**摘 要**

本文提出了一种基于LeNet的改进卷积网络模型用于图像分类。MathorNet建立在以前使用高深卷积网有效地分类目标的成果上。相较于LeNet，无论是训练集还是测试集，准确度都更高，消耗时间是其2倍（在GPU条件不相同的情况下）。选用的数据是CIARF-10，Batch大小为100，训练集的Epoch设置为100，每个Epoch大约40s。在训练集上的最高精度可达100%，最终在测试集上的精度为84.33%，高于LeNet的76.23%。 MathorNet的Python（PyTorch）实现开源发布在：<https://github.com/mathors/MathorNet/>

**关键词：**图像分类；LeNet；MathorNet；CIARF-10

**目 录**

[1 概述 1](#_Toc11000070)

[2 算法原理 2](#_Toc11000071)

[2.1 卷积（Convolution） 2](#_Toc11000072)

[2.2 池化（Pooling） 2](#_Toc11000073)

[2.3 激活函数 3](#_Toc11000074)

[2.4 全连接神经网络（Fully connected neural network） 4](#_Toc11000075)

[3 算法设计 5](#_Toc11000076)

[3.1 最终代码 6](#_Toc11000077)

[4 实验结果及对比分析 9](#_Toc11000078)

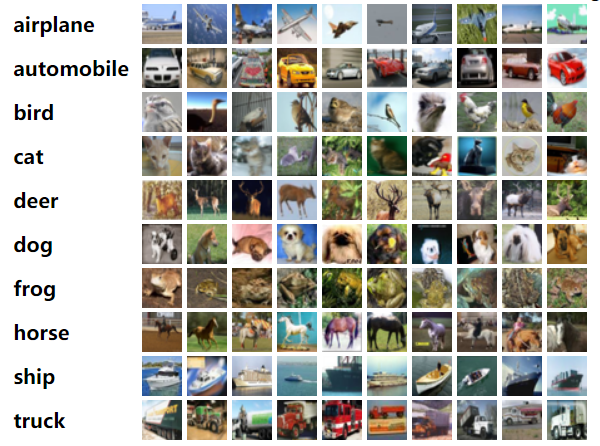
[5 结束语 10](#_Toc11000079)

[参考文献 11](#_Toc11000080)

# 1 概述

一直以来，深度卷积网络已经显著提高了图像分类和目标检测的准确性，CNN从90年代的LeNet[1]开始，21世纪初沉寂了10年，直到12年AlexNet[3]开始又再焕发第二春，从ZF Net[4]到VGG[5]，GoogLeNet[6]再到ResNet和最近的DenseNet，网络越来越深，架构越来越复杂，解决反向传播时梯度消失的方法也越来越巧妙。本文在LeNet的基础上，模仿VGG对于CNN层数的理解，通过加深卷积层从而达到模型精度的提高，并将此模型命名为MathorNet（本文作者英文名为Mathor）。

对比实验选用的数据集为CIFAR-10。CIFAR-10数据集由10类共60000个32x32彩色图像组成，每个类有6000个图像。有50000个训练图像和10000个测试图像



