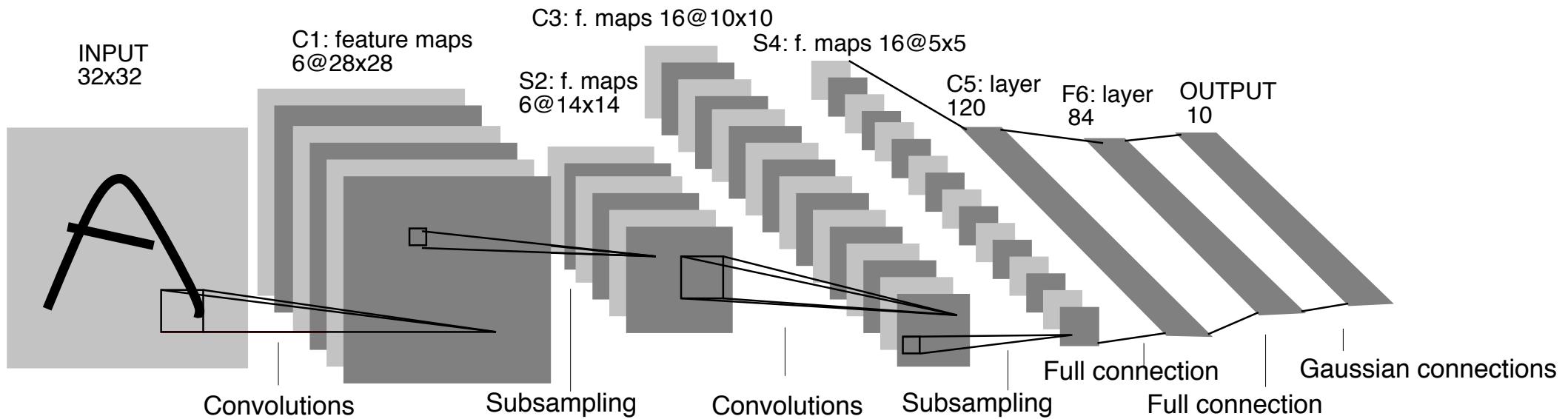


Common CNN Architectures

Shusen Wang

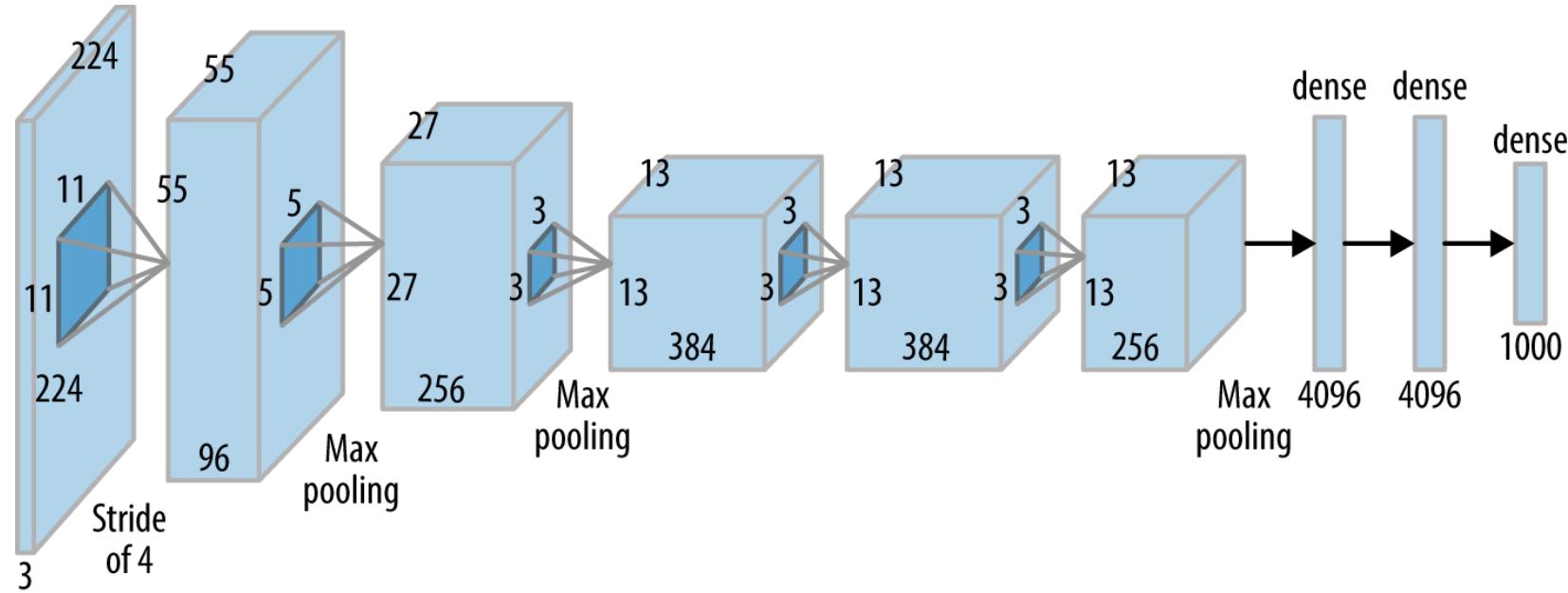
Classic CNN Architectures

LeNet-5 [Yan LeCun *et al.* 1998]



- Layers: 2 Conv + 2 FC layers.
- Paper: [Gradient-based learning applied to document recognition](#)

AlexNet [Alex Krizhevsky *et al.* 2012]



- Layers: 5 Conv + 3 FC layers.
- Number of trainable parameters: $60M$.
- Paper: [ImageNet Classification with Deep Convolutional Neural Networks](#)
- Keras implementation: <http://dandxy89.github.io/ImageModels/>

```

DROPOUT = 0.5
model_input = Input(shape = (227, 227, 3))

# First convolutional Layer (96x11x11)
z = Conv2D(filters = 96, kernel_size = (11,11), strides = (4,4), activation = "relu")(model_input)
z = MaxPooling2D(pool_size = (3,3), strides=(2,2))(z)
z = BatchNormalization()(z)

# Second convolutional Layer (256x5x5)
z = ZeroPadding2D(padding = (2,2))(z)
z = Convolution2D(filters = 256, kernel_size = (5,5), strides = (1,1), activation = "relu")(z)
z = MaxPooling2D(pool_size = (3,3), strides=(2,2))(z)
z = BatchNormalization()(z)

# Rest 3 convolutional layers
z = ZeroPadding2D(padding = (1,1))(z)
z = Convolution2D(filters = 384, kernel_size = (3,3), strides = (1,1), activation = "relu")(z)

z = ZeroPadding2D(padding = (1,1))(z)
z = Convolution2D(filters = 384, kernel_size = (3,3), strides = (1,1), activation = "relu")(z)

z = ZeroPadding2D(padding = (1,1))(z)
z = Convolution2D(filters = 256, kernel_size = (3,3), strides = (1,1), activation = "relu")(z)

z = MaxPooling2D(pool_size = (3,3), strides=(2,2))(z)
z = Flatten()(z)

z = Dense(4096, activation="relu")(z)
z = Dropout(DROPOUT)(z)

z = Dense(4096, activation="relu")(z)
z = Dropout(DROPOUT)(z)

final_dim = 1 if N_CATEGORY == 2 else N_CATEGORY
final_act = "sigmoid" if N_CATEGORY == 2 else "softmax"
model_output = Dense(final_dim, activation=final_act)(z)

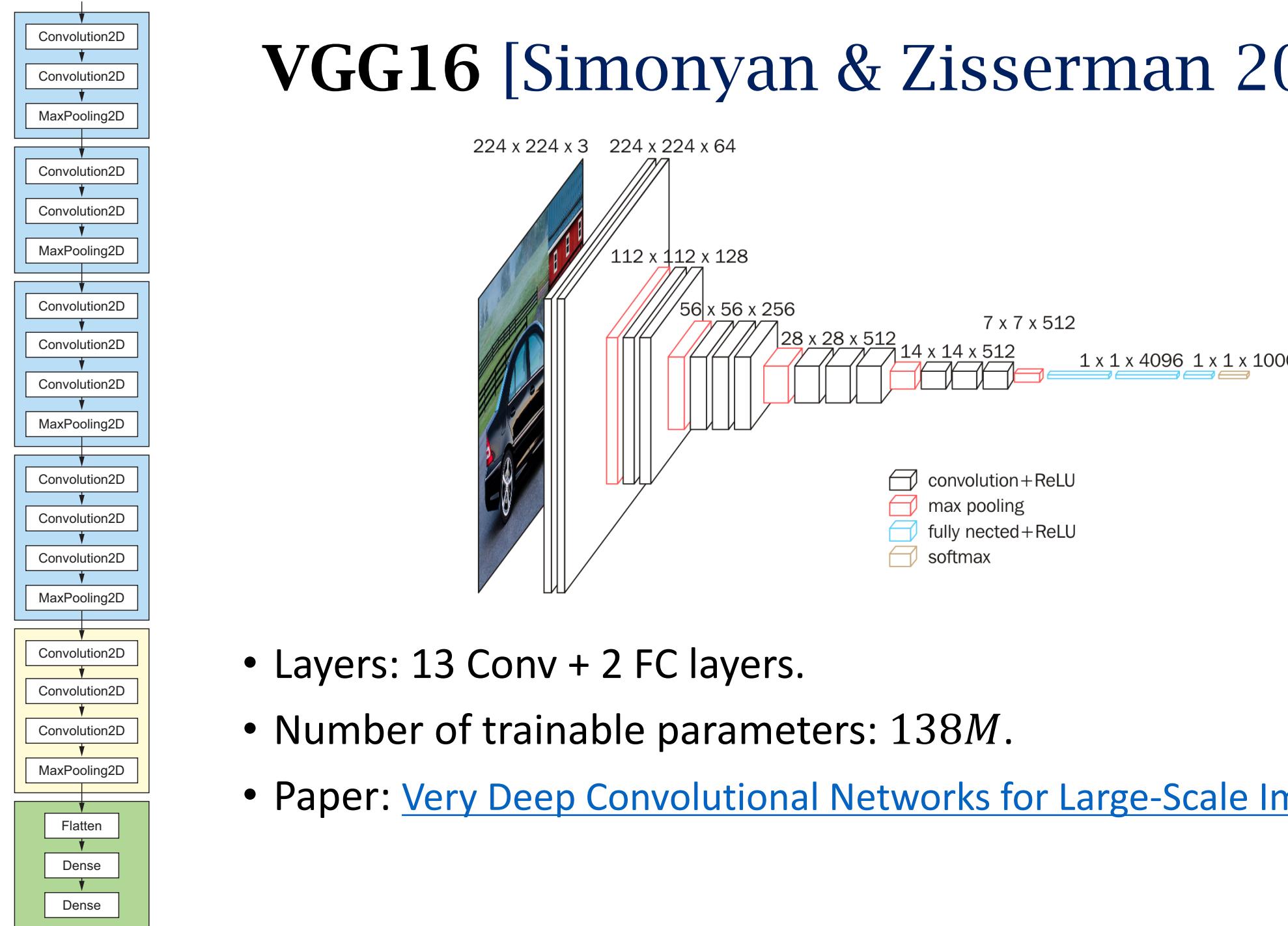
model = Model(model_input, model_output)
model.summary()

```

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 227, 227, 3)	0
conv2d_6 (Conv2D)	(None, 55, 55, 96)	34944
max_pooling2d_4 (MaxPooling2D)	(None, 27, 27, 96)	0
batch_normalization_3 (Batch Normalization)	(None, 27, 27, 96)	384
zero_padding2d_5 (ZeroPadding2D)	(None, 31, 31, 96)	0
conv2d_7 (Conv2D)	(None, 27, 27, 256)	614656
max_pooling2d_5 (MaxPooling2D)	(None, 13, 13, 256)	0
batch_normalization_4 (Batch Normalization)	(None, 13, 13, 256)	1024
zero_padding2d_6 (ZeroPadding2D)	(None, 15, 15, 256)	0
conv2d_8 (Conv2D)	(None, 13, 13, 384)	885120
zero_padding2d_7 (ZeroPadding2D)	(None, 15, 15, 384)	0
conv2d_9 (Conv2D)	(None, 13, 13, 384)	1327488
zero_padding2d_8 (ZeroPadding2D)	(None, 15, 15, 384)	0
conv2d_10 (Conv2D)	(None, 13, 13, 256)	884992
max_pooling2d_6 (MaxPooling2D)	(None, 6, 6, 256)	0
flatten_2 (Flatten)	(None, 9216)	0
dense_3 (Dense)	(None, 4096)	37752832
dropout_3 (Dropout)	(None, 4096)	0
dense_4 (Dense)	(None, 4096)	16781312
dropout_4 (Dropout)	(None, 4096)	0
dense_5 (Dense)	(None, 100)	409700

Total params: 58,692,452.0
Trainable params: 58,691,748.0
Non-trainable params: 704.0

VGG16 [Simonyan & Zisserman 2014]

