MobileNets: Efficient Convolutional Neural Networks for Mobile Vision

Abstract

We present a class of efficient models called MobileNets for mobile and embedded vision applications. MobileNets are based on a streamlined architecture that uses **depth-wise separable convolutions** to build light weight deep neural networks. We introduce two simple global hyper-
parameters that efficiently **trade off** between **latency** and accuracy. These hyper-parameters allow the model builder to choose the right sized model for their application based on the constraints of the problem. We present extensive experiments on resource and accuracy **tradeoffs** and show strong performance compared to other popular models on ImageNet classification. We then **demonstrate** the effectiveness of MobileNets across a wide range of applications and use cases including object detection, finegrain classification, face attributes and large scale geo-localization.

我们提出了一类高效的模型，称为MobileNets的高效模型，用于移动和嵌入式视觉应用。移动网络是基于一个精简的架构，使用深度聪明的可分离卷积来建立轻量级的深度神经网络。我们引入了两个简单的全局超参数，有效地在延迟和准确性之间进行权衡。这些超参数允许模型构建者根据问题的约束条件，为他们的应用选择合适规模的模型。我们提出了广泛的资源和精度的权衡实验，并显示了与其他流行的模型相比在ImageNet分类的强大性能。然后，我们展示了移动网络在广泛的应用中的有效性。在广泛的应用和用例中证明了MobileNets的有效性。包括物体检测、细粒度分类脸部属性和大规模的地理定位。

1. Introduction

Convolutional neural networks have become ubiquitous in computer vision ever since AlexNet [19] popularized deep convolutional neural networks by winning the ImageNet Challenge: ILSVRC 2012 [24]. The general trend has been to make deeper and more complicated networks in order to achieve higher accuracy [27, 31, 29, 8]. However, these advances to improve accuracy are not necessarily making networks more efficient with respect to size and
speed. In many real world applications such as robotics, self-driving car and augmented reality, the recognition tasks need to be carried out in a timely fashion on a computationally limited platform. This paper describes an efficient network architecture and a set of two hyper-parameters in order to build very small, low latency models that can be easily matched to the design requirements for mobile and embedded vision applications. Section 2 reviews prior work in building small models. Section 3 describes the MobileNet architecture and two hyper-parameters width multiplier and resolution multiplier to define smaller and more efficient MobileNets. Section 4 describes experiments on ImageNet as well a variety of different applications and use cases. Section 5 closes with a summary and conclusion.

卷积神经网络在计算机视觉中已经变得无处不在了。自从AlexNet[19]通过赢得Ima-school的比赛而普及了深度卷积神经网络以来，卷积神经网络在计算机视觉中已经无处不在。深度卷积神经网络，赢得了Ima-geNet Challenge: ILSVRC 2012 [24]。总的趋势是制造更深、更复杂的网络以达到更高的精度[27, 31, 29, 8]。然而
这些提高精度的进步并不意味着网络在尺寸和速度方面更有效率。在许多现实世界的应用中，如机器人技术、自动驾驶汽车和增强功能，识别任务需要在一个计算能力有限的平台上及时执行。本文描述了一个有效的网络结构和一组两个超参数，以建立非常小的、低延迟的模型。可以很容易地与移动和嵌入式视觉应用的设计要求相匹配。应用的设计要求。第2节回顾了先前在建立小型模型方面的工作。第3节描

述了MobileNet的结构和两个超参数宽度乘法器和分辨率乘法器来定义更小更有效的模型。第4节第4节描述了在ImageNet上的实验以及各种不同的应用和用例。第5节结束的摘要和结论。

2. Prior Work

There has been rising interest in building small and efficient neural networks in the recent literature, e.g. [16, 34,12, 36, 22]. Many different approaches can be generally **categorized** into either **compressing** pretrained networks or training small networks directly. This paper proposes a class of network architectures that allows a model developer to specifically choose a small network that matches the **resource restrictions** (latency, size) for their application. MobileNets primarily focus on **optimizing for latency** but also **yield small networks**. Many papers on small networks focus only on size but do not consider speed. MobileNets are built primarily from **depthwise separable convolutions initially introduced in [26] and subsequently used in Inception models [13] to reduce the computation in the first few layers**. Flattened networks [16] build a network out of **fully factorized** convolutions and showed the potential of extremely **factorized networks. Independent of this current paper,** Factorized Networks[34] introduces a similar factorized convolution as well as the use of **topological connections. Subsequently**, the Xception network [3] demonstrated how to scale up depthwise separable filters to out perform Inception V3 networks. Another small network is Squeezenet [12] which uses a bottleneck approach to design a very small network. Other reduced computation networks include structured transform networks [28] and deep fried convnets [37].A different approach for obtaining small networks is **shrinking, factorizing or compressing** pretrained networks. Compression based on product **quantization [36], hashing [2], and pruning, vector quantization and Huffman coding** [5] have been proposed in the literature. Additionally various factorizations have been proposed to speed up pre-trained networks [14, 20]. Another method for training small networks is **distillation** [9] which uses a larger network to teach a smaller network. It is **complementary** to our approach and is covered in some of our use cases in section 4. Another emerging approach is low bit networks[4, 22, 11].

