MobileNetV2: Inverted Residuals and Linear Bottlenecks

Mark Sandler Andrew Howard Menglong Zhu Andrey Zhmoginov Liang-Chieh Chen
Google Inc.
{sandler, howarda, menglong, azhmogin, lcchen}@google.com

Abstract

In this paper we describe a new mobile architecture, MobileNetV2, that improves the state of the art performance of mobile models on multiple tasks and benchmarks as well as across a spectrum of different model sizes. We also describe efficient ways of applying these mobile models to object detection in a novel framework we call SSDLite. Additionally, we demonstrate how to build mobile semantic segmentation models through a reduced form of DeepLabv3 which we call Mobile DeepLabv3.

is based on an inverted residual structure where the shortcut connections are between the thin bottleneck layers. The intermediate expansion layer uses lightweight depthwise convolutions to filter features as a source of non-linearity. Additionally, we find that it is important to remove non-linearities in the narrow layers in order to maintain representational power. We demonstrate that this improves performance and provide an intuition that led to this design.

Finally, our approach allows decoupling of the input/output domains from the expressiveness of the transformation, which provides a convenient framework for further analysis. We measure our performance on ImageNet [1] classification, COCO object detection [2], VOC image segmentation [3]. We evaluate the trade-offs between accuracy, and number of operations measured by multiply-adds (MAdd), as well as actual latency, and the number of parameters.

1. Introduction

Neural networks have revolutionized many areas of machine intelligence, enabling superhuman accuracy for challenging image recognition tasks. However, the drive to improve accuracy often comes at a cost: modern state of the art networks require high computational resources beyond the capabilities of many mobile and embedded applications.

This paper introduces a new neural network architecture that is specifically tailored for mobile and resource constrained environments. Our network pushes the state of the art for mobile tailored computer vision models, by significantly decreasing the number of operations and memory needed while retaining the same accuracy.

Our main contribution is a novel layer module: the inverted residual with linear bottleneck. This module takes as an input a low-dimensional compressed representation which is first expanded to high dimension and filtered with a lightweight depthwise convolution. Features are subsequently projected back to a low-dimensional representation with a *linear convolution*. The official implementation is available as part of TensorFlow-Slim model library in [4].

This module can be efficiently implemented using standard operations in any modern framework and allows our models to beat state of the art along multiple performance points using standard benchmarks. Furthermore, this convolutional module is particularly suitable for mobile designs, because it allows to significantly reduce the memory footprint needed during inference by never fully materializing large intermediate tensors. This reduces the need for main memory access in many embedded hardware designs, that provide small amounts of very fast software controlled cache memory.

2. Related Work

Tuning deep neural architectures to strike an optimal balance between accuracy and performance has been an area of active research for the last several years. Both manual architecture search and improvements in training algorithms, carried out by numerous teams has lead to dramatic improvements over early designs such as AlexNet [5], VGGNet [6], GoogLeNet [7]. , and ResNet [8]. Recently there has been lots of progress in algorithmic architecture exploration included hyperparameter optimization [9, 10, 11] as well as various

methods of network pruning [12, 13, 14, 15, 16, 17] and connectivity learning [18, 19]. A substantial amount of work has also been dedicated to changing the connectivity structure of the internal convolutional blocks such as in ShuffleNet [20] or introducing sparsity [21] and others [22].

Recently, [23, 24, 25, 26], opened up a new direction of bringing optimization methods including genetic algorithms and reinforcement learning to architectural search. However one drawback is that the resulting networks end up very complex. In this paper, we pursue the goal of developing better intuition about how neural networks operate and use that to guide the simplest possible network design. Our approach should be seen as complimentary to the one described in [23] and related work. In this vein our approach is similar to those taken by [20, 22] and allows to further improve the performance, while providing a glimpse on its internal operation. Our network design is based on MobileNetV1 [27]. It retains its simplicity and does not require any special operators while significantly improves its accuracy, achieving state of the art on multiple image classification and detection tasks for mobile applications.

3. Preliminaries, discussion and intuition

3.1. Depthwise Separable Convolutions

Depthwise Separable Convolutions are a key building block for many efficient neural network architectures [27, 28, 20] and we use them in the present work as well. The basic idea is to replace a full convolutional operator with a factorized version that splits convolution into two separate layers. The first layer is called a depthwise convolution, it performs lightweight filtering by applying a single convolutional filter per input channel. The second layer is a 1×1 convolution, called a pointwise convolution, which is responsible for building new features through computing linear combinations of the input channels.