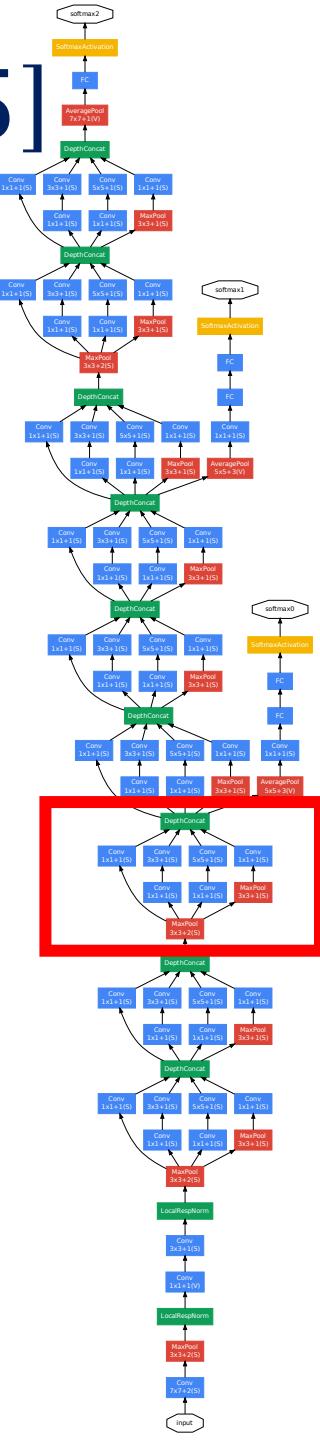


- Layers: 13 Conv + 2 FC layers.
- Number of trainable parameters: $138M$.
- Paper: [Very Deep Convolutional Networks for Large-Scale Image Recognition](#)

“Modern” CNN Architectures

GoogLeNet/Inception [Szegedy *et al.* 2015]

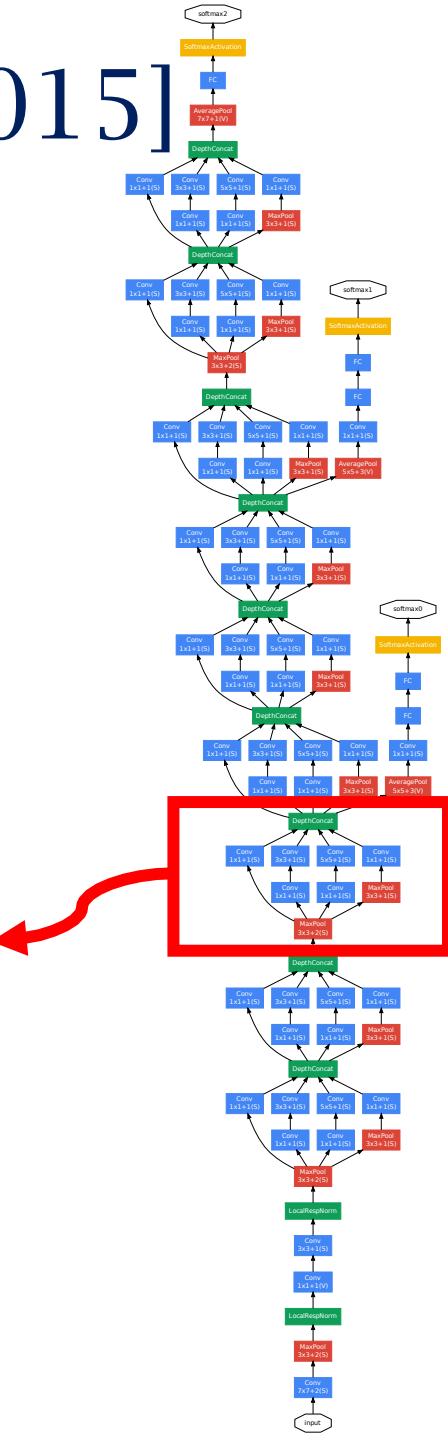
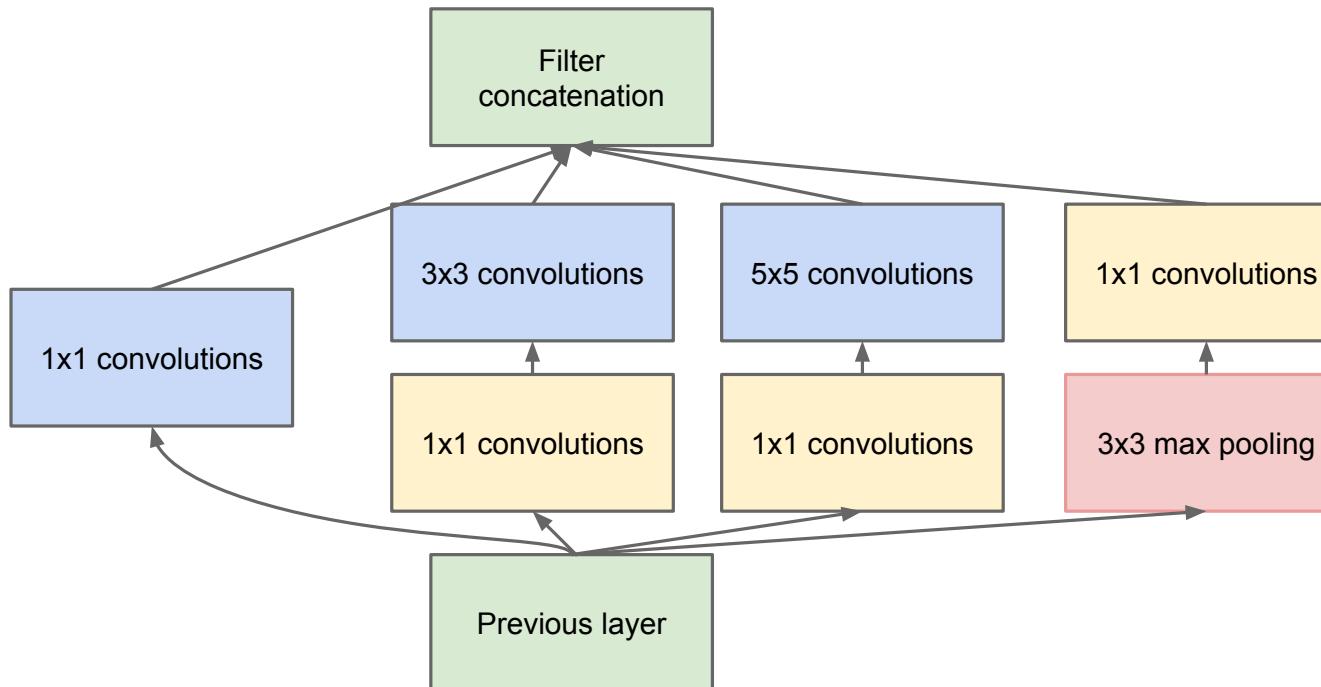
- Stack of the “**Inception**” modules.
- Only $5M$ parameters. (VGG16 has $138M$ parameters!)



GoogLeNet/Inception [Szegedy *et al.* 2015]

- Stack of the “**Inception**” modules.
- Only $5M$ parameters. (VGG16 has $138M$ parameters!)

An **inception module**



GoogLeNet/Inception [Szegedy *et al.* 2015]

```
from keras.layers import Input, Conv2D, MaxPooling2D, concatenate

x_input = Input(shape=(d1, d2, 64*4))

tower1 = Conv2D(64, (1,1), padding='same', activation='relu')(x_input)

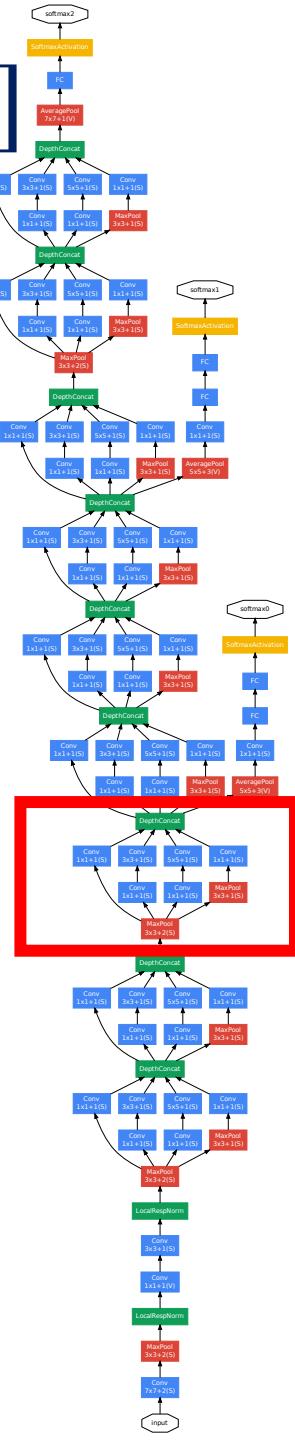
tower2 = Conv2D(64, (1,1), padding='same', activation='relu')(x_input)
tower2 = Conv2D(64, (3,3), padding='same', activation='relu')(tower2)

tower3 = Conv2D(64, (1,1), padding='same', activation='relu')(x_input)
tower3 = Conv2D(64, (5,5), padding='same', activation='relu')(tower3)

tower4 = MaxPooling2D((3,3), strides=(1,1), padding='same')(x_input)
tower4 = Conv2D(64, (1,1), padding='same', activation='relu')(tower4)

x_output = concatenate([tower1, tower2, tower3, tower4], axis = 3)
```

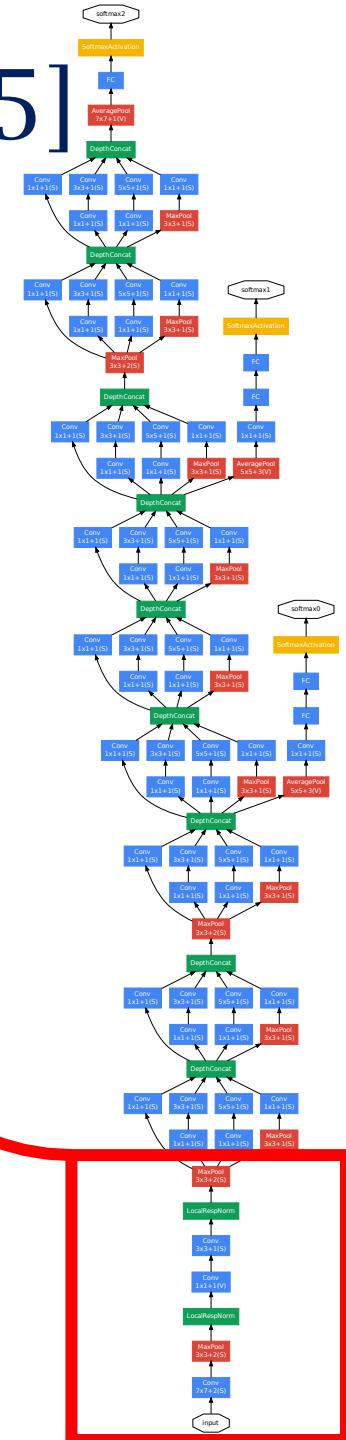
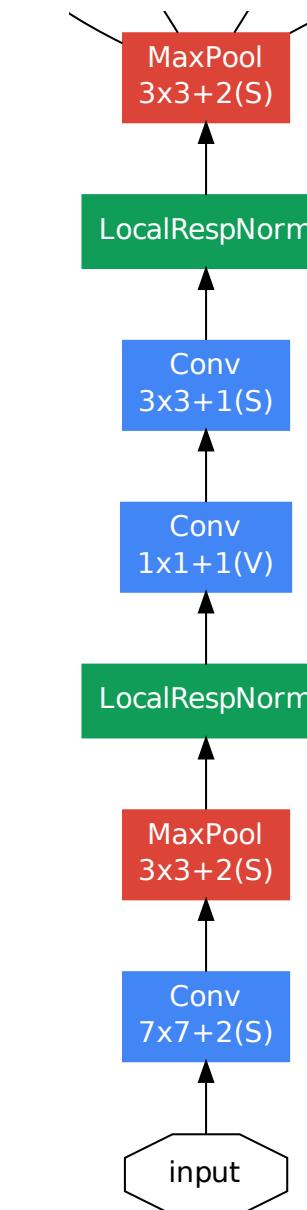
Output: $d_1 \times d_2 \times 256$ (the same as input)



GoogLeNet/Inception [Szegedy *et al.* 2015]

