MobileNets:
Efficient Convolutional Neural Networks for Mobile Vision Applications

MobileNets

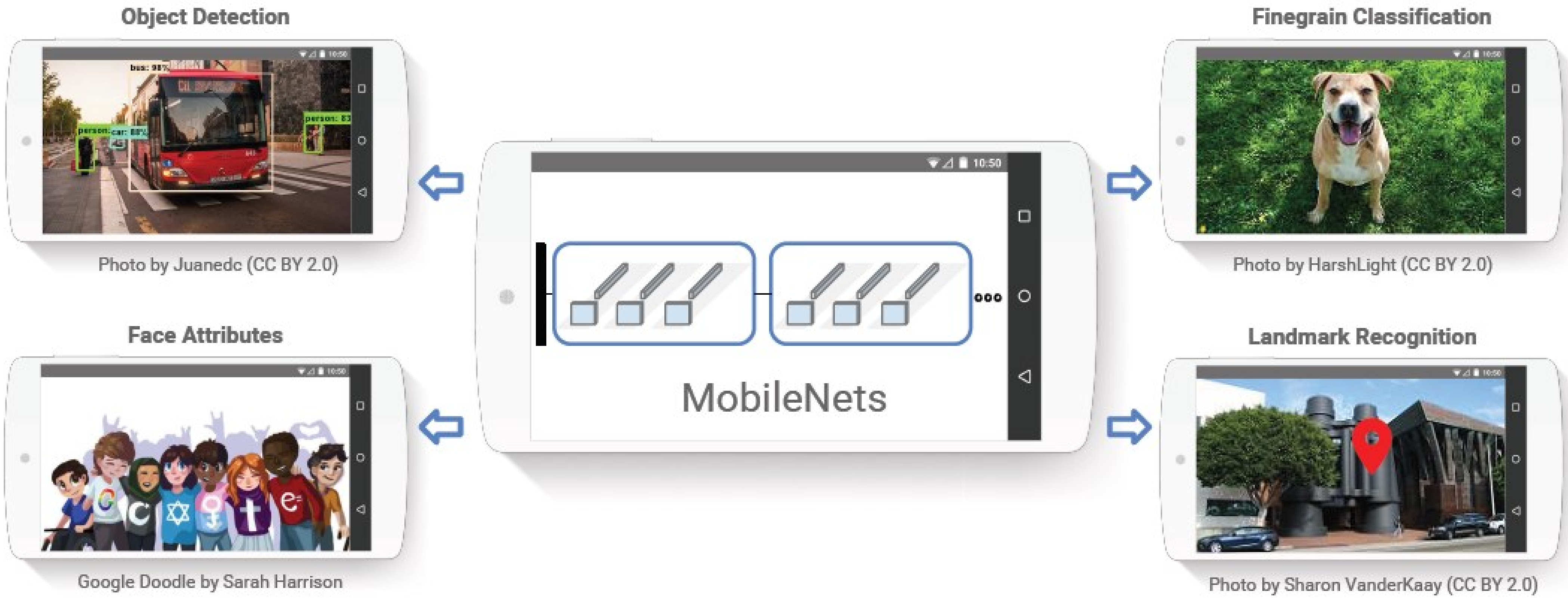


Figure 1. MobileNet models can be applied to various recognition tasks for efficient on device intelligence.

Key Requirements for Commercial Computer Vision Usage

- Data-centers(Clouds)
  - Rarely safety-critical
  - Low power is nice to have
  - Real-time is preferable
- Gadgets – Smartphones, Self-driving cars, Drones, etc.
  - Usually safety-critical(except smartphones)
  - Low power is must-have
  - Real-time is required

What's the "Right" Neural Network for Use in a Gadget?

- Desirable Properties
  - Sufficiently high accuracy
  - Low computational complexity
  - Low energy usage
  - Small model size

Why Small Deep Neural Networks?

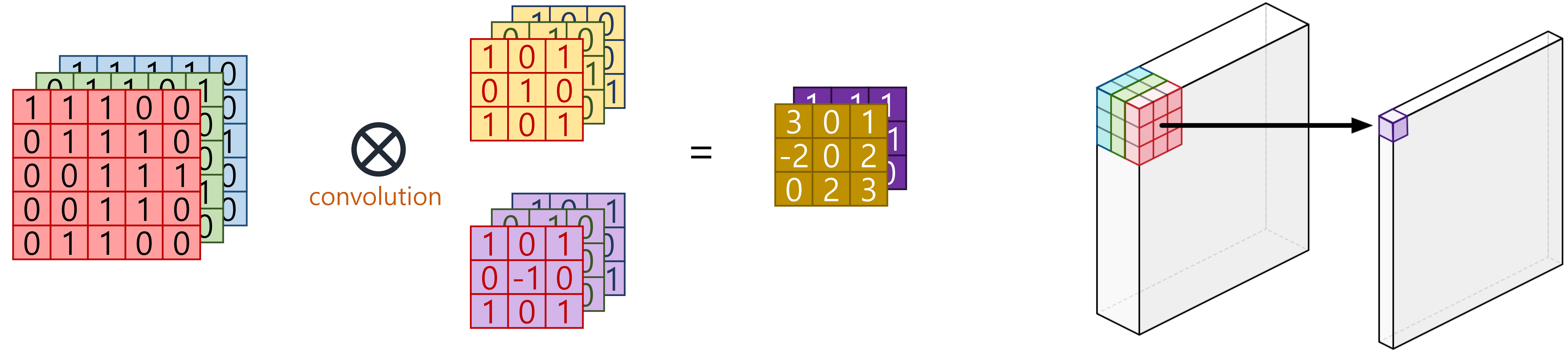
- Small DNNs train faster on distributed hardware
- Small DNNs are more deployable on embedded processors
- Small DNNs are easily updatable Over-The-Air(OTA)

Techniques for Small Deep Neural Networks

- Remove Fully-Connected Layers
- Kernel Reduction (3x3 → 1x1)
- Channel Reduction
- Evenly Spaced Downsampling
- Depthwise Separable Convolutions
- Shuffle Operations
- Distillation & Compression

Key Idea : Depthwise Separable Convolution!