Standard convolution takes an $h_i \times w_i \times d_i$ input tensor L_i, and applies convolutional kernel $K \in \mathcal{R}^{k \times k \times d_i \times d_j}$ to produce an $h_i \times w_i \times d_j$ output tensor L_j. Standard convolutional layers have the computational cost of $h_i \cdot w_i \cdot d_i \cdot d_j \cdot k \cdot k$.

Depthwise separable convolutions are a drop-in replacement for standard convolutional layers. Empirically they work almost as well as regular convolutions but only cost:

$$h_i \cdot w_i \cdot d_i(k^2 + d_j) \tag{1}$$

which is the sum of the depthwise and 1×1 pointwise convolutions. Effectively depthwise separable convolution reduces computation compared to traditional layers by almost a factor of k^2[1]. MobileNetV2 uses $k = 3$ (3×3 depthwise separable convolutions) so the computational cost is 8 to 9 times smaller than that of standard convolutions at only a small reduction in accuracy [27].

3.2. Linear Bottlenecks

Consider a deep neural network consisting of n layers L_i each of which has an activation tensor of dimensions $h_i \times w_i \times d_i$. Throughout this section we will be discussing the basic properties of these activation tensors, which we will treat as containers of $h_i \times w_i$ "pixels" with d_i dimensions. Informally, for an input set of real images, we say that the set of layer activations (for any layer L_i) forms a "manifold of interest". It has been long assumed that manifolds of interest in neural networks could be embedded in low-dimensional subspaces. In other words, when we look at all individual d-channel pixels of a deep convolutional layer, the information encoded in those values actually lie in some manifold, which in turn is embeddable into a low-dimensional subspace[2].

At a first glance, such a fact could then be captured and exploited by simply reducing the dimensionality of a layer thus reducing the dimensionality of the operating space. This has been successfully exploited by MobileNetV1 [27] to effectively trade off between computation and accuracy via a width multiplier parameter, and has been incorporated into efficient model designs of other networks as well [20]. Following that intuition, the width multiplier approach allows one to reduce the dimensionality of the activation space until the manifold of interest spans this entire space. However, this intuition breaks down when we recall that deep convolutional neural networks actually have non-linear per coordinate transformations, such as ReLU. For example, ReLU applied to a line in 1D space produces a 'ray', where as in $\mathcal{R}^n$ space, it generally results in a piece-wise linear curve with n-joints.

It is easy to see that in general if a result of a layer transformation $\mathrm{ReLU}(Bx)$ has a non-zero volume S, the points mapped to $\mathrm{interior}\, S$ are obtained via a linear transformation B of the input, thus indicating that the part of the input space corresponding to the full dimensional output, is limited to a linear transformation. In other words, deep networks only have the power of a linear classifier on the non-zero volume part of the

[1] more precisely, by a factor $k^2 d_j/(k^2 + d_j)$

[2] Note that dimensionality of the manifold differs from the dimensionality of a subspace that could be embedded via a linear transformation.

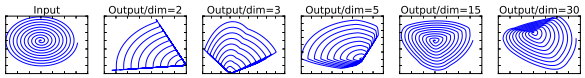


Figure 1: Examples of ReLU transformations of low-dimensional manifolds embedded in higher-dimensional spaces. In these examples the initial spiral is embedded into an n-dimensional space using random matrix T followed by ReLU, and then projected back to the 2D space using T^{-1}. In examples above $n = 2, 3$ result in information loss where certain points of the manifold collapse into each other, while for $n = 15$ to 30 the transformation is highly non-convex.

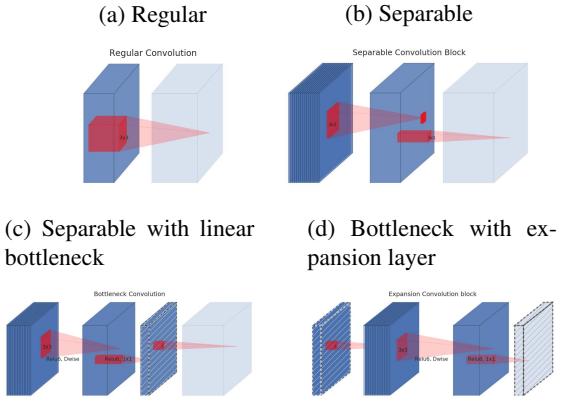


Figure 2: Evolution of separable convolution blocks. The diagonally hatched texture indicates layers that do not contain non-linearities. The last (lightly colored) layer indicates the beginning of the next block. Note: 2d and 2c are equivalent blocks when stacked. Best viewed in color.

output domain. We refer to supplemental material for a more formal statement.

