机器视角：长文揭秘图像处理和卷积神经网络架构

2017-07-06 机器之心

选自 Analyticsvidhya

机器之心编译

作者：DISHASHREE GUPTA

> *近日，Dishashree Gupta 在 Analyticsvidhya 上发表了一篇题为《Architecture of Convolutional Neural Networks (CNNs) demystified》的文章，对用于图像识别和分类的卷积神经网络架构作了深度揭秘；作者在文中还作了通盘演示，期望对 CNN 的工作机制有一个深入的剖析。机器之心对本文进行了编译，原文链接见文末。*

引言

先坦白地说，有一段时间我无法真正理解深度学习。我查看相关研究论文和文章，感觉深度学习异常复杂。我尝试去理解神经网络及其变体，但依然感到困难。

接着有一天，我决定一步一步，从基础开始。我把技术操作的步骤分解开来，并手动执行这些步骤（和计算），直到我理解它们如何工作。这相当费时，且令人紧张，但是结果非凡。

现在，我不仅对深度学习有了全面的理解，还在此基础上有了好想法，因为我的基础很扎实。随意地应用神经网络是一回事，理解它是什么以及背后的发生机制是另外一回事。

今天，我将与你共享我的心得，展示我如何上手卷积神经网络并最终弄明白了它。我将做一个通盘的展示，从而使你对 CNN 的工作机制有一个深入的了解。

在本文中，我将会讨论 CNN 背后的架构，其设计初衷在于解决图像识别和分类问题。同时我也会假设你对神经网络已经有了初步了解。

目录

1.机器如何看图?

2.如何帮助神经网络识别图像?

3.定义卷积神经网络

- 卷积层
- 池化层
- 输出层

4.小结

5.使用 CNN 分类图像

1. 机器如何看图?

人类大脑是一非常强大的机器，每秒内能看（捕捉）多张图，并在意识不到的情况下就完成了对这些图的处理。但机器并非如此。机器处理图像的第一步是理解，理解如何表达一张图像，进而读取图片。

简单来说，每个图像都是一系列特定排序的图点（像素）。如果你改变像素的顺序或颜色，图像也随之改变。举个例子，存储并读取一张上面写着数字 4 的图像。

基本上，机器会把图像打碎成像素矩阵，存储每个表示位置像素的颜色码。在下图的表示中，数值 1 是白色，256 是最深的绿色（为了简化，我们示例限制到了一种颜色）。

25	2	1	44
223	7	6	60
196	8	2	148
249	1	3	40
60	7	1	154
59	1	7	213
214	7	3	163
89	182	219	13
74	146	113	72
89	18	244	85
1	4	8	97
3	4	2	121
2	1	2	131
7	6	8	47
3	5	5	126
7	6	8	121
5	3	1	237

一旦你以这种格式存储完图像信息，下一步就是让神经网络理解这种排序与模式。

2. 如何帮助神经网络识别图像?

表征像素的数值是以特定的方式排序的。

25	2	1	44
223	7	6	60
196	8	2	148
249	1	3	40
60	7	1	154
59	1	7	213
214	7	3	163
89	182	219	13
74	146	113	72
89	18	244	85
1	4	8	97
3	4	2	121
2	1	2	131
7	6	8	47
3	5	5	126
7	6	8	121
5	3	1	237

假设我们尝试使用全连接网络识别图像，该如何做?

全连接网络可以通过平化它，把图像当作一个数组，并把像素值当作预测图像中数值的特征。明确地说，让网络理解理解下面图中发生了什么，非常的艰难。

25	223	196	249	60	.....	47	126	121	237

即使人类也很难理解上图中表达的含义是数字 4。我们完全丢失了像素的空间排列。

我们能做什么呢？可以尝试从原图像中提取特征，从而保留空间排列。

案例 1

这里我们使用一个权重乘以初始像素值。

25	2	1	44			=B2*$G$5	6	3	132
223	7	6	60			669	21	18	180
196	8	2	148		Weight	588	24	6	444
249	1	3	40		3	747	3	9	120
60	7	1	154			180	21	3	462
59	1	7	213			177	3	21	639
214	7	3	163			642	21	9	489
89	182	219	13			267	546	657	39
74	146	113	72			222	438	339	216
89	18	244	85			267	54	732	255
1	4	8	97			3	12	24	291
3	4	2	121			9	12	6	363
2	1	2	131			6	3	6	393
7	6	8	47			21	18	24	141
3	5	5	126			9	15	15	378
7	6	8	121			21	18	24	363
5	3	1	237			15	9	3	711