在最近的文献中，人们对建立小型和高效的神经网络的兴趣不断增加，例如[16, 34,12, 36, 22]. 许多不同的方法一般可以分为可分为压缩预训练的网络或直接训练小型网络。本文提出了一种网络架构，允许模型开发者专门选择符合其资源限制（延迟、大小）的小型网络。移动网络主要集中在对延迟的优化上，但是也产生小型网络。许多关于小型网络的论文只关注大小而不考虑速度。移动网络主要由深度可分离的最初在[26]中引入，随后在Inception模型[13]中使用，以减少前几层的计算。扁平化网络[16]构建了一个由完全因数化的卷积组成的网络，并显示了极度因子化网络的潜力。与本文无关目前的论文，因子化网络[34]引入了一个类似的
因子化卷积，以及使用拓扑连接。随后，Xception网络[3]证明了阐述了如何扩大深度可分离滤波器的规模，以超越表现Inception V3网络。另一个小型网络是
Squeezenet[12]，它使用了一个瓶颈方法来设计一个非常小的网络。一个非常小的网络。其他减少计算的网络包括结构化的变换网络[28]和深炸的convnets[37]。
获得小型网络的一个不同的方法是缩减、分解或压缩预训练的网络。基于乘积量化的压缩[36]、散列[2]以及剪枝、矢量量化和Huffman编码的压缩[5]已经在文献中被提出。此外，还提出了不同的此外，还提出了各种因式分解来加快预训练的网络[14, 20]。另一种用于训练训练小型网络的另一种方法是蒸馏法[9]，它使用一个较大的网工作来训练一个较小的网络。它是对我们方法的补充，并且在我们的第四节中的一些用例也有涉及。另一种新兴的方法是低比特网络[4, 22, 11].

3. MobileNet Architecture

In this section we first describe the core layers that MobileNet is built on which are depthwise separable filters. We then describe the MobileNet network structure and conclude with descriptions of the two model shrinking hyper-parameters width multiplier and resolution multiplier.

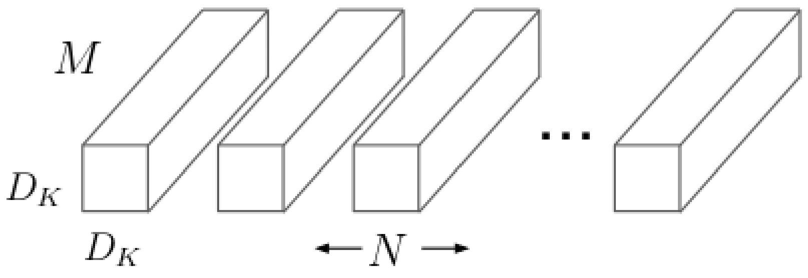
在这一节中，我们首先描述MobileNet的核心层。MobileNet是建立在深度可分离的过滤器之上的。然后，我们描述MobileNet的网络结构，并在此基础上描述两个模型收缩的超参数。最后描述了两个模型收缩的超参数宽度倍增器和分辨率倍增器。

3.1. Depthwise Separable Convolution

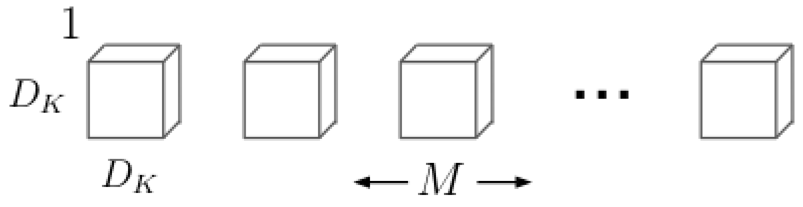
The MobileNet model is based on depthwise separable convolutions which is a form of factorized convolutions which **factorize a standard convolution into a depthwise convolution and a 1 × 1 convolution called a pointwise convolution. For MobileNets the depthwise convolution applies a single filter to each input channel. The pointwise convolution then applies a 1 × 1 convolution to combine the outputs the depthwise convolution. A standard convolution both filters and combines inputs into a new set of outputs in one step.** The depthwise separable convolution splits this into two layers, **a separate layer for filtering and a separate layer for combining**. This factorization has the effect of **drastically** reducing computation and model size. Figure 2 shows how a standard convolution 2(a) is factorized into a depthwise convolution 2(b) and a 1 × 1 pointwise convolu-tion 2(c). A standard convolutional layer takes as input a D F ×D F × M feature map F and produces a D F × D F × N feature map G where D F is the spatial width and height of a square input feature map 1 , M is the number of input channels (input depth), D G is the spatial width and height of a square output feature map and N is the number of output channel (output depth).