On the other hand, when ReLU collapses the channel, it inevitably loses information in *that channel*. However if we have lots of channels, and there is a a structure in the activation manifold that information might still be preserved in the other channels. In supplemental materials, we show that if the input manifold can be embedded into a significantly lower-dimensional subspace of the activation space then the ReLU transformation preserves the information while introducing the needed complexity into the set of expressible functions.

To summarize, we have highlighted two properties that are indicative of the requirement that the manifold of interest should lie in a low-dimensional subspace of the higher-dimensional activation space:

1. If the manifold of interest remains non-zero volume after ReLU transformation, it corresponds to a linear transformation.

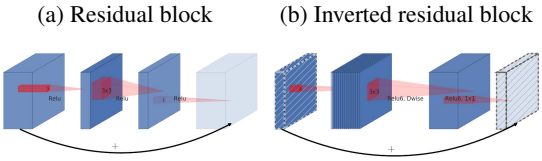


Figure 3: The difference between residual block [8, 30] and inverted residual. Diagonally hatched layers do not use non-linearities. We use thickness of each block to indicate its relative number of channels. Note how classical residuals connects the layers with high number of channels, whereas the inverted residuals connect the bottlenecks. Best viewed in color.

2. ReLU is capable of preserving complete information about the input manifold, but only if the input manifold lies in a low-dimensional subspace of the input space.

These two insights provide us with an empirical hint for optimizing existing neural architectures: assuming the manifold of interest is low-dimensional we can capture this by inserting *linear bottleneck* layers into the convolutional blocks. Experimental evidence suggests that using linear layers is crucial as it prevents non-linearities from destroying too much information. In Section 6, we show empirically that using non-linear layers in bottlenecks indeed hurts the performance by several percent, further validating our hypothesis[3]. We note that similar reports where non-linearity was helped were reported in [29] where non-linearity was removed from the input of the traditional residual block and that lead to improved performance on CIFAR dataset.

For the remainder of this paper we will be utilizing bottleneck convolutions. We will refer to the ratio between the size of the input bottleneck and the inner size as the *expansion ratio*.

3.3. Inverted residuals

The bottleneck blocks appear similar to residual block where each block contains an input followed by several bottlenecks then followed by expansion [8]. However, inspired by the intuition that the bottlenecks actually contain all the necessary information, while an expansion layer acts merely as an implementation detail that accompanies a non-linear transformation of the tensor, we use shortcuts directly between the bottlenecks.

[3]We note that in the presence of shortcuts the information loss is actually less strong.

Figure 3 provides a schematic visualization of the difference in the designs. The motivation for inserting shortcuts is similar to that of classical residual connections: we want to improve the ability of a gradient to propagate across multiplier layers. However, the inverted design is considerably more memory efficient (see Section 5 for details), as well as works slightly better in our experiments.

Running time and parameter count for bottleneck convolution The basic implementation structure is illustrated in Table 1. For a block of size $h \times w$, expansion factor t and kernel size k with d' input channels and d'' output channels, the total number of multiply add required is $h \cdot w \cdot d' \cdot t(d' + k^2 + d'')$. Compared with (1) this expression has an extra term, as indeed we have an extra 1×1 convolution, however the nature of our networks allows us to utilize much smaller input and output dimensions. In Table 3 we compare the needed sizes for each resolution between MobileNetV1, MobileNetV2 and ShuffleNet.

3.4. Information flow interpretation

One interesting property of our architecture is that it provides a natural separation between the input/output *domains* of the building blocks (bottleneck layers), and the *layer transformation* – that is a non-linear function that converts input to the output. The former can be seen as the *capacity* of the network at each layer, whereas the latter as the *expressiveness*. This is in contrast with traditional convolutional blocks, both regular and separable, where both expressiveness and capacity are tangled together and are functions of the output layer depth.

In particular, in our case, when inner layer depth is 0 the underlying convolution is the identity function thanks to the shortcut connection. When the expansion ratio is smaller than 1, this is a classical residual convolutional block [8, 30]. However, for our purposes we show that expansion ratio greater than 1 is the most useful.