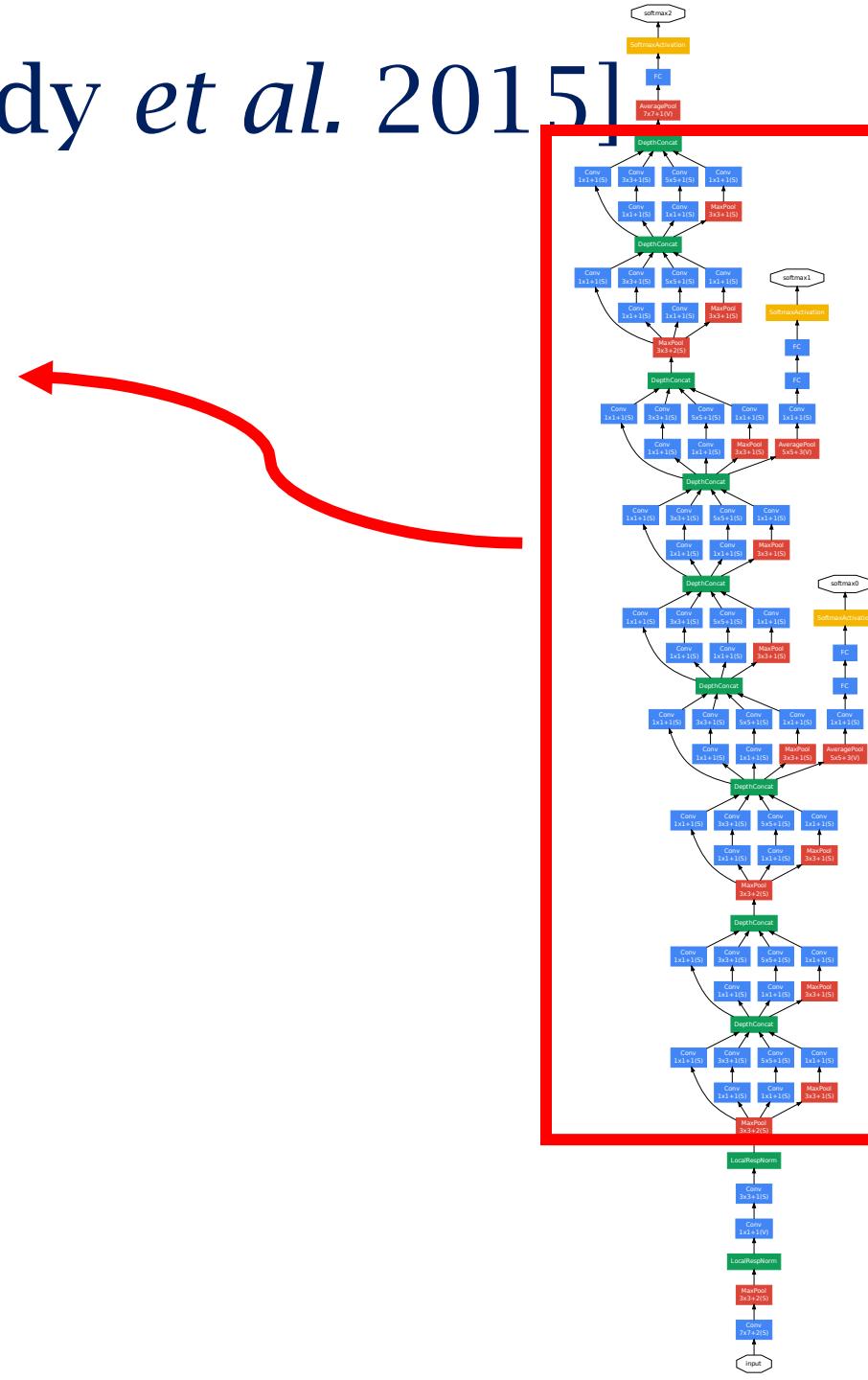
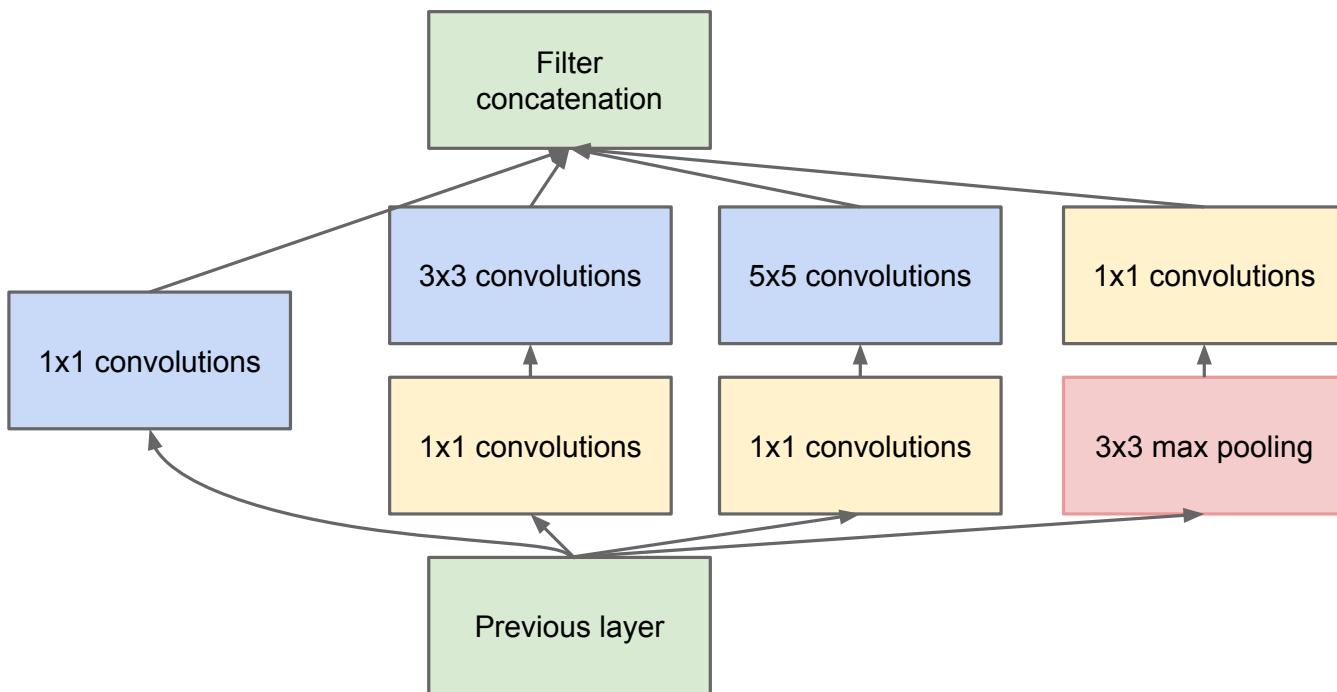
Bottom layers

- A simple ConvNet.
- 3 Conv + 2 Pooling + 2 Normalization Layers.



GoogLeNet/Inception [Szegedy *et al.* 2015]

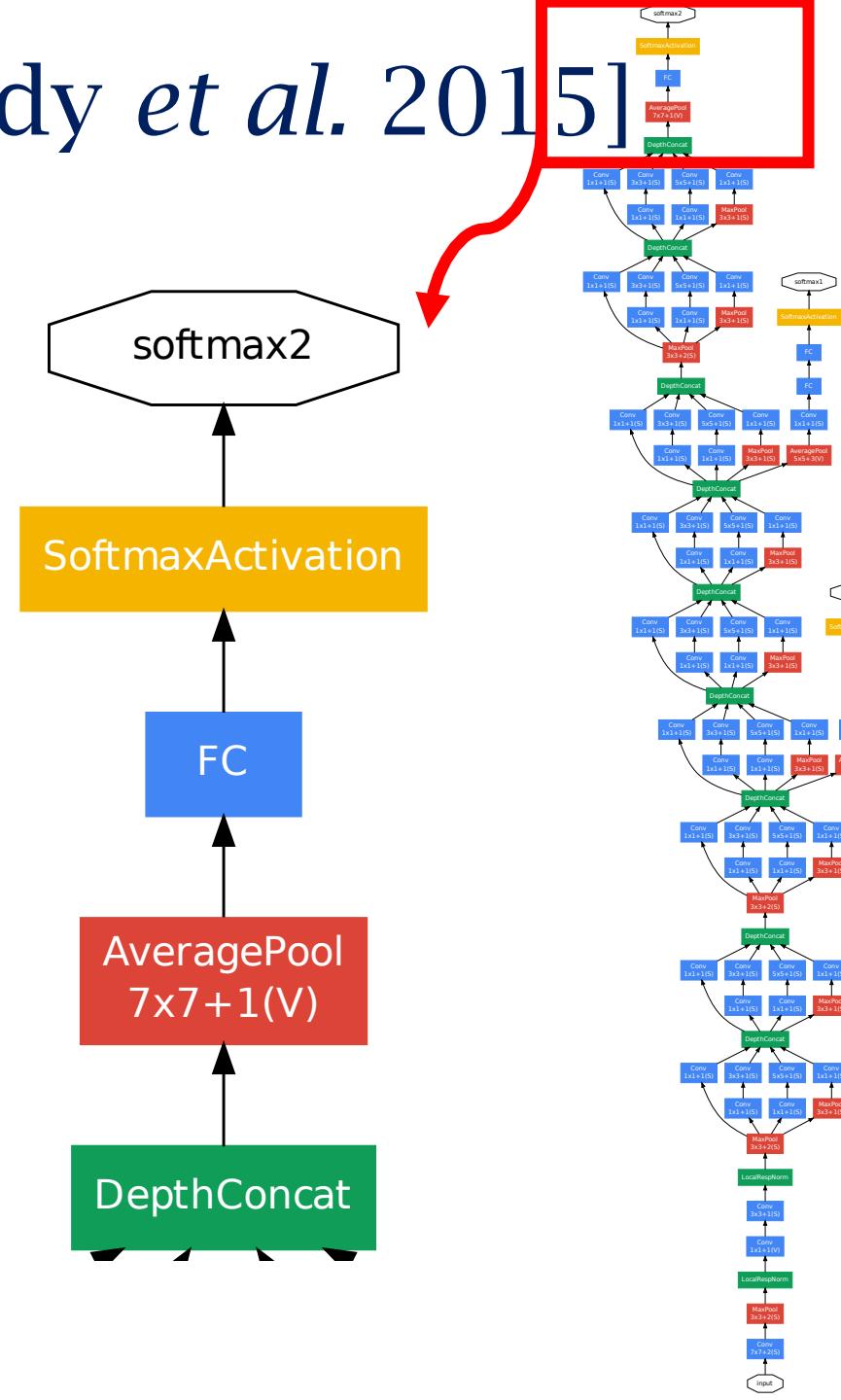
Stack of 9 Inception modules



GoogLeNet/Inception [Szegedy *et al.* 2015]

Output layers

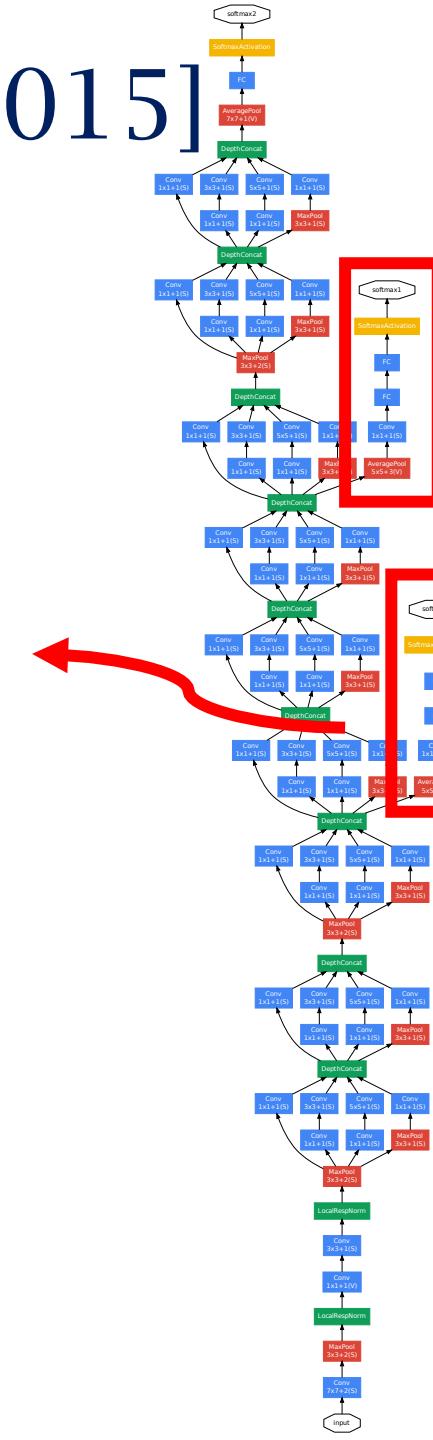
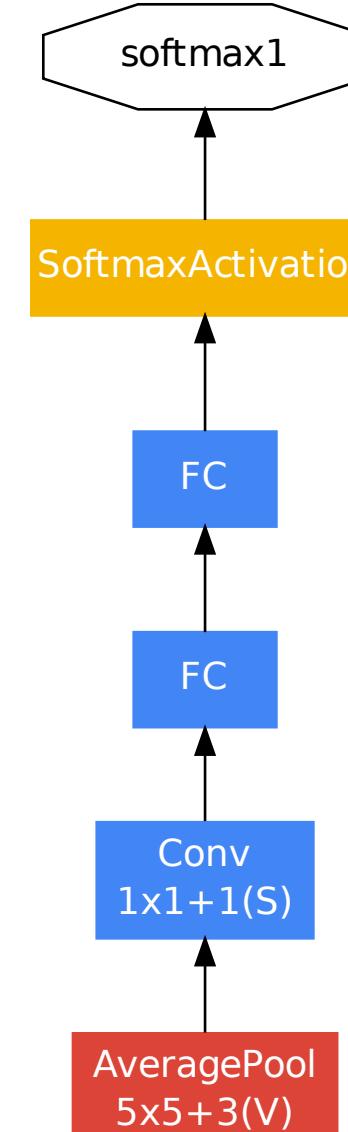
- AveragePool Layer for reducing the number of parameters.
- A Softmax classifier built on the AveragePool Layer.



GoogLeNet/Inception [Szegedy *et al.* 2015]



- Inject additional gradient at lower layers.
 - Make the optimization easier.



How Deep Can We Go?

- LeNet-5: 2 Conv + 2 FC layers.
 - AlexNet: 5 Conv + 3 FC layers.
 - VGG16: 13 Conv + 2 FC layers.
- 
- classic architectures (sequential)

Question: Why VGG16? Why not deeper classic architectures?

How Deep Can We Go?

