

转载

手把手带你走进卷积神经网络！

2019-06-17 18:59:27 CSDN资讯 阅读数 939



点击“上方蓝字”关注CSDN

这是一篇关于CNN（卷积神经网络）的简单指南，本文将介绍CNN如何工作，以及如何在Python中从头开始构建一个CNN。



© 视觉中国

作者 | victorzhou

译者 | 虎说，责编 | 郭芮

出品 | CSDN (ID: CSDNnews)

以下为译文：

在过去的几年中，有很多关于卷积神经网络（CNN）的讨论，尤其是因为它们已经彻底改变了计算机视觉领域。在这篇文章中，我们将基于神经网络的基本背景知识，探索CNN是什么，理解它们是如何工作的，并使用Python中的numpy从头开始构建一个真正的卷积神经网络。

本文假设读者有一定的神经网络的基本知识。如果你想要了解一些关于神经网络的知识，你可以读一下我的关于对神经网络的介绍（<https://victorzhou.com/blog/intro-to-neural-networks/>）。

现在，让我们开始今天的话题。



动机

CNN的经典用例是执行图像分类，例如查看宠物的图像并确定它是猫还是狗。这是一项看似非常简单的任务，你可能会这样的疑惑：为什么不使用普通的神经网络呢？不得不说这是一个好问题。

原因1：图像很大

目前用于计算机视觉问题的图像通常为224x224甚至更大。想象一下，构建一个神经网络来处理224x224彩色图像：包括图像中的3个颜色通道（RGB），即 $224 \times 224 \times 3 = 150528$ 个输入权重！这种网络中的典型隐藏层可能有1024个节点，因此我们必须仅为第一层训练 $150528 \times 1024 = 15$ 亿个权重。想象一下拥有15亿个权重的神经网络，是不是太大了？这几乎是不可能完成训练的。

最重要的是其实我们并不需要那么多的权重，相反我们仅知道像素点其邻居的点才是最有用的。因为图像中的物体是由小的局部特征组成的，如圆形虹膜或一张纸的方角。对于第一个隐藏层中的每个节点来说，查看每个像素似乎是很浪费的！

原因2：位置可变

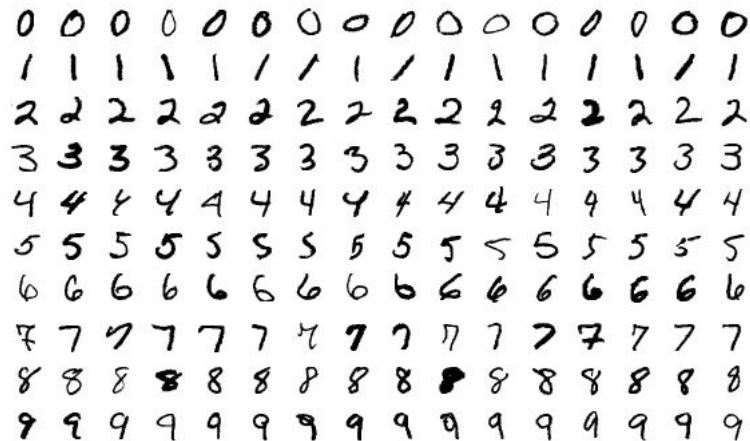
如果你训练了一个网络来检测狗，那么无论图像出现在哪张照片中，你都希望它能够检测到狗。想象一下，训练一个在某个狗图像上运行良好的网络，然后为它提供相同图像的略微移位版本，此时的网络会有完全不同的反应！

那么CNN是如何帮助我们解决这些问题的呢？不要着急我们很快就会看到CNN如何帮助我们缓解这些问题！

2

数据集

在这篇文章中，我们将解决计算机视觉的“Hello，World！”：MNIST手写数字分类问题。很简单：给定图像，将其分类为数字。



来自MNIST数据集的样本图像

MNIST数据集集中的每个图像都是28x28，其中包含了一个居中的灰度数字。说实话，一个正常的神经网络实际上可以很好地解决这个问题。你可以将每个图像视为 $28 \times 28 = 784$ 维向量，将其输入到784-dim图层，堆叠一些隐藏图层，最后输出10个节点的输出图层，每个数字1个。

