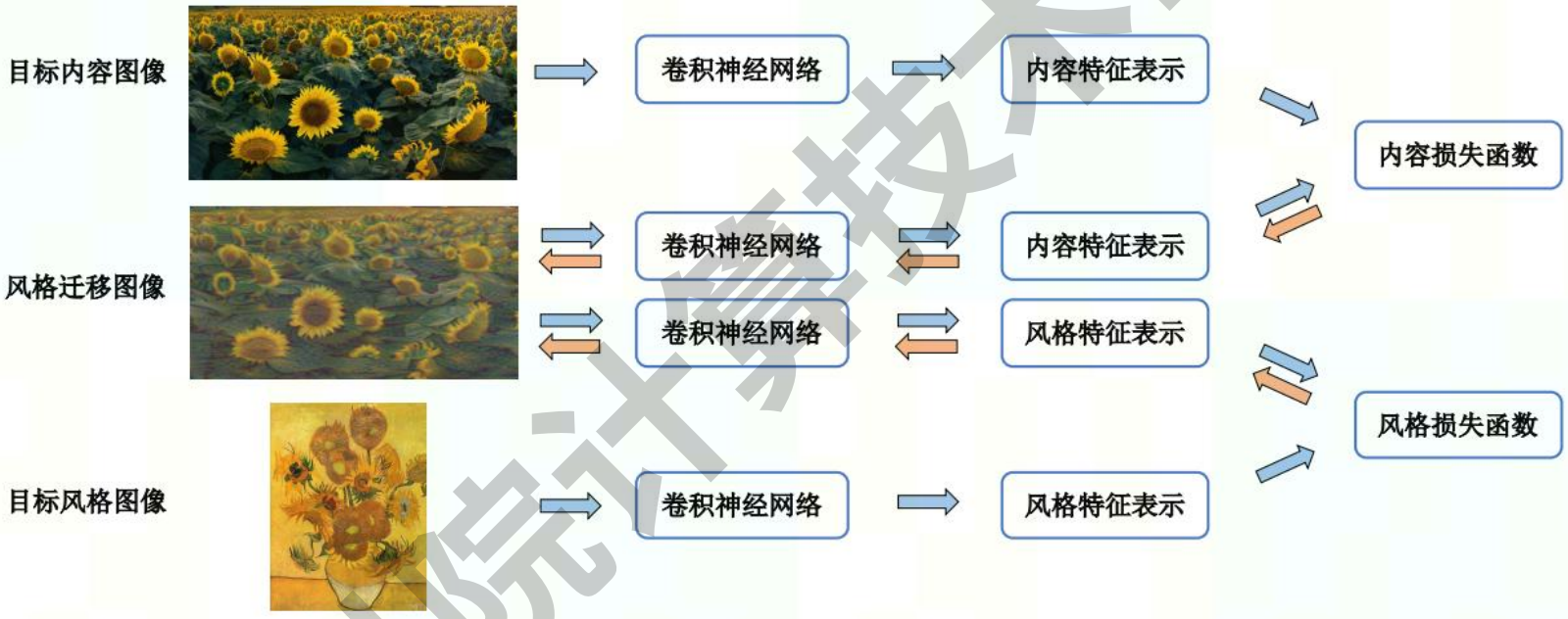


图 3.3　非实时风格迁移

3.3 实验环境

硬件环境	CPU
软件环境	Python 编译环境及相关的扩展库，包括 Python 3.6.12、Pillow 7.2.0、Scipy1.2.0、NumPy 1.19.5（本实验不需使用 Pytorch 等深度学习框架）

3.4 实验内容

VGG19 网络实现非实时风格迁移

利用 VGG19 模型提取内容图像和风格图像的特征

计算风格迁移图像与目标风格（内容）图像的风格（内容）损失

对损失进行反向传播，更新风格迁移图像（也称为生成图像）

通过多次迭代，不断减小生成图像与目标风格（内容）图像的风格（内容）损失，最终获得风格化后的图像

3.5 实验步骤

首先确定超参数

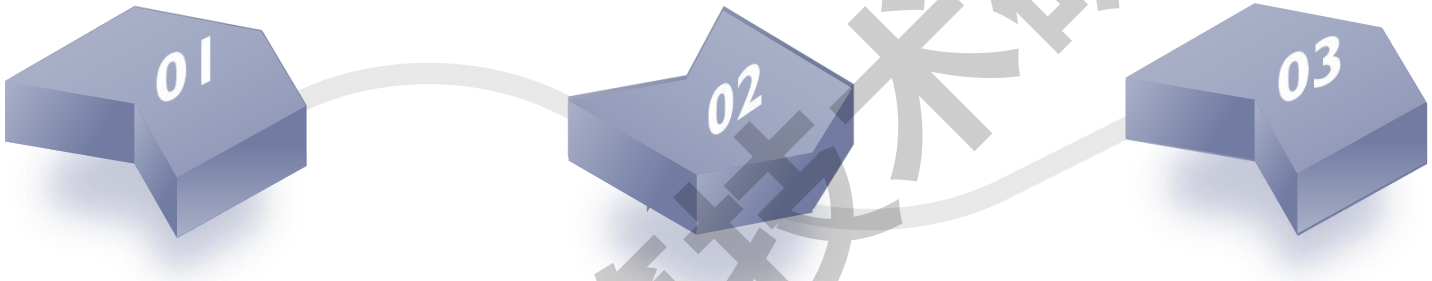
建立 VGG19 的网络结构并加载官方模型参数，同时实例化非实时风格迁移中的内容损失计算、风格损失计算和 Adam 优化器。

