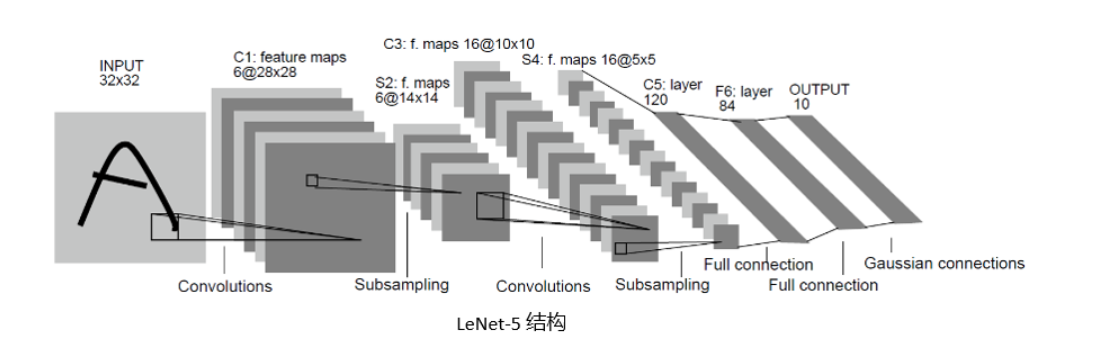
## 基于LeNet-5的MNIST手写体识别实验报告（MindSpore框架）

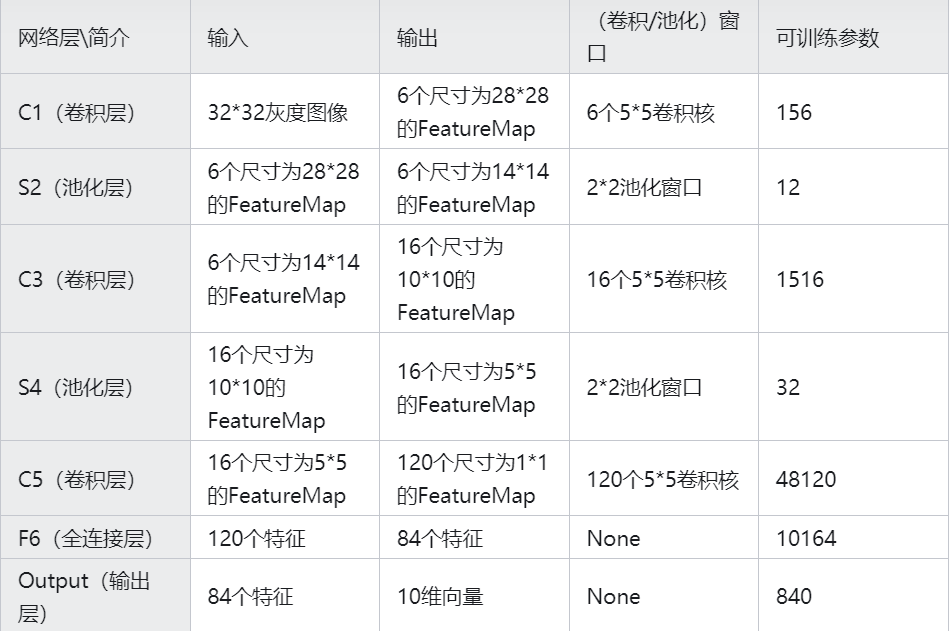
**姓名：卓威 学号：123106222860**

LeNet是卷积神经网络的开山之作，也是将深度学习推向繁荣的一座里程碑。Yann LeCun于上世纪90年代提出了LeNet，他首次采用了卷积层、池化层这两个全新的神经网络组件；LeNet在手写字符识别任务上取得了瞩目的准确率。LeNet网络有一系列的版本，其中以LeNet-5版本最为著名，也是LeNet系列中效果最佳的版本。LeNet-5使用5个卷积层来学习图像特征；卷积层的权重共享特点使得它相较于全连接层，节省了相当多的计算量与内存空间；同时卷积层的局部连接特点可以保证图像的空间相关性。