这样做可以完成任务，因为MNIST数据集包含的都是些居中的小图像，因此我们不会遇到上述的大小或移位问题。但是，请记住，大多数现实世界的图像分类问题并不容易。

那么就让我们进入CNN吧！

3

卷积

什么是卷积神经网络？

它们基本上是使用卷积层的神经网络，即Conv层，它们基于卷积的数学运算。Conv图层由一组过滤器组成，你可以将其视为2d数字矩阵。这是一个示例3x3过滤器：

-1	0	1
-2	0	2
-1	0	1

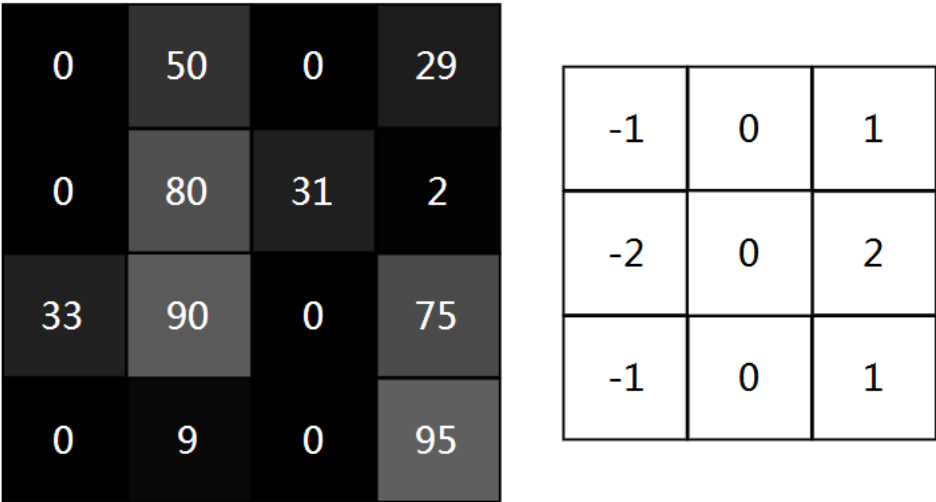
一个3x3过滤器

我们可以通过将滤波器与输入图像进行卷积来产生输出图像。这包括：

- 在某个位置覆盖图像顶部的过滤器；
- 在过滤器中的值与图像中的相应值之间执行逐元素乘法；
- 所有元素（ element-wise products ）求和，此和是输出图像中目标像素的输出值。
- 重复所有位置。

注释：实际上我们（以及许多CNN实现）在技术上使用互相关（ Cross-correlation ）而不是卷积，但它们几乎完全相同。

这4步描述有点抽象，所以让我们举个例子吧，考虑这个微小的4x4灰度图像和这个3x3过滤器：



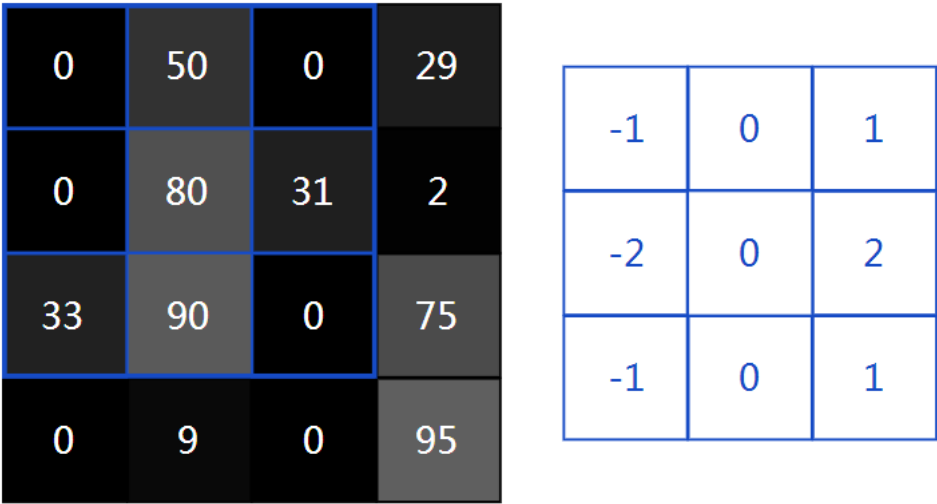
4x4图像（左）和3x3滤镜（右）

图像中的数字表示像素强度，其中0是黑色，255是白色。我们将对输入图像和过滤器进行卷积以生成2x2输出图像：

?	?
?	?

