MobileNetv3

2018

Searching for MobileNetV3

论文的出发点和背景

本文的目标是开发出尽可能最好的移动计算机视觉架构，以优化移动设备上的精度延迟权衡

本文开始探索自动化搜索算法和网络设计如何协同工作，利用互补的方法来提高整体水平。通过这个过程，我们创建了两个新的发布的MobileNet模型:MobileNetV3-Large和MobileNetV3-Small

SqueezeNet:广泛使用1x1卷积与压缩和扩展模块，主要专注于减少参数的数量。

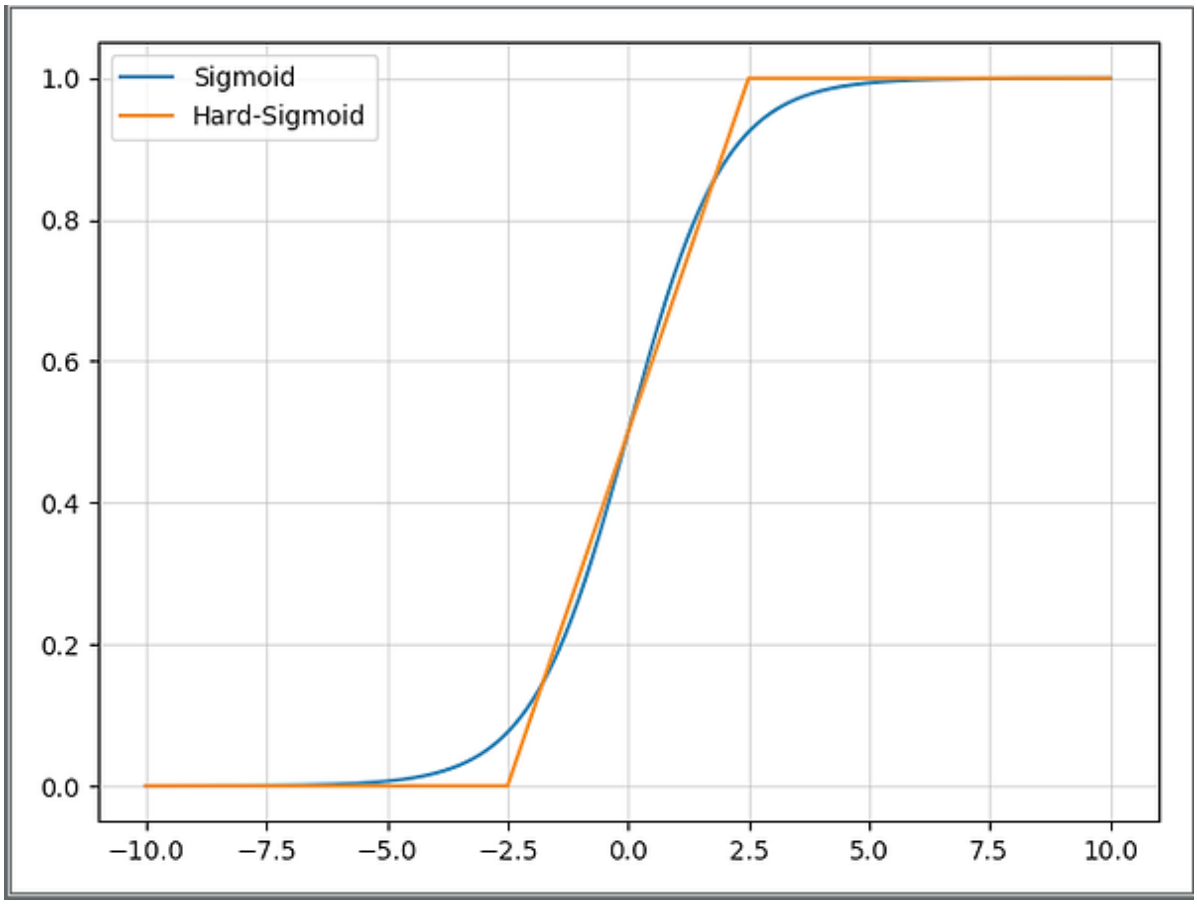
MobileNetV1:采用深度可分离卷积来大大提高计算效率。

MobileNetV2:引入反向残差&线性瓶颈

ShiftNet:提出了与点向卷积交错的移位操作来取代昂贵的空间卷积。

MnasNet 建立在MobileNetV2结构之上，通过在瓶颈结构中引入基于挤压和激励的轻量级注意模块。

对于MobileNetV3，我们使用这些层的组合作为构建块，以便构建最有效的模型。层也升级与修改的swish非线性。挤压和激励以及swish非线性都使用了sigmoid，它的计算效率很低，而且很难在定点算法中保持精度，因此我们将其替换为硬sigmoid