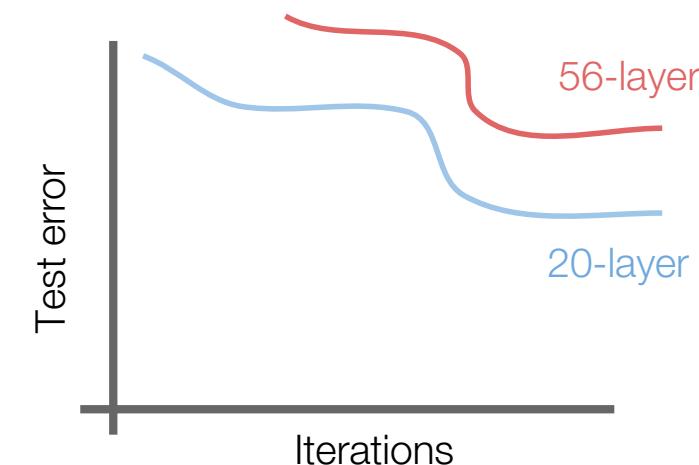
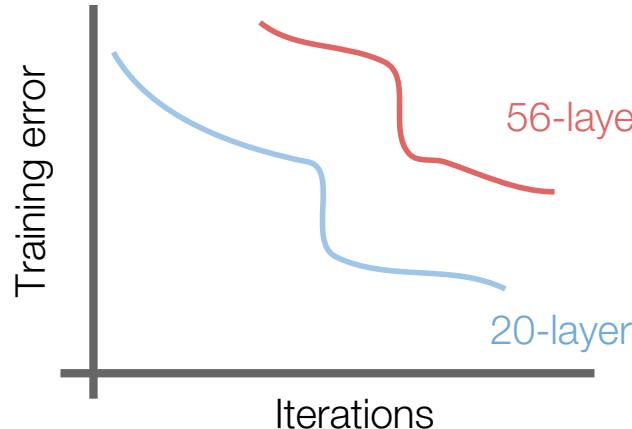
- LeNet-5: 2 Conv + 2 FC layers.
- AlexNet: 5 Conv + 3 FC layers.
- VGG16: 13 Conv + 2 FC layers.



classic architectures (sequential)

Question: Why VGG16? Why not deeper classic architectures?

Answer: Deeper nets have worse training and test errors.



How Deep Can We Go?

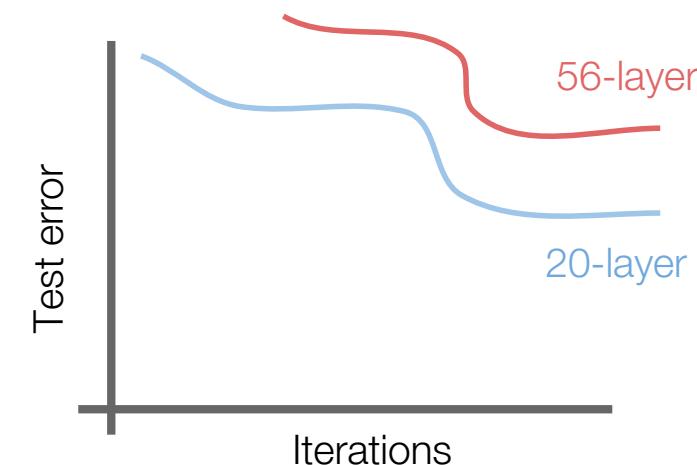
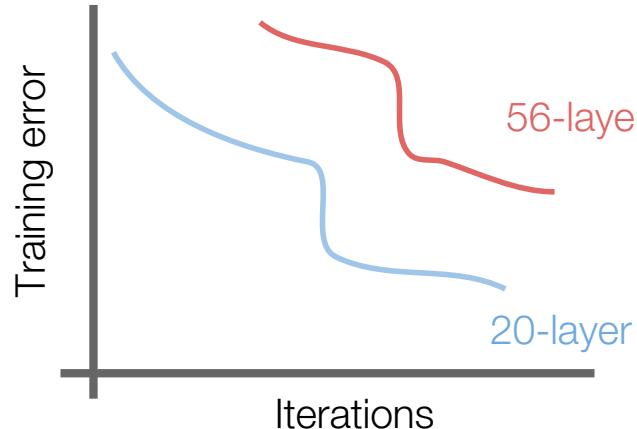
- LeNet-5: 2 Conv + 2 FC layers.
- AlexNet: 5 Conv + 3 FC layers.
- VGG16: 13 Conv + 2 FC layers.



classic architectures (sequential)

Question: What makes deeper VGG nets worse?

Answer: It is bad optimization, not overfitting.



How Deep Can We Go?

- LeNet-5: 2 Conv + 2 FC layers.
 - AlexNet: 5 Conv + 3 FC layers.
 - VGG16: 13 Conv + 2 FC layers.
- 
- classic architectures (sequential)

Question: What makes deeper VGG nets worse?

- The vanishing gradient problem.
 - Derivative of the loss function w.r.t. a bottom layer can be vanishingly small.
 - It makes deep nets difficult to optimize. The bottom layers are not well trained.

How Deep Can We Go?

- LeNet-5: 2 Conv + 2 FC layers.
 - AlexNet: 5 Conv + 3 FC layers.
 - VGG16: 13 Conv + 2 FC layers.
 - Inception: 21 Conv + 1 FC layers.

classic architectures (sequential)

Question: Why can Inception go deeper than VGG16?

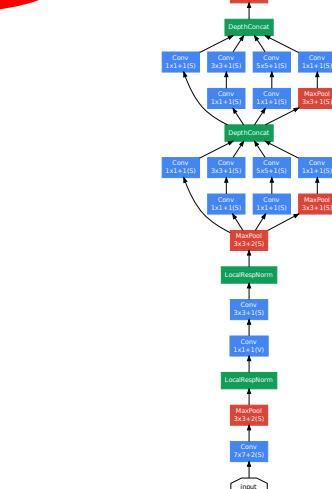
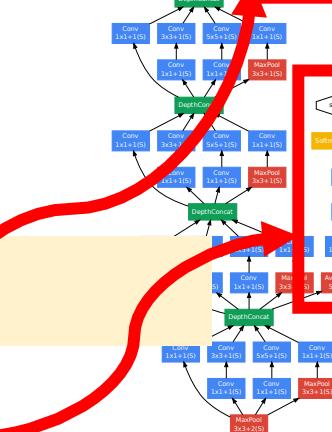
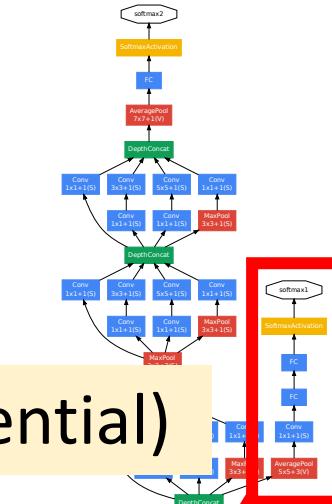
How Deep Can We Go?

- LeNet-5: 2 Conv + 2 FC layers.
 - AlexNet: 5 Conv + 3 FC layers.
 - VGG16: 13 Conv + 2 FC layers.
 - Inception: 21 Conv + 1 FC layers.

classic architectures (sequential)

Question: Why can Inception go deeper than VGG16?

- Auxiliary outputs inject additional gradient at lower layers.



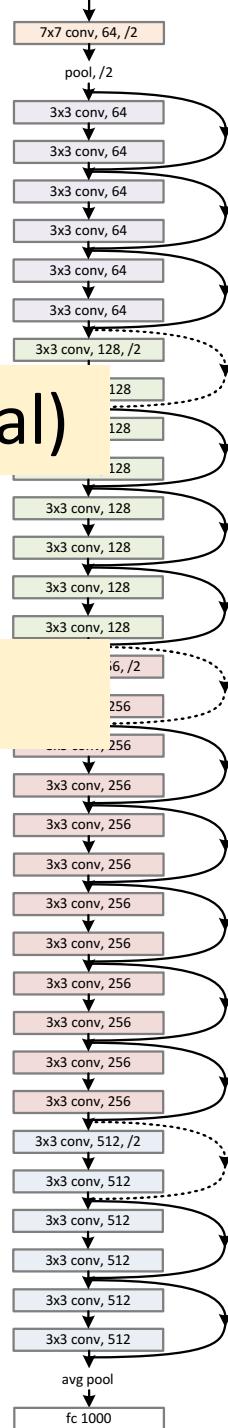
How Deep Can We Go?

- LeNet-5: 2 Conv + 2 FC layers.
- AlexNet: 5 Conv + 3 FC layers.
- VGG16: 13 Conv + 2 FC layers.
- Inception: 21 Conv + 1 FC layers.
- ResNet: Up to 151 Conv + 1 FC layers.

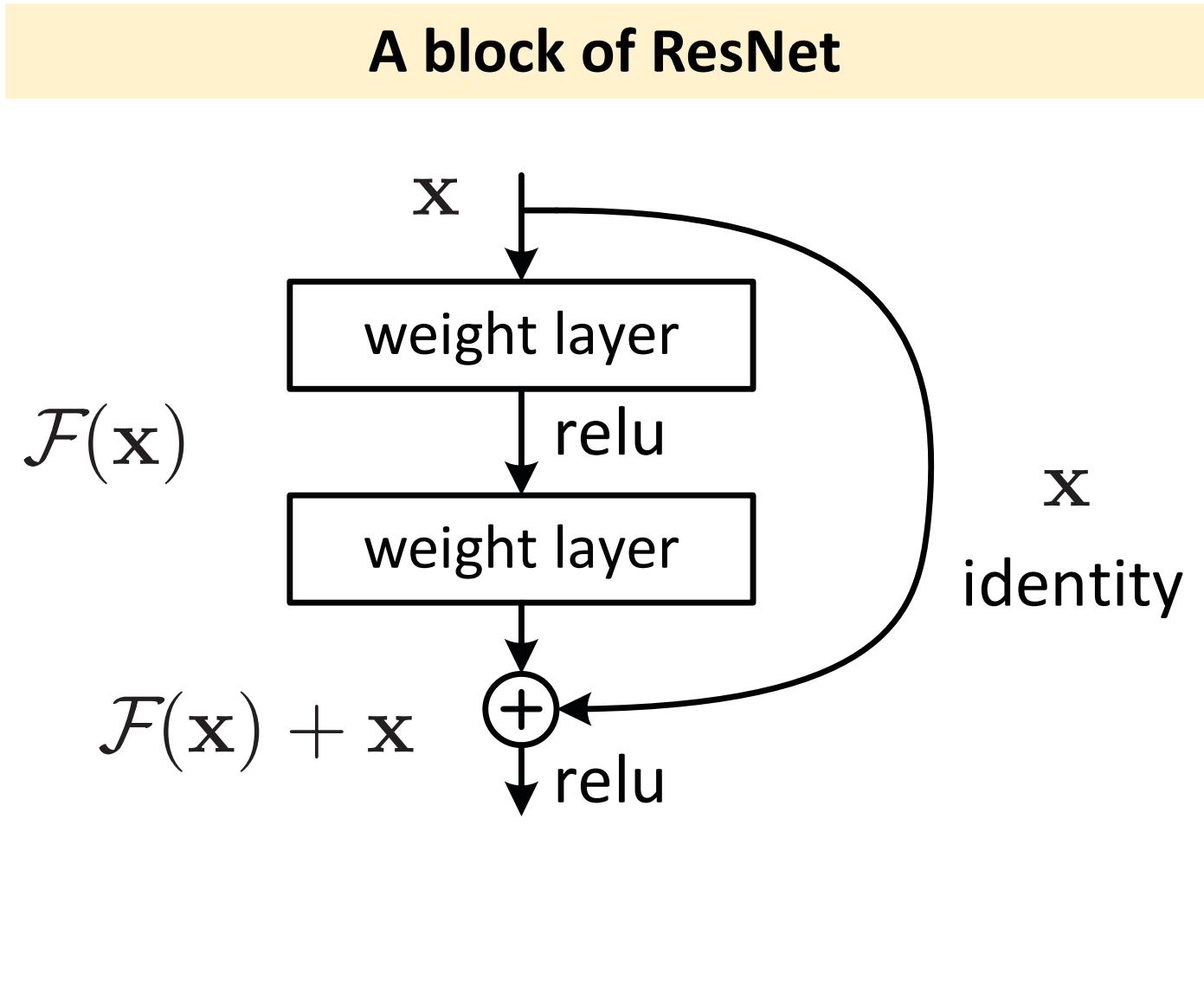
classic architectures (sequential)

modern architectures

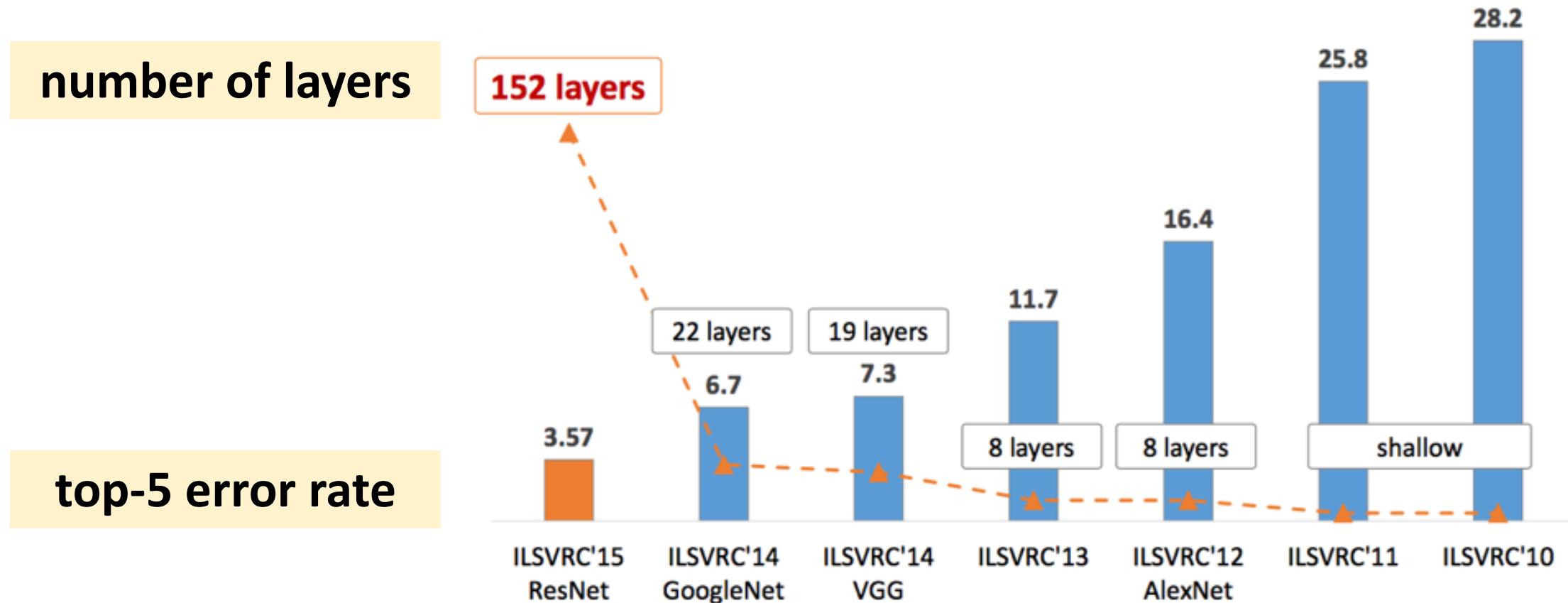
ResNet's key idea: skip connections for curing vanishing gradient.