2x2输出图像

首先，让我们将过滤器叠加在图片的左上角：



第1步：将过滤器（右）叠加在图像上方（左）

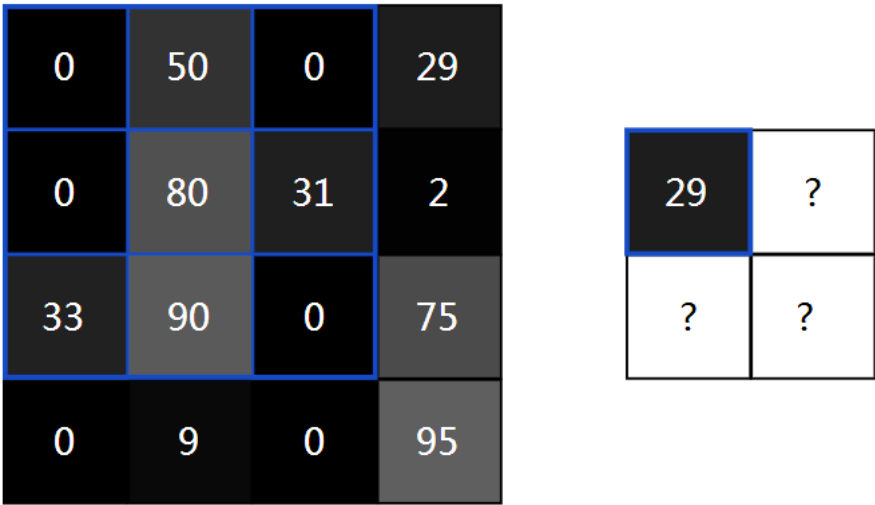
接下来，我们在重叠图像值和过滤器值之间执行逐元素乘法。以下是结果，从左上角开始向右，然后向下：

Image Value	Filter Value	Result
0	-1	0
50	0	0
0	1	0
0	-2	0
80	0	0
31	2	62
33	-1	-33
90	0	0
0	1	0

第2步：执行逐元素乘法。

接下来，我们总结所有结果。这很容易： $62 - 33 = 29$ 。

最后，我们将结果放在输出图像的目标像素中。由于我们的过滤器覆盖在输入图像的左上角，因此我们的目标像素是输出图像的左上角像素：



我们做同样的步骤来生成输出图像的其余部分：

3.1这有用吗？

用过滤器卷积图像有什么作用？我们可以先使用我们一直使用的示例3x3滤波器，这通常被称为垂直索贝尔滤波器：

-1	0	1
-2	0	2
-1	0	1

垂直索贝尔滤波器

以下是垂直Sobel滤波器的示例：



垂直Sobel滤波器卷积的图像

同样，还有一个水平Sobel滤波器：

1	2	1
0	0	0
-1	-2	-1

水平Sobel滤波器



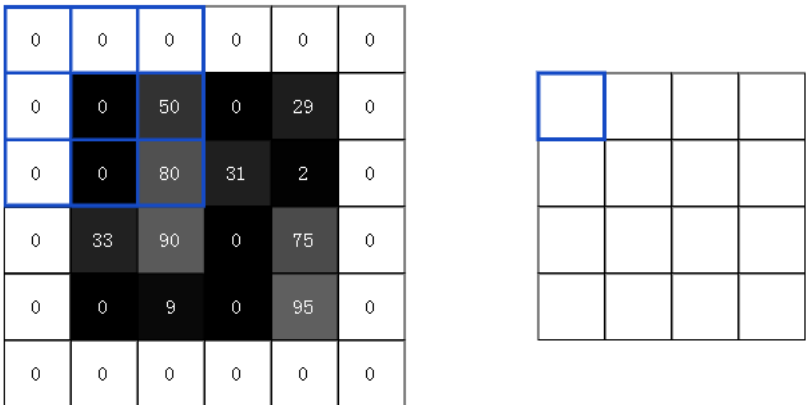
水平Sobel滤波器卷积的图像

到底发生了什么？Sobel滤波器是边缘检测器。垂直Sobel滤波器检测的是垂直边缘，水平Sobel滤波器的是检测水平边缘。现在可以轻松解释输出图像：输出图像中的亮像素（具有高值的像素）表示原始图像中存在强边缘。

你能看出为什么边缘检测图像可能比原始图像更有用吗？回想一下我们的MNIST手写数字分类问题。在MNIST上训练的CNN可以寻找数字1，例如，通过使用边缘检测滤波器并检查图像中心附近的两个突出的垂直边缘。通常，卷积有助于我们查找特定的本地化图像特征。