开始非实时风格迁移的迭代训练过程。

计算内容损失和风格损失函数时使用 VGG19 的哪些层、数据预加载模块相关的图像缩放分辨率、训练有关的学习率大小、迭代次数、损失函数权重系数等

读取给定的内容图像和风格图像进行预处理，并计算相应的特征图作为计算内容损失和风格损失函数的标记，同时利用内容图像初始化生成图像

述补全layer_1.py、layer_2.py、layer_3.py、style_transfer.py代码



环境申请

```
# 登录云平台
ssh root@xxx.xxx.xxx.xxx -p xxxxx
# 进入 code_chap_2_3_student 目录
cd /opt/code_chap_2_3/code_chap_2_3_student
```

代码实现

```
# 进入实验目录
cd exp_3_1_vgg
# 补全 layer_1.py, layer_2.py, layer_3.py, style_transfer.py
vim stu_upload/layer_1.py
vim stu_upload/layer_2.py
vim stu_upload/layer_3.py
vim stu_upload/style_transfer.py
```

运行实验

```
# 运行完整实验
python main_exp_3_3.py
```

3.6 实验评估

- 本实验的评估标准设定如下：
- • 60 分标准：正确实现利用四重循环计算卷积层和池化层的前向传播和反向传播过程。给定卷积层和最大池化层的前向传播和反向传播输入矩阵和参数值，可以得到正确的前向传播输出矩阵和反向传播输出梯度。同时分别给出卷积层和最大池化层的前向传播时间和反向传播时间。
- • 80 分标准：正确实现内容损失函数和风格损失函数的计算，计算得出风格迁移后的图片。给定生成图像、目标内容图像和目标风格图像，可以计算得到正确的内容损失值和风格损失值；可以得到正确的内容损失和风格损失对生成图像的更新梯度，生成风格迁移后的图像。
- • 100 分标准：对使用四重循环计算卷积层和池化层的实现方式进行改进，提升计算速度。给定卷积层和池化层的前向传播和反向传播输入矩阵和参数值，可以得到正确的前向传播输出矩阵和反向传播输出梯度，同时分别给出卷积层和池化层的前向传播时间和反向传播时间及其对应的加

3.7 实验思考

- 1) 在 CPU 平台上使用四重循环计算卷积层前向传播和反向传播的速度较慢，如何利用高效的矩阵运算库，将四重循环中卷积核与特征图的内积运算（即向量运算）转化为矩阵运算，从而减少循环次数，加速卷积层的运算速度？
- 2) 统计训练过程中每次迭代时每层的前向传播和反向传播时间及其在每层的时间占比，哪些层的时间占比较高？前向传播和反向传播的计算瓶颈在哪些层？
- 3) 在第3.1节和第3.2节的实验评估中，均使用给定输入值并与正确的输出值进行比较的方式来确定某个层的实现是否正确。如果正确的输出值无法获知，如何从梯度定义的角度检查层的实现是否正确？（可参考由梯度定义引申出的梯度的数值近似实现方法）
- 4) 风格迁移的结果通常是由人直接进行判断，这是一种主观的定性判断方式，不同的人判断结果可能会有较大偏差。如何设计合理的定量判断方法，可以较为客观的评价风格迁移结果的优劣？

智能计算系统实验教程

第四章 编程框架实验

中国科学院计算技术研究所
张凯歌 工程师
zhangkaige@ict.ac.cn
手机：18810366140

提纲

- 1.基于VGG19实现图像分类
- 2.实时风格迁移的推断
- 3.实时风格迁移的训练
- 4.自定义PyTorch CPU算子

1.基于VGG19实现图像分类

- 1.1实验目的
- 1.2背景知识
- 1.3实验环境
- 1.4实验内容
- 1.5实验步骤
- 1.6实验评估
- 1.7实验思考

1.1 实验目的

- 掌握使用 PyTorch 编程框架处理深度学习任务的流程
- 熟悉 PyTorch 中常用数据结构的使用方法
- 掌握 PyTorch 中常用 API 的使用方法，包括卷积、激活等相关操作
- 与第3章的实验比较，理解使用编程框架实现深度学习算法的便捷性及高效性