ResNet [He *et al.* 2015]



Winners of ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



Classification Error: Top-1 vs. Top-5



ConvNet
→

Label: “**wolf**”

class	pred. probability
• husky	0.50
• wolf	0.20
• dog	0.18
• fox	0.08
• snow	0.01
• fur	0.01
• forest	0.01
•	
•	
•	

Classification Error: Top-1 vs. Top-5



Label: “**wolf**”

ConvNet
→

class	pred. probability
• husky	0.50
• wolf	0.20
• dog	0.18
• fox	0.08
• snow	0.01
• fur	0.01
• forest	0.01

Evaluated by the Top-1 error,
this prediction is **wrong!**

Classification Error: Top-1 vs. Top-5



ConvNet
→

class	pred. probability
• husky	0.50
• wolf	0.20
• dog	0.18
• fox	0.08
• snow	0.01

- fur 0.01
- forest 0.01

Label: “**wolf**”

Evaluated by the Top-5 error,
the prediction is **correct!**

Beyond the Accuracy

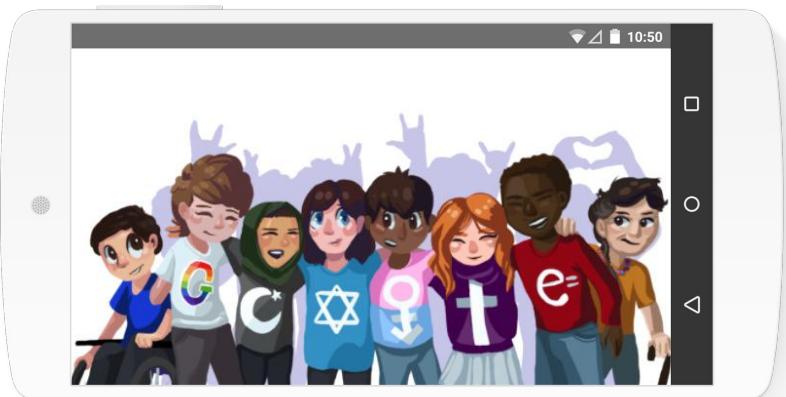
Deep Learning on the Edge

Object Detection



Photo by Juanedc (CC BY 2.0)