3.1. 深度可分离卷积

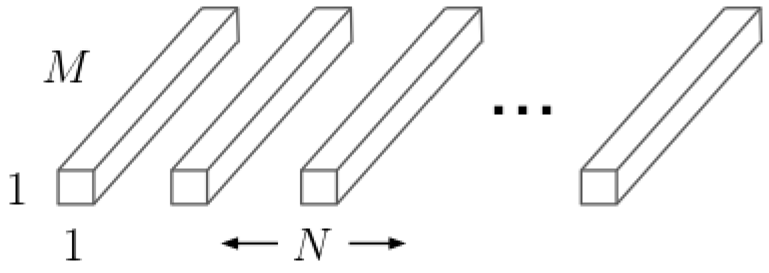
MobileNet模型是基于深度可分离卷积，这是一种因数化卷积的形式，**它将标准卷积分解为深度卷积和1×1卷积，称为点卷积。对于MobileNets来说，深度卷积对每个输入通道应用一个过滤器。 然后，逐点卷积应用1×1卷积来结合深度卷积的输出。标准卷积在一个步骤中既过滤又将输入合并为一组新的输出**。深度可分离卷积将其分成两层，**一个单独的过滤层和一个单独的组合层**。这种因子化的效果是地减少了计算和模型的大小。图2显示了标准卷积2(a)如何被分解成深度卷积2(b)和1×1点卷积2(c)。标准卷积层将D F×D F×M特征图F作为输入，并产生D F×D F×N特征图G，其中D F是正方形输入特征图1的空间宽度和高度。的空间宽度和高度，M是输入通道的数量（输入深度），D G是方形输出特征图的空间宽度和高度，N是输出通道的数量（输出深度）。



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 2. The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.

The standard convolutional layer is parameterized by convolution kernel K of size D K ×D K ×M ×N where D K is the spatial dimension of the kernel assumed to be square and M is number of input channels and N is the number of output channels as defined previously The output feature map for standard convolution assuming stride one and padding is computed as:

标准卷积层的参数是大小为D K×D K×M×N的卷积核，其中D K是假定为正方形的核的空间维度，M是输入通道的数量，N是前面定义的输出通道的数量，标准卷积的输出特征图假定为跨度1和填充，计算如下。

$$\mathbf{G}_{k,l,n} = \sum_{i,j,m} \mathbf{K}_{i,j,m,n} \cdot \mathbf{F}_{k+i-1,l+j-1,m} \qquad (1)$$

Standard convolutions have the computational cost of:

标准卷积的计算成本为：

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \quad (2)$$

where the computational cost depends multiplicatively on the number of input channels M , the number of output channels N the kernel size D k × D k and the feature map size D F × D F . MobileNet models address each of these terms and their interactions. **First it uses depthwise separable convolutions to break the interaction between the number of output channels and the size of the kernel.** The standard convolution operation has the effect of filtering features based on the convolutional kernels and combining features in order to produce a new representation. The filtering and combination steps can be split into two steps via the use of factorized convolutions called depthwise separable convolutions for substantial reduction in computational cost. Depthwise separable convolution are made up of two layers: depthwise convolutions and pointwise convolutions. We use depthwise convolutions to apply a single filter per each input channel (input depth). Pointwise convolution, a simple 1×1 convolution, is then used to create a linear combination of the output of the depthwise layer. MobileNets use both batchnorm and ReLU nonlinearities for both layers. Depthwise convolution with one filter per input channel (input depth) can be written as:

其中，计算成本取决于输入通道的数量M，输出通道的数量N，内核大小D k×D k和特征图大小D F×D F的乘法。MobileNet模型解决了这些条款中的每一项及其相互作用。首先，**它使用深度可分离卷积来打破输出通道数量和内核大小之间的相互作用**。标准的卷积操作具有基于卷积核过滤特征和组合特征的效果，以便产生一个新的表示。过滤和组合步骤可以通过使用被称为深度可分离卷积的因子化卷积分成两个步骤，从而大大降低计算成本。深度可分卷积是由两层组成的：深度卷积和点卷积。我们使用纵深卷积来为每个输入通道（输入深度）应用一个过滤器。点卷积，一个简单的1×1卷积，然后被用来创建深度层输出的线性组合。MobileNets对两层都使用 batchnorm 和 ReLU 非线性。每个输入通道（输入深度）有一个滤波器的纵深卷积可以写成：。

$$\hat{\mathbf{G}}_{k,l,m} = \sum_{i,j} \hat{\mathbf{K}}_{i,j,m} \cdot \mathbf{F}_{k+i-1,l+j-1,m} \quad (3)$$

where K̂ is the depthwise convolutional kernel of size D K × D K × M where the m th filter in K̂ is applied to the m th channel in F to produce the m th channel of the filtered output feature map Ĝ. Depthwise convolution has a computational cost of:

其中K̂是深度卷积核，大小为D K×D K×M，K̂中的第m个滤波器被应用于F中的第m个通道，以产生过滤后的输出特征图Ĝ的第m个通道。深度卷积的计算成本为。

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$$

Depthwise convolution is extremely efficient relative to standard convolution. However it only filters input channels, it does not combine them to create new features. So an additional layer that computes a linear combination of the output of depthwise convolution via 1 × 1 convolution is needed in order to generate these new features. The combination of depthwise convolution and 1 × 1(pointwise) convolution is called depthwise separable convolution which was originally introduced in [26]. Depthwise separable convolutions cost:

深度卷积相对于标准卷积来说是非常有效的。然而，它只是对输入通道进行过滤，并没有将它们结合起来以创造新的特征。所以需要一个额外的层，通过1×1卷积来计算深度卷积的输出的线性组合，以产生这些新的特征。深度卷积和1×1（pointwise）卷积的组合被称为深度可分离卷积，最初是在[26]中提出的。深度可分离卷积的成本。

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$$

which is the sum of the depthwise and 1 × 1 pointwise convolutions. By expressing convolution as a two step process of filtering and combining we get a reduction in computation of:

它是深度卷积和1×1点卷积的总和。通过将卷积表达为过滤和组合的两步过程，我们得到了计算量的减少。

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F}$$
$$= \frac{1}{N} + \frac{1}{D_K^2}$$

MobileNet uses 3 × 3 depthwise separable convolutions which uses between 8 to 9 times less computation than standard convolutions at only a small reduction in accuracy as seen in Section 4. Additional factorization in spatial dimension such as in[16, 31] does not save much additional computation as very little computation is spent in depthwise convolutions.

MobileNet使用3×3的深度可分离卷积，这比标准卷积的计算量少了8到9倍，但在第4节中看到，准确度只减少了一点。像[16,31]那样在空间维度上的额外因式分解并没有节省多少额外的计算，因为在深度卷积中花费的计算量非常少。

3.2. Network Structure and Training

The MobileNet structure is built on depthwise separable convolutions as mentioned in the previous section except for the first layer which is a full convolution. By defining the network in such simple terms we are able to **easily explore network topologies** to find a good network. The MobileNet architecture is defined in Table 1. **All layers are followed by a batchnorm [13] and ReLU nonlinearity with the exception of the final fully connected layer which has no nonlinearity and feeds into a softmax layer for classification.** Figure 3 **contrasts** a layer with **regular convolutions, batchnorm and ReLU nonlinearity** to the factorized layer with depthwise convolution, **1 × 1 pointwise convolution as well as batchnorm and ReLU after each convolutional layer. Down sampling is handled with strided convolution in the depthwise convolutions as well as in the first layer**. **A final average pooling reduces the spatial resolution to 1 before the fully connected layer. Counting depthwise and pointwise convolutions as separate layers, MobileNet has 28 layers.**

MobileNet的结构是建立在上一节提到的深度可分离卷积上的，除了第一层是全卷积。通过用如此简单的术语定义网络，我们能够很容易地探索网络拓扑结构，找到一个好的网络。MobileNet架构的定义见表1。所有的层后面都有一个 batchnorm[13]和ReLU非线性，只有最后的全连接层没有非线性，它被送入一个softmax层进行分类。图3对比了具有常规卷积、批判性规范和ReLU非线性的层与具有深度卷积、1×1点状卷积以及每个卷积层之后的批判性规范和ReLU的因子化层。在深度卷积和第一层中，向下采样是通过分层卷积处理的。在全连接层之前，最后的平均集合将空间分辨率降低到1。将深度卷积和点卷积算作独立的层，MobileNet有28层。