1.2 背景知识

- 1. PyTorch
- PyTorch 是由 Meta 开发并于 2017 年 11 月开源的深度学习框架，适用于 Python、C++等编程语言，可以用于部署并实施大规模机器学习模型。
- PyTorch 提供了一系列高性能的 API，方便程序员高效地实现深度学习算法。以卷积神经网络 VGG为例，对于每个卷积层，首先输入与权重做卷积运算，然后加上偏置，最后通过非线性激活函数 ReLU 输出。PyTorch 中则提供了一系列封装好的 API，可以方便地实现上述操作。
- 2.基于 DLP 的 PyTorch
- 基于 DLP 的PyTorch 对原生 PyTorch 的主要修改有：添加 MLU 设备、部分算子的分发方式、Torchvision访问权限和 CATCH 拓展包。
- 基于 DLP 的 PyTorch 训练模式和推理模式。

1.3 实验环境

硬件环境	CPU 、DLP
软件环境	Torch1.6.0，CNNL 高性能算子库，CNRT 运行时库，以及 python 环境及相关的扩展库，包括python3.7.4，Pillow9.2.0, Numpy1.21.6，Scipy1.5.4

1.4 实验内容

01

利用 PyTorch 的 API，实现第3.1节中基于 VGG19 进行图像分类的实验，运行平台包括CPU 和 DLP.

02

比较两种平台实现的差异。

1.5 实验步骤

步骤1

逐层定义需要执行的操作，每一层的输出作为下一层的输入搭建完整的 VGG19 网络。

步骤2

将.mat 格式的 VGG19 模型生成为.pth 格式。

步骤3

利用 Image 模块内置的 open 函数读入待处理图像，并将图像进行预处理。

步骤4

在 CPU 和DLP上使用 PyTorch 加载 VGG19 网络参数进行推理，得到分类概率。

补全lgenerate_pth.py、evaluate_cpu.py、evaluate_cnnl_mfus.py代码

环境申请

```
# 登录云平台
ssh root@xxx.xxx.xxx.xxx -p xxxxx
# 进入/opt/code_chap_4_student目录
cd /opt/code_chap_4_student
```

代码实现

```
# 进入实验目录
cd exp_4_1_vgg19_student
# 补全生成.pth文件的代码
vim stu_upload/generate_pth.py
# 补全CPU 上进行推理的代码
vim stu_upload/evaluate_cpu.py
#补全DLP 上进行推理的代码
vim stu_upload/evaluate_cnnl_mfus.py
```

运行实验

CPU和DLP运行

```
#生成.pth模型，并在cpu上进行推理
bash run_cpu.sh

#在DLP上使用Eager模式进行推理
bash run_mlu.sh
```

1.6 实验评估

- 本实验的评估标准设定如下：
- • 60 分标准：在 CPU 平台上正确实现输入图像的读入、卷积层和池化层的定义等。可以正确生成.pth 文件。
- • 80 分标准：在 CPU 平台上正确加载.pth 文件模型参数，实现 CPU 上的 VGG19 网络的推理；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。
- • 100 分标准：在 DLP 平台上正确实现 VGG19 网络的推理；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；处理时间相比 CPU 平台具有明显的提升。

1.7 实验思考

- 1) 本实验与第3.1节中使用 Python 语言实现的图像分类相比，在识别精度、识别速度等方面有哪些差异？为什么会有这些差异?

2.实时风格迁移的推断

- 2.1实验目的
- 2.2背景知识
- 2.3实验环境
- 2.4实验内容
- 2.5实验步骤
- 2.6实验评估
- 2.7实验思考

2.1 实验目的

- 掌握使用 PyTorch 定义完整网络结构的方法
- 掌握使用 PyTorch 加载模型结构及参数的方法
- 以实时风格迁移算法为例，掌握在 CPU 平台上使用 PyTorch 进行神经网络推断的方法
- 掌握在 DLP 平台上使用 PyTorch 对模型进行神经网络推断的方法