Recap – Convolution Operation

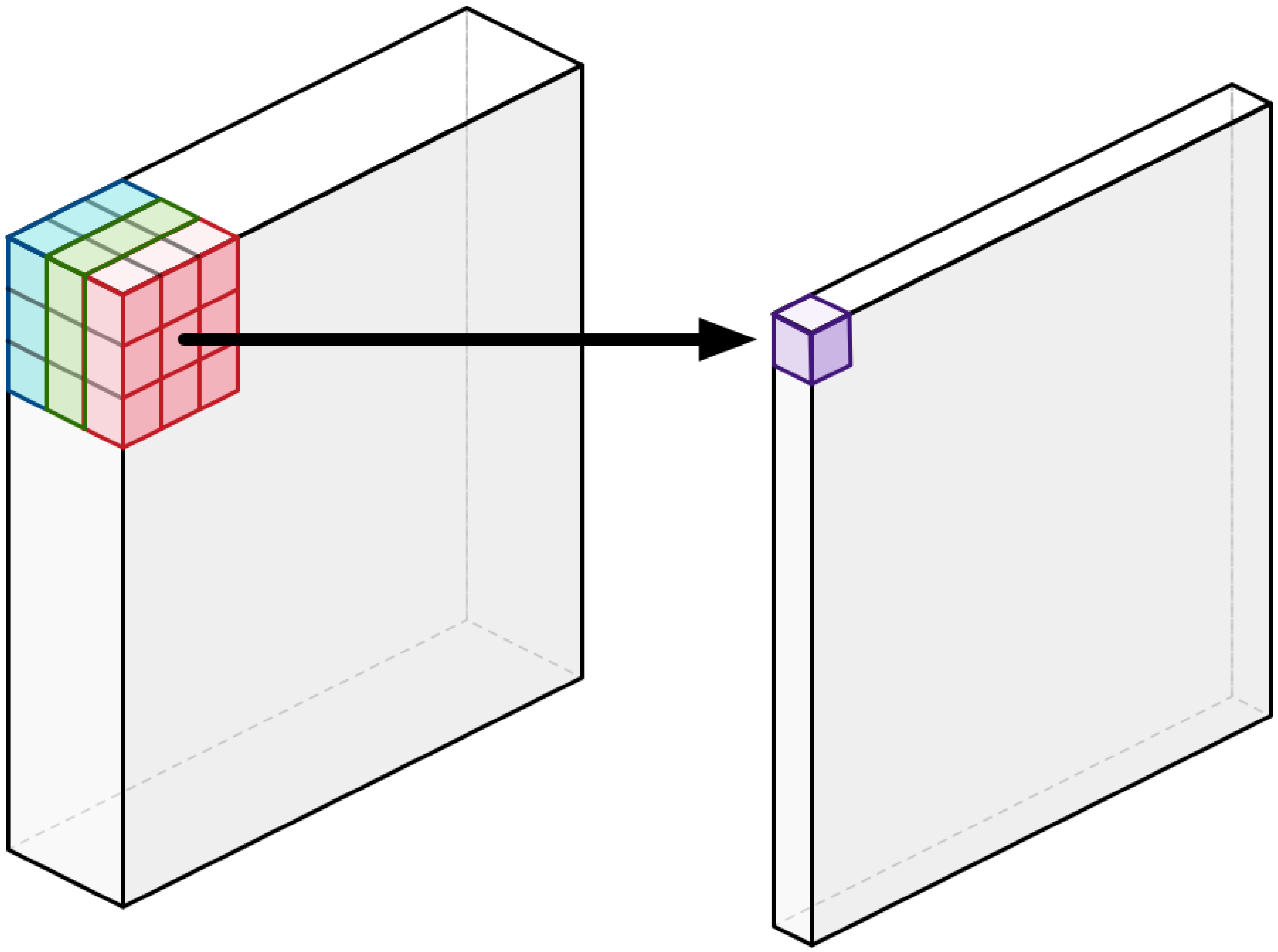


Input channel : 3

# of filters : 2

Output channel : 2

Standard Convolution

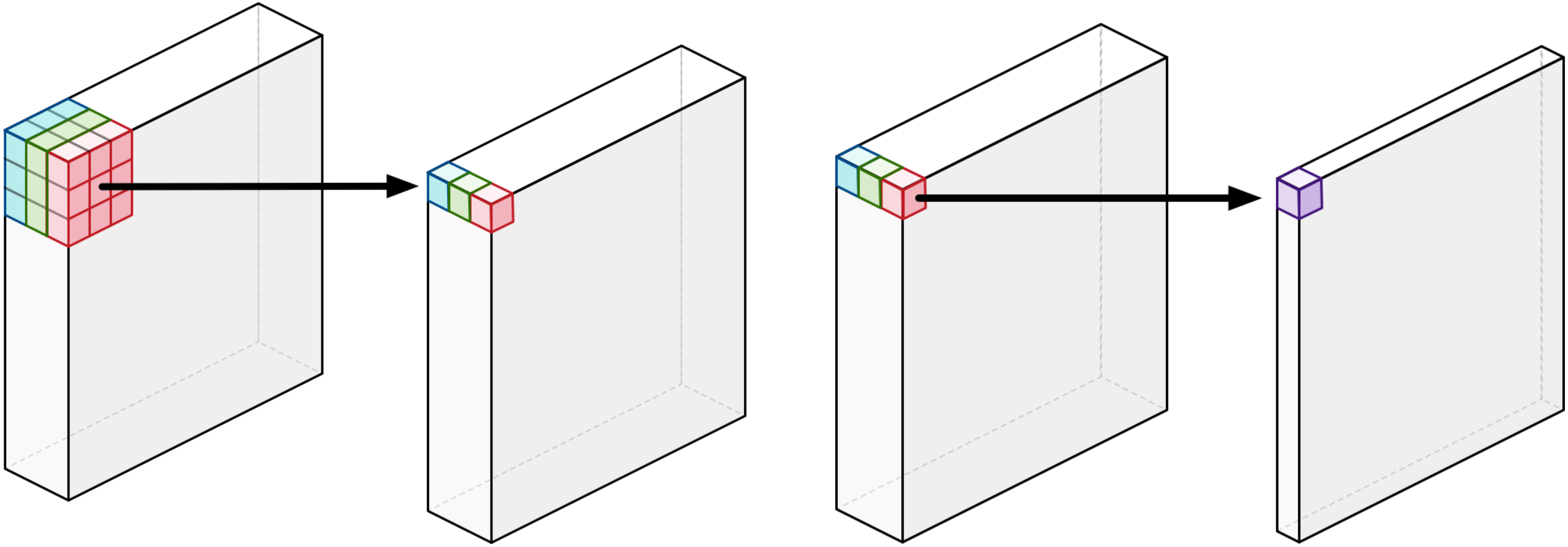


Depthwise convolution

Figures from http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/