3.2填充（Padding）

还记得先用3x3过滤器对4x4的输入图像进行卷积，以产生2x2输出图像吗？通常，我们希望输出图像的大小与输入图像的大小相同。为此，我们在图像周围添加零，以便我们可以在更多位置叠加过滤器。3x3滤镜需要1个像素的填充：



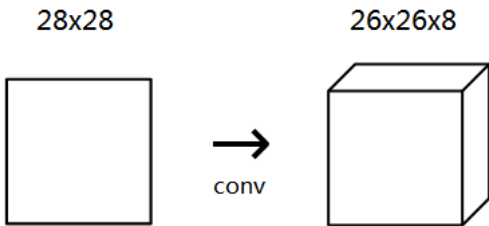
4x4输入与3x3滤波器卷积，使用填充以产生4x4输出

这称为“相同”填充，因为输入和输出具有相同的尺寸。不使用任何填充，这是我们一直在做的，有时也被称为“有效”填充。

3.3 Conv图层

既然我们知道图像卷积是如何工作的以及为什么用它，那让我们看看它是在CNN中实际使用的。如前所述，CNN包括使用一组过滤器将输入图像转换为输出图像的conv layer。conv层的主要参数是它具有的过滤器数量。

对于我们的MNIST CNN，我们将使用一个带有8个过滤器的小conv layer作为我们网络中的初始层。这意味着它会将28x28输入图像转换为26x26x8输出向量：



提醒：输出为26x26x8而不是28x28x8，因为我们使用有效填充，这会将输入的宽度和高度减少2。

conv layer中的4个过滤器中的每一个都会产生26x26输出，因此堆叠在一起它们构成26x26x8的向量。

3.4执行卷积

是时候将我们学到的东西放到代码中了！我们将实现一个conv layer的前馈部分，该部分负责处理带有输入图像的过滤器，以产生输出向量。为简单起见，我们假设过滤器是3x3（其实5x5和7x7过滤器也很常见）。

让我们开始实现一个conv layer类：

该类只有一个参数：过滤器的数量。在构造函数中，我们存储过滤器的数量并使用NumPy的randn（）方法初始化随机过滤器数组。

注意：在初始化期间初始值很重要，如果初始值太大或太小，则训练网络将无效。要了解更多信息，请阅读Xavier Initialization（<https://www.quora.com/What-is-an-intuitive-explanation-of-the-Xavier-Initialization-for-Deep-Neural-Networks>）。

接下来，实际卷积：

```
class Conv3x3:
    # ...

    def iterate_regions(self, image):
        """
        Generates all possible 3x3 image regions using valid padding.
        - image is a 2d numpy array
        """
        h, w = image.shape

        for i in range(h - 2):
            for j in range(w - 2):
                im_region = image[i:(i + 3), j:(j + 3)]
                yield im_region, i, j

    def forward(self, input):
        """
        Performs a forward pass of the conv layer using the given input.
        Returns a 3d numpy array with dimensions (h, w, num_filters).
        - input is a 2d numpy array
        """
        h, w = input.shape
        output = np.zeros((h - 2, w - 2, self.num_filters))

        for im_region, i, j in self.iterate_regions(input):
            output[i, j] = np.sum(im_region * self.filters, axis=(1, 2))

        return output
```

iterate_regions()是一个辅助生成器方法，为我们生成所有有效的3x3图像区域，这对于稍后实现此类的后向部分非常有用。

上面是实际执行卷积的代码行。让我们分解一下：

- im_region：一个包含相关图像区域的3x3阵列。
- self.filters：一个3d数组。
- im_region*self.filtersself.filters：我们使用numpy的广播（broadcasting）功能以元素方式乘以两个数组，结果是具有相同尺寸的3d数组。
- axis=(1,2)num_filters：我们使用np.sum（）上一步的结果，产生一个长度为1d的数组，其中每个元素包含相应过滤器的卷积结果。
- 我们将结果分配给output[i,j]，其中包含输出中像素的卷积结果（i，j）。

对输出中的每个像素执行上面的序列，直到我们获得我们想要的结果！让我们的代码进行测试运行：

```
import mnist
from conv import Conv3x3

# The mnist package handles the MNIST dataset for us!
# Learn more at https://github.com/datapythonista/mnist
train_images = mnist.train_images()
train_labels = mnist.train_labels()

conv = Conv3x3(8)
output = conv.forward(train_images[0])
print(output.shape) # (26, 26, 8)
```