#### 网络架构



###### 网络层级



###### 1.2 输入层

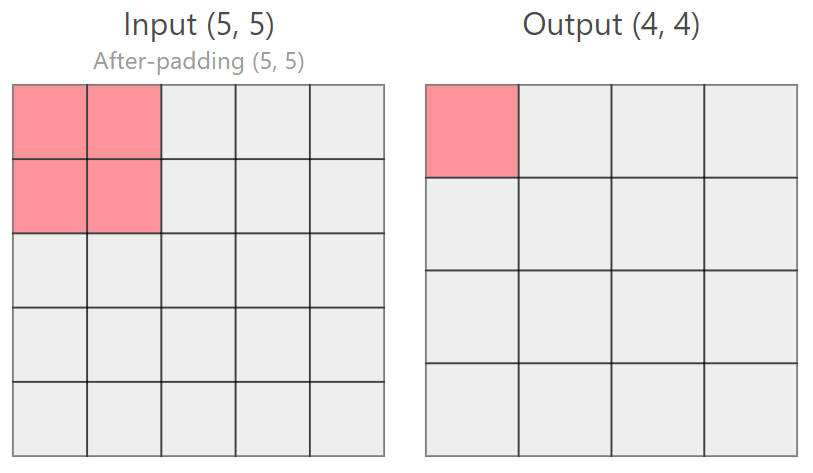
输入层为单通道的灰度图像，图像尺寸为32\*32。

###### 1.3 卷积层

卷积层是 CNN 的基础，因为它们包含学习的内核（权重），它提取区分不同图像的特征——这就是为什么将其成为分类器的原因。当与卷积层交互时，前一层和卷积层之间相互链接。

通过卷积层，可使得信号特征增强，降低噪声；故其主要作用是降低图像噪声，提取图像重要特征。此外，卷积层能够保证图像的平移不变性，卷积运算保留了图像的空间信息，卷积后的图像位置关系没有改变。

卷积层利用卷积核对图像进行卷积运算；每个卷积核都是可训练的，一般会在卷积核上添加一个偏置参数。卷积运算是卷积核以步长stride在原图上进行移动，将其与卷积核进行元素乘法，并求和乘法结果，最后加上偏置，得到卷积结果。



###### 1.4池化层Max Pooling

**池化层一般不同训练，它的主要作用是减少数据，在降低数据维度的同时保留特征图中重要的特征信息；同时也避免了网络参数太多而造成的过拟合问题。**不同的 CNN 架构中的池化层类型很多，但它们的目的都是逐渐减小网络的空间范围，从而减少网络的参数和整体计算量。 Max-Pooling 操作需要在架构设计期间选择内核大小和步长。 选择后，该操作会在输入上滑动具有指定步长的内核，同时仅从输入中选择每个内核切片的最大值以生成输出值。

###### 1.5全连接层

全连接层一般接在卷积神经网络的最后，用于提取卷积和池化之后的特征向量；并基于提取的特征向量进行图像分类。所以全连接层即充当了特征提取器，又充当了分类器的角色。

#### 代码实现（基于MindSpore框架）

from mindvision.dataset import Mnist

# 下载并处理MNIST数据集

download\_train = Mnist(path="./MNIST\_Data", split="train", batch\_size=32, repeat\_num=1, shuffle=True, resize=32, download=False)

download\_eval = Mnist(path="./MNIST\_Data", split="test", batch\_size=32, resize=32, download=False)

dataset\_train = download\_train.run()

dataset\_eval = download\_eval.run()

# 搭建LeNet-5 网络

import mindspore.nn as nn

class LeNet5(nn.Cell):

"""

LeNet-5网络结构

"""

def \_\_init\_\_(self, num\_class=10, num\_channel=1):

super(LeNet5, self).\_\_init\_\_()

# 卷积层，输入的通道数为num\_channel，输出的通道数为6，卷积核大小为5\*5

self.conv1 = nn.Conv2d(num\_channel, 6, 5, pad\_mode='valid')

# 卷积层，输入的通道数为6，输出的通道数为16，卷积核大小为5\*5

self.conv2 = nn.Conv2d(6, 16, 5, pad\_mode='valid')

# 全连接层，输入个数为16\*5\*5，输出个数为120

self.fc1 = nn.Dense(16 \* 5 \* 5, 120)

# 全连接层，输入个数为120，输出个数为84

self.fc2 = nn.Dense(120, 84)

# 全连接层，输入个数为84，分类的个数为num\_class

self.fc3 = nn.Dense(84, num\_class)