This interpretation allows us to study the expressiveness of the network separately from its capacity and we believe that further exploration of this separation is warranted to provide a better understanding of the network properties.

4. Model Architecture

Now we describe our architecture in detail. As discussed in the previous section the basic building block is a bottleneck depth-separable convolution with residuals. The detailed structure of this block is shown in Table 1. The architecture of MobileNetV2 contains the initial fully convolution layer with 32 filters, followed by 19 *residual bottleneck* layers described in the Table 2. We use ReLU6 as the non-linearity because of its robustness when used with low-precision computation [27]. We always use kernel size 3×3 as is standard for modern networks, and utilize dropout and batch normalization during training.

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Table 1: *Bottleneck residual block* transforming from k to k' channels, with stride s, and expansion factor t.

With the exception of the first layer, we use constant expansion rate throughout the network. In our experiments we find that expansion rates between 5 and 10 result in nearly identical performance curves, with smaller networks being better off with slightly smaller expansion rates and larger networks having slightly better performance with larger expansion rates.

For all our main experiments we use expansion factor of 6 applied to the size of the input tensor. For example, for a bottleneck layer that takes 64-channel input tensor and produces a tensor with 128 channels, the intermediate expansion layer is then $64 \cdot 6 = 384$ channels.

Trade-off hyper parameters As in [27] we tailor our architecture to different performance points, by using the input image resolution and width multiplier as tunable hyper parameters, that can be adjusted depending on desired accuracy/performance trade-offs. Our primary network (width multiplier 1, 224×224), has a computational cost of 300 million multiply-adds and uses 3.4 million parameters. We explore the performance trade offs, for input resolutions from 96 to 224, and width multipliers of 0.35 to 1.4. The network computational cost ranges from 7 multiply adds to 585M MAdds, while the model size vary between 1.7M and 6.9M parameters.

One minor implementation difference, with [27] is that for multipliers less than one, we apply width multiplier to all layers except the very last convolutional layer. This improves performance for smaller models.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Table 2: MobileNetV2 : Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated n times. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor t is always applied to the input size as described in Table 1.

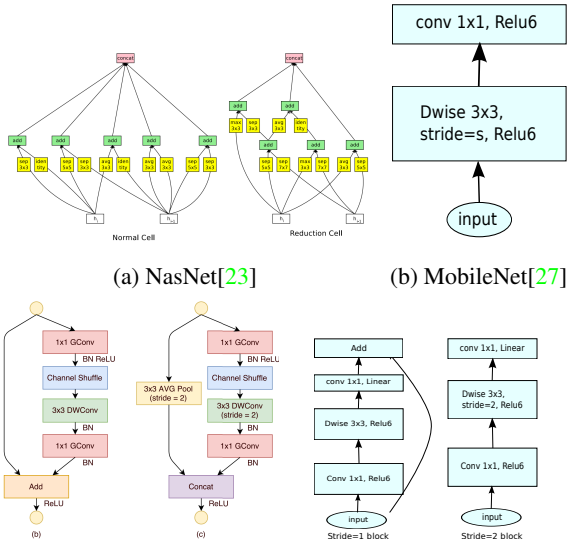
Size	MobileNetV1	MobileNetV2	ShuffleNet (2x,g=3)
112x112	1/O(1)	1/O(1)	1/O(1)
56x56	128/800	32/200	48/300
28x28	256/400	64/100	400/600K
14x14	512/200	160/62	800/310
7x7	1024/199	320/32	1600/156
1x1	1024/2	1280/2	1600/3
max	800K	**200K**	600K

Table 3: The max number of channels/memory (in Kb) that needs to be materialized at each spatial resolution for different architectures. We assume 16-bit floats for activations. For ShuffleNet, we use $2x, g = 3$ that matches the performance of MobileNetV1 and MobileNetV2. For the first layer of MobileNetV2 and ShuffleNet we can employ the trick described in Section 5 to reduce memory requirement. Even though ShuffleNet employs bottlenecks elsewhere, the non-bottleneck tensors still need to be materialized due to the presence of shortcuts between non-bottleneck tensors.

5. Implementation Notes

5.1. Memory efficient inference

The inverted residual bottleneck layers allow a particularly memory efficient implementation which is very important for mobile applications. A standard efficient implementation of inference that uses for instance



(a) NasNet[23] (b) MobileNet[27]

(c) ShuffleNet [20] (d) Mobilenet V2

Figure 4: Comparison of convolutional blocks for different architectures. ShuffleNet uses Group Convolutions [20] and shuffling, it also uses conventional residual approach where inner blocks are narrower than output. ShuffleNet and NasNet illustrations are from respective papers.