**图1.1 CIFAR-10数据概览图**

主要步骤为以下4个步骤：

（1）加载数据集

（2）定义神经网络

（3）定义Loss函数和优化器

（4）训练

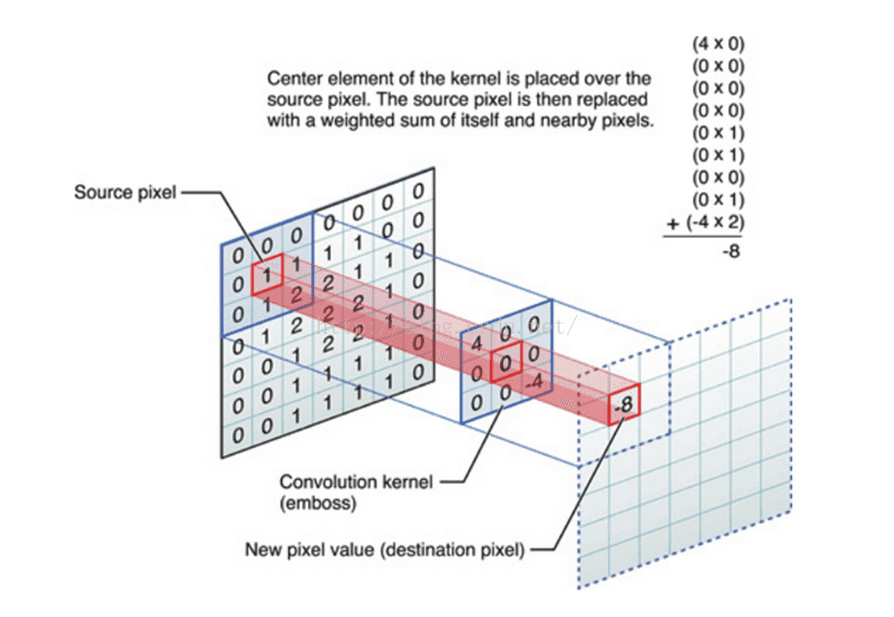
（5）测试

# 2 算法原理

接下来对该算法的四个核心步骤进行分析。

## 2.1 卷积（Convolution）

卷积是数字图像处理中常用的操作，对数字图像做卷积操作其实就是利用卷积核（卷积模板）在图像上滑动，将图像点上的像素灰度值与对应的卷积核上的数值相乘，然后将所有相乘后的值相加作为卷积核中间像素对应的图像上像素的灰度值，并最终滑动完所有图像的过程。



**图2.1 卷积过程图**

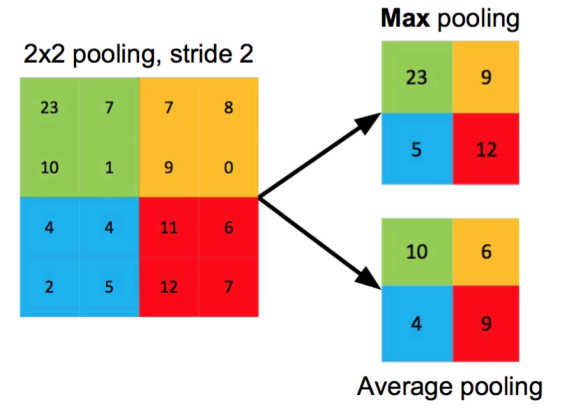
卷积公式如下：

PyTorch自带卷积函数Conv2d，因此可以快速完成图像卷积操作，值得一提的是，卷积后的图像大小会发生变化，公式为：

其中，代表卷积核的大小，表示在输入图像周围加0的圈数，为卷积核每次移动的步长。

## 2.2 池化（Pooling）

在完成卷积特征提取之后，对每一层的矩阵划分出几个不重合的区域，然后在每个区域上计算该区域内特征的均值或最大值，然后用这些均值或最大值参与后续的训练，这个过程就是池化。池化最大的优点就是显著减少参数数量



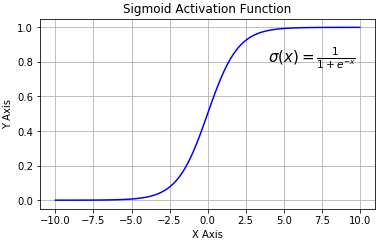
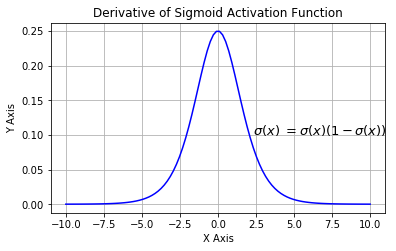
**图2.2 池化过程图**

对于CNN来说，最大的挑战就是参数太多。以我们使用的CIFAR-10数据集为例，每张图片的大小为3×32×32，共有288个参数，一张图片就这么多，全部图片加起来，参数数量是巨大的，这就导致模型在训练时，反向传播计算梯度太慢，参数更新也很慢。但是通过池化操作，可以有效减少参数数量。对于一个H×H大小的灰度图片，假设池化核大小为2，步长也为2，池化后的图片大小为。