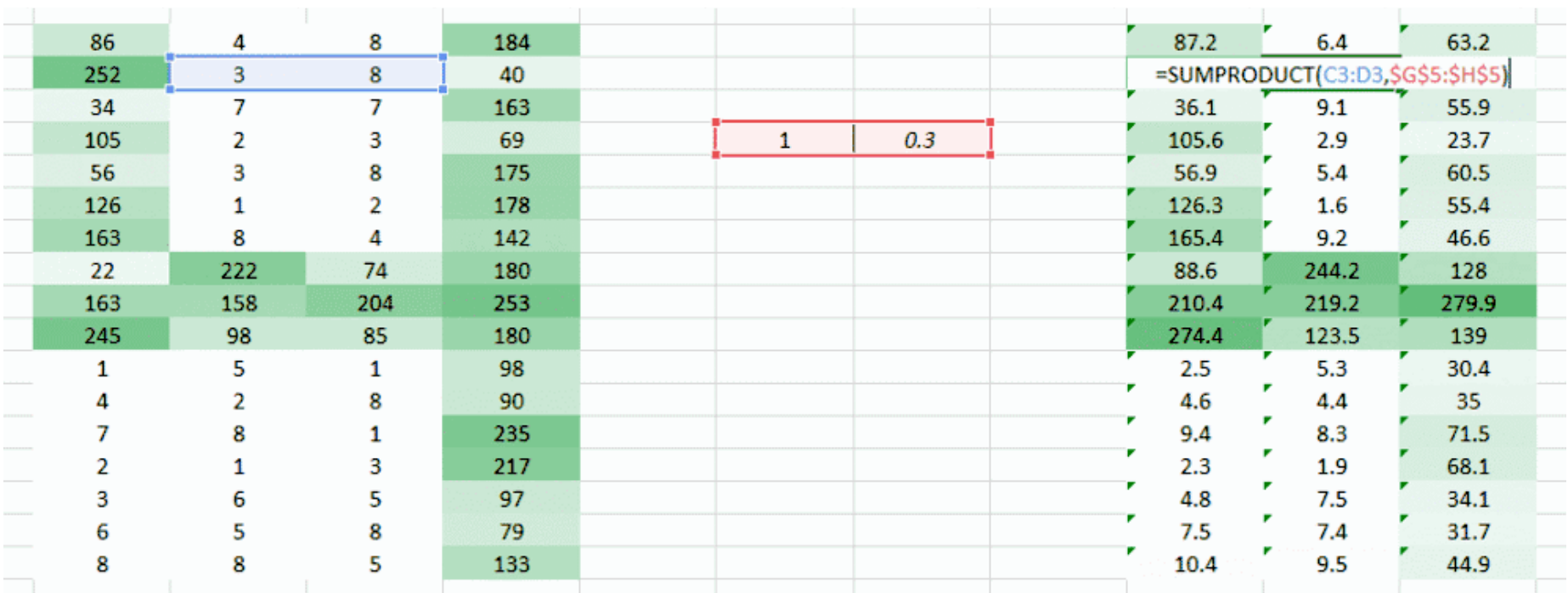
现在裸眼识别出这是「4」就变得更简单了。但把它交给全连接网络之前，还需要平整化（flatten) 它，要让我们能够保留图像的空间排列。

75	669	588	747	180	.....	141	378	363	711

案例 2

现在我们可以看到，把图像平整化完全破坏了它的排列。我们需要想出一种方式在没有平整化的情况下把图片馈送给网络，并且还要保留空间排列特征，也就是需要馈送像素值的 2D/3D 排列。

我们可以尝试一次采用图像的两个像素值，而非一个。这能给网络很好的洞见，观察邻近像素的特征。既然一次采用两个像素，那也就需要一次采用两个权重值了。



希望你能注意到图像从之前的 4 列数值变成了 3 列。因为我们现在一次移用两个像素（在每次移动中像素被共享），图像变的更小了。虽然图像变小了，我们仍能在很大程度上理解这是「4」。而且，要意识到的一个重点是，我们采用的是两个连贯的水平像素，因此只会考虑水平的排列。

这是我们从图像中提取特征的一种方式。我们可以看到左边和中间部分，但右边部分看起来不那么清楚。主要是因为两个问题：

1. 图片角落左边和右边是权重相乘一次得到的。
2. 左边仍旧保留，因为权重值高；右边因为略低的权重，有些丢失。

现在我们有两个问题，需要两个解决方案。

案例 3

遇到的问题是图像左右两角只被权重通过一次。我们需要做的是让网络像考虑其他像素一样考虑角落。我们有一个简单的方法解决这一问题：把零放在权重运动的两边。

0	86	4	8	184	0			25.8	87.2	6.4	63.2
0	252	3	8	40	0			75.6	252.9	5.4	20
0	34	7	7	163	0			10.2	36.1	9.1	55.9
0	105	2	3	69	0	1	0.3	31.5	105.6	2.9	23.7
0	56	3	8	175	0			16.8	56.9	5.4	60.5
0	126	1	2	178	0			37.8	126.3	1.6	55.4
0	163	8	4	142	0			48.9	165.4	9.2	46.6
0	22	222	74	180	0			6.6	88.6	244.2	128
0	163	158	204	253	0			48.9	210.4	219.2	279.9
0	245	98	85	180	0			73.5	274.4	123.5	139
0	1	5	1	98	0			0.3	2.5	5.3	30.4
0	4	2	8	90	0			1.2	4.6	4.4	35
0	7	8	1	235	0			2.1	9.4	8.3	71.5
0	2	1	3	217	0			0.6	2.3	1.9	68.1
0	3	6	5	97	0			0.9	4.8	7.5	34.1
0	6	5	8	79	0			1.8	7.5	7.4	31.7
0	8	8	5	133	0			2.4	10.4	9.5	44.9