TensorFlow[31] or Caffe [32], builds a directed acyclic compute hypergraph G, consisting of edges representing the operations and nodes representing tensors of intermediate computation. The computation is scheduled in order to minimize the total number of tensors that needs to be stored in memory. In the most general case, it searches over all plausible computation orders $\Sigma(G)$ and picks the one that minimizes

$$M(G) = \min_{\pi \in \Sigma(G)} \max_{i \in 1..n} \left[\sum_{A \in R(i,\pi,G)} |A| \right] + \text{size}(\pi_i).$$

where $R(i, \pi, G)$ is the list of intermediate tensors that are connected to any of $\pi_i \ldots \pi_n$ nodes, $|A|$ represents the size of the tensor A and $size(i)$ is the total amount of memory needed for internal storage during operation i.

For graphs that have only trivial parallel structure (such as residual connection), there is only one non-trivial feasible computation order, and thus the total amount and a bound on the memory needed for infer-

ence on compute graph G can be simplified:

$$M(G) = \max_{op \in G} \left[\sum_{A \in \text{op}_{inp}} |A| + \sum_{B \in \text{op}_{out}} |B| + |op| \right] \quad (2)$$

Or to restate, the amount of memory is simply the maximum total size of combined inputs and outputs across all operations. In what follows we show that if we treat a bottleneck residual block as a single operation (and treat inner convolution as a disposable tensor), the total amount of memory would be dominated by the size of bottleneck tensors, rather than the size of tensors that are internal to bottleneck (and much larger).

Bottleneck Residual Block A bottleneck block operator $\mathcal{F}(x)$ shown in Figure 3b can be expressed as a composition of three operators $\mathcal{F}(x) = [A \circ \mathcal{N} \circ B]x$, where A is a linear transformation $A : \mathcal{R}^{s \times s \times k} \to \mathcal{R}^{s \times s \times n}$, $\mathcal{N}$ is a non-linear per-channel transformation: $\mathcal{N} : \mathcal{R}^{s \times s \times n} \to \mathcal{R}^{s' \times s' \times n}$, and B is again a linear transformation to the output domain: $B : \mathcal{R}^{s' \times s' \times n} \to \mathcal{R}^{s' \times s' \times k'}$.

For our networks $\mathcal{N} = \text{ReLU6} \circ \text{dwise} \circ \text{ReLU6}$, but the results apply to any per-channel transformation. Suppose the size of the input domain is $|x|$ and the size of the output domain is $|y|$, then the memory required to compute $F(X)$ can be as low as $|s^2k| + |s'^2k'| + O(\max(s^2, s'^2))$.

The algorithm is based on the fact that the inner tensor $\mathcal{I}$ can be represented as concatenation of t tensors, of size n/t each and our function can then be represented as

$$\mathcal{F}(x) = \sum_{i=1}^{t} (A_i \circ N \circ B_i)(x)$$

by accumulating the sum, we only require one intermediate block of size n/t to be kept in memory at all times. Using $n = t$ we end up having to keep only a single channel of the intermediate representation at all times. The two constraints that enabled us to use this trick is (a) the fact that the inner transformation (which includes non-linearity and depthwise) is per-channel, and (b) the consecutive non-per-channel operators have significant ratio of the input size to the output. For most of the traditional neural networks, such trick would not produce a significant improvement.

We note that, the number of multiply-adds operators needed to compute $F(X)$ using t-way split is independent of t, however in existing implementations we find that replacing one matrix multiplication with several smaller ones hurts runtime performance due to increased cache misses. We find that this approach is the most helpful to be used with t being a small constant between 2 and 5. It significantly reduces the memory requirement, but still allows one to utilize most of the efficiencies gained by using highly optimized matrix multiplication and convolution operators provided by deep learning frameworks. It remains to be seen if special framework level optimization may lead to further runtime improvements.