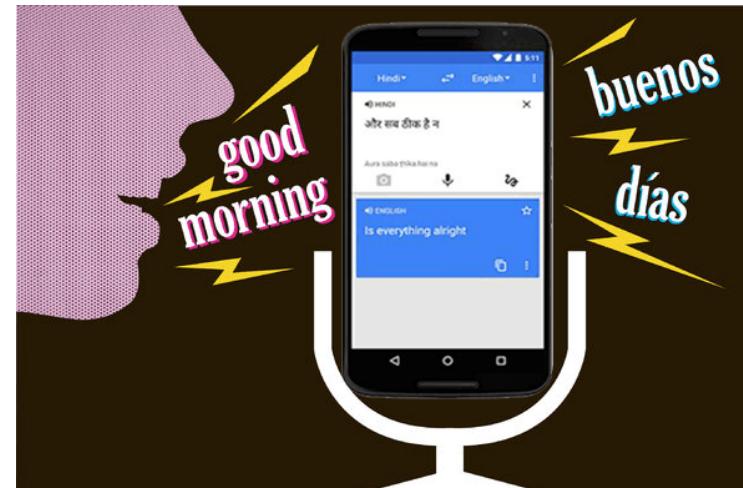
Face Attributes



Google Doodle by Sarah Harrison

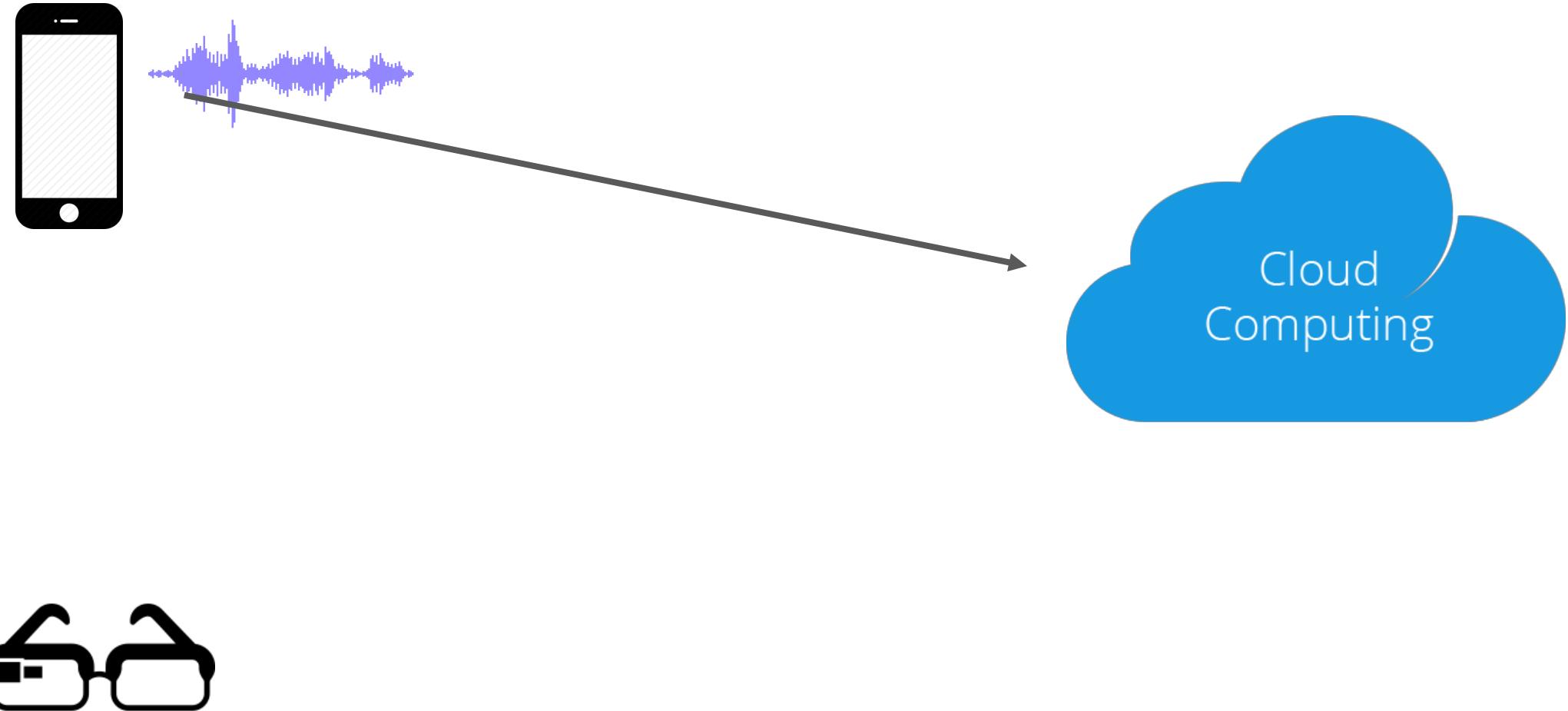


OCR & Translation

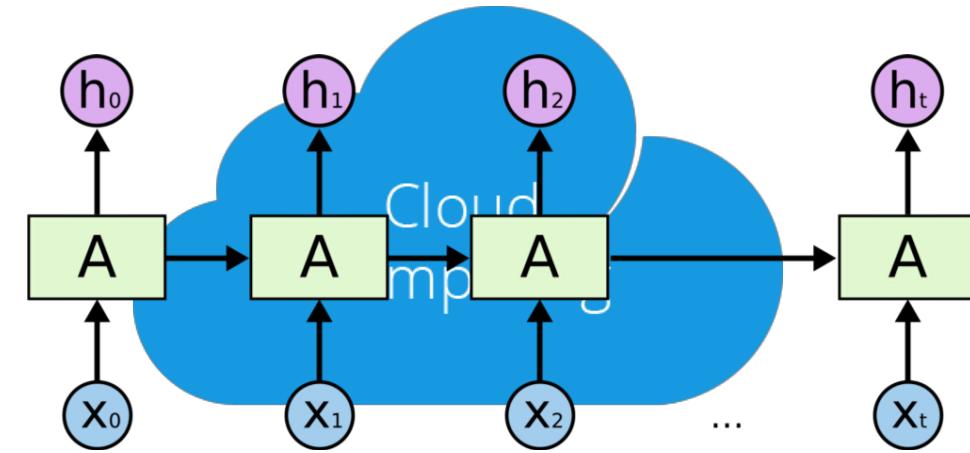


Speech Recognition & Translation

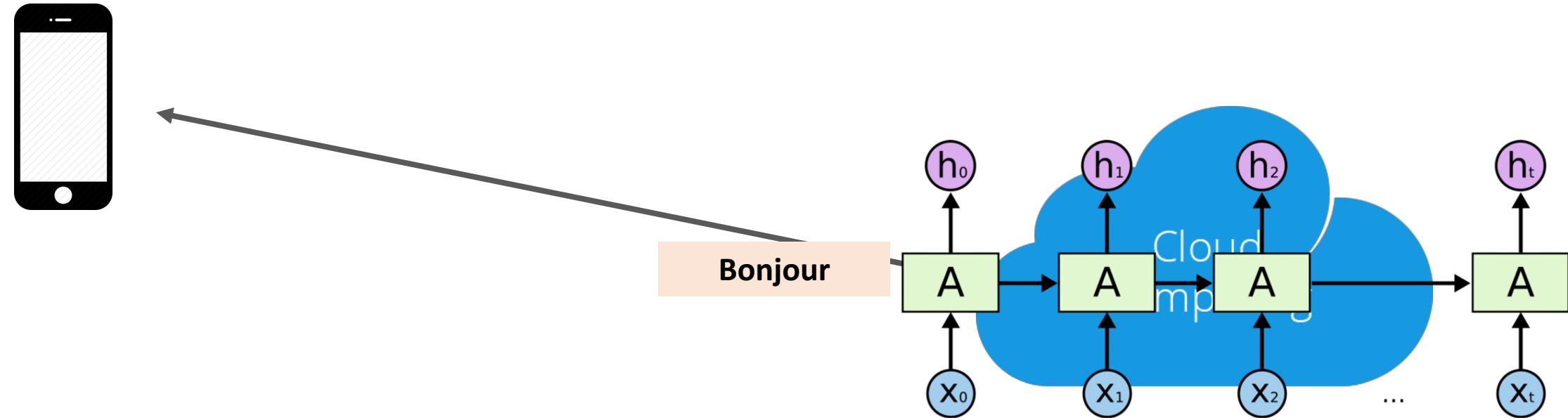
Solution 1: Could Computing



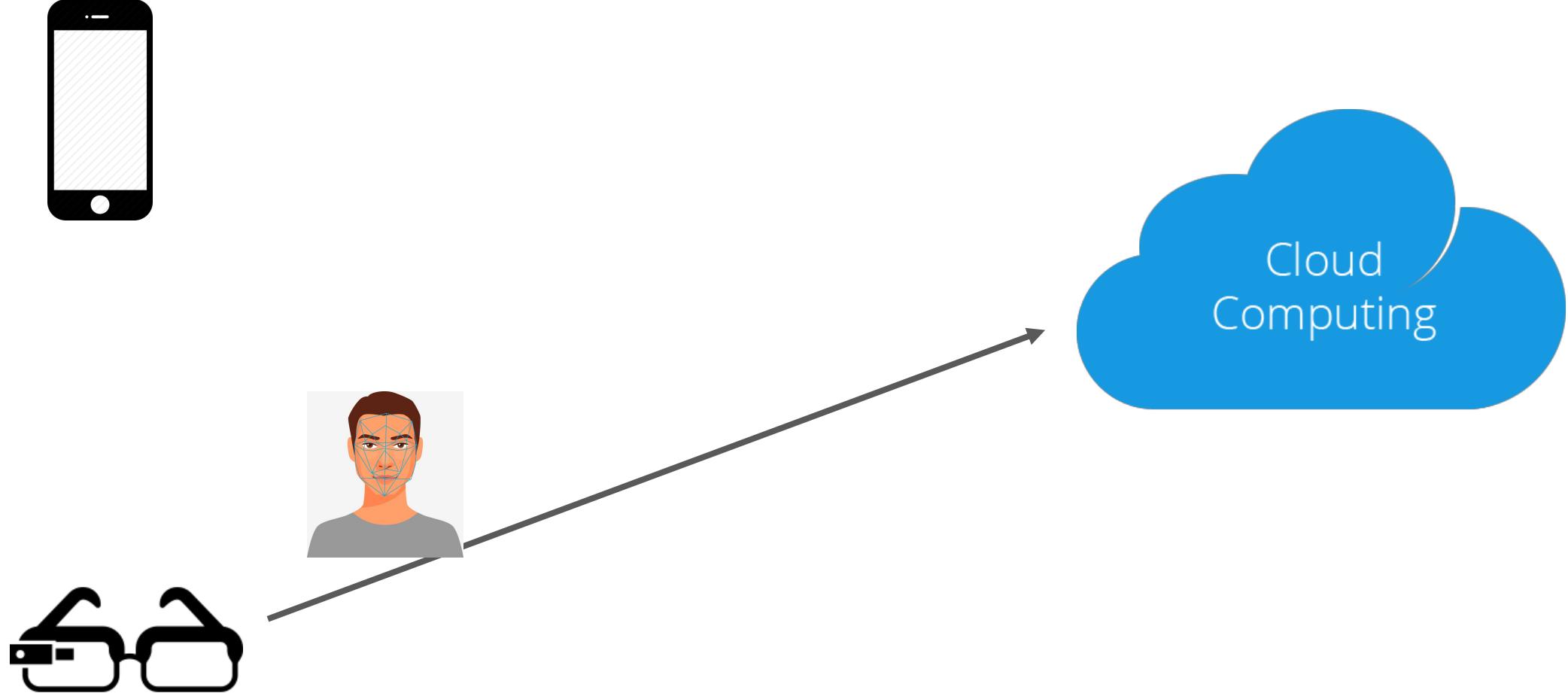
Solution 1: Could Computing



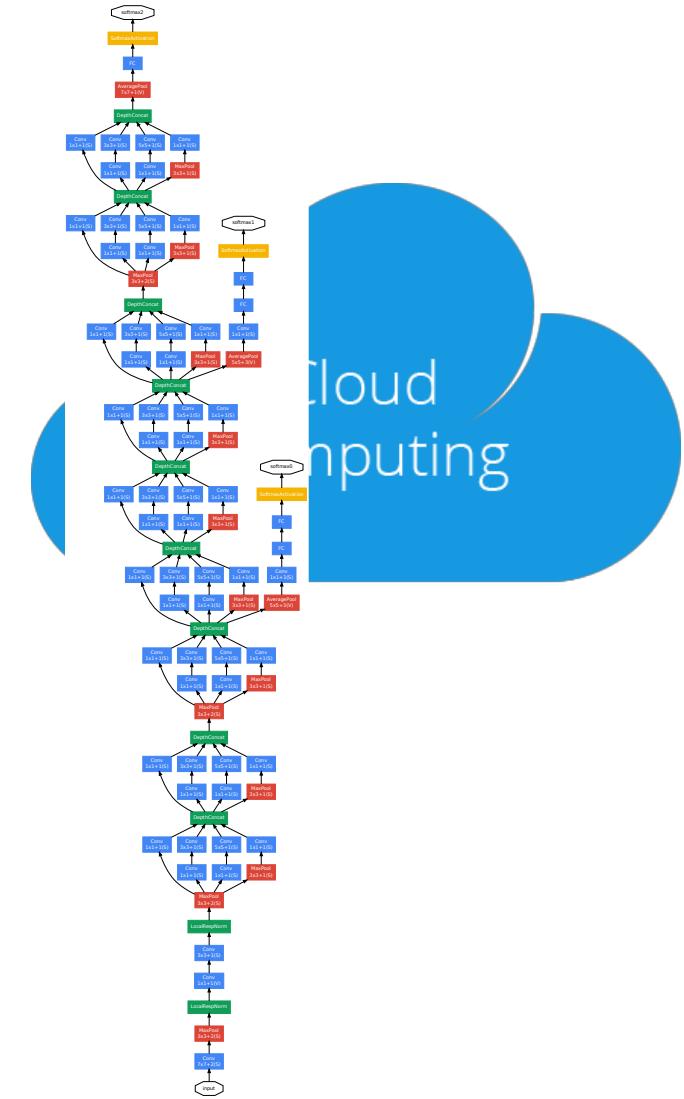
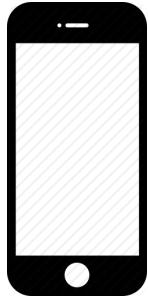
Solution 1: Could Computing



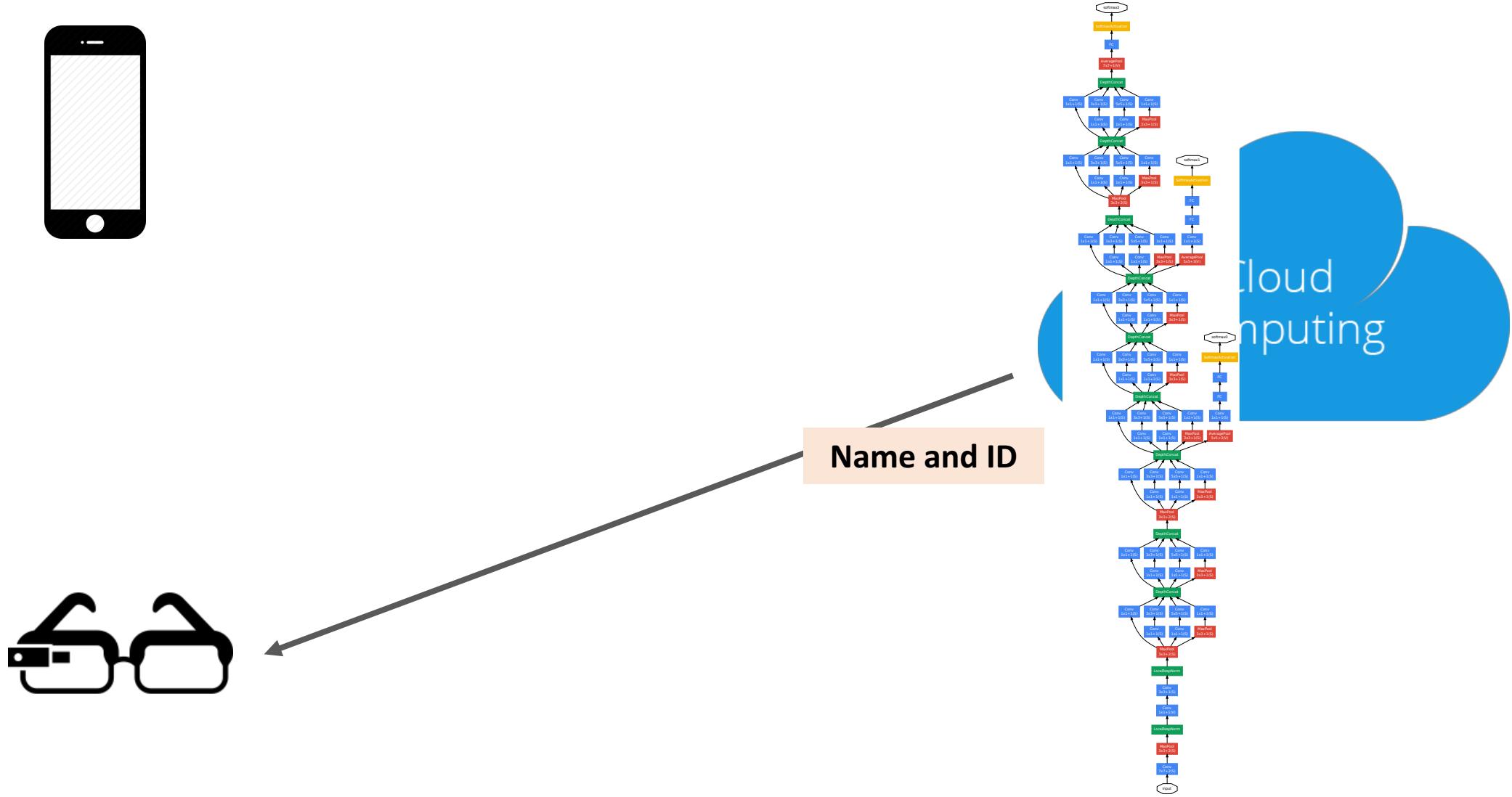
Solution 1: Could Computing



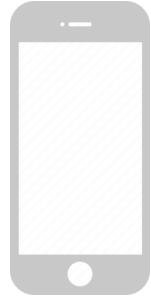
Solution 1: Could Computing



Solution 1: Could Computing



Solution 1: Could Computing

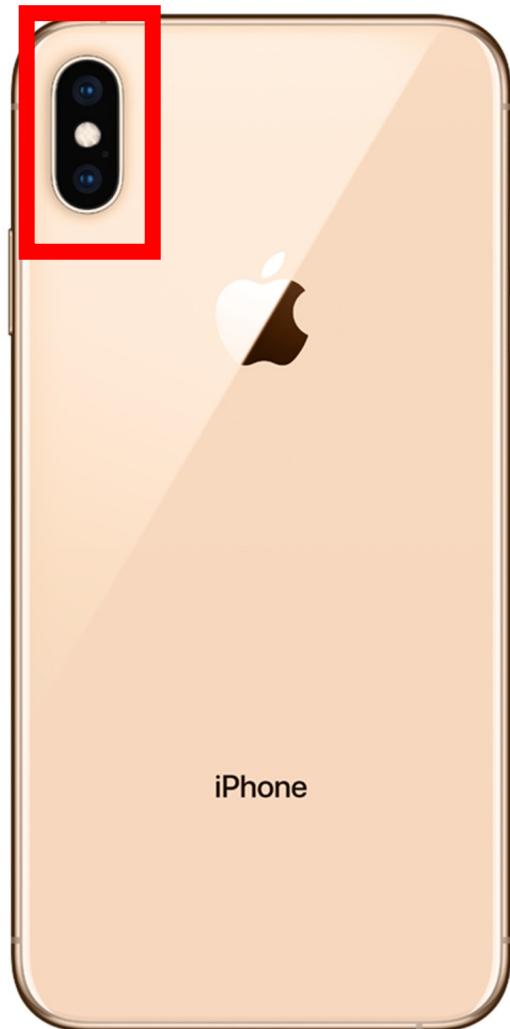


Downsides:

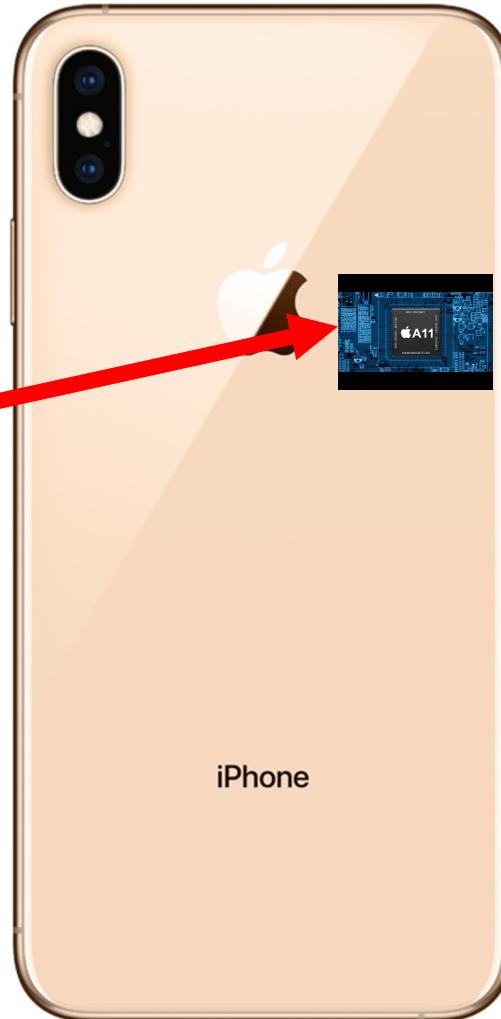
1. Sending data between client and server is relatively slow (in comparing with local computation on the device.)
2. Sending data through 3G/4G/5G network may be charged.
3. User's privacy.