2.2 背景介绍

02

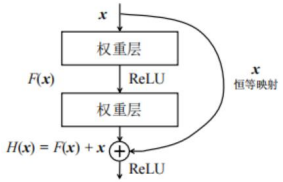
2. 基于DLP的pytorch推理

01

1. 实时风格迁移推断算法

该网络主要包含几部分：下采样卷积层、残差层、上采样卷积层以及输出层。

残差块结构：



残差块增加了从输入到输出的直连（shortcut connection），其卷积拟合的是输出与输入的差（即残差）

归一化：有效避免梯度爆炸或消失，从而训练出较深的神经网络

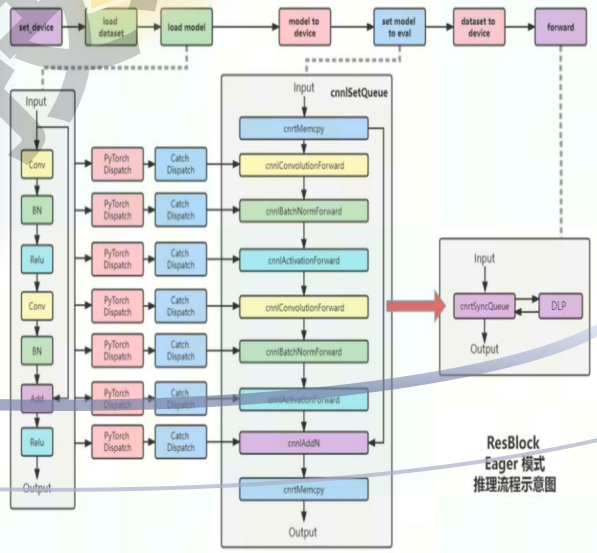


图 4.6　基于 DLP 的 PyTorch 推理运行流程

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}}, \quad \mu_{ti} = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm}, \quad \sigma_{ti}^2 = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - \mu_{ti})^2 \qquad (4.2)$$

2.3 实验环境

硬件环境	CPU 、DLP
软件环境	Torch1.6.0，CNNL 高性能算子库，CNRT 运行时库，以及 python 环境及相关的扩展库，包括python3.7.4，Pillow9.2.0, Numpy1.21.6，Scipy1.5.4

2.4 实验内容

网络推理

本实验利用 PyTorch 加载预先保存好的实时风格迁移模型文件（.pth 文件），在 CPU 和DLP 平台上运行实时风格迁移的推断（即图像转换网络的推断）得到风格化图像，并比较两种平台在实现上和运行性能上的差异

2.5 实验步骤

步骤1 补全数据加载模块

数据加载模块读取 zip 文件中的数据，并将数据转换为图像格式，经过预处理，将图像转换为符合 PyTorch 处理格式的 tensor。

步骤2 补全网络结构模块

实时风格迁移推断的过程中，需要利用训练好的图像转换网络对任意输入图像做网络推断。

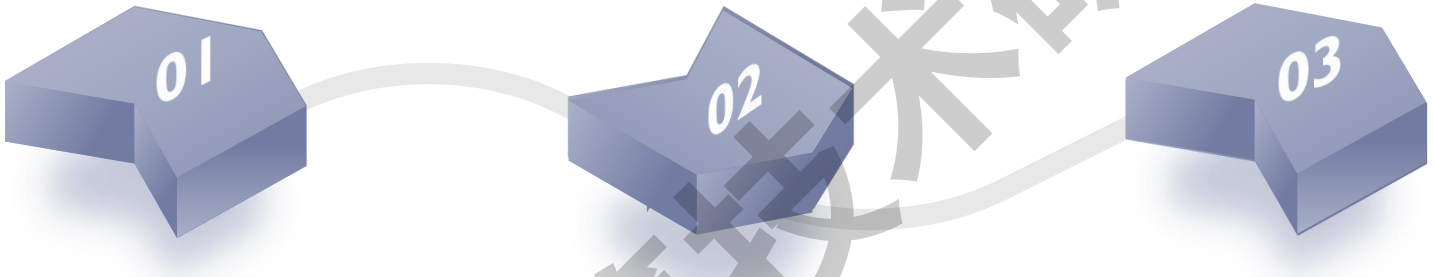
步骤3 补全CPU推理模块

在 CPU 上使用 PyTorch 加载图像转换网络参数，进行推理，并得到分类概率。

步骤4 补全DLP推理模块

在 DLP 上使用 PyTorch 加载图像转换网络参数，利用 CNNL 后端进行推理，得到分类概率。

补全 exp_4_2_fast_style_transfer_infer/stu_upload 中的 evaluate_cpu.py、evaluate_cnnl_mfus.py 代码



环境申请

```
# 登录云平台
ssh root@xxx.xxx.xxx.xxx -p xxxxx
# 进入/opt/code_chap_4_student目录
cd /opt/code_chap_4_student
```

代码实现

```
# 进入实验目录
cd exp_4_2_fast_style_transfer_infer_student
# 补全CPU上进行推理的代码
vim stu_upload/evaluate_cpu.py
#补全DLP上进行推理的代码
vim stu_upload/evaluate_cnnl_mfus.py
```

运行实验

```
#在cpu上进行推理
bash run_cpu.sh

#在DLP上进行推理
bash run_mlu.sh
```

2.6 实验评估

- 本实验的评估标准设定如下：
- • 60 分标准：在 CPU 平台上正确实现数据加载模块，并且实现网络结构模块的构建。
- • 80 分标准：在 60 分标准的基础上，在 CPU 平台上正确实现实时风格迁移的推断过程；给定输入图像、权重参数，可以实时计算并输出风格迁移后的图像，同时给出对图像进行实时风格迁移的时间。
- • 100 分标准：在 80 分标准的基础上，在 DLP 平台上实现实时风格迁移，给定输入图像、权重参数，能够实时输出风格迁移后的图像，同时 DLP 上比 CPU 平台上的推理时间有 10 倍以上的提升。