# ReLU激活函数

self.relu = nn.ReLU()

# 池化层

self.max\_pool2d = nn.MaxPool2d(kernel\_size=2, stride=2)

# 多维数组展平为一维数组

self.flatten = nn.Flatten()

def construct(self, x):

# 使用定义好的运算构建前向网络

x = self.conv1(x)

x = self.relu(x)

x = self.max\_pool2d(x)

x = self.conv2(x)

x = self.relu(x)

x = self.max\_pool2d(x)

x = self.flatten(x)

x = self.fc1(x)

x = self.relu(x)

x = self.fc2(x)

x = self.relu(x)

x = self.fc3(x)

return x

network = LeNet5(num\_class=10)

import mindspore.nn as nn

# 定义损失函数

net\_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

# 定义优化器函数

net\_opt = nn.Momentum(network.trainable\_params(), learning\_rate=0.01, momentum=0.9)

import mindspore as ms

# 设置模型保存参数，模型训练保存参数的step为1875。

config\_ck = ms.CheckpointConfig(save\_checkpoint\_steps=1875, keep\_checkpoint\_max=10)

# 应用模型保存参数

ckpoint = ms.ModelCheckpoint(prefix="lenet", directory="./lenet", config=config\_ck)

from mindvision.engine.callback import LossMonitor

import mindspore as ms

from mindspore.train.callback import SummaryCollector

summary\_collector = SummaryCollector(summary\_dir='./summary\_dir', collect\_freq=1)

# 初始化模型参数

model = ms.Model(network, loss\_fn=net\_loss, optimizer=net\_opt, metrics={'accuracy'})

# 训练网络模型，并保存为lenet-1\_1875.ckpt文件

model.train(10, dataset\_train, callbacks=[ckpoint, LossMonitor(0.01, 1875),summary\_collector], dataset\_sink\_mode=False)

acc = model.eval(dataset\_eval,callbacks=[summary\_collector])

print("{}".format(acc))

import mindspore as ms

# 加载已经保存的用于测试的模型

param\_dict = ms.load\_checkpoint("./lenet/lenet-1\_1875.ckpt")

# 加载参数到网络中

ms.load\_param\_into\_net(network, param\_dict)

import numpy as np

import mindspore as ms

import matplotlib.pyplot as plt

mnist = Mnist("./MNIST\_Data", split="train", batch\_size=6, resize=32)

dataset\_infer = mnist.run()

ds\_test = dataset\_infer.create\_dict\_iterator()

data = next(ds\_test)

images = data["image"].asnumpy()

labels = data["label"].asnumpy()

plt.figure()

for i in range(1, 7):

plt.subplot(2, 3, i)

plt.imshow(images[i-1][0], interpolation="None", cmap="gray")

plt.show()

# 使用函数model.predict预测image对应分类

output = model.predict(ms.Tensor(data['image']))

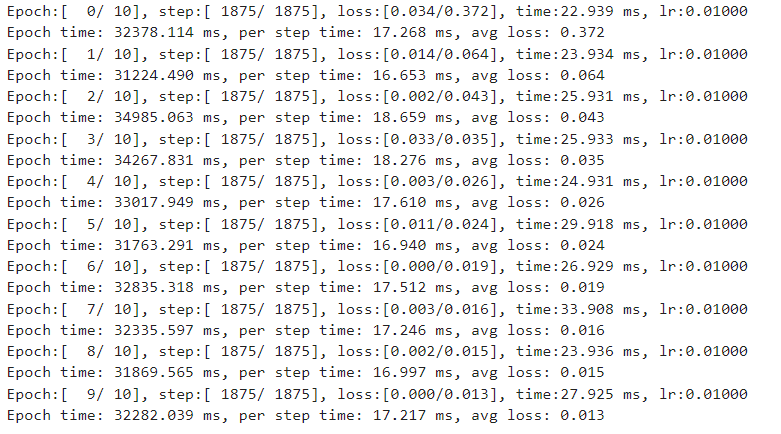
predicted = np.argmax(output.asnumpy(), axis=1)

# 输出预测分类与实际分类

print(f'Predicted: "{predicted}", Actual: "{labels}"')

#### 实验结果

###### 3.1 训练过程



###### 3.2 验证模型