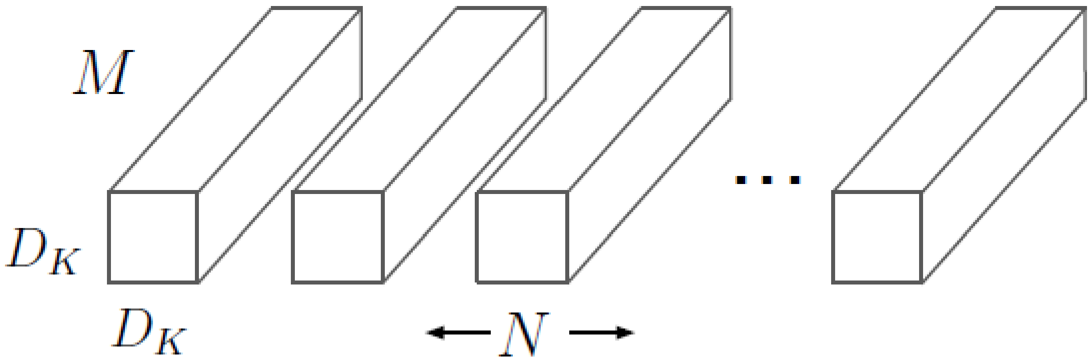
Depthwise Separable Convolution

- Depthwise Convolution + Pointwise Convolution(1x1 convolution)

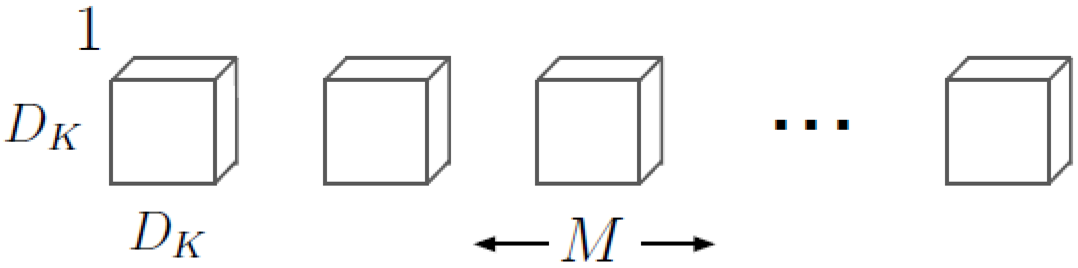


Depthwise convolution **Pointwise convolution**

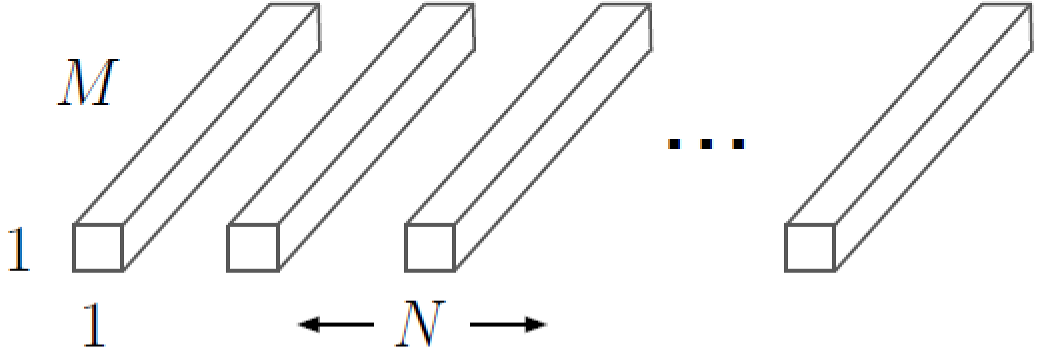
Standard Convolution vs Depthwise Separable Convolution