184	0
40	0
163	0
69	0
175	0
178	0
142	0
180	0
253	0
180	0
98	0
90	0
235	0
217	0
97	0
79	0
133	0

1	0.3

25.8	87.2	6.4	63.2	=SUMPRODUCT(F2:G2,$I$5:$J$5)
75.6	252.9	5.4	20	
10.2	36.1	9.1	55.9	163
31.5	105.6	2.9	23.7	69
16.8	56.9	5.4	60.5	175
37.8	126.3	1.6	55.4	178
48.9	165.4	9.2	46.6	142
6.6	88.6	244.2	128	180
48.9	210.4	219.2	279.9	253
73.5	274.4	123.5	139	180
0.3	2.5	5.3	30.4	98
1.2	4.6	4.4	35	90
2.1	9.4	8.3	71.5	235
0.6	2.3	1.9	68.1	217
0.9	4.8	7.5	34.1	97
1.8	7.5	7.4	31.7	79
2.4	10.4	9.5	44.9	133

你可以看到通过添加零，来自角落的信息被再训练。图像也变得更大。这可被用于我们不想要缩小图像的情况下。

案例 4

这里我们试图解决的问题是右侧角落更小的权重值正在降低像素值，因此使其难以被我们识别。我们所能做的是采取多个权重值并将其结合起来。

(1,0.3) 的权重值给了我们一个输出表格

87.2	6.4	63.2
252.9	5.4	20
36.1	9.1	55.9
105.6	2.9	23.7
56.9	5.4	60.5
126.3	1.6	55.4
165.4	9.2	46.6
88.6	244.2	128
210.4	219.2	279.9
274.4	123.5	139
2.5	5.3	30.4
4.6	4.4	35
9.4	8.3	71.5
2.3	1.9	68.1
4.8	7.5	34.1
7.5	7.4	31.7
10.4	9.5	44.9

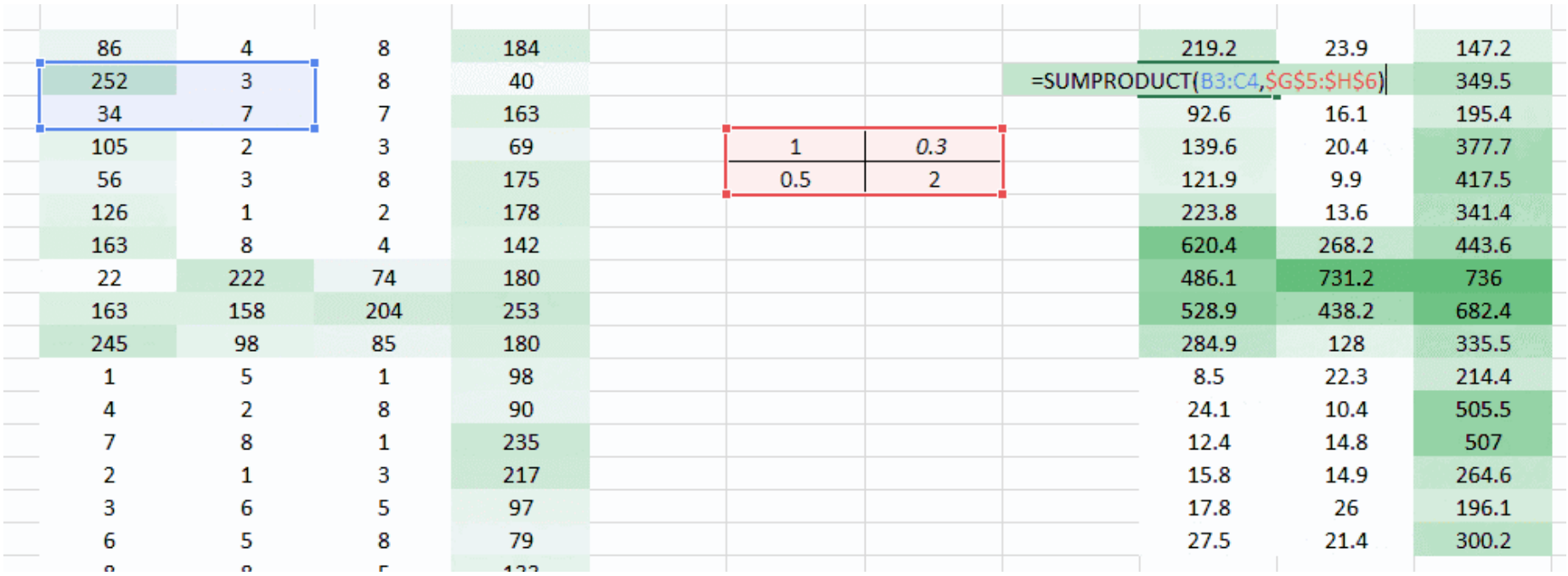
同时表格 (0.1,5) 的权重值也将给我们一个输出表格。

28.6	40.4	920.8
40.2	40.3	200.8
38.4	35.7	815.7
20.5	15.2	345.3
20.6	40.3	875.8
17.6	10.1	890.2
56.3	20.8	710.4
1112.2	392.2	907.4
806.3	1035.8	1285.4
514.5	434.8	908.5
25.1	5.5	490.1
10.4	40.2	450.8
40.7	5.8	1175.1
5.2	15.1	1085.3
30.3	25.6	485.5
25.6	40.5	395.8
40.8	25.8	665.5