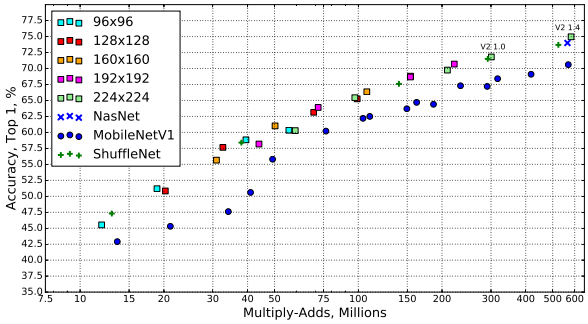
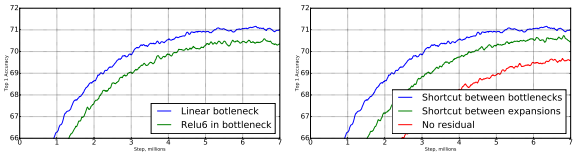


Figure 5: Performance curve of MobileNetV2 vs MobileNetV1, ShuffleNet, NAS. For our networks we use multipliers 0.35, 0.5, 0.75, 1.0 for all resolutions, and additional 1.4 for for 224. Best viewed in color.



(a) Impact of non-linearity in the bottleneck layer.

(b) Impact of variations in residual blocks.

Figure 6: The impact of non-linearities and various types of shortcut (residual) connections.

6. Experiments

6.1. ImageNet Classification

Training setup We train our models using TensorFlow[31]. We use the standard RMSPropOptimizer with both decay and momentum set to 0.9. We use batch normalization after every layer, and the standard weight decay is set to 0.00004. Following MobileNetV1[27] setup we use initial learning rate of 0.045, and learning rate decay rate of 0.98 per epoch. We use 16 GPU asynchronous workers, and a batch size of 96.

Results We compare our networks against MobileNetV1, ShuffleNet and NASNet-A models. The statistics of a few selected models is shown in Table 4 with the full performance graph shown in Figure 5.

6.2. Object Detection

We evaluate and compare the performance of MobileNetV2 and MobileNetV1 as feature extractors [33] for object detection with a modified version of the Single Shot Detector (SSD) [34] on COCO dataset [2]. We also compare to YOLOv2 [35] and original SSD (with VGG-16 [6] as base network) as baselines. We do not compare performance with other architectures such as Faster-RCNN [36] and RFCN [37] since our focus is on mobile/real-time models.

SSDLite: In this paper, we introduce a mobile friendly variant of regular SSD. We replace all the regular convolutions with separable convolutions (depthwise followed by 1×1 projection) in SSD prediction layers. This design is in line with the overall design of MobileNets and is seen to be much more computationally efficient. We call this modified version SSDLite. Compared to regular SSD, SSDLite dramatically reduces both parameter count and computational cost as shown in Table 5.

For MobileNetV1, we follow the setup in [33]. For MobileNetV2, the first layer of SSDLite is attached to the expansion of layer 15 (with output stride of 16). The second and the rest of SSDLite layers are attached on top of the last layer (with output stride of 32). This setup is consistent with MobileNetV1 as all layers are attached to the feature map of the same output strides.

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	**3.4M**	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	**72.0**	**3.4M**	**300M**	**75ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**143ms**

Table 4: Performance on ImageNet, comparison for different networks. As is common practice for ops, we count the total number of Multiply-Adds. In the last column we report running time in milliseconds (ms) for a single large core of the Google Pixel 1 phone (using TF-Lite). We do not report ShuffleNet numbers as efficient group convolutions and shuffling are not yet supported.

	Params	MAdds
SSD[34]	14.8M	1.25B
SSDLite	**2.1M**	**0.35B**

Table 5: Comparison of the size and the computational cost between SSD and SSDLite configured with MobileNetV2 and making predictions for 80 classes.

Network	mAP	Params	MAdd	CPU
SSD300[34]	23.2	36.1M	35.2B	-
SSD512[34]	26.8	36.1M	99.5B	-
YOLOv2[35]	21.6	50.7M	17.5B	-
MNet V1 + SSDLite	22.2	5.1M	1.3B	270ms
MNet V2 + SSDLite	22.1	**4.3M**	**0.8B**	200ms

Table 6: Performance comparison of MobileNetV2 + SSDLite and other realtime detectors on the COCO dataset object detection task. MobileNetV2 + SSDLite achieves competitive accuracy with significantly fewer parameters and smaller computational complexity. All models are trained on `trainval35k` and evaluated on `test-dev`. SSD/YOLOv2 numbers are from [35]. The running time is reported for the large core of the Google Pixel 1 phone, using an internal version of the TF-Lite engine.