2.7 实验思考

- 1) 对于给定的输入图像集合、权重参数，在不改变图转换网络结构的前提下如何提升推断速度？
- 2) 请使用性能剖析/监控等工具分析在 DLP 平台上进推断的性能瓶颈。如何利用多核 DLP 架构提升整体的吞吐率？
- 3) 批归一化、实例归一化、层归一化以及组归一化这些归一化策略有什么本质的区别，分别适合什么样的应用场景？

3.实时风格迁移的训练

- 3.1实验目的
- 3.2背景知识
- 3.3实验环境
- 3.4实验内容
- 3.5实验步骤
- 3.6实验评估
- 3.7实验思考

3.1 实验目的

- 掌握如何使用 PyTorch 实现实时风格迁移模型的训练。具体包括：
- 1) 掌握使用 PyTorch 定义损失函数的方法；
- 2) 掌握使用 PyTorch 存储网络模型的方法；
- 3) 以实时风格迁移算法为例，掌握使用 PyTorch 进行神经网络训练的方法。

3.1 实验目的

掌握 PyTorch
实时风格迁移模型的训练

01 掌握使用 PyTorch 定义损失函数的方法；

02 掌握使用 PyTorch 存储网络模型的方法；

03 以实时风格迁移算法为例，掌握使用 PyTorch 进行神经网络训练的方法

3.2 背景介绍

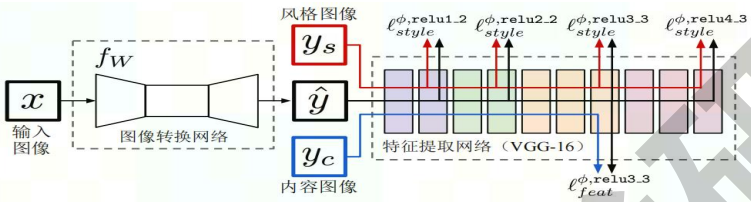


图 4.7　实时图像风格迁移算法的流程[11]

1.实时风格迁移训练算法

首先，输入的内容图像 x 经过图像转换网络输出风格化图像其次，利用特征提取网络（原始网络为在 ImageNet数据集上预训练好的 VGG16，这里改为 VGG19）分别提取生成图像 $\hat{y}$ 、风格图像 y_s 和内容图像 y_c 的特征，并利用这些特征计算损失函数；然后，通过迭代地调整图像转换网络的参数来最小化损失函数，最终完成对图像转换网络的训练。

2.基于 CPU 的 PyTorch 训练

首先定义网络基本运算单元；其次构建图像转换网络以及特征提取网络，其构建方法和第4.2节所采用的方法一致，随后定义损失函数，然后创建优化器，定义模型训练方法，最后迭代地执行模型的训练过程。此外，在模型训练过程中或当模型训练完成后，可以使用 torch.save() 函数来将当前时刻的模型参数保存到磁盘指定路径下

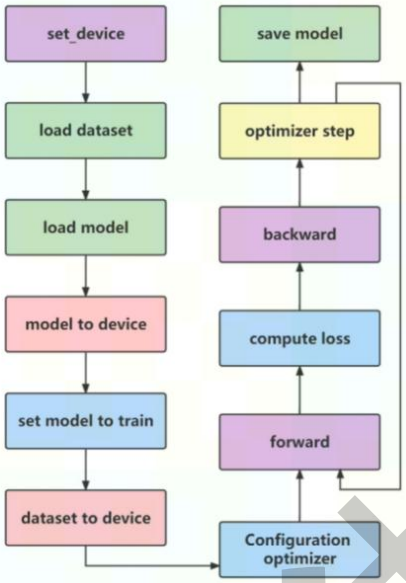
3.基于 DLP 的 PyTorch 训练

在 DLP 上，PyTorch 在单机单卡浮点模式下训练网络使用 CNNL 作为后端，无需进行量化设置和分布式设置，灵活简单，可快速逐层调试；权重梯度与学习率相乘，并进行权重更新，得到新权重，从而完成一次训练过程。最后保存好训练的模型。

3.3 实验环境

硬件环境	CPU 、DLP
软件环境	Torch1.6.0，CNNL 高性能算子库，CNRT 运行时库，以及 python 环境及相关的扩展库，包括python3.7.4，Pillow9.2.0, Numpy1.21.6，Scipy1.5.4

3.4 实验内容



1. **Set_device**：指定 MLU 硬件设备ID

2. **Load Dataset**：加载推理数据集至 Host 端内存

3. **Load Model**：加载网络模型和权重至 Host 端内存

4. **Model To Device**：将模型权重 copy 至 Device 端

5. **Set Model To Train**：将模型转换为 training 模式，调用 PyTorch Dispatch 和 Catch DisPatch 模块，完成算子分发

6. **Dataset To Device**：为网络模型输入创建 CnnlTensor 和对应的 Device 端内存空间，并完成 Host 至 Device 拷贝