创新思路

本文的目标是开发最佳的移动计算机视觉架构，以优化移动设备上的精确延迟交换。为了实现这一点，我们引入了(1)互补搜索技术，(2)适用于移动设备的非线性函数的新高效版本，(3)新的高效网络设计，(4)一个新的高效分割解码器。

论文方法

NetAdapt

1. Starts with a seed network architecture found by platform-aware NAS.
2. For each step:
   (a) Generate a set of new *proposals*. Each proposal represents a modification of an architecture that generates at least δ reduction in latency compared to the previous step.
   (b) For each proposal we use the pre-trained model from the previous step and populate the new proposed architecture, truncating and randomly initializing missing weights as appropriate. Fine-tune each proposal for T steps to get a coarse estimate of the accuracy.
   (c) Selected best proposal according to some metric.
3. Iterate previous step until target latency is reached.

对于MobileNetV3，我们使用平台感知的NAS，通过优化每个网络块来搜索全局网络结构。然后，我们使用NetAdapt算法来搜索每一层的过滤器的数量。

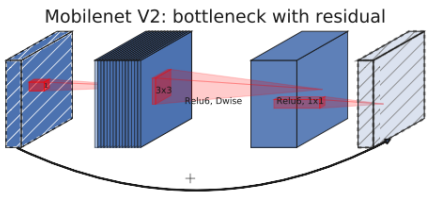


Figure 3. MobileNetV2 [39] layer (Inverted Residual and Linear Bottleneck). Each block consists of narrow input and output (bottleneck), which don't have nonlinearity, followed by expansion to a much higher-dimensional space and projection to the output. The residual connects bottleneck (rather than expansion).

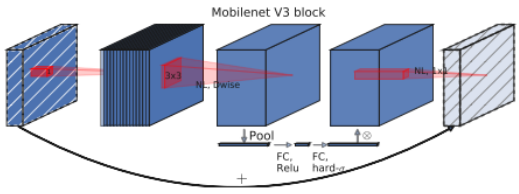


Figure 4. MobileNetV2 + Squeeze-and-Excite [20]. In contrast with [20] we apply the squeeze and excite in the residual layer. We use different nonlinearity depending on the layer, see section 5.2 for details.

引入了一种新的非线性，h-swish，它是最近的swish非线性的改进版本，它的计算速度更快，也更有利于量化。

一旦通过架构搜索找到模型，我们就会发现，**一些最后的层以及一些较早的层比其他层更昂贵**。我们建议对体系结构进行一些修改，**以减少这些慢层的延迟**，同时保持准确性。这些修改超出了当前搜索空间的范围。

第一个修改将重新处理网络的最后几层是如何交互的，以便生成最终层功能更有效率。**目前的模型基于MobileNetV2的倒瓶颈结构和变体，使用1x1卷积作为最后一层，以扩展到高维特征空间。这一层非常重要，因为它具有丰富的预测功能。**然而，这是以**额外的延迟**为代价的。

为了减少延迟并保留高维特性，我们将该层移到最终的平均池之后。最后一组特征现在计算为1x1空间分辨率，而不是7x7空间分辨率。这种设计选择的结果是，在计算和延迟方面，特性的计算变得几乎是免费的。

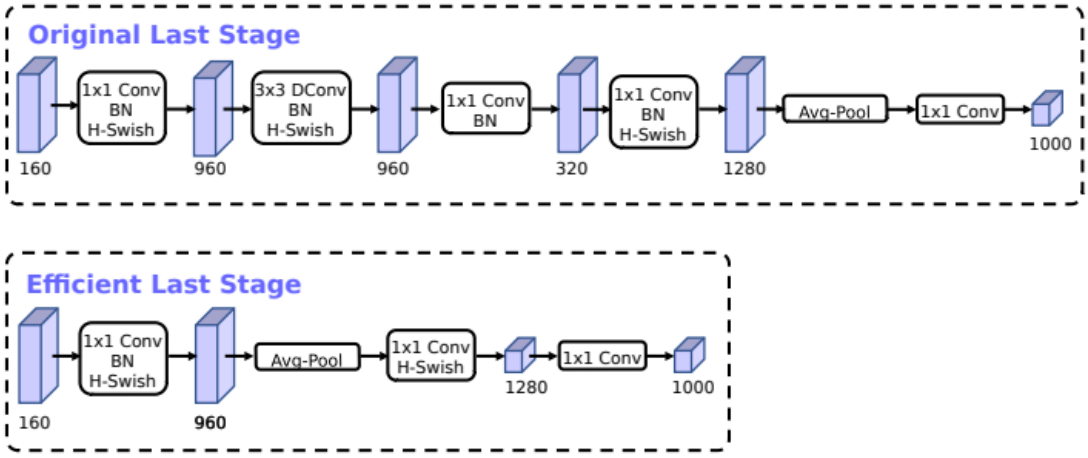


Figure 5. Comparison of original last stage and efficient last stage. This more efficient last stage is able to drop three expensive layers at the end of the network at no loss of accuracy.