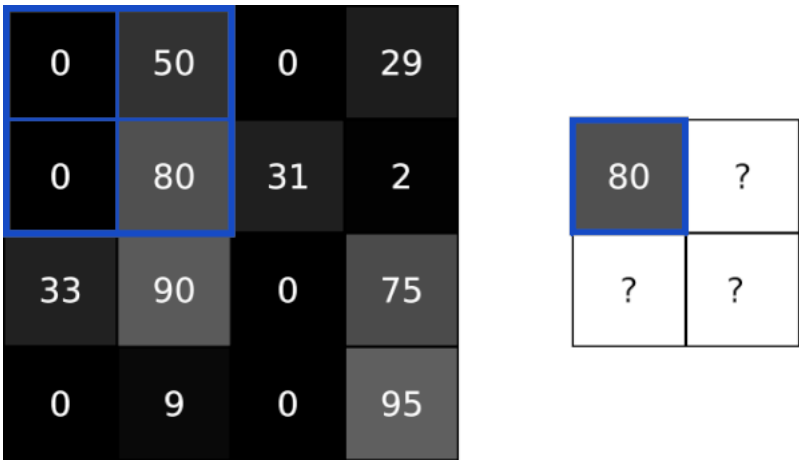
注意：为简单起见，在我们的Conv3x3实现中，我们假设输入是一个2d numpy数组，因为这是我们的MNIST图像的存储方式。这对我们有用，我们将它用作网络中的第一层，但大多数CNN都有更多的Conv层。如果我们要构建一个需要Conv3x3多次使用的更大的网络，我们必须使输入成为3d numpy数组。



池化 (Pooling)

图像中的相邻像素倾向于具有相似的值，因此conv layer通常也会为输出中的相邻像素产生类似的值。但是conv layer中输出的大部分信息都是冗余的，例如，如果我们使用边缘检测过滤器并在某个位置找到强边缘，那么我们也可能会在距离原始像素1个像素偏移的位置找到相对较强的边缘。但是，我们可能并没有找到任何新的东西。

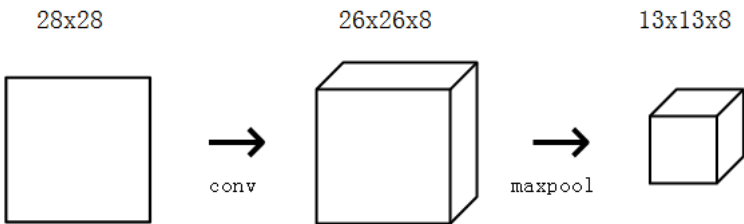
池化层解决了这个问题。他们所做的就是减少通过猜测在输入中产生的汇总值。该池通常是通过简单的操作完成max，min或average等这些操作。以下是池化大小为2的Max Pooling图层的示例：



4x4图像上的最大池（池大小为2）以产生2x2输出

为了执行最大池化，我们以2x2块（因为池大小=2）遍历输入图像，并将最大值放入相应像素的输出图像中。

对于我们的MNIST CNN，我们将在初始conv layer之后放置一个池大小为2的Max Pooling层，这样池化层就会将26x26x8输入转换为13x13x8输出：



4.1执行池化

我们将使用MaxPool2与上一节中的conv类相同的方法实现一个类：


```

import numpy as np

class MaxPool2:
    # A Max Pooling layer using a pool size of 2.

    def iterate_regions(self, image):
        """
        Generates non-overlapping 2x2 image regions to pool over.
        - image is a 2d numpy array
        """
        h, w, _ = image.shape
        new_h = h // 2
        new_w = w // 2

        for i in range(new_h):
            for j in range(new_w):
                im_region = image[(i * 2):(i * 2 + 2), (j * 2):(j * 2 + 2)]
                yield im_region, i, j

    def forward(self, input):
        """
        Performs a forward pass of the maxpool layer using the given input.
        Returns a 3d numpy array with dimensions (h / 2, w / 2, num_filters).
        - input is a 3d numpy array with dimensions (h, w, num_filters)
        """
        h, w, num_filters = input.shape
        output = np.zeros((h // 2, w // 2, num_filters))

        for im_region, i, j in self.iterate_regions(input):
            output[i, j] = np.amax(im_region, axis=(0, 1))

        return output