两张图像的结合版本将会给我们一个清晰的图片。因此，我们所做的是简单地使用多个权重而不是一个，从而再训练图像的更多信息。最终结果将是上述两张图像的一个结合版本。

案例 5

我们到现在通过使用权重，试图把水平像素（horizontal pixel）结合起来。但是大多数情况下我们需要在水平和垂直方向上保持空间布局。我们采取 2D 矩阵权重，把像素在水平和垂直方向上结合起来。同样，记住已经有了水平和垂直方向的权重运动，输出会在水平和垂直方向上低一个像素。



86	4	8	184
252	3	8	40
34	7	7	163
105	2	3	69
56	3	8	175
126	1	2	178
163	8	4	142
22	222	74	180
163	158	204	253
245	98	85	180
1	5	1	98
4	2	8	90
7	8	1	235
2	1	3	217
3	6	5	97
6	5	8	79

1	*0.3*
0.5	2

219.2	23.9	147.2
92.6	16.1	195.4
139.6	20.4	377.7
121.9	9.9	417.5
223.8	13.6	341.4
620.4	268.2	443.6
486.1	731.2	736
528.9	438.2	682.4
284.9	128	335.5
8.5	22.3	214.4
24.1	10.4	505.5
12.4	14.8	507
15.8	14.9	264.6
17.8	26	196.1
27.5	21.4	300.2

特别感谢 Jeremy Howard 启发我创作了这些图像。

因此我们做了什么?

上面我们所做的事是试图通过使用图像的空间的安排从图像中提取特征。为了理解图像，理解像素如何安排对于一个网络极其重要。上面我们所做的也恰恰是一个卷积网络所做的。我们可以采用输入图像，定义权重矩阵，并且输入被卷积以从图像中提取特殊特征而无需损失其有关空间安排的信息。

这个方法的另一个重大好处是它可以减少图像的参数数量。正如所见，卷积图像相比于原始图像有更少的像素。

3.定义一个卷积神经网络

我们需要三个基本的元素来定义一个基本的卷积网络

1. 卷积层
2. 池化层（可选）
3. 输出层

卷积层

在这一层中，实际所发生的就像我们在上述案例 5 中见到的一样。假设我们有一个 6*6 的图像。我们定义一个权值矩阵，用来从图像中提取一定的特征。

INPUT IMAGE

18	54	51	239	244	188
55	121	75	78	95	88
35	24	204	113	109	221
3	154	104	235	25	130
15	253	225	159	78	233
68	85	180	214	245	0

WEIGHT

1	0	1
0	1	0
1	0	1

429

我们把权值初始化成一个 3*3 的矩阵。这个权值现在应该与图像结合，所有的像素都被覆盖至少一次，从而来产生一个卷积化的输出。上述的 429，是通过计算权值矩阵和输入图像的 3*3 高亮部分以元素方式进行的乘积的值而得到的。

INPUT IMAGE

18	54	51	239	244	188
55	121	75	78	95	88
35	24	204	113	109	221
3	154	104	235	25	130
15	253	225	159	78	233
68	85	180	214	245	0

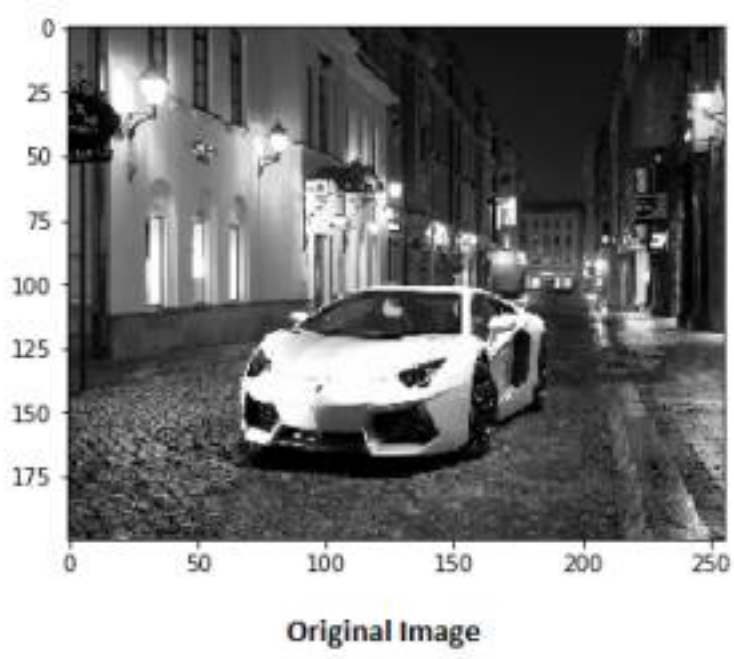
WEIGHT

1	0	1
0	1	0
1	0	1

429	505	686	856
261	792	412	640

现在 6*6 的图像转换成了 4*4 的图像。想象一下权值矩阵就像用来刷墙的刷子。首先在水平方向上用这个刷子进行刷墙，然后再向下移，对下一行进行水平粉刷。当权值矩阵沿着图像移动的时候，像素值再一次被使用。实际上，这样可以使参数在卷积神经网络中被共享。