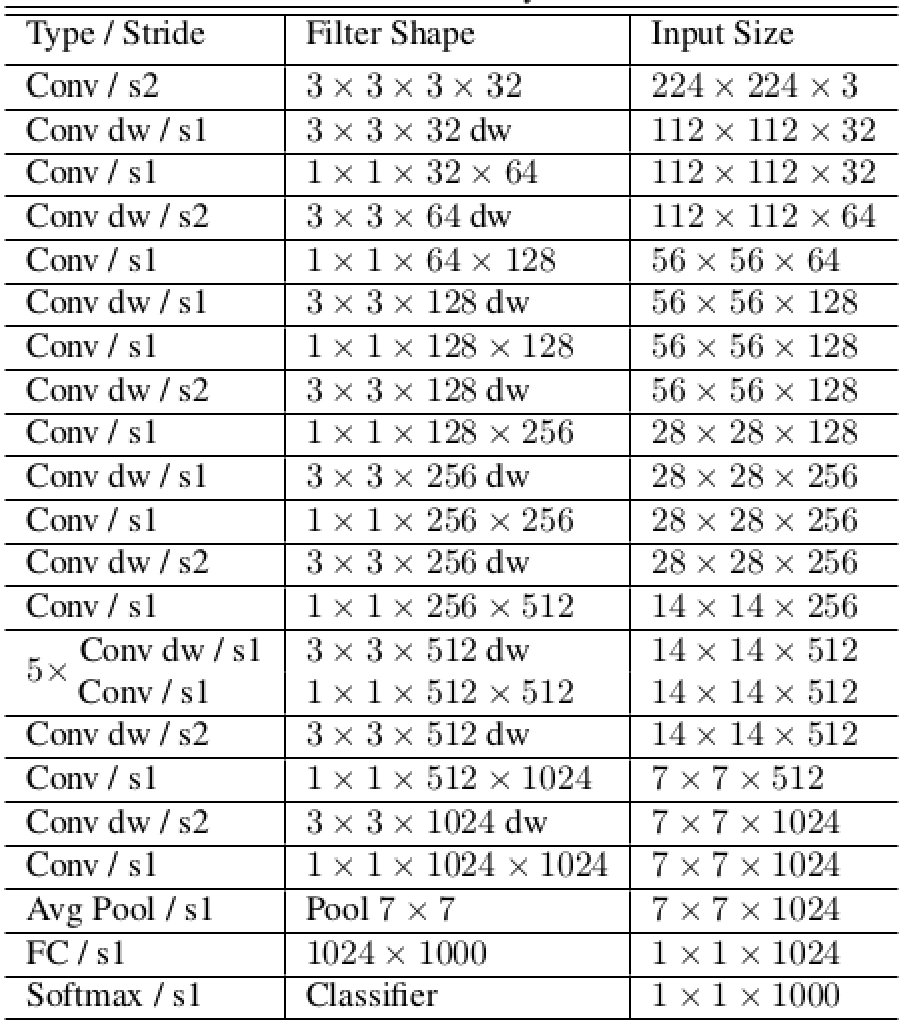


Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

```python
class MobileNetV1(nn.Module):
    def __init__(self):
        super(MobileNetV1, self).__init__()

        def conv_bn(inp, oup, stride):
            return nn.Sequential(
                nn.Conv2d(inp, oup, 3, stride, 1, bias=False),
                nn.BatchNorm2d(oup),
                nn.ReLU(inplace=True)
            )

        def conv_dw(inp, oup, stride):
            return nn.Sequential(
                nn.Conv2d(inp, inp, 3, stride, 1, groups=inp, bias=False),
                nn.BatchNorm2d(inp),
                nn.ReLU(inplace=True),

                nn.Conv2d(inp, oup, 1, 1, 0, bias=False),
                nn.BatchNorm2d(oup),
                nn.ReLU(inplace=True),
            )

        self.model = nn.Sequential(
            conv_bn(  3,  32, 2), #标准卷积，3x3(卷积核大小) x3(in) x32(out)
in:224x224x3 out:112x112x32
            conv_dw( 32,  64, 1), #深度可分离卷积，1. Conv2d(32, 32, 3, 1, 1,
groups=32) 2. Conv2d(32, 64, 1, 1, groups=0)
            conv_dw( 64, 128, 2),
            conv_dw(128, 128, 1),
            conv_dw(128, 256, 2),
            conv_dw(256, 256, 1),
            conv_dw(256, 512, 2),

            conv_dw(512, 512, 1),
            conv_dw(512, 512, 1),
            conv_dw(512, 512, 1), # in:14x14x256 out:14x14x512
            conv_dw(512, 512, 1),
            conv_dw(512, 512, 1),

            conv_dw(512, 1024, 2), # in:14x14x512 out: 7x7x1024
            conv_dw(1024, 1024, 1),
            nn.AvgPool2d(7),  # in: 7x7x1024 out: 1x1x1024
        )
        self.fc = nn.Linear(1024, 1000)  #in: 1x1x1024  out: 1x1x1000

    def forward(self, x):
        x = self.model(x)
        x = x.view(-1, 1024)
        x = self.fc(x)
        return x
```