```

此类与我们之前实现的Conv3x3类工作方式类似。特别注意的是为了找到给定图像区域的最大值，我们使用np.amax（）。我们设置axis=(0,1)，因为我们只希望在前两个维度（高度和宽度）上进行最大化，而不是第三个维度，num_filters。

我们来试试吧！

```

import mnist
from conv import Conv3x3
from maxpool import MaxPool2

# The mnist package handles the MNIST dataset for us!
# Learn more at https://github.com/datapythonista/mnist
train_images = mnist.train_images()
train_labels = mnist.train_labels()

conv = Conv3x3(8)
pool = MaxPool2()

output = conv.forward(train_images[0])
output = pool.forward(output)
print(output.shape) # (13, 13, 8)

MNIST CNN马上就要完成了！

```



Softmax

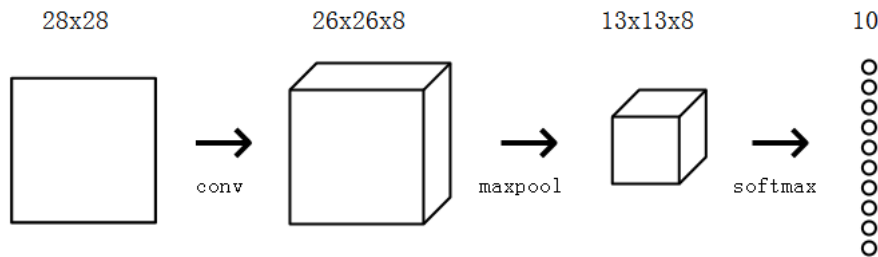
为了完成我们的CNN，我们需要让它能够实际进行预测。我们将通过使用标准最终层来实现多类分类问题：Softmax层，一个使用softmax激活函数的标准全连接（密集）层。

提示：完全连接层将每个节点连接到前一层的每个输出。我们在之前的神经网络的介绍中使用了完全连接的图层。

Softmax将任意实际值转换为概率。如果你对其背后的数学有兴趣，自己可以简单了解下，因为它很简单。

5.1用法

我们将使用一个带有10个节点的softmax层，每个节点都代表一个数字，softmax层是我们CNN的最后一层。图层中的每个节点都将连接到输入，在使用softmax变换之后，概率最高的节点表示的数字将成为CNN的输出！



5.2交叉熵损失函数

你可能会这样的疑问，为什么还要将输出转换为概率呢？最高输出值是否总是具有最高概率？如果你有这样的疑问，那说明你的感觉很对。我们实际上不需要使用softmax来预测数字，因为我们可以选择网络输出最高的数字！

softmax真正做的是帮助我们量化我们对预测的确定程度，这在训练和评估我们的CNN时非常有用。更具体地说，它可以帮我们确定每个预测的正确程度。

5.3实施Softmax

让我们实现一个Softmax图层层：

```

import numpy as np

class Softmax:
    # A standard fully-connected layer with softmax activation.

    def __init__(self, input_len, nodes):
        # We divide by input_len to reduce the variance of our initial values
        self.weights = np.random.randn(input_len, nodes) / input_len
        self.biases = np.zeros(nodes)

    def forward(self, input):
        """
        Performs a forward pass of the softmax layer using the given input.
        Returns a 1d numpy array containing the respective probability values.
        - input can be any array with any dimensions.
        """
        input = input.flatten()

        input_len, nodes = self.weights.shape

        totals = np.dot(input, self.weights) + self.biases
        exp = np.exp(totals)
        return exp / np.sum(exp, axis=0)
  
```

我们现在已经完成了CNN的整个编码工作! 把它放在一起:

```
import mnist
import numpy as np
from conv import Conv3x3
from maxpool import MaxPool2
from softmax import Softmax

# We only use the first 1k testing examples (out of 10k total)
# in the interest of time. Feel free to change this if you want.
test_images = mnist.test_images()[1000]
test_labels = mnist.test_labels()[1000]

conv = Conv3x3(8)          # 28x28x1 -> 26x26x8
pool = MaxPool2()          # 26x26x8 -> 13x13x8
softmax = Softmax(13 * 13 * 8, 10) # 13x13x8 -> 10

def forward(image, label):
    """
    Completes a forward pass of the CNN and calculates the accuracy and
    cross-entropy loss.
    - image is a 2d numpy array
    - label is a digit
    """
    # We transform the image from [0, 255] to [-0.5, 0.5] to make it easier
    # to work with. This is standard practice.
    out = conv.forward((image / 255) - 0.5)
    out = pool.forward(out)
    out = softmax.forward(out)

    # Calculate cross-entropy loss and accuracy. np.log() is the natural log.
    loss = -np.log(out[label])
    acc = 1 if np.argmax(out) == label else 0

    return out, loss, acc

print('MNIST CNN initialized!')

loss = 0
num_correct = 0
for i, (im, label) in enumerate(zip(test_images, test_labels)):
    # Do a forward pass.
    _, l, acc = forward(im, label)
    loss += l
    num_correct += acc

# Print stats every 100 steps.
if i % 100 == 99:
    print(
        '[Step %d] Past 100 steps: Average Loss %.3f | Accuracy: %d%%' %
        (i + 1, loss / 100, num_correct)
    )
    loss = 0
    num_correct = 0
```

