# Intro

MobileNets are a family of mobile-first computer vision models for [TensorFlow](https://www.tensorflow.org/), designed to effectively maximize accuracy while being mindful of the restricted resources for an on-device or embedded application. This architecture was proposed by Google.

MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of a variety of use cases. They can be built upon for classification, detection, embeddings, and segmentation, similar to how other popular large scale models, such as [Inception](https://arxiv.org/pdf/1602.07261.pdf), are used.

# Key features:

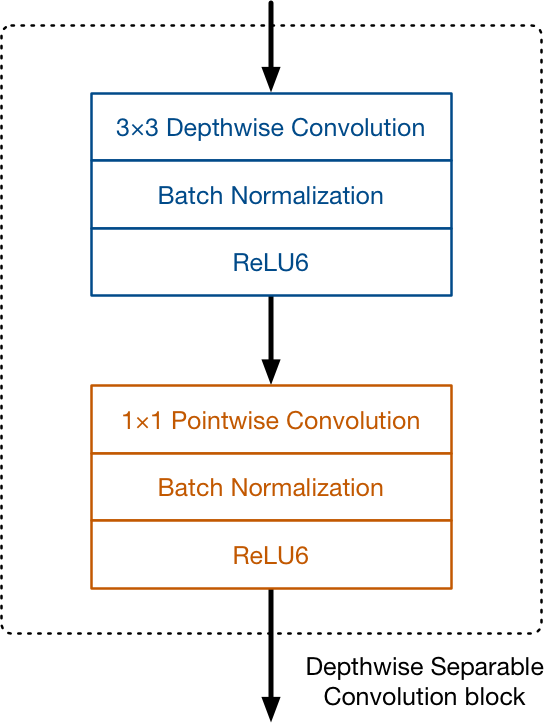
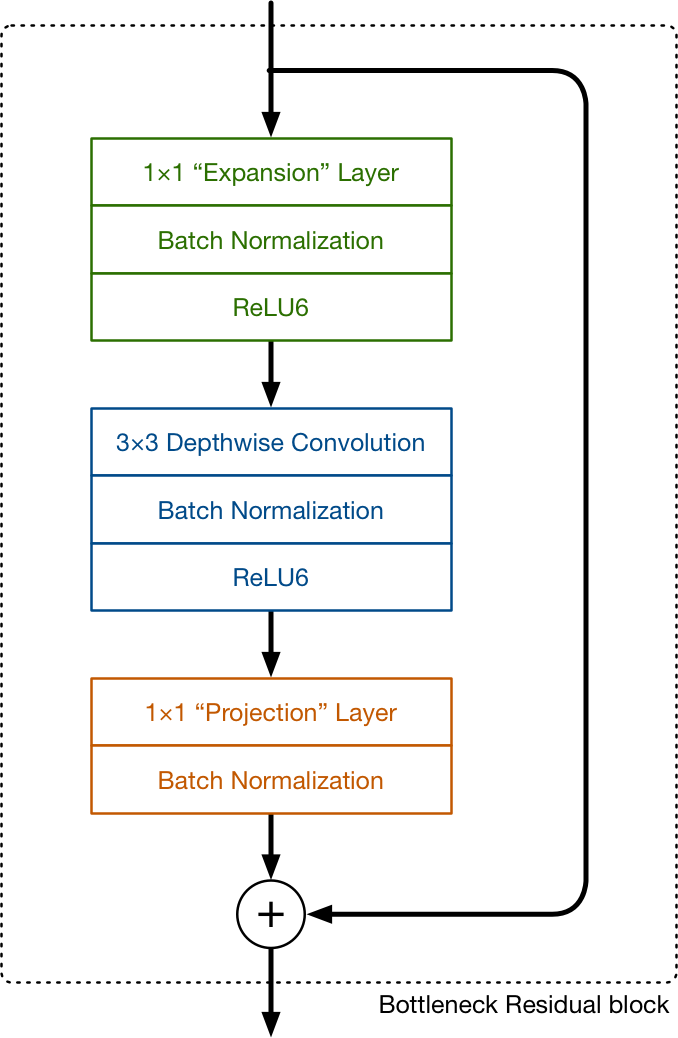
* This architecture uses **depthwise separable convolutions** which significantly **reduces the number of parameters** when compared to the network with normal convolutions with the **same depth** in the networks. This results in light weight deep neural networks.
* The normal convolution is replaced by **depthwise convolution followed by pointwise convolution** which is called as **depthwise separable convolution**.
* This results in the reduction of number of parameters significantly and thereby **reduces the total number of floating point multiplication operations** which is **favorable in mobile and embedded vision applications** with less compute power.
* By **using depthwise separable convolutions**, there is some **sacrifice of accuracy** for **low complexity deep neural network**. For better understanding and for the metrics, one can read the paper mentioned below.