下面我们以一个真实图像为例。



Original Image

Convoluted image

权值矩阵在图像里表现的像一个从原始图像矩阵中提取特定信息的过滤器。一个权值组合可能用来提取边缘（edge）信息，另一个可能是用来提取一个特定颜色，下一个就可能就是对不需要的噪点进行模糊化。

先对权值进行学习，然后损失函数可以被最小化，类似于多层感知机（MLP）。因此需要通过对参数进行学习来从原始图像中提取信息，从而来帮助网络进行正确的预测。当我们有多个卷积层的时候，初始层往往提取较多的一般特征，随着网络结构变得更深，权值矩阵提取的特征越来越复杂，并且越来越适用于眼前的问题。

步长（stride）和边界（padding）的概念

像我们在上面看到的一样，过滤器或者说权值矩阵，在整个图像范围内一次移动一个像素。我们可以把它定义成一个超参数（hyperparameter），从而来表示我们想让权值矩阵在图像内如何移动。如果权值矩阵一次移动一个像素，我们称其步长为 1。下面我们看一下步长为 2 时的情况。

INPUT IMAGE

18	54	51	239	244
55	121	75	78	95
35	24	204	113	109
3	154	104	235	25
15	253	225	159	78

WEIGHT

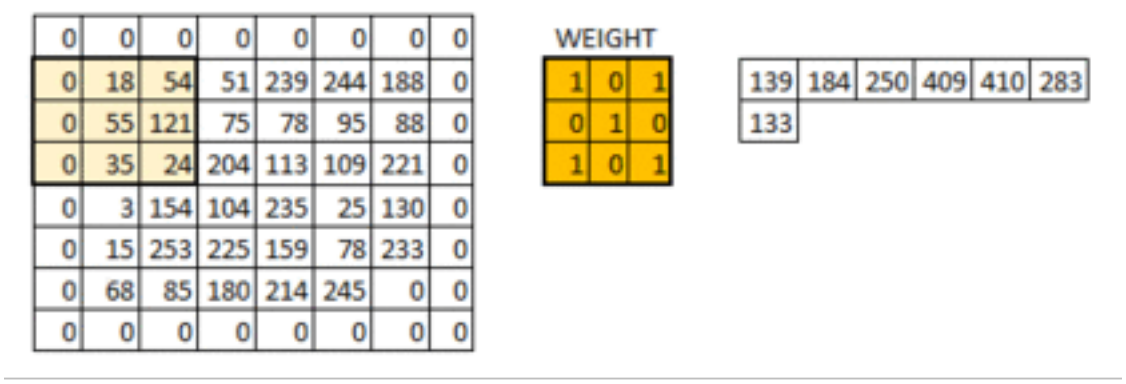
1	0	1
0	1	0
1	0	1

429	686

你可以看见当我们增加步长值的时候，图像的规格持续变小。在输入图像四周填充 0 边界可以解决这个问题。我们也可以在高步长值的情况下在图像四周填加不只一层的 0 边界。

0	0	0	0	0	0	0	0
0	18	54	51	239	244	188	0
0	55	121	75	78	95	88	0
0	35	24	204	113	109	221	0
0	3	154	104	235	25	130	0
0	15	253	225	159	78	233	0
0	68	85	180	214	245	0	0
0	0	0	0	0	0	0	0