执行cnn.py, 我们可以得到:

MNIST CNN initialized!

[Step 100] Past 100 steps: Average Loss 2.302 | Accuracy: 11%

[Step 200] Past 100 steps: Average Loss 2.302 | Accuracy: 8%

[Step 300] Past 100 steps: Average Loss 2.302 | Accuracy: 3%

[Step 400] Past 100 steps: Average Loss 2.302 | Accuracy: 12%

想亲自尝试或修改这些代码？在浏览器中运行此CNN，你也可以在Github上找到它。



结论

关于CNN的介绍就到此结束了！在这篇文章中，我们：

- 介绍了为什么CNN可能对某些问题更有用，例如图像分类；
- 介绍了MNIST手写数字数据集；
- 了解了Conv图层，它将过滤器与图像进行卷积，以产生更有用的输出；
- 了解了Pooling图层，它可以帮助修剪除最有用特征之外的内容；
- 实现了Softmax层，因此我们可以使用交叉熵损失函数。

原文：<https://victorzhou.com/blog/intro-to-cnns-part-1/>

本文为 CSDN 翻译，转载请注明来源出处。

入行AI,你需要一本Python机器学习入门

https://edu.csdn.net/topic/ai30?utm_source=csdn_bw

【End】

CSDN 5G免费沙龙来啦！

6月29日，微软（中国）首席技术官韦青、北京邮电大学信息与通信工程学院多媒体技术教研中心主任/博士生导师孙松林、爱立信中国研发部多天线高级专家朱怀松、爱立信中国研发部主任系统工程师刘阳等行业内顶尖的领军者、资深的技术专家们共聚一堂，共同探讨5G在物联网中的巨大潜能。

CSDN 程序员技术沙龙

第 2 期

5G 在物联网领域的 技术应用实践

🕒 2019年6月29日（周六）13:30-17:30

📍 北京市海淀区中关村E世界财富中心A座地下二层联合创业办公社
(People Squared)

演讲嘉宾:

13:30-13:40 主办方致辞

13:40-14:25 韦 青 微软(中国)首席技术官

14:25-15:10 孙松林 北京邮电大学 信息与通信工程学院
多媒体技术教研中心主任 博士生导师

15:10-15:20 休息

15:20-16:05 肖 江 金山云AIoT事业部高级研发总监

16:05-16:50 朱怀松 爱立信中国研发部多天线高级专家

刘 阳 爱立信中国研发部主任系统工程师

16:50-17:30 圆桌讨论



扫描二维码，免费报名!

热文推荐

华为准备替代安卓? 小米停止MIUI全球Beta计划; 首台类脑超级计算机2022有望诞生 | 极客头条

@程序员, 不容错过的 Vim 实用技巧请查收!

☞[微软发布 VS Code Java 安装程序，一键安装所有 Java 开发环境](#)

[那些去德国的程序员后来怎么样了？](#)


[独家对话V神! 质疑之下的以太坊路在何方？](#)

☞[苹果宣布加入CNCF；华为要求美国运营商支付专利费；微软删除最大的公开人脸识别数据集](#)

☞[阿里巴巴杨群：高并发场景下Python的性能挑战](#)

[新技术“红”不过十年，半监督学习为什么是个例外？](#)

☞[老码农冒死揭开编程黑幕：这些Bug让我认输，谁踩谁服！](#)

 [点击阅读原文，输入关键词，即可搜索您想要的 CSDN 文章。](#)



你点的每个“在看”，我都认真当成了喜欢

文章最后发布于: 2019-06-17 18:59:27

有 0 个人打赏