卷積神經網路的複雜度分析

視學演算法 2022-01-04 11:33

作者 | Michael Yuan@知乎（已授權）

來源 | https://zhuanlan.zhihu.com/p/31575074

編輯 | 極市平臺

導讀

在梳理CNN經典模型的過程中，作者理解到其實經典模型演進中的很多創新點都與改善模型計算複雜度緊密相關，因此今天就讓我們對卷積神經網路的複雜度分析簡單總結一下。

在梳理CNN經典模型的過程中，我理解到其實經典模型演進中的**很多創新點都與改善模型計算複雜度緊密相關**，因此今天就讓我們對卷積神經網路的複雜度分析簡單總結一下下。

> 本文主要關注的是**針對模型本身的複雜度分析**（其實並不是很複雜啦~）。 如果想要進一步評估模型**在計算平臺上的理論計算性能**，則需要瞭解 Roofline Model 的相關理論，歡迎閱讀本文的進階版： Roofline Model 與深度學習模型的性能分析。 （連結：https://zhuanlan.zhihu.com/p/34204282 ）

"複雜度分析"其實沒有那麼複雜啦~

時間複雜度

即模型的運算次數，可用 **FLOPs** 衡量，也就是浮點運算次數（FLoating-point OPerations）。

1.1 單個捲積層的時間複雜度

- M 每個卷積核輸出特徵圖的邊長 $(Feature\ Map)$
- K 每個捲積核的邊長 $(Kernel)$
- C_{in} 每個卷積核的通道數，也即輸入通道數，也即上一層的輸出通道數。
- C_{out} 本卷積層具有的卷積核個數，也即輸出通道數。
- 可見，每個捲積層的時間複雜度由**輸出**特徵圖面積M^2、卷積核面 K^2、輸入C_{in}和輸出通道數 C_{out} 完全決定。
- 其中，**輸出**特徵圖尺寸本身又由**輸入**矩陣尺寸X 、卷積核尺寸 、 $Padding$、 $Stride$ 這四個參數所決定，表示如下： K

$$M = (X - K + 2 * Padding)/Stride + 1$$

- 注1：為了簡化表達式中的變數個數，這裡統一假設輸入和卷積核的形狀都是正方形。
- 注2：嚴格來講每層應該還包含1個$Bias$參數，這裡為了簡潔就省略了。

1.2 卷積神經網路整體的時間複雜度

$$\textbf{Time} \sim O\left(\sum_{l=1}^{D} M_l^2 \cdot K_l^2 \cdot C_{l-1} \cdot C_l\right)$$

- D 神經網路所具有的捲積層數，也即網路的深度。
- l 神經網路第 l 個捲積層
- C_l 神經網路第 l 個卷積層的輸出通道數 C_{out} ，也即該層的卷積核個數。
- 對於第 l 個捲積層而言，其輸入通道數 C_{in} 就是第 $(l-1)$ 個卷積層的輸出通道數。
- 可見，CNN整體的時間複雜度並不神秘，只是所有捲積層的時間複雜度累加而已。
- 簡而言之，層內連乘，層間累加。

範例：用 Numpy 手動簡單實現二維卷積

假設 Stride = 1， Padding = 0， img 和 kernel 都是 np.ndarray.

```
def conv2d(img, kernel):
    height, width, in_channels = img.shape
    kernel_height, kernel_width, in_channels, out_channels = kernel.shape
    out_height = height - kernel_height + 1
    out_width = width - kernel_width + 1
    feature_maps = np.zeros(shape=(out_height, out_width, out_channels))
```

```
    for oc in range(out_channels):                # Iterate out_channels (# of
        for h in range(out_height):               # Iterate out_height
            for w in range(out_width):            # Iterate out_width
                for ic in range(in_channels):     # Iterate in_channels
                    patch = img[h: h + kernel_height, w: w + kernel_width, ic
                    feature_maps[h, w, oc] += np.sum(patch * kernel[:, :, ic,

    return feature_maps
```

空间复杂度

空间复杂度（访存量），严格来讲包括两部分：总参数量 + 各层输出特征图。

- **参数量**：模型所有带参数的层的权重参数总量（即**模型体积**，下式第一个求和表达式）
- **特征图**：模型在实时运行过程中每层所计算出的输出特征图大小（下式第二个求和表达式）

$$\textbf{Space} \sim O\left(\sum_{l=1}^{D} K_l^2 \cdot C_{l-1} \cdot C_l \sum_{l=1}^{D} M^2 \cdot C_l\right)$$