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



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

Standard Convolution vs Depthwise Separable Convolution

- Standard convolutions have the computational cost of
  - $D_K \times D_K \times M \times N \times D_F \times D_F$
- Depthwise separable convolutions cost
  - $D_K \times D_K \times M \times D_F \times D_F + M \times N \times D_F \times D_F$
- Reduction in computations
  - $1 / N + 1 / D_K^2$
  - If we use 3x3 depthwise separable convolutions, we get between 8 to 9 times less computations

D_K : width/height of filters
D_F : width/height of feature maps
M : number of input channels
N : number of output channels(number of filters)

Comparison Standard Vs. Depthwise

$$\frac{No.Mults\ in\ Depthwise\ Separable\ Conv}{No.Mults\ in\ Standard\ Conv} = \frac{M \times D_G^2\ (\ D_K^2 + N)}{N\ \times D_G \times D_G \times D_K \times D_K \times M}$$

$$\frac{No.\ Mults\ in\ Depthwise\ Separable\ Conv}{No.\ Mults\ in\ Standard\ Conv} = \frac{D_K^2 + N}{(\ D_K^2 \times N)} = \frac{1}{N} + \frac{1}{D_K^2}$$

Depthwise Separable Convolutions

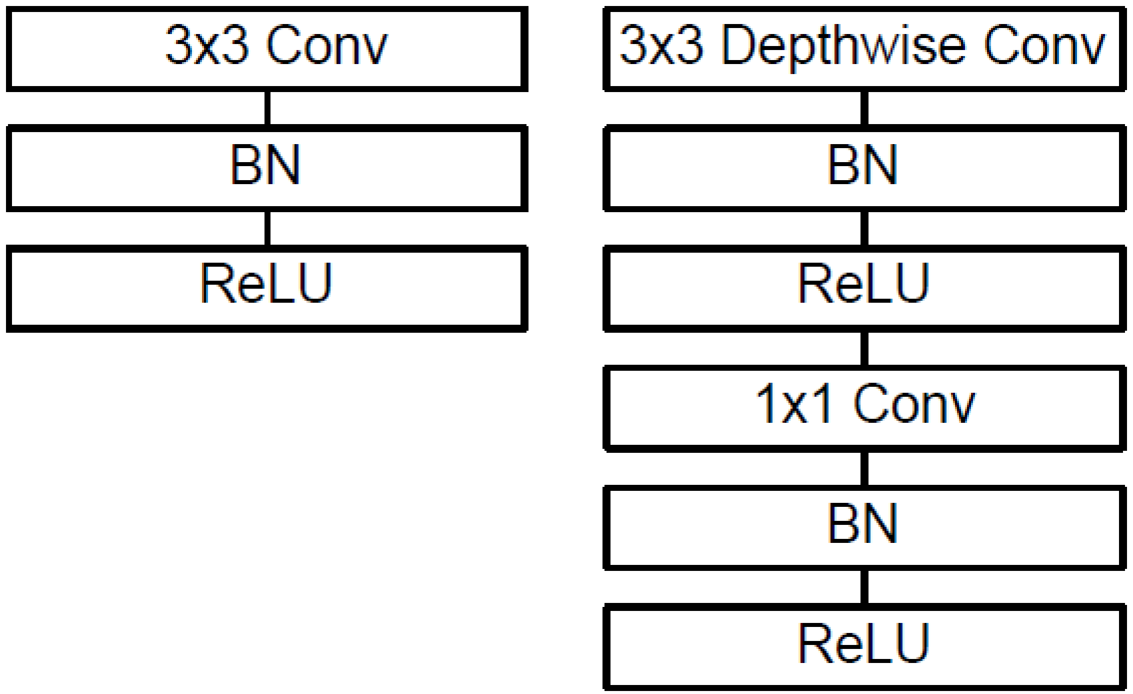


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Model Structure

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	$1 \times 1 \times 1000$

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%

Width Multiplier & Resolution Multiplier

- Width Multiplier – Thinner Models
  - 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 – where α with typical settings of 1, 0.75, 0.6 and 0.25
- Resolution Multiplier – Reduced Representation
  - The second hyper-parameter to reduce the computational cost of a neural network is a resolution multiplier ρ
  - $0<\rho\leq 1$, which is typically set of implicitly so that input resolution of network is 224, 192, 160 or 128($\rho = 1,\ 0.857, 0.714, 0.571$)
- Computational cost:

$$D_K \times D_K \times \alpha M \times \rho D_F \times \rho D_F + \alpha M \times \alpha N \times \rho D_F \times \rho D_F$$

Width Multiplier & Resolution Multiplier

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 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

Experiments – Model Choices

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Table 5. Narrow vs Shallow MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.75 MobileNet	68.4%	325	2.6
Shallow MobileNet	65.3%	307	2.9