7. **Configuration optimizer**：配置人工智能训练优化器，如 SGD、Adam 等

8. **Forward**：逐 cnnl 前向算子启动 CNNL Runtime 执行**前向推理**

9. **Compute Loss**：将 forward 的输出与 target 一同输入 cnnl loss 算子，计算 loss

10. **Backward**：根据 loss 结果进行反向推理，逐 cnnl 反向算子启动 CNNL Runtime

11. **Optimizer Step**：将反向推理得到的权重梯度乘以学习率，并进行权重更新，得到新的权重，完成一次训练

12. **Save Model**：保存训练好的模型

实验内容

利用 PyTorch 的 API 实现卷积层、残差块等基本运算单元，构建如图 4.7所示的实时风格迁移网络，通过特征提取网络构建损失函数，并基于该损失函数来迭代地训练图像转换网络[15]，最终获得较好的训练效果。同时，比较 CPU 以及 DLP 实现实时风格迁移训练的不同之处。

3.5 实验步骤

步骤1 补全数据加载模块

在数据加载模块中，使用 cv2 读取风格图片，并将风格图片进行预处理，从而将图像转换为符合 PyTorch 处理格式的 tensor

步骤2 补全网络结构模块

网络结构模块主要由特征提取网络和图像转换网络构成；特征提取网络采用 VGG19 网络。在实现过程中，它通torchvision.models来调用 VGG19 模型，为了后续重建损失的输出，这里输出第 2、4、8、12 层卷积后的特征。

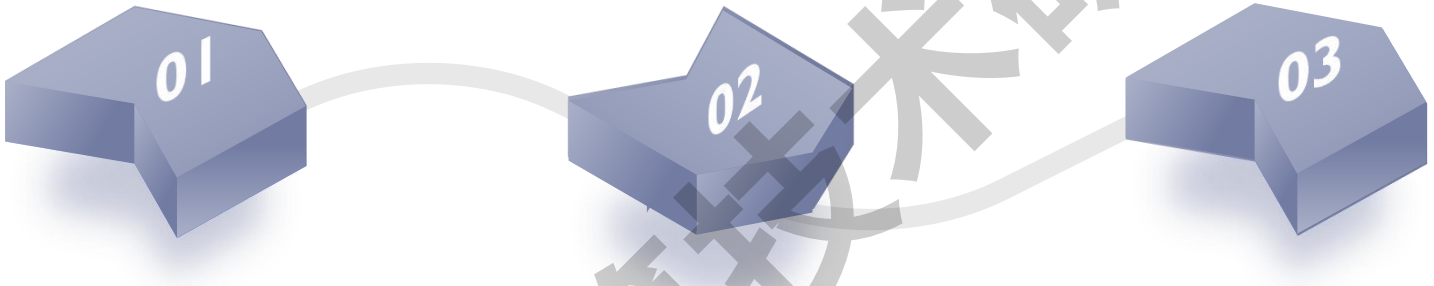
步骤3 补全CPU训练模块

该部分读入输入风格图像及内容图像，并构建损失函数，通过优化器的创建及训练方法的定义来完成反向传播的过程，从而实现CPU 上的训练并正确保存模型文件以及生成的图片

步骤4 补全DLP训练模块

该部分对输入风格图像及内容图像进行特征提取，通过损失函数最小化在 DLP 平台上实现实时风格迁移的训练过程。

补全exp_4_3_fast_style_transfer_train/stu_upload中的train.py和 train-mlu.py 的代码



环境申请

```
# 登录云平台
ssh root@xxx.xxx.xxx.xxx -p xxxxx
# 进入/opt/code_chap_4_student目录
cd /opt/code_chap_4_student
```

代码实现

```
# 进入实验目录
cd exp_4_3_fast_style_transfer_train
# 补全CPU训练的代码
vim stu_upload/train.py
#补全DLP训练的代码
vim stu_upload/train-mlu.py
```

运行实验

```
#在cpu上进行训练
bash run_train_cpu.sh
```

```
#在dlp上进行训练
bash run_train_mlu.sh
```

3.6 实验评估

- 本实验的评估标准设定如下：
- • 60 分标准：正确读入风格图像、内容图像，正确实现特征提取网络、图像转换网络。
- • 80 分标准：完成损失函数的构建、前向传播反向传播的过程。给定输入图像、风格图像，在 CPU 上可以通过训练过程使得损失值逐渐减少，最终能够得到正确的风格迁移图像。
- • 100 分标准：在 DLP 平台上能够正确实现实时风格迁移的训练过程，给定输入图像、风格图像，在 DLP 上可以通过训练过程使得损失值逐渐减少，最终能够得到正确的风格迁移图像。