- 总参数量只与卷积核的尺寸 K、通道数 C、层数 D 相关，而与**输入数据的大小无关。**
- 输出特征图的空间占用比较容易，就是其空间尺寸 M^2 和通道数 C 的连乘。
- 注：实际上有些层（例如 ReLU）其实是可以通过原位运算完成的，此时就不用统计输出特征图这一项了。

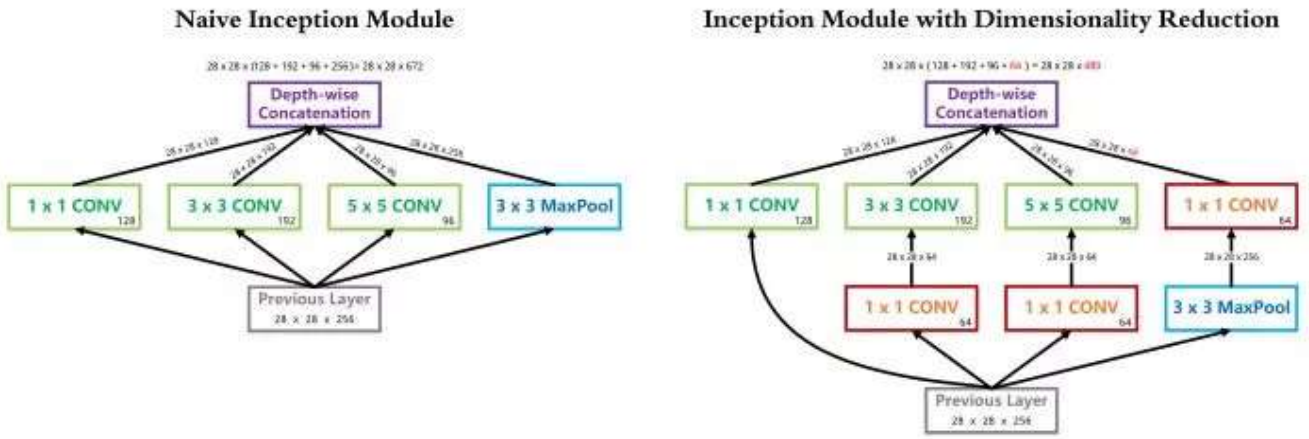
复杂度对模型的影响

- 时间复杂度决定了模型的训练/预测时间。如果复杂度过高，则会导致模型训练和预测耗费大量时间，既无法快速的验证想法和改善模型，也无法做到快速的预测。
- 空间复杂度决定了模型的参数数量。由于维度诅咒的限制，模型的参数越多，训练模型所需的数据量就越大，而现实生活中的数据集通常不会太大，这会导致模型的训练更容易过拟合。
- 当我们需要裁剪模型时，由于卷积核的空间尺寸通常已经很小（3x3），而网络的深度又与模型的表征能力紧密相关，不宜过多削减，因此模型裁剪通常最先下手的地方就是通道数。

Inception 系列模型是如何优化复杂度的

通过五个小例子说明模型的演进过程中是如何优化复杂度的。

4.1 InceptionV1 中的 1×1 卷积降维同时优化时间复杂度和空间复杂度



（图像被压缩的惨不忍睹...）

- InceptionV1 借鉴了 Network in Network 的思想，在一个 Inception Module 中构造了四个并行的不同尺寸的卷积/池化模块（上图左），有效的提升了网络的宽度。但是这么做也造成了网络的时间和空间复杂度的激增。对策就是添加 1 x 1 卷积（上图右红色模块）将输入通道数先降到一个较低的值，再进行真正的卷积。
- 以 InceptionV1 论文中的 (3b) 模块为例（可以点击上图看超级精美的大图），输入尺寸为 $28 \times 28 \times 256$，1×1 卷积核 个，3×3 卷积核 个，5×5 卷积核 个，卷积核一律采用 Same Padding 确保输出不改变尺寸。 **128 192 96**
- 在 3×3 卷积分支上加入 64 个 1×1 卷积前后的时间复杂度对比如下式：

$$\textbf{Before} \quad 28^2 \cdot 3^2 \cdot 256 \cdot 192 = 3.5 \times 10^8$$

$$\textbf{After} \quad 28^2 \cdot 1^2 \cdot 256 \cdot 64 + 28^2 \cdot 3^2 \cdot 64 \cdot 192 = 1 \times 10^8$$