我们能够将卷积核的数量减少到16个，同时保持与使用ReLU或swish的32个卷积核相同的精度。这节省了额外的3毫秒和1000万 MAdds。

虽然swish这种非线性激活函数提高了精度，但在嵌入式环境中，它的成本是非零的，因为在移动设备上计算sigmoid函数要昂贵得多。

我们将sigmoid替换为其分段线性硬模拟之后得到swish的hard版本

$$\text{h-swish}[x] = x\frac{\text{ReLU6}(x+3)}{6}$$

在我们的实验中，用量化模式下的swish替换h-swish使推理延迟增加了15%。

在本文中，我们提出了另一种轻量级分割头，称为Lite R-ASPP(或LR-ASPP)。Lite R-ASPP是对R-ASPP的改进，它部署全局平均池的方式类似于挤压-激励模块

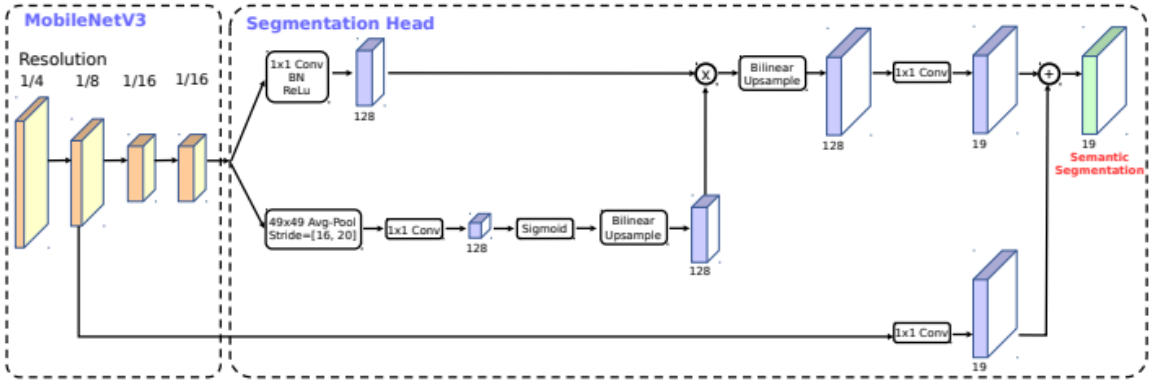
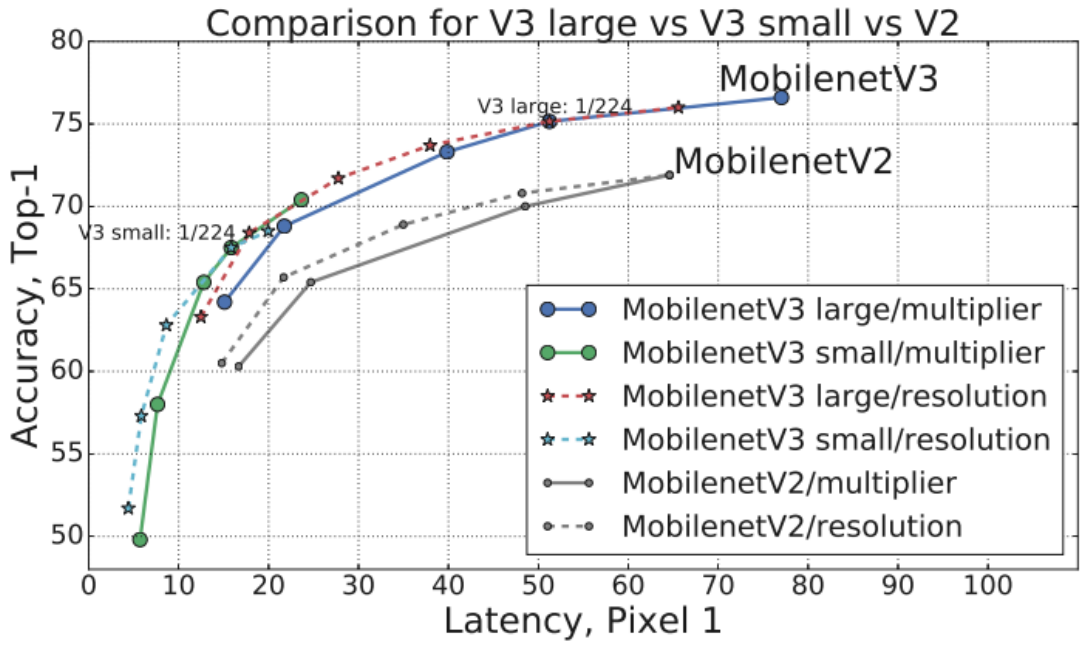


Figure 10. Building on MobileNetV3, the proposed segmentation head, Lite R-ASPP, delivers fast semantic segmentation results while mixing features from multiple resolutions.