3.7 实验思考

- 1) 整个实时风格迁移算法中包含了图像转换网络和特征提取网络两部分，其中特征提取网络的参数是已经预训练好的。在使用 PyTorch 设计算法时，应该如何操作才能使得训练时 PyTorch 内置的优化器仅针对图像转换网络的参数进行优化？
- 2) 对于给定的输入图像集合，在不改变图像转换网络以及特征提取网络结构的前提下应如何提升训练速度？
- 3) 在图像转换网络中使用实例归一化方法，相比批归一化方法，对生成的图像质量会产生怎样的影响？
- 4) 为什么计算风格损失时需要将多层卷积层的输出求和，而计算内容损失时只需要计算某层卷积层的输出？
- 5) 在定义损失函数时，如果改变内容损失、风格损失的content_weight 和 style_weight，将如何影响最后的迁移效果？
- 6) 使用 Tensorboard 看一下计算图，分析一下代码中可以优化的地方。
- 7) 查询每秒浮点运算次数（FLOPS）的计算公式，尝试下计算非实时风格迁移的FLOPS。

4.自定义PyTorch CPU算子

- 4.1实验目的
- 4.2背景知识
- 4.3实验环境
- 4.4实验内容
- 4.5实验步骤
- 4.6实验评估
- 4.7实验思考

4.1 实验目的

- 掌握如何在 PyTorch 中新增自定义的 hsigmoid 算子。具体包括：
- 1) 熟悉 PyTorch 整体设计机理；
- 2) 通过在 PyTorch 框架中添加自定义的 hsigmoid 算子，加深对 PyTorch 算子实现机制
- 的理解，以 PyTorch 中添加自定义 CPU 算子为例，为后续在 PyTorch 中集成添加自定义的DLP 算子奠定基础。

4.2 背景介绍

激活函数是深度神经网络中每一个神经元线性加权计算结果的非线性处理，赋予神经网络非线性映射能力。

常见的激活函数有 Sigmoid、Tanh、ReLU、Leaky ReLU 等

$$hsigmoid(x) = \frac{1}{1+e^{-x}}$$

Sigmoid 函数具有以下优点：

1) 输出值在 0 到 1 之间，方便观察模型的输出是否符合预期；

2) Sigmoid 函数的导数在 0 到 0.25 之间，可以控制反向传播过程中权值的更新幅度；

3) Sigmoid 函数的输出接近 0 或 1 时，输出的导数接近 0，可以避免梯度消失的问题。

4.2 背景介绍

激活函数是深度神经网络中每一个神经元线性加权计算结果的非线性处理，赋予神经网络非线性映射能力。

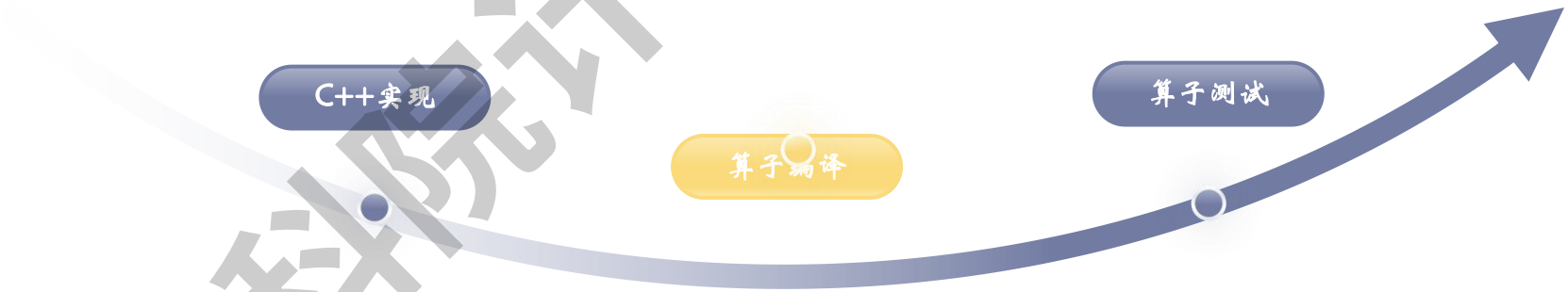
常见的激活函数有 Sigmoid、Tanh、ReLU、Leaky ReLU 等

Sigmoid 函数具有以下优点：

1) 输出值在 0 到 1 之间，方便观察模型的输出是否符合预期；

2) Sigmoid 函数的导数在 0 到 0.25 之间，可以控制反向传播过程中权值的更新幅度；

3) Sigmoid 函数的输出接近 0 或 1 时，输出的导数接近 0，可以避免梯度消失的问题。



4.3 实验环境

硬件环境	CPU
软件环境	Torch1.6.0以及 python 环境及相关的扩展库，包括python3.7.4，Pillow9.2.0, Numpy1.21.6，Scipy1.5.4