我们可以看见在我们给图像填加一层 0 边界后，图像的原始形状是如何被保持的。由于输出图像和输入图像是大小相同的，所以这被称为 same padding。



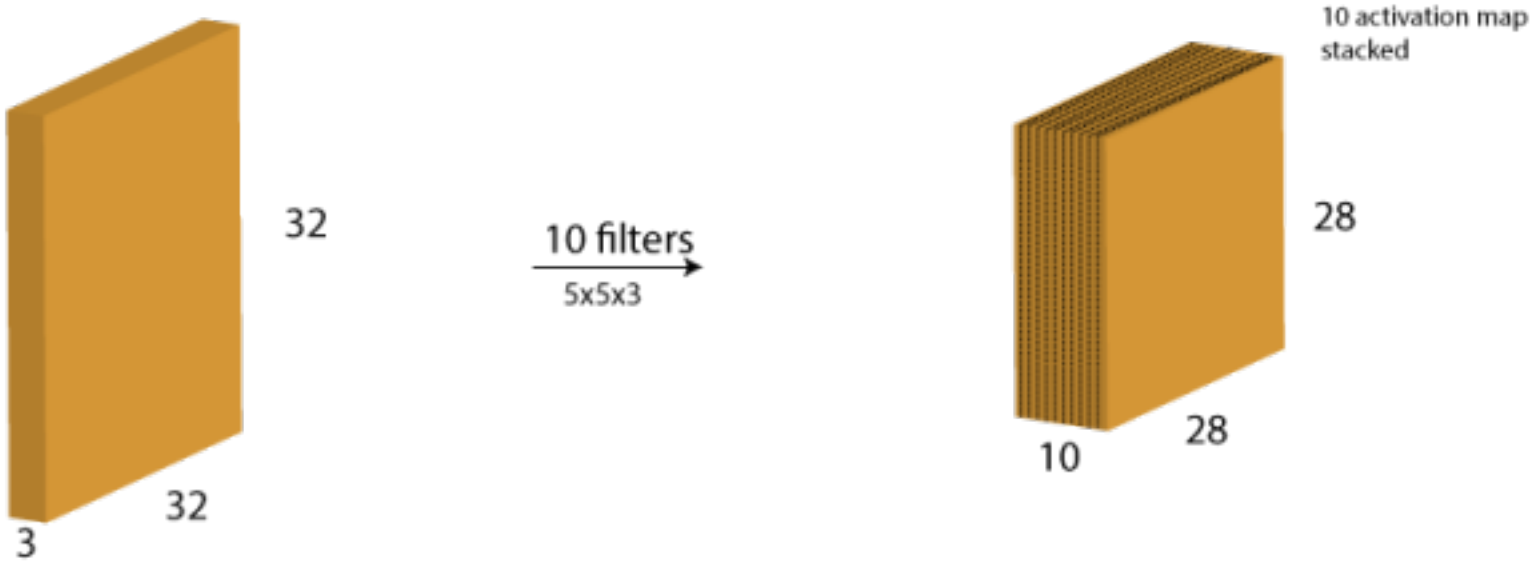
这就是 same padding（意味着我们仅考虑输入图像的有效像素）。中间的 4*4 像素是相同的。这里我们已经利用边界保留了更多信息，并且也已经保留了图像的原大小。

多过滤与激活图

需要记住的是权值的纵深维度（depth dimension）和输入图像的纵深维度是相同的。权值会延伸到输入图像的整个深度。因此，和一个单一权值矩阵进行卷积会产生一个单一纵深维度的卷积化输出。大多数情况下都不使用单一过滤器（权值矩阵），而是应用维度相同的多个过滤器。

每一个过滤器的输出被堆叠在一起，形成卷积图像的纵深维度。假设我们有一个 32*32*3 的输入。我们使用 5*5*3，带有 valid padding 的 10 个过滤器。输出的维度将会是 28*28*10。

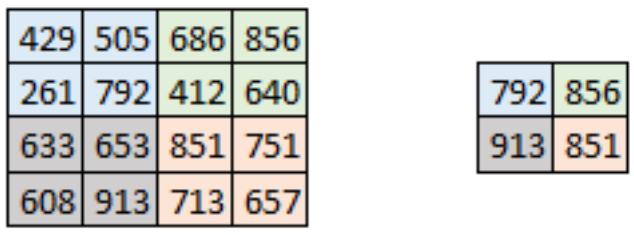
如下图所示：



激活图是卷积层的输出。

池化层

有时图像太大，我们需要减少训练参数的数量，它被要求在随后的卷积层之间周期性地引进池化层。池化的唯一目的是减少图像的空间大小。池化在每一个纵深维度上独自完成，因此图像的纵深保持不变。池化层的最常见形式是最大池化。