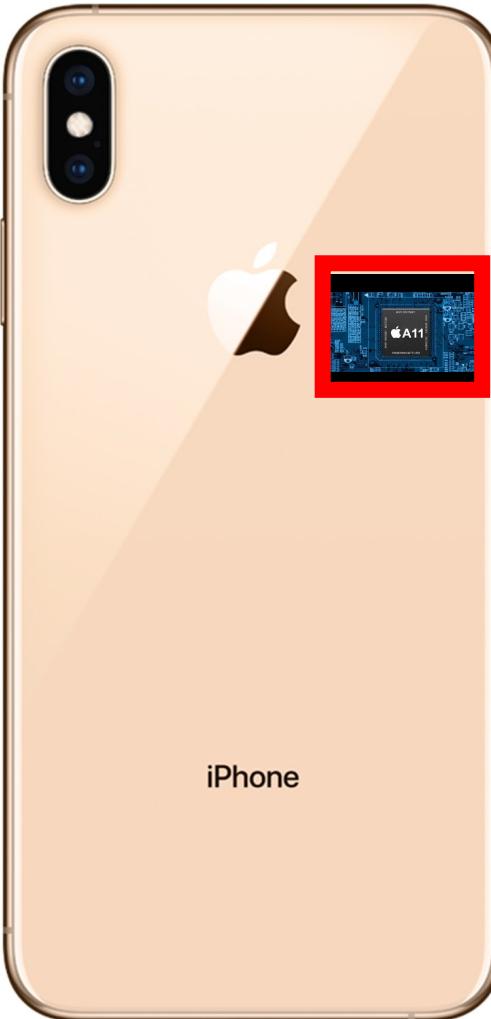
Solution 2: Edge Computing



Solution 2: Edge Computing



Solution 2: Edge Computing

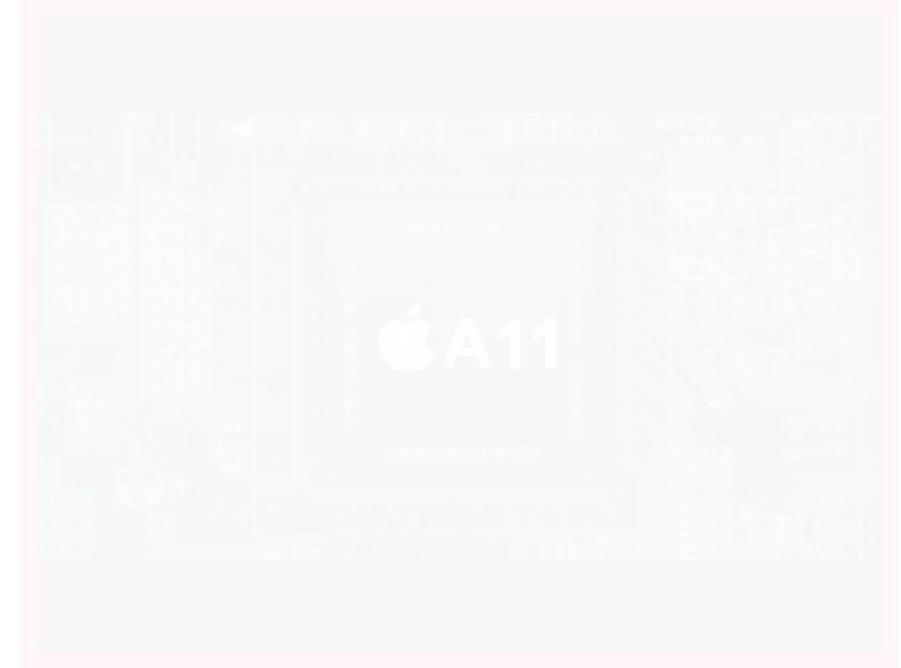
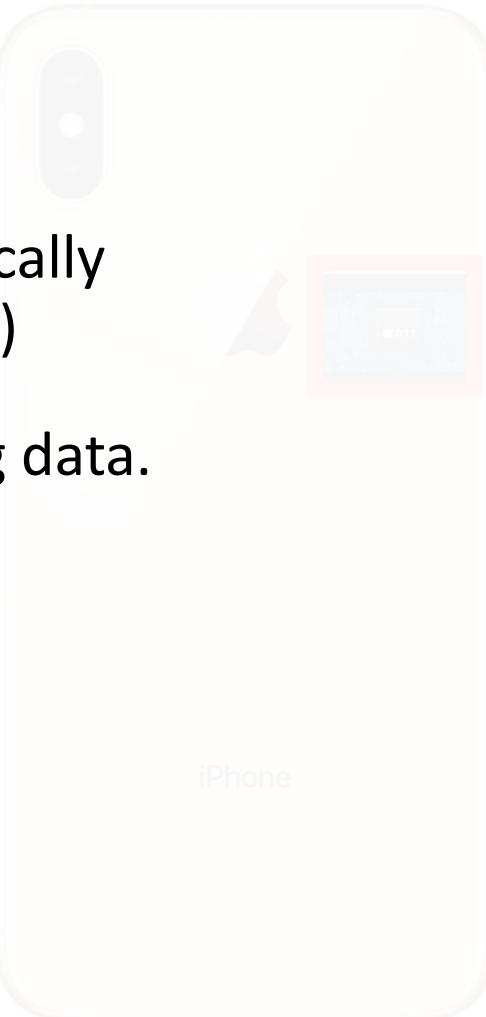


**Run a ConvNet on the chip
to make prediction.
(The ConvNet has been
trained on the server.)**

Solution 2: Edge Computing

Advantages:

1. Faster. (Computation is typically faster than communication.)
2. Avoid sending and receiving data.
 - Save money.
 - Works without internet.
3. Protect user's privacy.



Solution 2: Edge Computing

Advantages:

1. Faster. (Computation is typically faster than communication.)
2. Avoid sending and receiving data.
 - Save money.
 - Works without internet.
3. Protect user's privacy.

Disadvantages:

1. Energy consumption. (Heavy matrix and tensor computation.)
2. Cost local memory and storage. (Deep ConvNets has at least millions of parameters.)

Run a ConvNet on the chip
to make prediction.
(The ConvNet has been
trained on the server.)

Cloud Computing V.S. Edge Computing

- Edge computing is more popular for the deep learning tasks (at present).
- Research directions of edge computing:
 1. Less float point operations (FLOPs). (Thus less energy consumption.)
 2. Less network parameters. (Thus less memory and storage.)

MobileNet [Howard *et al.* 2017]

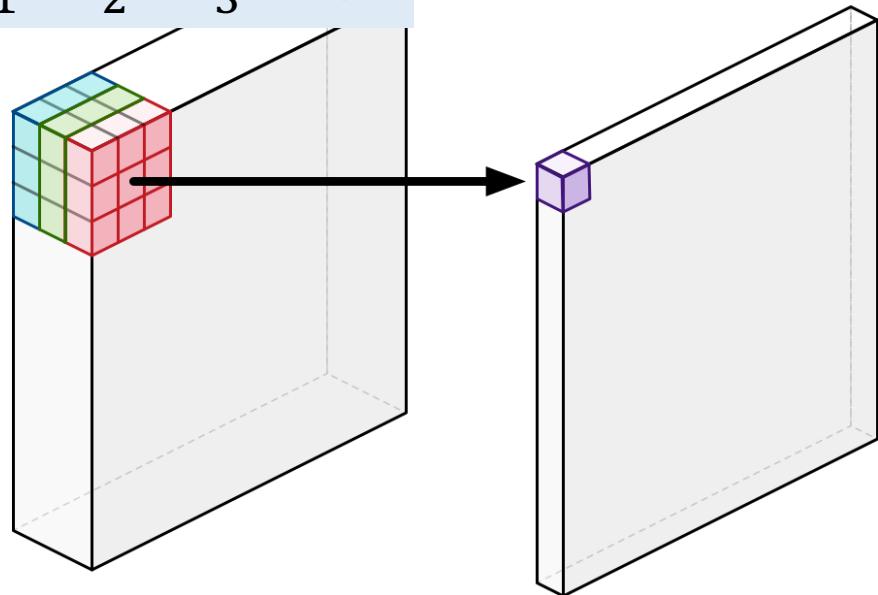
- Motivations
 - Small enough to fit in an iPhone (just $4M$ parameters).
 - Less computation than the standard ConvNets.
- Key idea: Depthwise separable convolution.
- Paper: <https://arxiv.org/pdf/1704.04861.pdf>
- Further reading:
 - <http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/>
 - <https://medium.com/@yu4u/why-mobilenet-and-its-variants-e-g-shufflenet-are-fast-1c7048b9618d>

Depthwise Separable Convolution

convolution

`keras.layers.Conv2D`

One $k_1 \times k_2 \times d_3$ filter



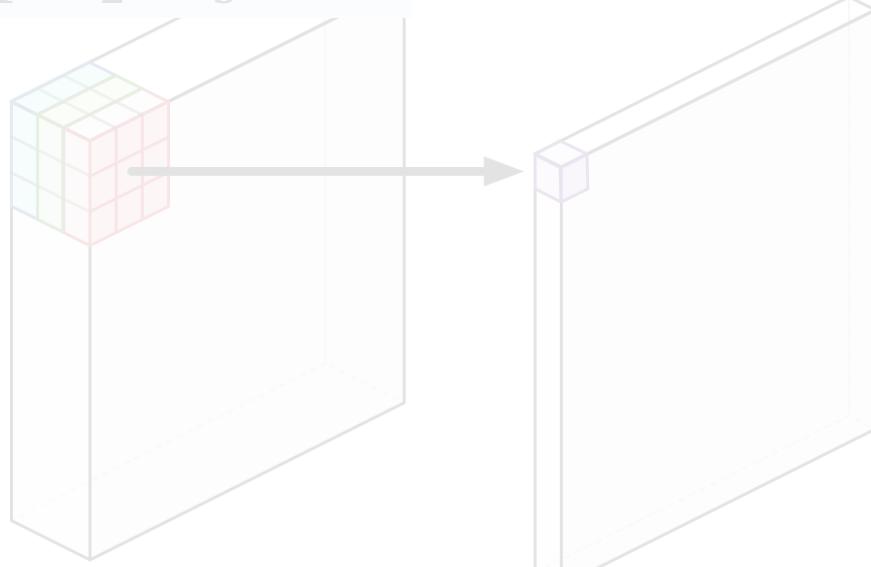
$d_1 \times d_2 \times d_3$ input

Depthwise Separable Convolution

convolution

`keras.layers.Conv2D`

One $k_1 \times k_2 \times d_3$ filter



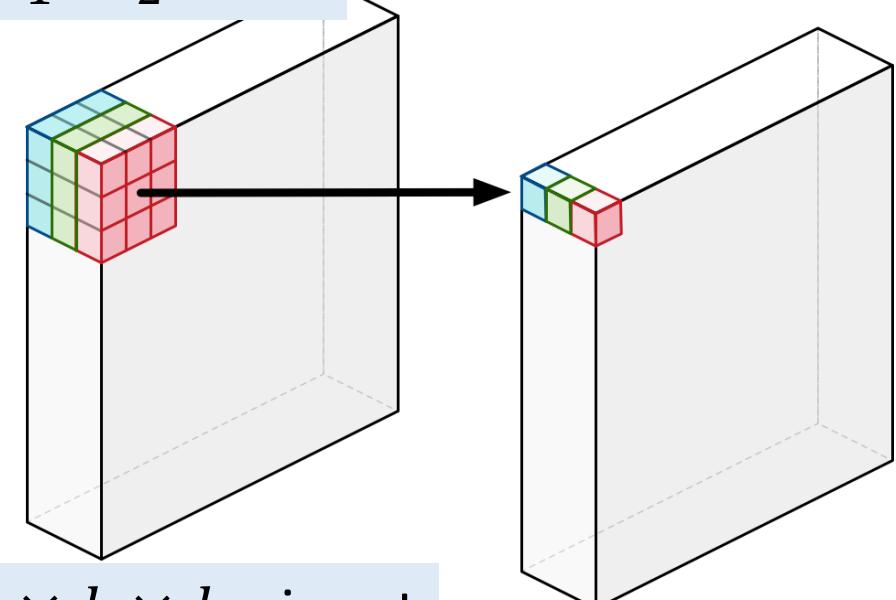
$d_1 \times d_2 \times d_3$ input

$d_1 \times d_2 \times 1$ output

depthwise convolution

`keras.layers.DepthwiseConv2D`

One $k_1 \times k_2$ filter



$d_1 \times d_2 \times d_3$ input

In this example, $d_3 = 3$.

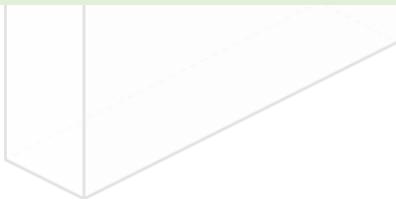
Depthwise Separable Convolution

convolution

`keras.layers.Conv2D`

For $i = 1$ to d_3 (each of the slices):

- Compute the convolution of the i -th slice (shape $d_1 \times d_2$) and the filter (shape $k_1 \times k_2$).
- Output: a matrix (shape $d_1 \times d_2$).



$d_1 \times d_2 \times d_3$ input

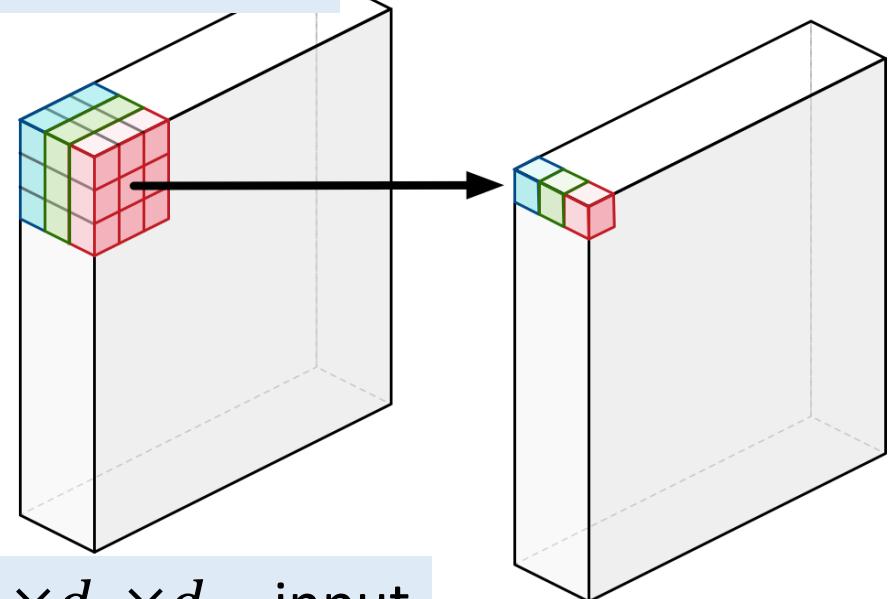


$d_1 \times d_2 \times 1$ output

depthwise convolution

`keras.layers.DepthwiseConv2D`

One $k_1 \times k_2$ filter



$d_1 \times d_2 \times d_3$ input

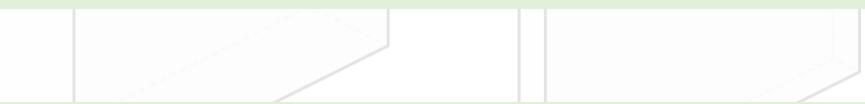
Depthwise Separable Convolution

convolution

`keras.layers.Conv2D`

For $i = 1$ to d_3 (each of the slices):

- Compute the convolution of the i -th slice (shape $d_1 \times d_2$) and the filter (shape $k_1 \times k_2$).
- Output: a matrix (shape $d_1 \times d_2$).