# How it works in details:

## MobileNet version 1

The big idea behind MobileNet V1 is that convolutional layers, which are essential to computer vision tasks but are quite expensive to compute, can be replaced by so-called **depthwise separable** convolutions.

The job of the convolution layer is split into two subtasks: first there is a [depthwise convolution](https://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/) layer that filters the input, followed by a 1×1 (or pointwise) convolution layer that combines these filtered values to create new features.

v1 structure v2 structure

Together, the depthwise and pointwise convolutions form a “depthwise separable” convolution block. It does approximately the same thing as traditional convolution but is much faster. Unlike a regular convolution it does not combine the input channels but it performs convolution on each channel separately. For an image with 3 channels, a depthwise convolution creates an output image that also has 3 channels. Each channel gets its own set of weights. The purpose of the depthwise convolution is to *filter* the input channels. Think edge detection, color filtering, and so on.

The full architecture of MobileNet V1 consists of a regular 3×3 convolution as the very first layer, followed by 13 times the above building block.

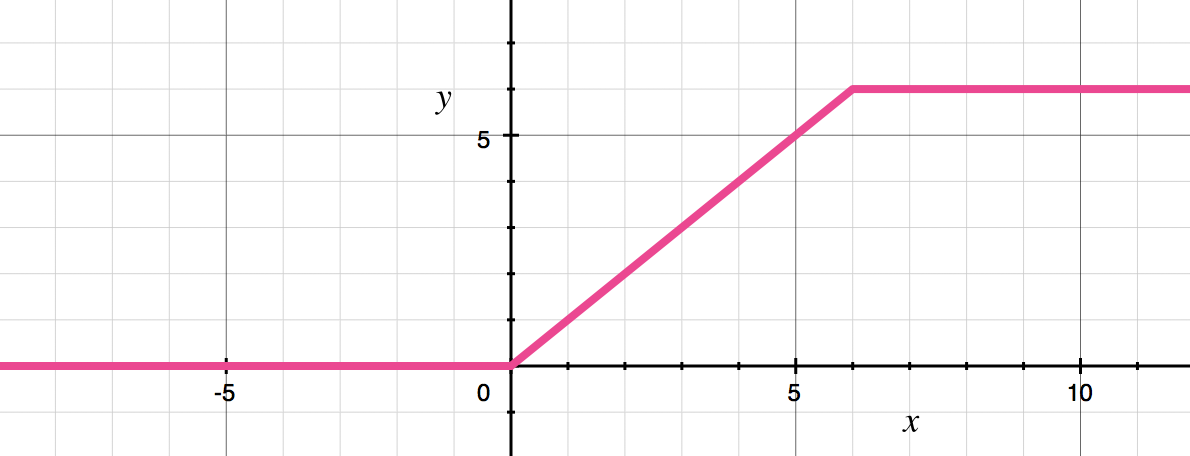
There are no pooling layers in between these depth-wise separable blocks. Instead, some of the depth-wise layers have a stride of 2 to reduce the spatial dimensions of the data. When that happens, the corresponding pointwise layer also doubles the number of output channels. If the input image is 224×224×3 then the output of the network is a 7×7×1024 feature map.

As is common in modern architectures, the convolution layers are followed by batch normalization.

The activation function used by MobileNet is **ReLU6**.

The authors of the MobileNet paper found that ReLU6 is more robust than regular ReLU when using low-precision computation.

It also makes the shape of the function look more like a sigmoid:



In a classifier based on MobileNet, there is typically a global average pooling layer at the very end, followed by a fully-connected classification layer or an equivalent 1×1 convolution, and a softmax.

The most important of these hyper-parameters is the **depth multiplier**, confusingly also known as the “width multiplier”. This changes how many channels are in each layer. Using a depth multiplier of 0.5 will halve the number of channels used in each layer, which cuts down the number of computations by a factor of 4 and the number of learnable parameters by a factor 3. It is therefore much faster than the full model but also less accurate.

Thanks to the innovation of depthwise separable convolutions, MobileNet has to do about 9 times less work than comparable neural nets with the same accuracy.

## MobileNet version 2

[MobileNet V2](https://arxiv.org/abs/1801.04381) still uses depthwise separable convolutions, its main building block(see v1 structure)

This time there are three convolutional layers in the block. The last two are the ones we already know: a depthwise convolution that filters the inputs, followed by a 1×1 pointwise convolution layer. However, this 1×1 layer now has a different job.