Both MobileNet models are trained and evaluated with Open Source TensorFlow Object Detection API [38]. The input resolution of both models is 320×320. We benchmark and compare both mAP (COCO challenge metrics), number of parameters and number of Multiply-Adds. The results are shown in Table 6. MobileNetV2 SSDLite is not only the most efficient model, but also the most accurate of the three. Notably, MobileNetV2 SSDLite is $20\times$ more efficient and $10\times$ smaller while still outperforms YOLOv2 on COCO dataset.

6.3. Semantic Segmentation

In this section, we compare MobileNetV1 and MobileNetV2 models used as feature extractors with DeepLabv3 [39] for the task of mobile semantic segmentation. DeepLabv3 adopts atrous convolution [40, 41, 42], a powerful tool to explicitly control the resolution of computed feature maps, and builds five parallel heads including (a) Atrous Spatial Pyramid Pooling module (ASPP) [43] containing three 3×3 convolutions with different atrous rates, (b) 1×1 convolution head, and (c) Image-level features [44]. We denote by

output_stride the ratio of input image spatial resolution to final output resolution, which is controlled by applying the atrous convolution properly. For semantic segmentation, we usually employ *output_stride* $= 16$ or 8 for denser feature maps. We conduct the experiments on the PASCAL VOC 2012 dataset [3], with extra annotated images from [45] and evaluation metric mIOU.

To build a mobile model, we experimented with three design variations: (1) different feature extractors, (2) simplifying the DeepLabv3 heads for faster computation, and (3) different inference strategies for boosting the performance. Our results are summarized in Table 7. We have observed that: (a) the inference strategies, including multi-scale inputs and adding left-right flipped images, significantly increase the MAdds and thus are not suitable for on-device applications, (b) using *output_stride* $= 16$ is more efficient than *output_stride* $= 8$, (c) MobileNetV1 is already a powerful feature extractor and only requires about $4.9 - 5.7$ times fewer MAdds than ResNet-101 [8] (*e.g.*, mIOU: 78.56 *vs* 82.70, and MAdds: 941.9B *vs* 4870.6B), (d) it is more efficient to build DeepLabv3 heads on top of the second last feature map of MobileNetV2 than on the original last-layer feature map, since the second to last feature map contains 320 channels instead of 1280, and by doing so, we attain similar performance, but require about 2.5 times fewer operations than the MobileNetV1 counterparts, and (e) DeepLabv3 heads are computationally expensive and removing the ASPP module significantly reduces the MAdds with only a slight performance degradation. In the end of the Table 7, we identify a potential candidate for on-device applications (in bold face), which attains 75.32% mIOU and only requires 2.75B MAdds.

6.4. Ablation study

Inverted residual connections. The importance of residual connection has been studied extensively [8, 30, 46]. The new result reported in this paper is that the shortcut connecting bottleneck perform better than shortcuts connecting the expanded layers (see Figure 6b for comparison).

Importance of linear bottlenecks. The linear bottleneck models are strictly less powerful than models with non-linearities, because the activations can always operate in linear regime with appropriate changes to biases and scaling. However our experiments shown in Figure 6a indicate that linear bottlenecks improve performance, providing support that non-linearity destroys information in low-dimensional space.

Network	OS	ASPP	MF	mIOU	Params	MAdds
MNet V1	16	✓		75.29	11.15M	14.25B
	8	✓	✓	78.56	11.15M	941.9B
MNet V2*	16	✓		75.70	4.52M	5.8B
	8	✓	✓	78.42	4.52M	387B
MNet V2*	16			**75.32**	**2.11M**	**2.75B**
	8		✓	77.33	2.11M	152.6B
ResNet-101	16	✓		80.49	58.16M	81.0B
	8	✓	✓	82.70	58.16M	4870.6B

Table 7: MobileNet + DeepLabv3 inference strategy on the PASCAL VOC 2012 *validation* set. **MNet V2***: Second last feature map is used for DeepLabv3 heads, which includes (1) Atrous Spatial Pyramid Pooling (**ASPP**) module, and (2) 1×1 convolution as well as image-pooling feature. **OS**: *output_stride* that controls the output resolution of the segmentation map. **MF**: Multi-scale and left-right flipped inputs during test. All of the models have been pretrained on COCO. The potential candidate for on-device applications is shown in bold face. PASCAL images have dimension 512×512 and atrous convolution allows us to control output feature resolution without increasing the number of parameters.