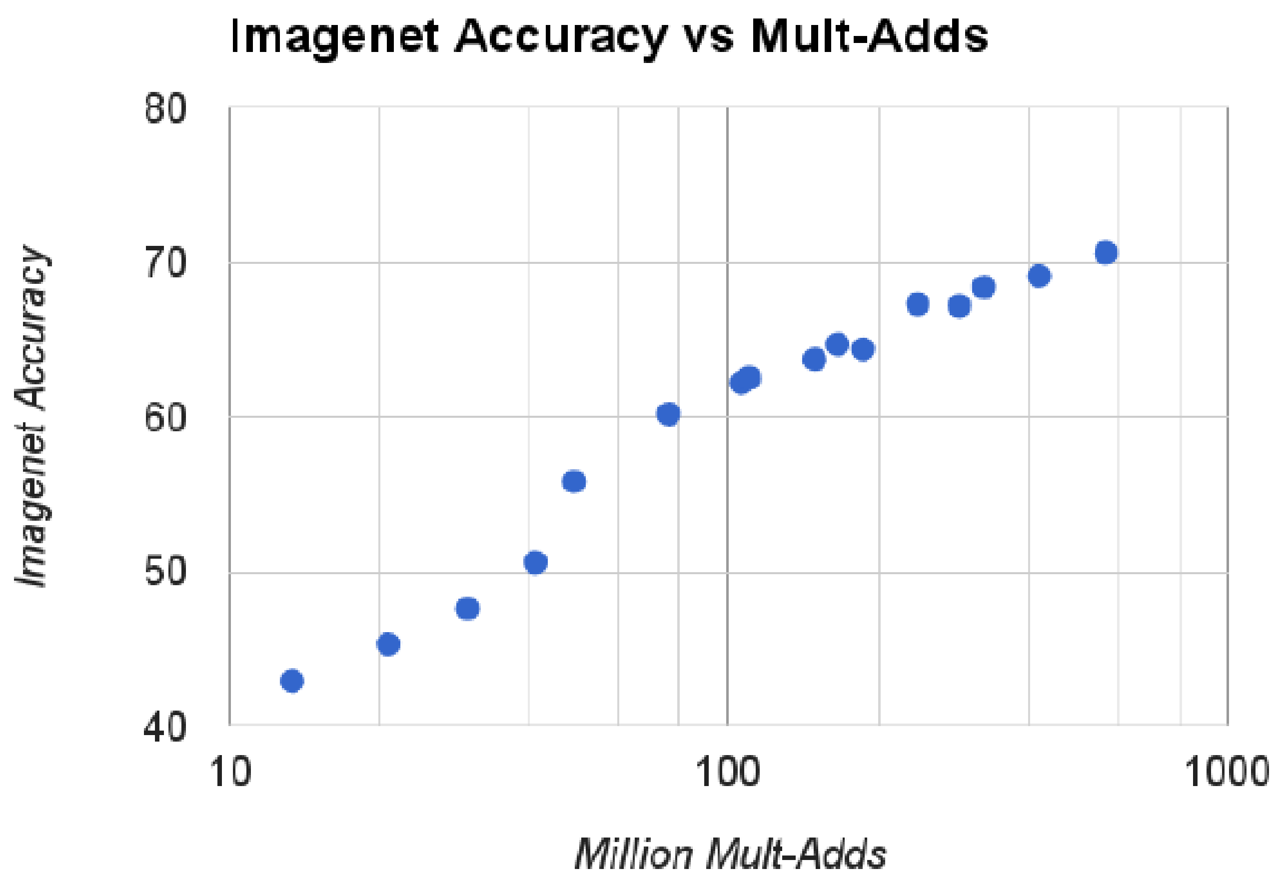
Table 6. MobileNet Width Multiplier

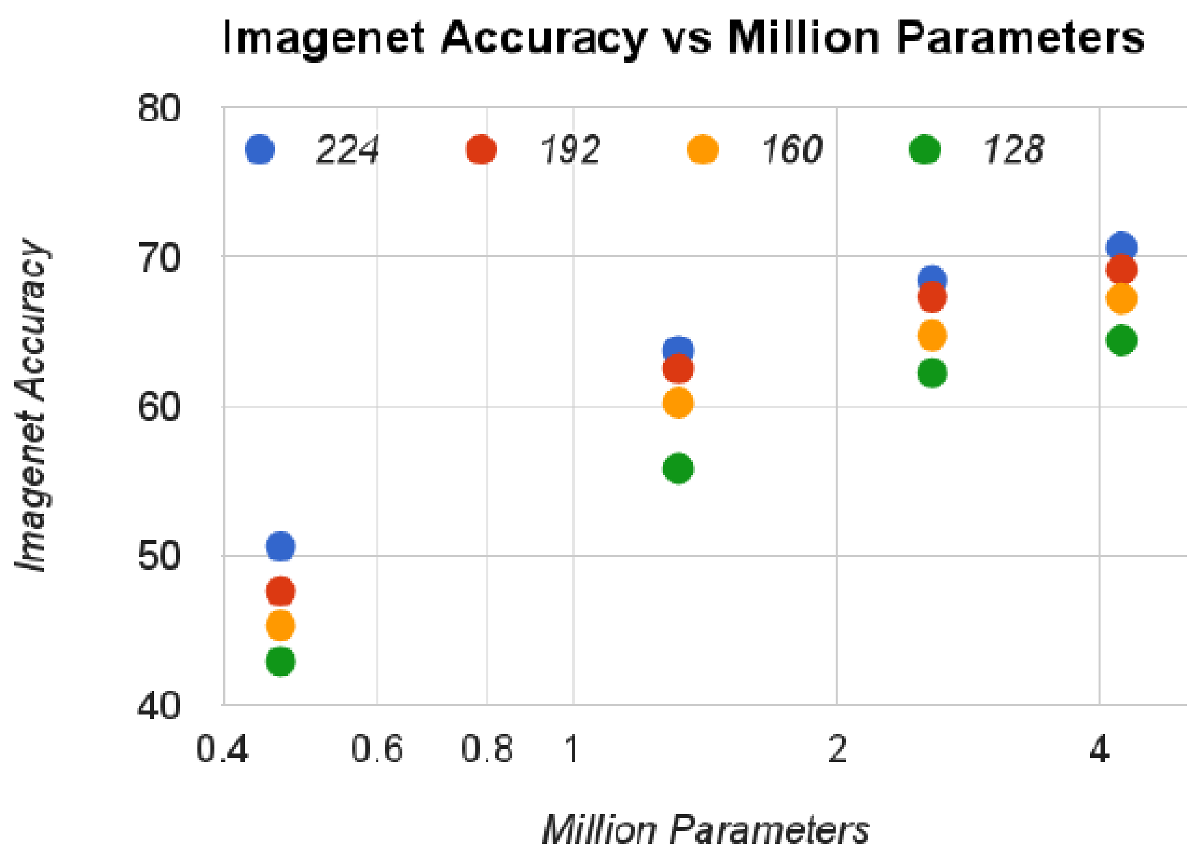
Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