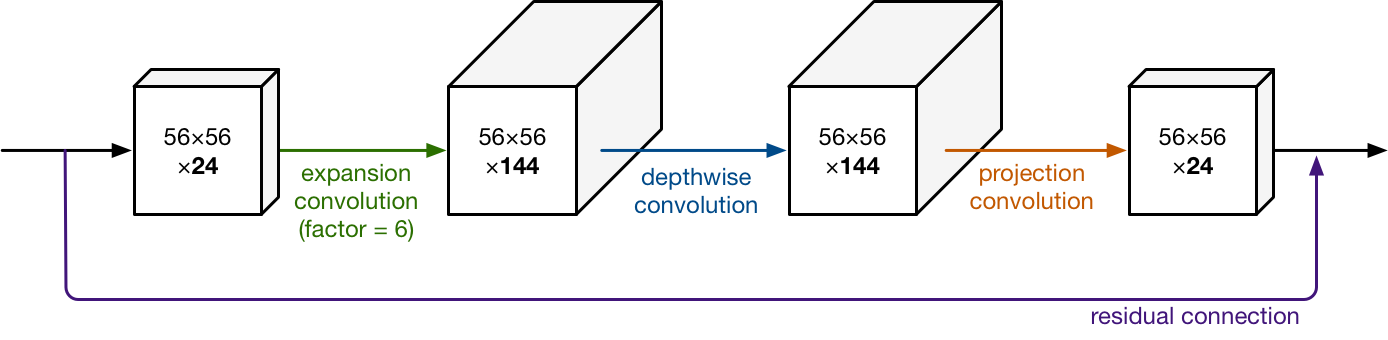
In V1 the pointwise convolution either kept the number of channels the same or doubled them. In V2 it does the opposite: it makes the number of channels smaller. This is why this layer is now known as the **projection layer** — it projects data with a high number of dimensions (channels) into a tensor with a much lower number of dimensions.

For example, the depthwise layer may work on a tensor with 144 channels, which the projection layer will then shrink down to only 24 channels. This kind of layer is also called a **bottleneck layer** because it reduces the amount of data that flows through the network. (This is where the “bottleneck residual block” gets its name from: the output of each block is a bottleneck.)

The first layer is the new kid in the block. This is also a 1×1 convolution. Its purpose is to expand the number of channels in the data before it goes into the depthwise convolution. Hence, this **expansion layer** always has more output channels than input channels — it pretty much does the opposite of the projection layer.

Exactly by how much the data gets expanded is given by the **expansion factor**. This is one of those hyper parameters for experimenting with different architecture trade-offs. The default expansion factor is 6.

For example, if there is a tensor with 24 channels going into a block, the expansion layer first converts this into a new tensor with 24 \* 6 = 144 channels. Next, the depth-wise convolution applies its filters to that 144-channel tensor. And finally, the projection layer projects the 144 filtered channels back to a smaller number, say 24 again.



So the input and the output of the block are low-dimensional tensors, while the filtering step that happens inside block is done on a high-dimensional tensor.

The second new thing in MobileNet V2’s building block is the **residual connection**. This works just like in ResNet and exists to help with the flow of gradients through the network. (The residual connection is only used when the number of channels going into the block is the same as the number of channels coming out of it, which is not always the case as every few blocks the output channels are increased.)

Each layer has batch normalization and the activation function is ReLU6. However, the output of the projection layer does not have an activation function applied to it. Since this layer produces low-dimensional data, the authors of the paper found that using a non-linearity after this layer actually destroyed useful information.

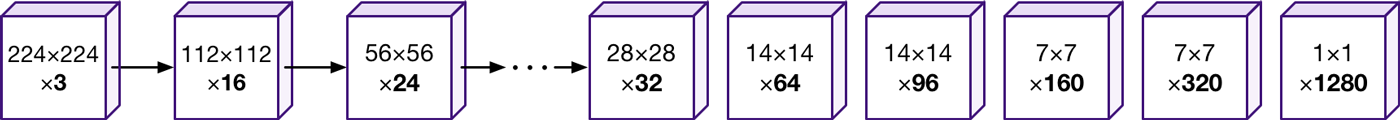
The full MobileNet V2 architecture, then, consists of 17 of these building blocks in a row. This is followed by a regular 1×1 convolution, a global average pooling layer, and a classification layer. (Small detail: the very first block is slightly different, it uses a regular 3×3 convolution with 32 channels instead of the expansion layer.)

## Motivation for changes un version 2

Why did the authors of MobileNet V2 make these choices?

**The idea behind V1 was to replace expensive convolutions with cheaper ones, even if it meant using more layers. That was a great success.** The main changes in the V2 architecture are the residual connections and the expand/projection layers.