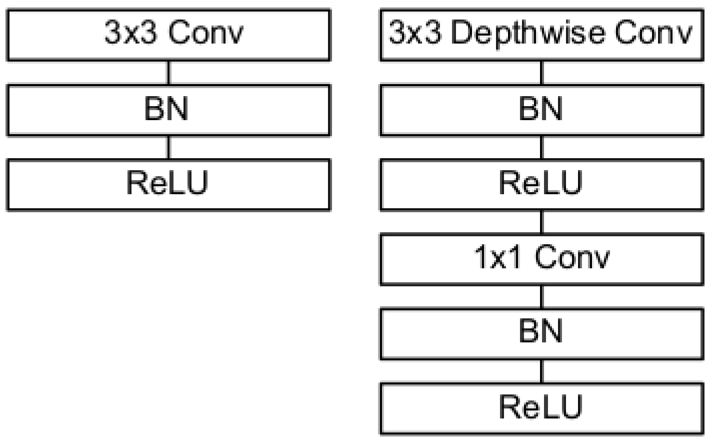


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

3.2. Network Structure and Training

The MobileNet structure is built on depthwise separable convolutions as mentioned in the previous section except for the first layer which is a full convolution. By defining the network in such simple terms we are able to easily explore **network topologies** to find a good network. The MobileNet
architecture is defined in Table 1. All layers are followed by a batchnorm [13] and ReLU nonlinearity with the exception of the final fully connected layer which has no nonlinearity and feeds into a softmax layer for classification. Figure 3 contrasts a layer with regular convolutions, batchnorm and
ReLU nonlinearity to the factorized layer with depthwise convolution, 1 × 1 pointwise convolution as well as batchnorm and ReLU after each convolutional layer. Down sampling is handled with strided convolution in the depthwise convolutions as well as in the first layer. A final average pooling reduces the spatial resolution to 1 before the fully connected layer. Counting depthwise and pointwise convolutions as separate layers, MobileNet has 28 layers.

MobileNet的结构是建立在上一节提到的深度可分离卷积上的，除了第一层是全卷积。通过用如此简单的术语定义网络，我们能够很容易地探索网络拓扑结构，找到一个好的网络。MobileNet架构的定义见表1。所有的层后面都有一个 batchnorm[13]和ReLU非线性，只有最后的全连接层没有非线性，它被送入一个softmax层进行分类。**图3对比了一个具有常规卷积、批量规范和ReLU非线性的层。ReLU非线性的层与具有深度卷积、1×1点卷积以及在每个卷积层之后的批判性规范和ReLU的因子化层进行对比。**在深度卷积和第一层中，向下采样是用分层卷积处理的。最后的平均在全连接层之前，最后的平均集合将空间分辨率降低到1。将深度卷积和点卷积算作独立的层，MobileNet有28层。

It is not enough to simply define networks **in terms of** a small number of Mult-Adds. It is also important to make sure these operations can be efficiently **implementable**. For instance **unstructured** sparse matrix operations are not typically faster than dense matrix operations until a very high level of sparsity. Our model structure puts nearly all of the computation into dense 1 × 1 convolutions. This can be implemented with highly optimized general matrix multiply (GEMM) functions. Often convolutions are implemented by a GEMM but require an initial reordering in memory called im2col in order to map it to a GEMM. For instance, this approach is used in the popular Caffe package [15]. 1 × 1 convolutions do not require this reordering in memory and can be implemented directly with GEMM which is one of the most optimized numerical linear algebra algorithms. MobileNet spends 95% of it's computation time in 1 × 1 convolutions which also has 75% of the parameters as can be seen in Table 2. Nearly all of the additional parameters are in the fully connected layer.