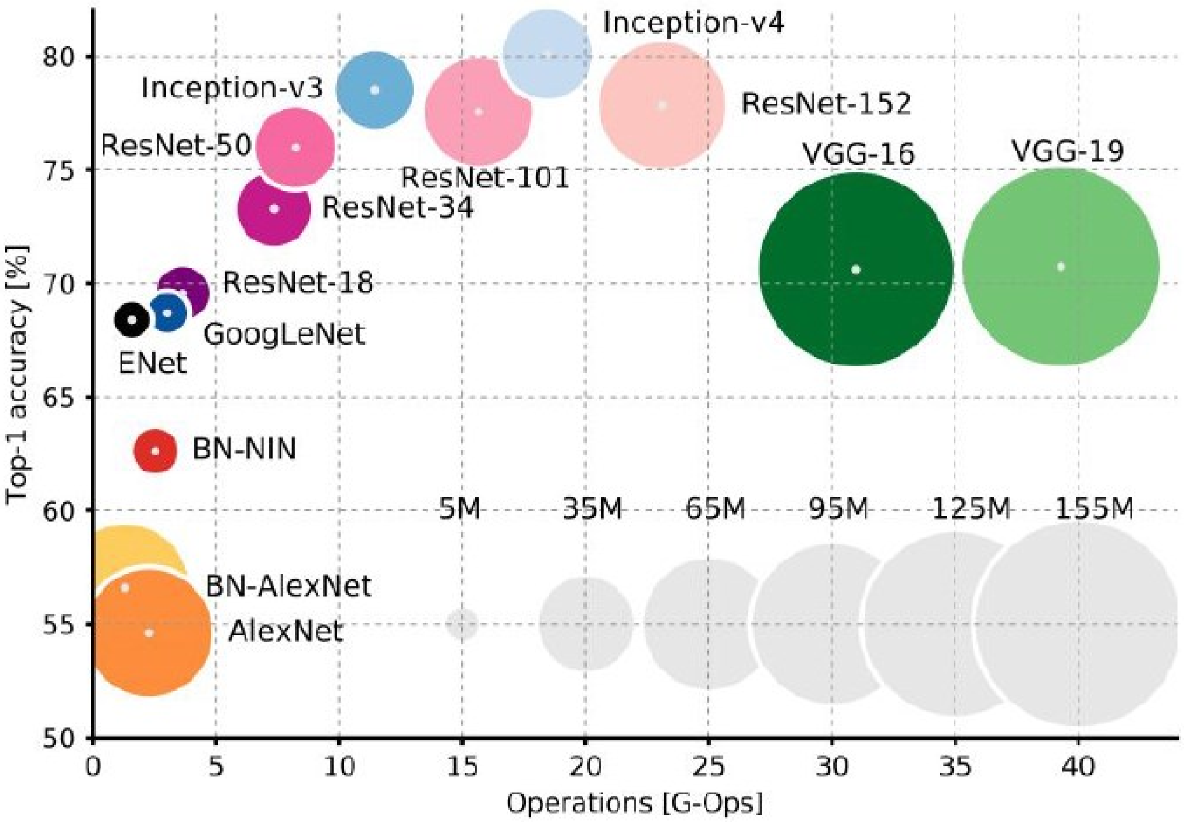
Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