实际效果



Network	Top-1	P-1	P-2	P-3
V3-Large 1.0	**73.8**	44	42.5	31.7
V2 1.0	70.9	52	48.3	37.0
V3-Small	**64.9**	15.5	14.9	10.7
V2 0.35	57.2	16.7	15.6	11.9

Table 4. Quantized performance. All latencies are in ms. The inference latency is measured using a single large core on the respective Pixel 1/2/3 device.

Network	Top-1	MAdds	Params	P-1	P-2	P-3
V3-Large 1.0	**75.2**	**219**	5.4M	**51**	**61**	**44**
V3-Large 0.75	73.3	155	4.0M	39	46	40
MnasNet-A1	75.2	315	3.9M	71	86	61
Proxyless[5]	74.6	320	4.0M	72	84	60
V2 1.0	72.0	300	3.4M	64	76	56
V3-Small 1.0	67.4	66	2.9M	15.8	19.4	14.4
V3-Small 0.75	65.4	44	2.4M	12.8	15.6	11.7
Mnas-small [43]	64.9	65.1	1.9M	20.3	24.2	17.2
V2 0.35	60.8	59.2	1.6M	16.6	19.6	13.9

Table 3. Floating point performance on the Pixel family of phones (P-n denotes a Pixel-n phone). All latencies are in ms and are measured using a single large core with a batch size of one. Top-1 accuracy is on ImageNet.

Backbone	mAP	Latency (ms)	Params (M)	MAdds (B)
V1	22.2	228	5.1	1.3
V2	22.1	162	4.3	0.80
MnasNet	23.0	174	4.88	0.84
V3	22.0	137	4.97	0.62
V3†	22.0	119	3.22	0.51
V2 0.35	13.7	66	0.93	0.16
V2 0.5	16.6	79	1.54	0.27
MnasNet 0.35	15.6	68	1.02	0.18
MnasNet 0.5	18.5	85	1.68	0.29
V3-Small	16.0	52	2.49	0.21
V3-Small†	16.1	43	1.77	0.16

Table 6. Object detection results of SSDLite with different backbones on COCO test set. †: Channels in the blocks between $C4$ and $C5$ are reduced by a factor of 2.

个人理解

该篇文章主要做了三个方面的工作：

1.更新block

> 加入和SE模块，同时使用了hard-σ函数，由于sigmoid的计算耗时且复杂，所以swish函数虽然效果不错，但是还有改进的余地，最终得到了h-swish函数

2.NAS

> 一直觉得这是有钱人才能做的实验。。

3.重新设计耗时层

> 减少第一层的通道数，从32变为16
>
> 简化last stage

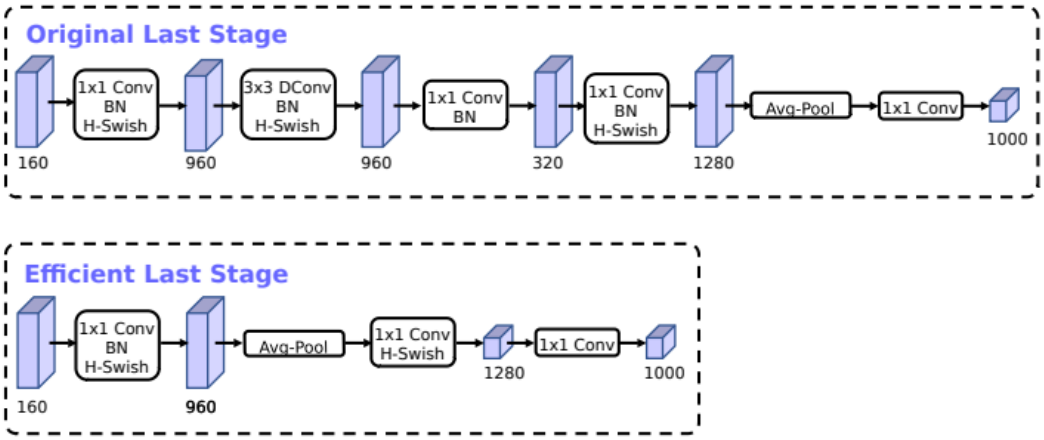


Figure 5. Comparison of original last stage and efficient last stage. This more efficient last stage is able to drop three expensive layers at the end of the network at no loss of accuracy.