If we look at the data as it flows through the network, notice how the number of channels stays fairly small between the blocks:

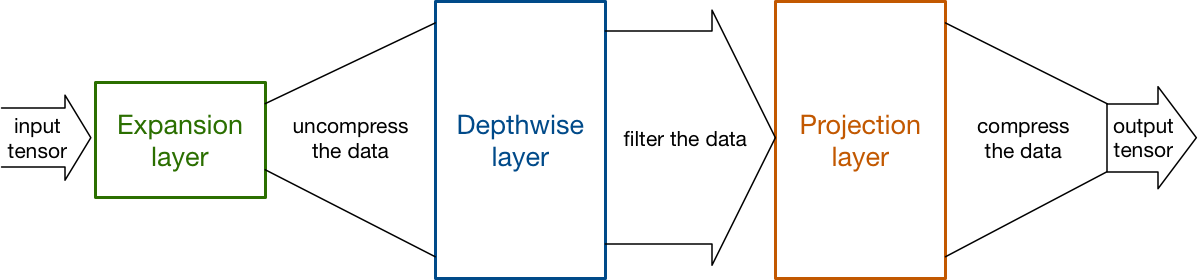


As is usual for this kind of model, the number of channels is increased over time (and the spatial dimensions cut in half). But overall, the tensors remain relatively small, thanks to the bottleneck layers that make up the connections between the blocks. Compared to this, V1 lets its tensors become much larger (up to 7×7×1024).

**Using low-dimension tensors is the key to reducing the number of computations. After all, the smaller the tensor, the fewer multiplications the convolutional layers have to do.**

However, *only* using low-dimensional tensors doesn’t work very well. Applying a convolutional layer to filter a low-dimensional tensor won’t be able to extract a whole lot of information. So to filter the data we ideally want to work with large tensors. MobileNet V2’s block design gives us the best of both worlds.

Think of the low-dimensional data that flows between the blocks as being a compressed version of the real data. In order to run filters over this data, we need to uncompress it first. That’s what happens inside each block:

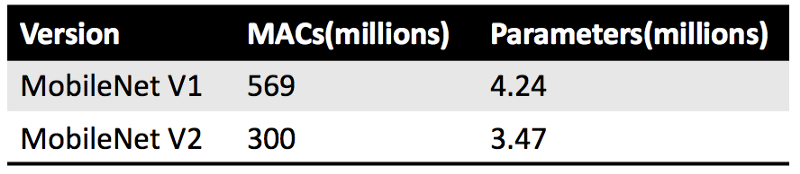


The expansion layer acts as a decompressor (like unzip) that first restores the data to its full form, then the depthwise layer performs whatever filtering is important at this stage of the network, and finally the projection layer compresses the data to make it small again.

The trick that makes this all work, of course, is that the expansions and projections are done using convolutional layers with learnable parameters, and so the model is able to learn how to best (de)compress the data at each stage in the network.

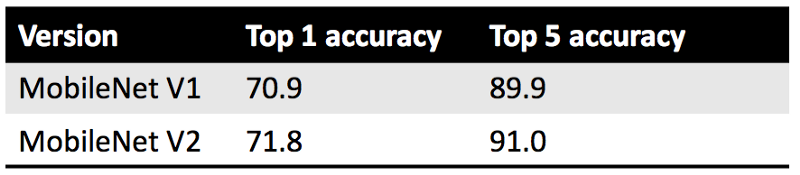
# Comparison of MobileNet Versions

MobileNet V2 is mostly an updated version of V1 that makes it even more efficient and powerful in terms of performance. Note: Lower is better



MACs are [multiply-accumulate operations](https://www.semanticscholar.org/topic/Multiply%E2%80%93accumulate-operation/408575), which measure how many calculations are needed to perform inference on a single 224×224 RGB image. (The larger the image, the more MACs are needed.)

From the number of MACs alone, V2 should be almost twice as fast as V1. However, it’s not just about the number of calculations. On mobile devices, [memory access](https://heartbeat.fritz.ai/profiling-your-app-with-android-studio-7accc268cb98) is much slower than computation. V2 only has 80% of the parameter count that V1 has hence making it better than V1. By seeing the result we can assume that V2 is almost twice as fast as V1 model.



In terms of accuracy, the reported top-1 and top-5 accuracy are on the [ImageNet classification dataset](http://www.image-net.org/challenges/LSVRC/2012/). (The [source](https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet) for these numbers claims they’re from the test set but looking at the code it appears to be the 50,000-image validation set.) To get the above numbers, the central region of the image was cropped to an area containing 87.5% of the original image, and then that crop was resized to 224×224 pixels. Only a single crop was used per image.

Conclusion: In all of these metrics, here MobileNet V2 is slightly, if not significantly, better than V1.

# Reference :

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

<https://arxiv.org/abs/1704.04861>

transfer learning

<https://towardsdatascience.com/transfer-learning-using-mobilenet-and-keras-c75daf7ff299>

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

<https://joshua19881228.github.io/2017-07-19-MobileNet/>

[Bag of Tricks for Image Classification with Convolutional Neural Networks](https://paperswithcode.com/paper/bag-of-tricks-for-image-classification-w2)