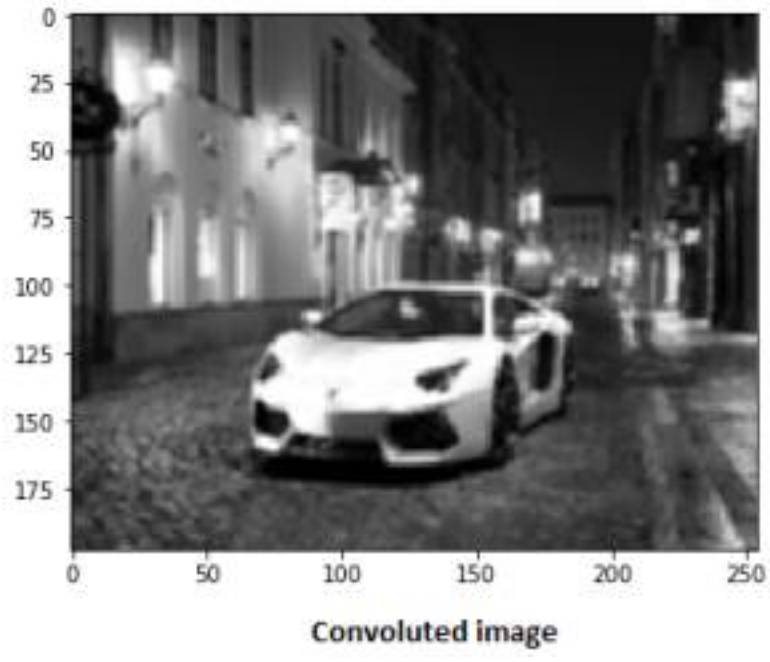


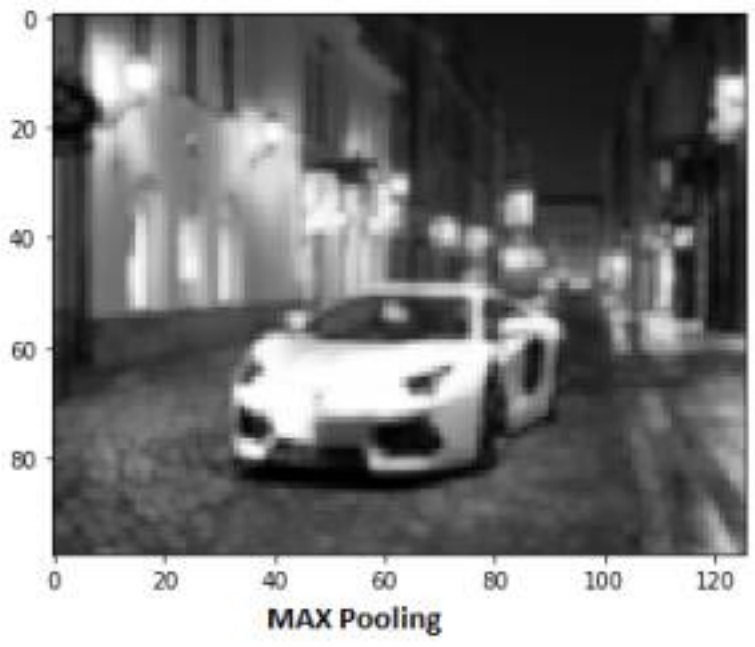
429	505	686	856
261	792	412	640
633	653	851	751
608	913	713	657

792	856
913	851

在这里，我们把步幅定为 2，池化尺寸也为 2。最大化执行也应用在每个卷机输出的深度尺寸中。正如你所看到的，最大池化操作后，4*4 卷积的输出变成了 2*2。

让我们看看最大池化在真实图片中的效果如何。





正如你看到的，我们卷积了图像，并最大池化了它。最大池化图像仍然保留了汽车在街上的信息。如果你仔细观察的话，你会发现图像的尺寸已经减半。这可以很大程度上减少参数。

同样，其他形式的池化也可以在系统中应用，如平均池化和 L2 规范池化。

输出维度

理解每个卷积层输入和输出的尺寸可能会有点难度。以下三点或许可以让你了解输出尺寸的问题。有三个超参数可以控制输出卷的大小。

1. 过滤器数量-输出卷的深度与过滤器的数量成正比。请记住该如何堆叠每个过滤器的输出以形成激活映射。激活图的深度等于过滤器的数量。
2. 步幅（Stride）-如果步幅是 1，那么我们处理图片的精细度就进入单像素级别了。更高的步幅意味着同时处理更多的像素，从而产生较小的输出量。
3. 零填充（zero padding）-这有助于我们保留输入图像的尺寸。如果添加了单零填充，则单步幅过滤器的运动会保持在原图尺寸。

我们可以应用一个简单的公式来计算输出尺寸。输出图像的空间尺寸可以计算为（[W-F + 2P] / S）+1。在这里，W 是输入尺寸，F 是过滤器的尺寸，P 是填充数量，S 是步幅数字。假如我们有一张 32*32*3 的输入图像，我们使用 10 个尺寸为 3*3*3 的过滤器，单步幅和零填充。

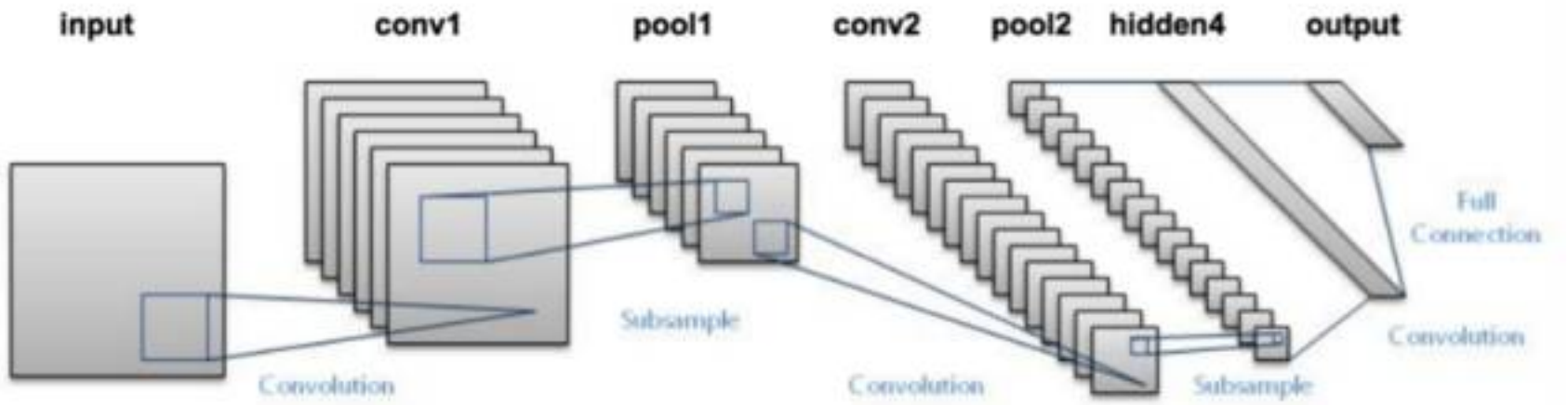
那么 W=32，F=3，P=0，S=1。输出深度等于应用的滤波器的数量，即 10，输出尺寸大小为 ([32-3+0]/1)+1 = 30。因此输出尺寸是 30*30*10。