仅仅用少量的Mult-Add来定义网络是不够的。同样重要的是，要确保这些操作可以有效地实现。例如，在非常高的稀疏程度之前，非结构化的稀疏矩阵运算通常不会比密集矩阵运算快。我们的模型结构将几乎所有的计算都放在密集的1×1卷积中。这可以通过高度优化的通用矩阵乘法（GEMM）函数来实现。通常情况下，卷积是由GEMM实现的，但需要在内存中进行初始的重新排序称为im2col，以便将其映射到GEMM。例如，这种方法在流行的Caffe包中使用[15]。1×1的卷积不需要在内存中进行这种重新排序，可以直接用GEMM来实现，而GEMM是最优化的数值线性代数算法之一。MobileNet在1×1卷积中花费了95%的计算时间，从表2中可以看出，它也有75%的参数。几乎所有的额外参数都在全连接层中。

MobileNet models were trained in TensorFlow [1] using RMSprop [33] with asynchronous gradient descent similar to Inception V3 [31]. However,contrary to training large models we use less regularization and data augmentation techniques because small models have less trouble with overfitting. When training MobileNets we do not use side heads or label smoothing and additionally reduce the amount image of distortions by limiting the size of small
crops that are used in large Inception training [31]. Additionally, we found that it was important to put very little or no weight decay (l2 regularization) on the depthwise filters since their are so few parameters in them. For the ImageNet benchmarks in the next section all models were trained with same training parameters regardless of the size of the model.

MobileNet模型是在TensorFlow[1]中使用RMSprop[33]与Inception V3[31]类似的异步梯度下降法训练的。然而，与训练大型模型相反，我们使用较少的正则化和数据增强技术，因为小型模型在过拟合方面的问题较少。当训练MobileNets时，我们不使用侧头或标签平滑，并通过限制小作物的大小来减少图像的失真量。
在大型Inception训练中使用的作物[31]。此外，我们发现，由于深度过滤器的参数很少，所以在深度过滤器上投入很少或没有权重衰减（l2正则化）很重要。对于下一节中的ImageNet基准，无论模型的大小，所有的模型都用相同的训练参数进行训练。

3.3. Width Multiplier: Thinner Models

Although the base MobileNet architecture is already small and low latency, many times a specific use case or application may require the model to be smaller and faster. In order to construct these smaller and less computationally expensive models we introduce a very simple parameter α called width multiplier. The role of the width multiplier α is to thin a network uniformly at each layer. For a given layer and width multiplier α, the number of input channels M becomes αM and the number of output channels N becomes αN .The computational cost of a depthwise separable convolution with width multiplier α is: D K · D K · αM · D F · D F + αM · αN · D F · D F(6) where α ∈ (0, 1] with typical settings of 1, 0.75, 0.5 and 0.25. α = 1 is the baseline MobileNet and α < 1 are reduced MobileNets. Width multiplier has the effect of re- ducing computational cost and the number of parameters quadratically by roughly α 2 . Width multiplier can be ap- plied to any model structure to define a new smaller model with a reasonable accuracy, latency and size trade off. It is used to define a new reduced structure that needs to be trained from scratch.

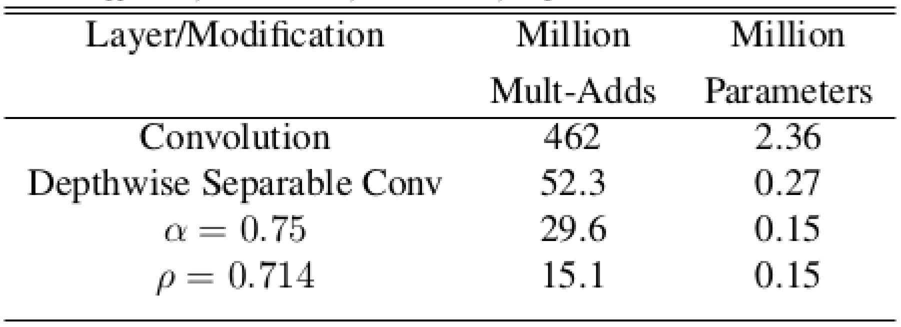
尽管基本的MobileNet架构已经是小而低延迟的，但很多时候，特定的用例或应用可能要求模型更小更快。为了构建这些更小的、计算成本更低的模型，我们引入了一个非常简单的参数α，称为宽度乘数。宽度乘数α的作用是在每一层均匀地稀释网络。对于一个给定的层和宽度乘数α，输入通道的数量M变成了αM，输出通道的数量N变成了αN.宽度乘数α的深度可分离卷积的计算成本是。D K - D K - αM - D F - D F + αM - αN - D F - D F(6) 其中α∈(0, 1]，典型设置为1、0.75、0.5和0.25。α=1是基线移动网，α<1是缩小的移动网。宽度倍增器的作用是重新减少计算成本和参数的数量，大约是α 2的四倍。宽度倍增器可以应用于任何模型结构，以定义一个新的更小的模型，具有合理的精度、延迟和大小的权衡。它被用来定义一个新的缩小的结构，需要从头开始训练。