7. Conclusions and future work

We described a very simple network architecture that allowed us to build a family of highly efficient mobile models. Our basic building unit, has several properties that make it particularly suitable for mobile applications. It allows very memory-efficient inference and relies utilize standard operations present in all neural frameworks.

For the ImageNet dataset, our architecture improves the state of the art for wide range of performance points.

For object detection task, our network outperforms state-of-art realtime detectors on COCO dataset both in terms of accuracy and model complexity. Notably, our architecture combined with the SSDLite detection module is $20\times$ less computation and $10\times$ less parameters than YOLOv2.

On the theoretical side: the proposed convolutional block has a unique property that allows to separate the network expressiviness (encoded by expansion layers) from its capacity (encoded by bottleneck inputs). Exploring this is an important direction for future research.

Acknowledgments We would like to thank Matt Streeter and Sergey Ioffe for their helpful feedback and discussion.

References

[1] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, December 2015. 1

[2] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014. 1, 7

[3] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserma. The pascal visual object classes challenge a retrospective. *IJCV*, 2014. 1, 8

[4] Mobilenetv2 source code. Available from https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet. 1

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Bartlett et al. [47], pages 1106–1114. 1

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1, 7

[7] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9. IEEE Computer Society, 2015. 1

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 1, 3, 4, 8

[9] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012. 1

[10] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In Bartlett et al. [47], pages 2960–2968. 1

[11] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. Scalable bayesian optimization using deep neural networks. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2171–2180. JMLR.org, 2015. 1

[12] Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In Stephen Jose Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pages 164–171. Morgan Kaufmann, 1992. 2

[13] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pages 598–605. Morgan Kaufmann, 1989. 2

[14] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1135–1143, 2015. 2

[15] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Shijian Tang, Erich Elsen, Bryan Catanzaro, John Tran, and William J. Dally. DSD: regularizing deep neural networks with dense-sparse-dense training flow. *CoRR*, abs/1607.04381, 2016. 2

[16] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1379–1387, 2016. 2

[17] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016. 2

[18] Karim Ahmed and Lorenzo Torresani. Connectivity learning in multi-branch networks. *CoRR*, abs/1709.09582, 2017. 2

[19] Tom Veniat and Ludovic Denoyer. Learning time-efficient deep architectures with budgeted super networks. *CoRR*, abs/1706.00046, 2017. 2

[20] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017. 2, 5

[21] Soravit Changpinyo, Mark Sandler, and Andrey Zhmoginov. The power of sparsity in convolutional neural networks. *CoRR*, abs/1702.06257, 2017. 2

[22] Min Wang, Baoyuan Liu, and Hassan Foroosh. Design of efficient convolutional layers using single intra-channel convolution, topological subdivisioning and spatial "bottleneck" structure. *CoRR*, abs/1608.04337, 2016. 2

[23] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017. 2, 5

[24] Lingxi Xie and Alan L. Yuille. Genetic CNN. *CoRR*, abs/1703.01513, 2017. 2

[25] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2902–2911. PMLR, 2017. 2

[26] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016. 2

[27] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 2, 4, 5, 6

[28] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 2

[29] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. *CoRR*, abs/1610.02915, 2016. 3

[30] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016. 3, 4, 8

[31] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 5, 6

[32] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 5

[33] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *CVPR*, 2017. 7

[34] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016. 7

[35] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016. 7

[36] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 7

[37] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016. 7

[38] Jonathan Huang, Vivek Rathod, Derek Chow, Chen Sun, and Menglong Zhu. Tensorflow object detection api, 2017. 7

[39] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017. 7

[40] Matthias Holschneider, Richard Kronland-Martinet, Jean Morlet, and Ph Tchamitchian. A real-time algorithm for signal analysis with the help of the wavelet transform. In *Wavelets: Time-Frequency Methods and Phase Space*, pages 289–297. 1989. 7

[41] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv:1312.6229*, 2013. 7

[42] George Papandreou, Iasonas Kokkinos, and Pierre-Andre Savalle. Modeling local and global deformations in deep learning: Epitomic convolution, multiple instance learning, and sliding window detection. In *CVPR*, 2015. 7

[43] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 2017. 7

[44] Wei Liu, Andrew Rabinovich, and Alexander C. Berg. Parsenet: Looking wider to see better. *CoRR*, abs/1506.04579, 2015. 7

[45] Bharath Hariharan, Pablo Arbeláez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *ICCV*, 2011. 8

[46] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016. 8

[47] Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors. *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, 2012. 9