- 同理，在 5×5 卷积分支上加入 64 个 1×1 卷积前后的时间复杂度对比如下式：

$$\textbf{Before} \quad 28^2 \cdot 5^2 \cdot 256 \cdot 96 = 4.8 \times 10^8$$

$$\textbf{After} \quad 28^2 \cdot 1^2 \cdot 256 \cdot 64 + 28^2 \cdot 5^2 \cdot 64 \cdot 96 = 1.3 \times 10^8$$

- 可见，使用 1×1 卷积降维可以降低时间复杂度3倍以上。该层完整的运算量可以在论文中查到，为 300 M，即 3×10^8。
- 另一方面，我们同样可以简单分析一下这一层参数量在使用 1 x 1 卷积前后的变化。可以看到，由于 1 x 1 卷积的添加，3 x 3 和 5 x 5 卷积核的参数量得以降低 4 倍，因此本层的参数量从 1000 K 降低到 300 K 左右。

$$\textbf{Before}\begin{cases}1^2\cdot 256\cdot 128\\ 3^2\cdot 256\cdot 192\\ 5^2\cdot 256\cdot 96\end{cases}\Longrightarrow\quad\textbf{After}\begin{cases}1^2\cdot 256\cdot 128\\ 1^2\cdot 256\cdot 64\\ 3^2\cdot 64\cdot 192\\ 1^2\cdot 256\cdot 64\\ 5^2\cdot 64\cdot 96\\ 1^2\cdot 256\cdot 64\end{cases}$$

4.2 InceptionV1 中使用 GAP 代替 $Flatten$

- 全连接层可以视为一种特殊的卷积层，其卷积核尺寸 与输入矩阵尺寸 一模一样。每个卷积核的输出特征图是一个标量点，即 。复杂度分析如下： K

$$\textbf{Time}\sim O(1^2\cdot X^2\cdot C_{in}\cdot C_{out})$$

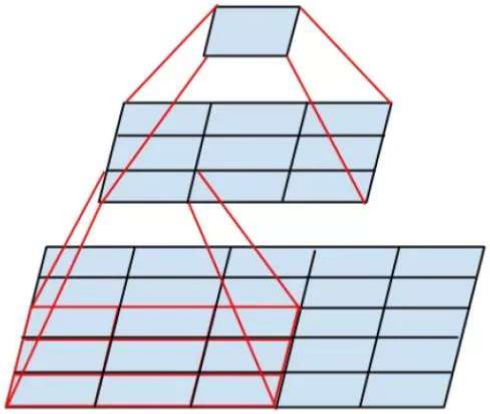
$$\textbf{Space}\sim O(X^2\cdot C_{in}\cdot C_{out})$$

- 可见，与真正的卷积层不同，**全连接层的空间复杂度与输入数据的尺寸密切相关。**因此如果输入图像尺寸越大，模型的体积也就会越大，这显然是不可接受的。例如早期的VGG系列模型，其 90% 的参数都耗费在全连接层上。
- InceptionV1 中使用的**全局平均池化 GAP** 改善了这个问题。由于每个卷积核输出的特征图在经过全局平均池化后都会直接精炼成一个标量点，因此全连接层的复杂度不再与输入图像尺寸有关，运算量和参数数量都得以大规模削减。复杂度分析如下：

$$\textbf{Time}\sim O(C_{in}\cdot C_{out})$$

$$\textbf{Space}\sim O(C_{in}\cdot C_{out})$$

4.3 InceptionV2 中使用两个 3×3 卷积级联替代 5×5 卷积分支



使用两个 3x3 卷积级联代替单个 5x5 卷积

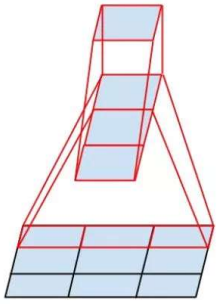
感受野不变

- 根据上面提到的二维卷积输入输出尺寸关系公式，可知：对于同一个输入尺寸，单个 5×5 卷积的输出与两个 3×3 卷积级联输出的尺寸完全一样，即感受野相同。
- 同样根据上面提到的复杂度分析公式，可知：这种替换能够非常有效的降低时间和空间复杂度。我们可以把辛辛苦苦省出来的这些复杂度用来提升模型的深度和宽度，使得我们的模型能够在复杂度不变的前提下，具有更大的容量，爽爽的。
- 同样以 InceptionV1 里的 (3b) 模块为例，替换前后的 5×5 卷积分支复杂度如下：

$$\textbf{Before} \quad 28^2 \cdot 1^2 \cdot 256 \cdot 64 + 28^2 \cdot 5^2 \cdot 64 \cdot 96 = 1.3 \times 10^8$$