3.4. Resolution Multiplier:

Reduced RepresentationThe second hyper-parameter to reduce the computational cost of a neural network is a resolution multiplier ρ. We ap-Table 3. Resource usage for modifications to standard convolution. Note that each row is a cumulative effect adding on top of the
previous row. This example is for an internal MobileNet layer ply this to the input image and the internal representation of every layer is subsequently reduced by the same multiplier.In practice we implicitly set ρ by setting the input resolution. We can now express the computational cost for the core
layers of our network as depthwise separable convolutions with width multiplier α and resolution multiplier ρ:

减少表示法第二个减少神经网络计算成本的超参数是分辨率乘数ρ。对标准卷积的修改的资源使用情况。注意，每一行都是在前一行的基础上增加的累积效应。上一行。这个例子是针对内部MobileNet层的，将其应用于输入图像，每一层的内部表示随后都被相同的乘数所减少。在实践中，我们通过设置输入分辨率隐含地设置ρ。我们现在可以将网络核心层的计算成本表示为我们现在可以将我们网络的核心层的计算成本表达为深度可分离的卷积，宽度乘数α和分辨率乘数ρ。

with $D_K = 3$, $M = 512$, $N = 512$, $D_F = 14$.



Layer/Modification	Million Mult-Adds	Million Parameters
Convolution	462	2.36
Depthwise Separable Conv	52.3	0.27
$\alpha = 0.75$	29.6	0.15
$\rho = 0.714$	15.1	0.15

$$D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F \quad (7)$$

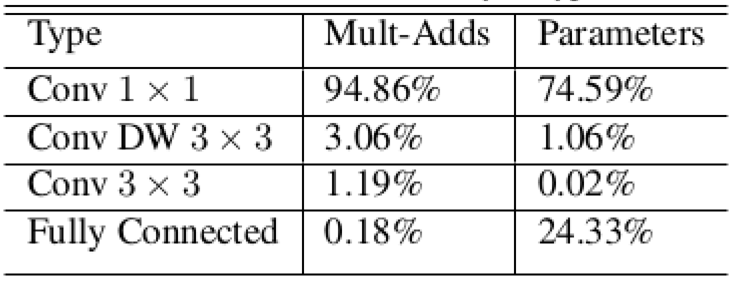
where ρ ∈ (0, 1] which is typically set implicitly so that the input resolution of the network is 224, 192, 160 or 128. ρ = 1 is the baseline MobileNet and ρ < 1 are reduced computation MobileNets. Resolution multiplier has the effect of reducing computational cost by ρ 2 .

As an example we can look at a typical layer in MobileNet and see how depthwise separable convolutions, width multiplier and resolution multiplier reduce the cost and parameters. Table 3 shows the computation and number of parameters for a layer as architecture shrinking methods are sequentially applied to the layer. The first row shows the Mult-Adds and parameters for a full convolutional layer with an input feature map of size 14 × 14 × 512 with a kernel K of size 3 × 3 × 512 × 512. We will look in detail in the next section at the trade offs between resources and accuracy.

其中，ρ∈（0，1）通常是隐式设置的，因此网络的输入分辨率为224、192、160或128。 ρ = 1是基线MobileNet，ρ < 1是减少计算的MobileNets。分辨率倍增器的作用是将计算成本降低ρ 2。

作为一个例子，我们可以看一下MobileNet中的一个典型层，看看深度可分离卷积、宽度乘法器和分辨率乘法器是如何减少成本和参数的。表3显示了一个层的计算和参数数量，因为架构缩减方法被依次应用于该层。第一行显示了一个全卷积层的Mult-Adds和参数，该层的输入特征图大小为14×14×512，内核K大小为3×3×512×512。我们将在下一节中详细研究资源和精度之间的权衡问题。

Table 2. Resource Per Layer Type



Type	Mult-Adds	Parameters
Conv 1 × 1	94.86%	74.59%
Conv DW 3 × 3	3.06%	1.06%
Conv 3 × 3	1.19%	0.02%
Fully Connected	0.18%	24.33%