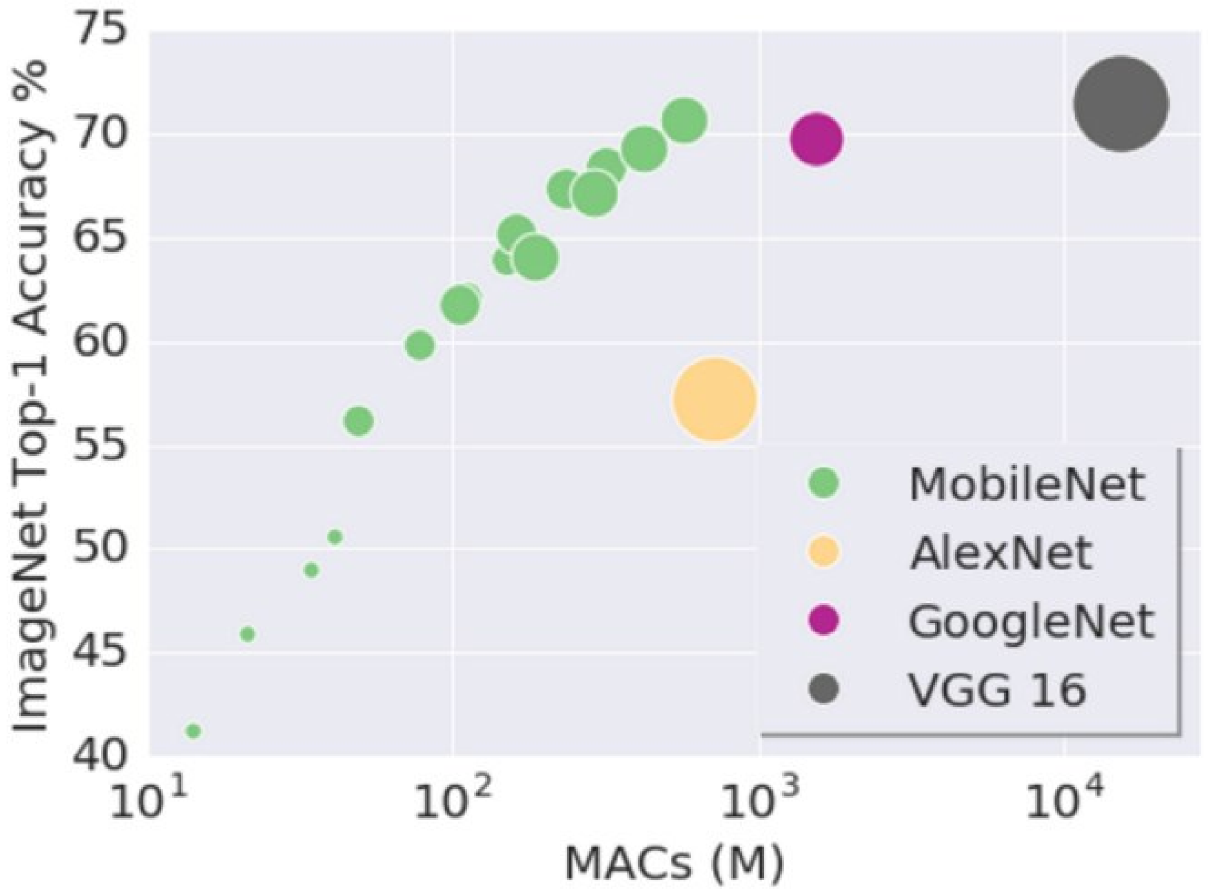
Model Shrinking Hyperparameters





Model Shrinking Hyperparameters





Results

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Table 9. Smaller MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

Results

Table 10. MobileNet for Stanford Dogs

Model	Top-1 Accuracy	Million Mult-Adds	Million Parameters
Inception V3 [18]	84%	5000	23.2
1.0 MobileNet-224	83.3%	569	3.3
0.75 MobileNet-224	81.9%	325	1.9
1.0 MobileNet-192	81.9%	418	3.3
0.75 MobileNet-192	80.5%	239	1.9

Table 11. Performance of PlaNet using the MobileNet architecture. Percentages are the fraction of the Im2GPS test dataset that were localized within a certain distance from the ground truth. The numbers for the original PlaNet model are based on an updated version that has an improved architecture and training dataset.

Scale	Im2GPS [7]	PlaNet [35]	PlaNet MobileNet
Continent (2500 km)	51.9%	77.6%	79.3%
Country (750 km)	35.4%	64.0%	60.3%
Region (200 km)	32.1%	51.1%	45.2%
City (25 km)	21.9%	31.7%	31.7%
Street (1 km)	2.5%	11.0%	11.4%

PlaNet : 52M parameters, 5.74B mult-adds
MobilNet : 13M parameters, 0.58M mult-adds

Results

Table 13. COCO object detection results comparison using different frameworks and network architectures. mAP is reported with COCO primary challenge metric (AP at IoU=0.50:0.05:0.95)

Framework Resolution	Model	mAP	Billion Mult-Adds	Million Parameters
SSD 300	deeplab-VGG	21.1%	34.9	33.1
	Inception V2	22.0%	3.8	13.7
	MobileNet	19.3%	1.2	6.8
Faster-RCNN 300	VGG	22.9%	64.3	138.5
	Inception V2	15.4%	118.2	13.3
	MobileNet	16.4%	25.2	6.1
Faster-RCNN 600	VGG	25.7%	149.6	138.5
	Inception V2	21.9%	129.6	13.3
	Mobilenet	19.8%	30.5	6.1

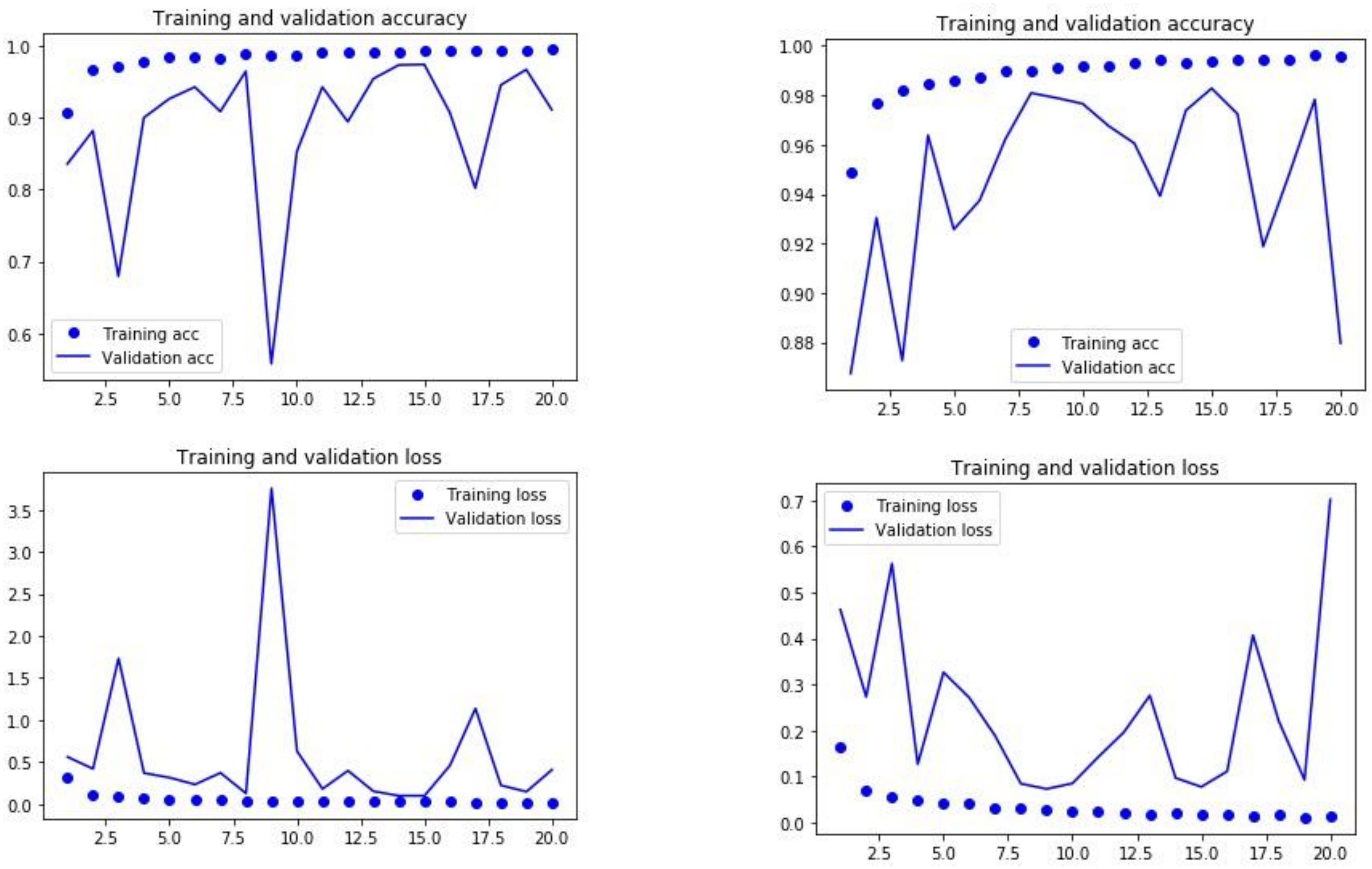
Figure 6. Example objection detection results using MobileNet SSD.