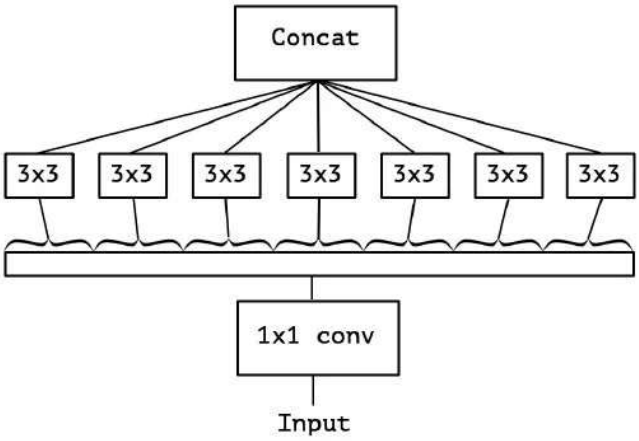
$$\textbf{After} \quad 28^2 \cdot 1^2 \cdot 256 \cdot 64 + 28^2 \cdot 3^2 \cdot 64 \cdot 96 \cdot 2 = 1.0 \times 10^8$$

4.4 InceptionV3 中使用 与 $N \times 1$ $1 \times N$ 卷积级联替代 $N \times N$ 卷积



- InceptionV3 中提出了卷积的 Factorization，在确保感受野不变的前提下进一步简化。
- 复杂度的改善同理可得，不再赘述。

4.5 Xception 中使用 Depth-wise Separable Convolution



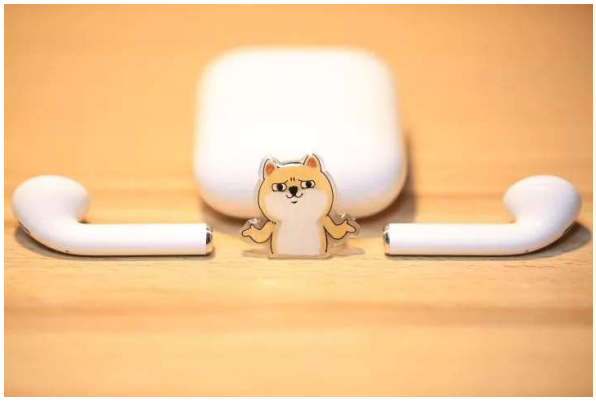
- 我们之前讨论的都是标准卷积运算，每个卷积核都对输入的所有通道进行卷积。
- Xception 模型挑战了这个思维定势，它让每个卷积核只负责输入的某一个通道，这就是所谓的 Depth-wise Separable Convolution。
- 从输入通道的视角看，标准卷积中每个输入通道都会被所有卷积核蹂躏一遍，而 Xception 中每个输入通道只会被对应的一个卷积核扫描，降低了模型的冗余度。
- 标准卷积与可分离卷积的时间复杂度对比：可以看到本质上是把连乘转化成为相加。

$$\textbf{Standard Convolution} \quad Time \sim O\left(M^2 \cdot K^2 \cdot C_{in} \cdot C_{out}\right)$$

$$\textbf{Depth-wise Separable Convolution} \quad Time \sim O\left(M^2 \cdot K^2 \cdot C_{in} + M^2 \cdot C_{in} \cdot C_{out}\right)$$

总结

通过上面的推导和经典模型的案例分析，我们可以清楚的看到其实很多创新点都是围绕模型复杂度的优化展开的，其基本逻辑就是乘变加。模型的优化换来了更少的运算次数和更少的参数数量，一方面促使我们能够构建更轻更快的模型（例如MobileNet），一方面促使我们能够构建更深更宽的网络（例如Xception），提升模型的容量，打败各种大怪兽，欧耶~



参考论文

- https://arxiv.org/abs/1412.1710
- https://arxiv.org/abs/1409.4842
- https://arxiv.org/abs/1502.03167
- https://arxiv.org/abs/1512.00567
- https://arxiv.org/abs/1610.02357

> 注：本文主要关注的是针对模型本身的复杂度分析。如果想要进一步评估模型在计算平台上的理论计算性能，则需要了解 Roofline Model 的相关理论，欢迎阅读本文的进阶版： Roofline Model与深度学习模型的性能分析。（文章链接：https://zhuanlan.zhihu.com/p/34204282 ）

【元旦手工】太赞了！一把剪刀一张纸，给你变出最美的新年剪纸创意！

幼儿园手工