4.4 实验内容

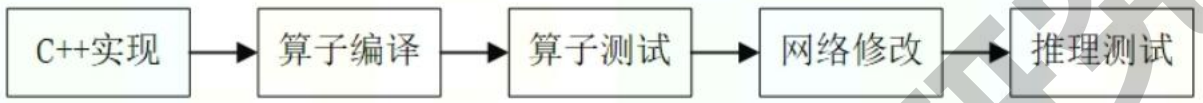


图 4.9　自定义 hsigmoid 算子添加实验流程图

1. **C++ 实现**　编写 hsigmoid 算子对应的 C++ extension 文件
2. **算子编译**　使用 setuptools 进行算子编译
3. **算子测试**　通过测试程序测试 hsigmoid 算子是否添加成功
4. **网络修改**　将网络结构中的 Sigmoid 算子更改为自定义添加的 hsigmoid 算子
5. **推理测试**　运行添加 hsigmoid 算子的实时风格迁移的代码，完成实验推理

4.5 实验步骤

步骤1 补全C++实现

自定义 hsigmoid 算子的 C++ 实现代码 hsigmoid.cpp

步骤2 补全算子编译

本实验利用 python 中提供的 setuptools 模块完成编译流程，将写有算子的 C++ 文件，编译成为一个动态链接库（在 Linux 平台是一个.so 后缀文件），可以让 python 调用其中实现的函数功能。

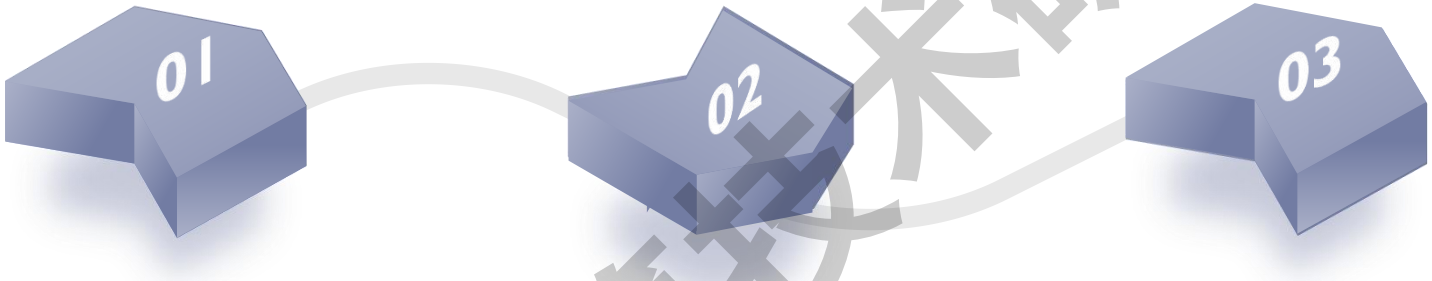
步骤3 补全算子测试

实验通过测试程序test_hsigmoid.py 对算子进行测试

步骤4 补全网络修改

本实验基于第4.2节实时风格迁移推断实验，其中的 Sigmoid 算子替换成为自定义的hsigmoid 算子

hsigmoid.cpp、setup.py、test_hsigmoid.py、evluate_cpu.py 文件



环境申请

```
# 登录云平台
ssh root@xxx.xxx.xxx.xxx -p xxxxx
# 进入/opt/code_chap_4_student目录
cd /opt/code_chap_4_student
```

代码实现

```
# 进入op_hsigmoid实验目录
cd exp_4_4_custom_pytorch_op_student/stu_upload/op_hsigmoid
# 补全hsigmoid.cpp文件
vim hsigmoid.cpp
# 补全setup.py文件
vim setup.py

# 进入stu_upload实验目录
cd exp_4_4_custom_pytorch_op_student/stu_upload
# 补全test_hsigmoid.py文件
vim test_hsigmoid.py
# 补全evluate_cpu.py文件
vim evluate_cpu.py
```

运行实验

```
# 进入op_hsigmoid实验目录
cd exp_4_4_custom_pytorch_op_student/stu_upload/op_hsigmoid
# 进行算子编译生成动态链接库
python setup.py build_ext --inplace
# 进入stu_upload实验目录
cd exp_4_4_custom_pytorch_op_student/stu_upload
# 进行算子测试
python test_hsigmoid.py
# 进行模型推理
python evaluate_cpu.py
```

4.6 实验评估

- 本实验的评估标准设定如下：
- • 60 分标准：完成使用 C++ extension 方法进行 PyTorch 上 hsigmoid 算子添加工作，能够正确生成动态链接库。
- • 80 分标准：在 60 分基础上，完成算子测试程序的补全，能够正确调用 hsigmoid 算子。
- • 100 分标准：在 80 分基础上，完成网络结构的修改，能够正确实现添加 hsimoid 算子进行实时风格迁移推理。

4.7 实验思考

- 1) 框架内部核心代码为何使用 C/C++ 语言，为何不使用高级语言（如 Python 等）？
- 2) PyTorch 算子添加方式有哪几种，有什么优劣势？
- 3) 将 C++ 算子编译成库文件有哪几种方式，有什么优劣势？
- 4) PyTorch 算子添加的方式与 Tensoflow 框架相比有何区别和共同点？

谢谢大家！

欢迎关注课程公众号