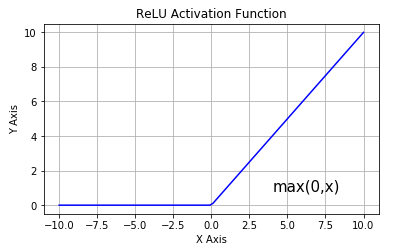
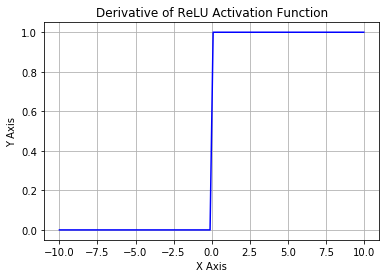
## 2.3 激活函数

如果没有激活函数，那么该网络仅能够表达线性映射，此时即便有再多的隐藏层，其整个网络跟单层神经网络也是等价的。因此也可以认为，只有加入了激活函数之后，深度神经网络才具备了分层的非线性映射学习能力。激活函数不是真的要去激活什么。在神经网络中，激活函数的作用是能够给神经网络加入一些非线性因素，使得神经网络可以更好地解决较为复杂的问题。

常用的激活函数有Sigmoid、ReLU等

**图2.3 Sigmoid函数 图2.4 Sigmoid函数的导数**

**图2.5 Sigmoid函数 图2.6 Sigmoid函数的导数**

关于激活函数的选取，LeNet使用的是Sigmoid，而从LeNet以后几乎所有的模型都是用的ReLu。Sigmoid的问题在于在反向传播的过程中，可能会存在梯度消失（即梯度趋于0）和梯度下降很慢的问题。而ReLU函数可以抵抗梯度消失的问题，而且由于使用了简单的阈值化，ReLU的计算效率很高

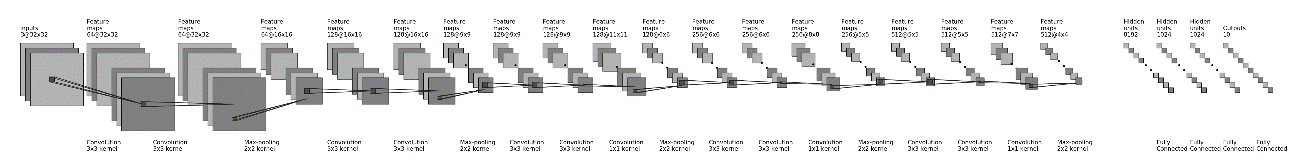
## 2.4 全连接神经网络（Fully connected neural network）

这里我并不想讲述神经网络的基本概念。我想提一下的是关于全连接层的设计。

阅读了AlexNet的论文以后，我发现它在其中有一个比较巧妙的设计——DropOut。由于全连接神经网络的效果太强，经常使得模型在训练过程中产生过拟合，因此AlexNet使用了一种叫做DropOut的方法来避免过拟合。事实证明这是十分有效的，后来的几乎所有模型都用上这一方法。DropOut是指，随机设定一个概率值，使得全连接神经网络中的每个神经元都有的概率在前向传播和反向下降的过程中失活（即不起任何作用）。这样表面上看会使得模型对于训练集的效果会变差，但实际上相当于变相给每个神经元的学习加压，增加的结果的鲁棒性，使得模型最终效果更好。