Tensorflow Implementation

```
def _depthwise_separable_conv(inputs,
                              num_pwc_filters,
                              width_multiplier,
                              sc,
                              downsample=False):
  """ Helper function to build the depth-wise separable convolution layer.
  """
  num_pwc_filters = round(num_pwc_filters * width_multiplier)
  _stride = 2 if downsample else 1

  # skip pointwise by setting num_outputs=None
  depthwise_conv = slim.separable_convolution2d(inputs,
                                                num_outputs=None,
                                                stride=_stride,
                                                depth_multiplier=1,
                                                kernel_size=[3, 3],
                                                scope=sc+'/depthwise_conv')

  bn = slim.batch_norm(depthwise_conv, scope=sc+'/dw_batch_norm')
  pointwise_conv = slim.convolution2d(bn,
                                      num_pwc_filters,
                                      kernel_size=[1, 1],
                                      scope=sc+'/pointwise_conv')
  bn = slim.batch_norm(pointwise_conv, scope=sc+'/pw_batch_norm')
  return bn
```

```
with tf.variable_scope(scope) as sc:
  end_points_collection = sc.name + '_end_points'
  with slim.arg_scope([slim.convolution2d, slim.separable_convolution2d],
                      activation_fn=None,
                      outputs_collections=[end_points_collection]):
    with slim.arg_scope([slim.batch_norm],
                        is_training=is_training,
                        activation_fn=tf.nn.relu):
      net = slim.convolution2d(inputs, round(32 * width_multiplier), [3, 3], stride=2, padding='SAME', scope='conv_1')
      net = slim.batch_norm(net, scope='conv_1/batch_norm')
      net = _depthwise_separable_conv(net, 64, width_multiplier, sc='conv_ds_2')
      net = _depthwise_separable_conv(net, 128, width_multiplier, downsample=True, sc='conv_ds_3')
      net = _depthwise_separable_conv(net, 128, width_multiplier, sc='conv_ds_4')
      net = _depthwise_separable_conv(net, 256, width_multiplier, downsample=True, sc='conv_ds_5')
      net = _depthwise_separable_conv(net, 256, width_multiplier, sc='conv_ds_6')
      net = _depthwise_separable_conv(net, 512, width_multiplier, downsample=True, sc='conv_ds_7')

      net = _depthwise_separable_conv(net, 512, width_multiplier, sc='conv_ds_8')
      net = _depthwise_separable_conv(net, 512, width_multiplier, sc='conv_ds_9')
      net = _depthwise_separable_conv(net, 512, width_multiplier, sc='conv_ds_10')
      net = _depthwise_separable_conv(net, 512, width_multiplier, sc='conv_ds_11')
      net = _depthwise_separable_conv(net, 512, width_multiplier, sc='conv_ds_12')

      net = _depthwise_separable_conv(net, 1024, width_multiplier, downsample=True, sc='conv_ds_13')
      net = _depthwise_separable_conv(net, 1024, width_multiplier, sc='conv_ds_14')
      net = slim.avg_pool2d(net, [7, 7], scope='avg_pool_15')

  end_points = slim.utils.convert_collection_to_dict(end_points_collection)
  net = tf.squeeze(net, [1, 2], name='SpatialSqueeze')
  end_points['squeeze'] = net
  logits = slim.fully_connected(net, num_classes, activation_fn=None, scope='fc_16')
  predictions = slim.softmax(logits, scope='Predictions')

  end_points['Logits'] = logits
  end_points['Predictions'] = predictions
```

https://github.com/Zehaos/MobileNet/blob/master/nets/mobilenet.py


Training and validation accuracy
Training acc
Validation acc
Training and validation accuracy
Training acc
Validation acc
Training and validation loss
Training loss
Validation loss
Training and validation loss
Training loss
Validation loss