输出层

在多层卷积和填充后，我们需要以类的形式输出。卷积和池化层只会提取特征，并减少原始图像带来的参数。然而，为了生成最终的输出，我们需要应用全连接层来生成一个等于我们需要的类的数量的输出。仅仅依靠卷积层是难以达到这个要求的。卷积层可以生成 3D 激活图，而我们只需要图像是否属于一个特定的类这样的内容。输出层具有类似分类交叉熵的损失函数，用于计算预测误差。一旦前向传播完成，反向传播就会开始更新权重与偏差，以减少误差和损失。

4. 小结

正如你所看到的，CNN 由不同的卷积层和池化层组成。让我们看看整个网络是什么样子：



- 我们将输入图像传递到第一个卷积层中，卷积后以激活图形式输出。图片在卷积层中过滤后的特征会被输出，并传递下去。
- 每个过滤器都会给出不同的特征，以帮助进行正确的类预测。因为我们需要保证图像大小的一致，所以我们使用同样的填充（零填充），否则填充会被使用，因为它可以帮助减少特征的数量。
- 随后加入池化层进一步减少参数的数量。
- 在预测最终提出前，数据会经过多个卷积和池化层的处理。卷积层会帮助提取特征，越深的卷积神经网络会提取越具体的特征，越浅的网络提取越浅显的特征。
- 如前所述，CNN 中的输出层是全连接层，其中来自其他层的输入在这里被平化和发送，以便将输出转换为网络所需的参数。
- 随后输出层会产生输出，这些信息会互相比较排除错误。损失函数是全连接输出层计算的均方根损失。随后我们会计算梯度错误。

- 错误会进行反向传播，以不断改进过滤器（权重）和偏差值。
- 一个训练周期由单次正向和反向传递完成。

5. 在 KERAS 中使用 CNN 对图像进行分类

让我们尝试一下，输入猫和狗的图片，让计算机识别它们。这是图像识别和分类的经典问题，机器在这里需要做的是看到图像，并理解猫与狗的不同外形特征。这些特征可以是外形轮廓，也可以是猫的胡须之类，卷积层会攫取这些特征。让我们把数据集拿来试验一下吧。

以下这些图片均来自数据集。




我们首先需要调整这些图像的大小，让它们形状相同。这是处理图像之前通常需要做的，因为在拍照时，让照下的图像都大小相同几乎不可能。

为了简化理解，我们在这里只用一个卷积层和一个池化层。注意：在 CNN 的应用阶段，这种简单的情况是不会发生的。

```
#import various packagesimport osimport numpy as npimport pandas as pdimport scipyimport skle
arnimport kerasfrom keras.models import Sequentialimport cv2from skimage import io
%matplotlib inline

#Defining the File Path

cat=os.listdir("/mnt/hdd/datasets/dogs_cats/train/cat")
```

```
dog=os.listdir("/mnt/hdd/datasets/dogs_cats/train/dog")
filepath="/mnt/hdd/datasets/dogs_cats/train/cat/"filepath2="/mnt/hdd/datasets/dogs_cats/train
/dog/"#Loading the Images

images=[]
label = []for i in cat:
image = scipy.misc.imread(filepath+i)
images.append(image)
label.append(0) #for cat imagesfor i in dog:
image = scipy.misc.imread(filepath2+i)
images.append(image)
label.append(1) #for dog images

#resizing all the imagesfor i in range(0,23000):
images[i]=cv2.resize(images[i],(300,300))

#converting images to arrays

images=np.array(images)
label=np.array(label)

# Defining the hyperparameters

filters=10filtersize=(5,5)

epochs =5batchsize=128input_shape=(300,300,3)

#Converting the target variable to the required sizefrom keras.utils.np_utils import to_categ
orical
label = to_categorical(label)

#Defining the model

model = Sequential()

model.add(keras.layers.InputLayer(input_shape=input_shape))

model.add(keras.layers.convolutional.Conv2D(filters, filtersize, strides=(1, 1), padding='val
id', data_format="channels_last", activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(keras.layers.Flatten())

model.add(keras.layers.Dense(units=2, input_dim=50,activation='softmax'))
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(images, label, epochs=epochs, batch_size=batchsize,validation_split=0.3)

model.summary()
```

在这一模型中，我只使用了单一卷积和池化层，可训练参数是 219,801。很好奇如果我在这种情况使用了 MLP 会有多少参数。通过增加更多的卷积和池化层，你可以进一步降低参数的数量。我们添加的卷积层越多，被提取的特征就会更具体和复杂。

在该模型中，我只使用了一个卷积层和池化层，可训练参数量为 219,801。如果想知道使用 MLP 在这种情况下会得到多少，你可以通过加入更多卷积和池化层来减少参数的数量。越多的卷积层意味着提取出来的特征更加具体，更加复杂。

结语

希望本文能够让你认识卷积神经网络，这篇文章没有深入 CNN 的复杂数学原理。如果希望增进了解，你可以尝试构建自己的卷积神经网络，借此来了解它运行和预测的原理。SYNCED

原文链接：https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/

阅读原文