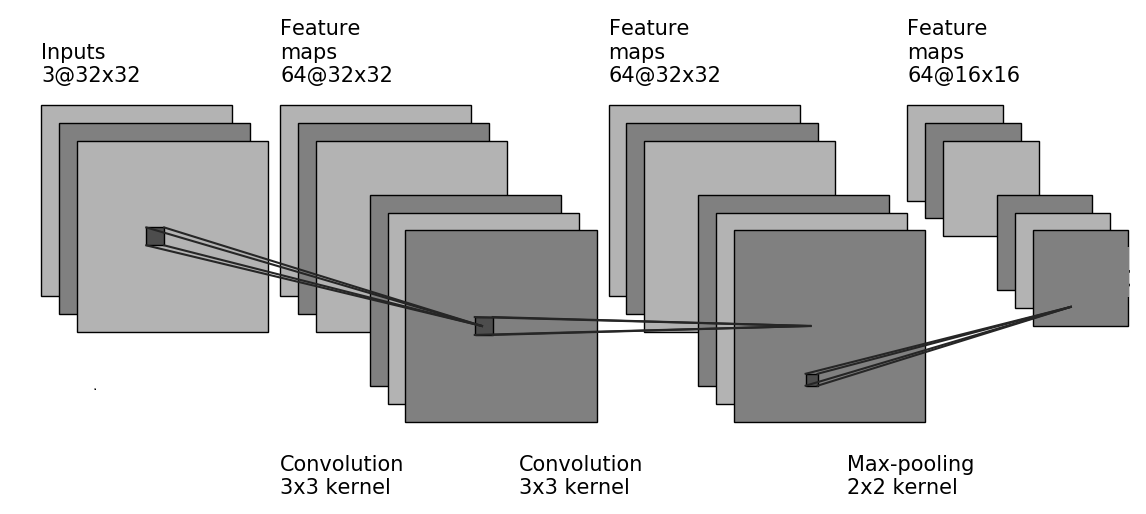
# 3 算法设计

首先给出MathorNet模型图



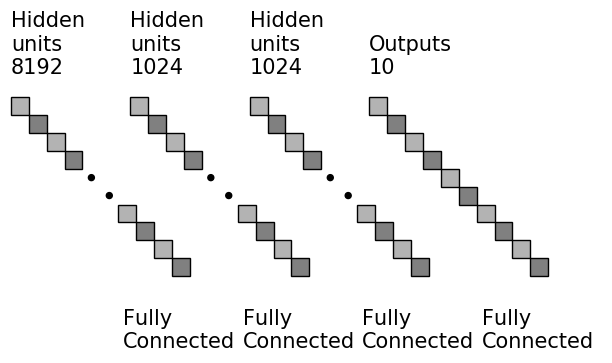
**图3.1 MathorNet模型**

由于MathorNet共有21层（18层卷积和池化层，3层全连接层），上面的图可能看不清楚，因此下面给出部分局部图并对其进行解释



**图3.2 MathorNet模型局部图1**

输入图像为大小为3×32×32。首先经过一次卷积，卷积核大小为3×3，共64个卷积核，并且对输入图像周围填上一圈0，即，输出图像为64×32×32。同样再经过一次卷积，依然得到64×32×32。然后进行一次Max-pooling，池化核大小为2×2，stride=2，输出图像大小为64×16×16



**图3.3 MathorNet模型局部图2**

经过大量的卷积和池化之后，最终将所有的参数排成一列，作为全连接神经网络的输入。从图上可以看出，最终输入参数共8192个，我设定第一层隐藏层有1024个神经元，第二层1024个，输出层10个，因为CIFAR-10共有10类标签。输出层每个神经元输出的是个概率值，代表这张图片为第i类物体的概率

## 3.1 最终代码

由于代码实在太长（将近200行），所以我在这里附上部分关键代码，其余的如果老师有兴趣请移步<https://github.com/mathors/MathorNet/blob/master/MathorNet.ipynb>

class Net(nn.Module):

def \_\_init\_\_(self):

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

self.conv1 = nn.Conv2d(3,64,3,padding=1)

self.conv2 = nn.Conv2d(64,64,3,padding=1)

self.pool1 = nn.MaxPool2d(2, 2)

self.bn1 = nn.BatchNorm2d(64)

self.relu1 = nn.ReLU()

self.conv3 = nn.Conv2d(64,128,3,padding=1)

self.conv4 = nn.Conv2d(128, 128, 3,padding=1)

self.pool2 = nn.MaxPool2d(2, 2, padding=1)

self.bn2 = nn.BatchNorm2d(128)

self.relu2 = nn.ReLU()

self.conv5 = nn.Conv2d(128,128, 3,padding=1)

self.conv6 = nn.Conv2d(128, 128, 3,padding=1)

self.conv7 = nn.Conv2d(128, 128, 1,padding=1)

self.pool3 = nn.MaxPool2d(2, 2, padding=1)

self.bn3 = nn.BatchNorm2d(128)

self.relu3 = nn.ReLU()

self.conv8 = nn.Conv2d(128, 256, 3,padding=1)

self.conv9 = nn.Conv2d(256, 256, 3, padding=1)

self.conv10 = nn.Conv2d(256, 256, 1, padding=1)

self.pool4 = nn.MaxPool2d(2, 2, padding=1)

self.bn4 = nn.BatchNorm2d(256)

self.relu4 = nn.ReLU()

self.conv11 = nn.Conv2d(256, 512, 3, padding=1)

self.conv12 = nn.Conv2d(512, 512, 3, padding=1)

self.conv13 = nn.Conv2d(512, 512, 1, padding=1)

self.pool5 = nn.MaxPool2d(2, 2, padding=1)

self.bn5 = nn.BatchNorm2d(512)

self.relu5 = nn.ReLU()

self.fc14 = nn.Linear(512\*4\*4,1024)

self.drop1 = nn.Dropout2d()

self.fc15 = nn.Linear(1024,1024)

self.drop2 = nn.Dropout2d()

self.fc16 = nn.Linear(1024,10)

def forward(self,x):

x = self.conv1(x)

x = self.conv2(x)

x = self.pool1(x)

x = self.bn1(x)

x = self.relu1(x)

x = self.conv3(x)

x = self.conv4(x)

x = self.pool2(x)

x = self.bn2(x)

x = self.relu2(x)

x = self.conv5(x)

x = self.conv6(x)

x = self.conv7(x)

x = self.pool3(x)

x = self.bn3(x)

x = self.relu3(x)

x = self.conv8(x)

x = self.conv9(x)

x = self.conv10(x)

x = self.pool4(x)

x = self.bn4(x)

x = self.relu4(x)

x = self.conv11(x)

x = self.conv12(x)

x = self.conv13(x)

x = self.pool5(x)

x = self.bn5(x)

x = self.relu5(x)

# print(" x shape ",x.size())

x = x.view(-1,512\*4\*4)

x = F.relu(self.fc14(x))

x = self.drop1(x)

x = F.relu(self.fc15(x))

x = self.drop2(x)

x = self.fc16(x)

return x

# 4 实验结果及对比分析

**表4.1 MathorNet和其它模型实验结果对比**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Network** | **GPU** | **Batch\_size** | **Epoch** | **Training time** | **Accuraacy(%)** |
| LeNet | GTX1080TI | 128 | 200 | 3600s | 76.23 |
| MathorNet | Tesla K80 | 100 | 200 | 8000s | 84.33 |
| Vgg19-Network | GTX1080TI | 128 | 200 | 6780 | 93.53 |

从实验结果上来看，虽然MathorNet模仿了VGG模型，但是无论在速度还是精度上都没有打败VGG。不过相较于LeNet来说已经提升太多了。同时，回想VGG的论文，它在其中提到了这么一句话：“我们猜想，模型的精度与其深度有关，因此我们考虑加深模型的深度”。虽然MathorNet没有打败VGG，但是相较于浅层的LeNet，MathorNet确实提升了不少精度，由此也能证明VGG论文中的想法

参考文献

[1] LeCun Y, Boser B, Denker J S, et al. Backpropagation applied to handwritten zip code recognition[J]. Neural computation, 1989, 1(4): 541-551.

[2] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.

[3] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C].Advances in neural information processing systems. 2012: 1097-1105.

[4] Zeiler M D, Fergus R. Visualizing and understanding convolutional networks[C].European conference on computer vision. Springer, Cham, 2014: 818-833.

[5] Zeiler M D, Fergus R. Visualizing and understanding convolutional networks[C].European conference on computer vision. Springer, Cham, 2014: 818-833.

[6] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[C]. Cvpr, 2015.

[7] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.