Thus the final output is $d_1 \times d_2 \times d_3$.

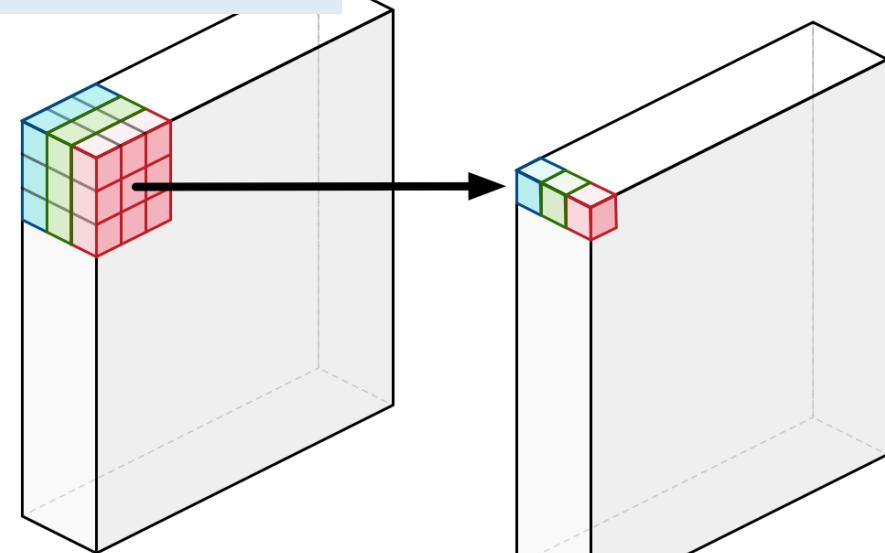
$d_1 \times d_2 \times d_3$ input

$d_1 \times d_2 \times 1$ output

depthwise convolution

`keras.layers.DepthwiseConv2D`

One $k_1 \times k_2$ filter



$d_1 \times d_2 \times d_3$ input

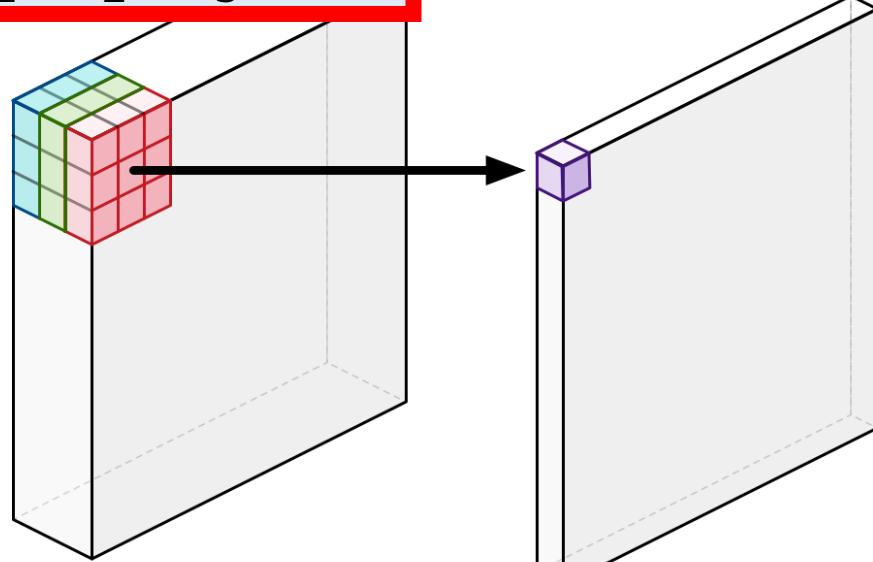
$d_1 \times d_2 \times d_3$ output

Depthwise Separable Convolution

convolution

`keras.layers.Conv2D`

One $k_1 \times k_2 \times d_3$ filter



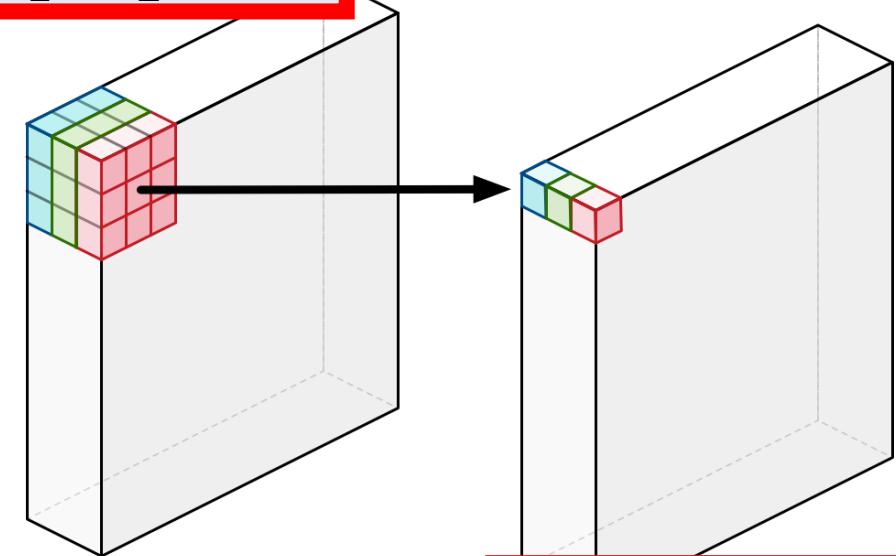
$d_1 \times d_2 \times d_3$ input

$d_1 \times d_2 \times 1$ output

depthwise convolution

`keras.layers.DepthwiseConv2D`

One $k_1 \times k_2$ filter



$d_1 \times d_2 \times d_3$ input

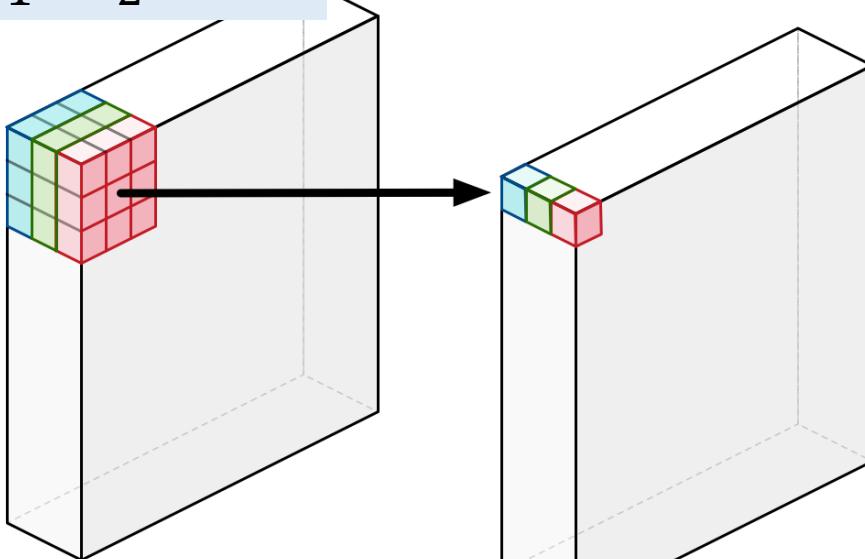
$d_1 \times d_2 \times d_3$ output

MobileNet: Depthwise Conv + 1×1 Conv

depthwise convolution

`keras.layers.DepthwiseConv2D`

One $k_1 \times k_2$ filter



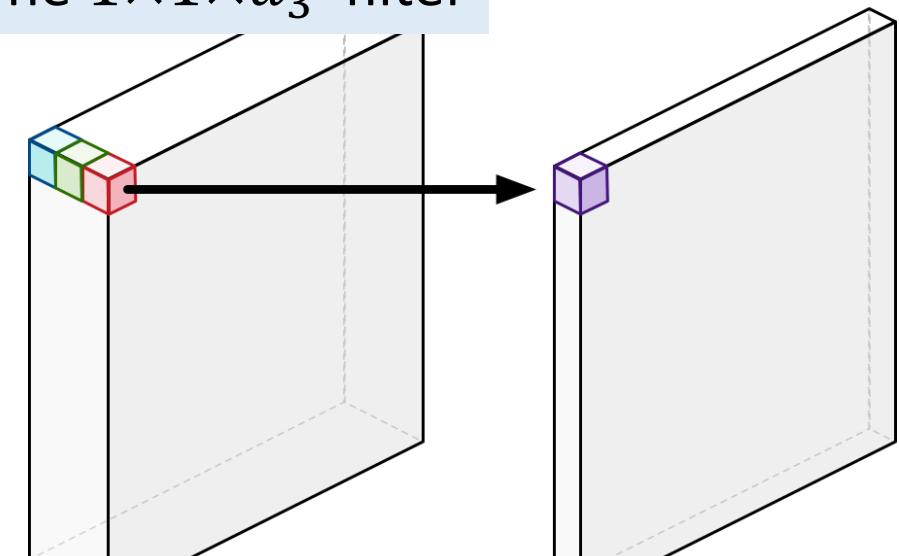
$d_1 \times d_2 \times d_3$ input

$d_1 \times d_2 \times d_3$ output

1×1 convolution

`keras.layers.Conv2D`

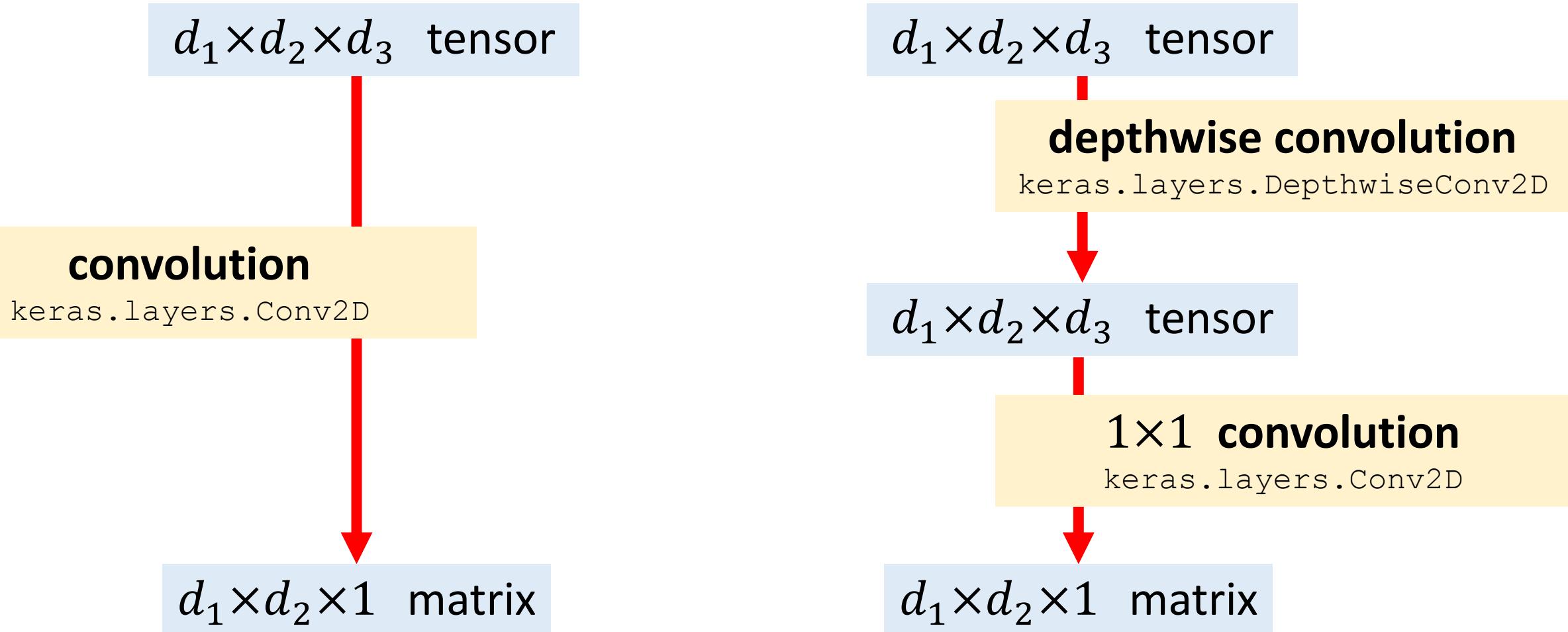
One $1 \times 1 \times d_3$ filter



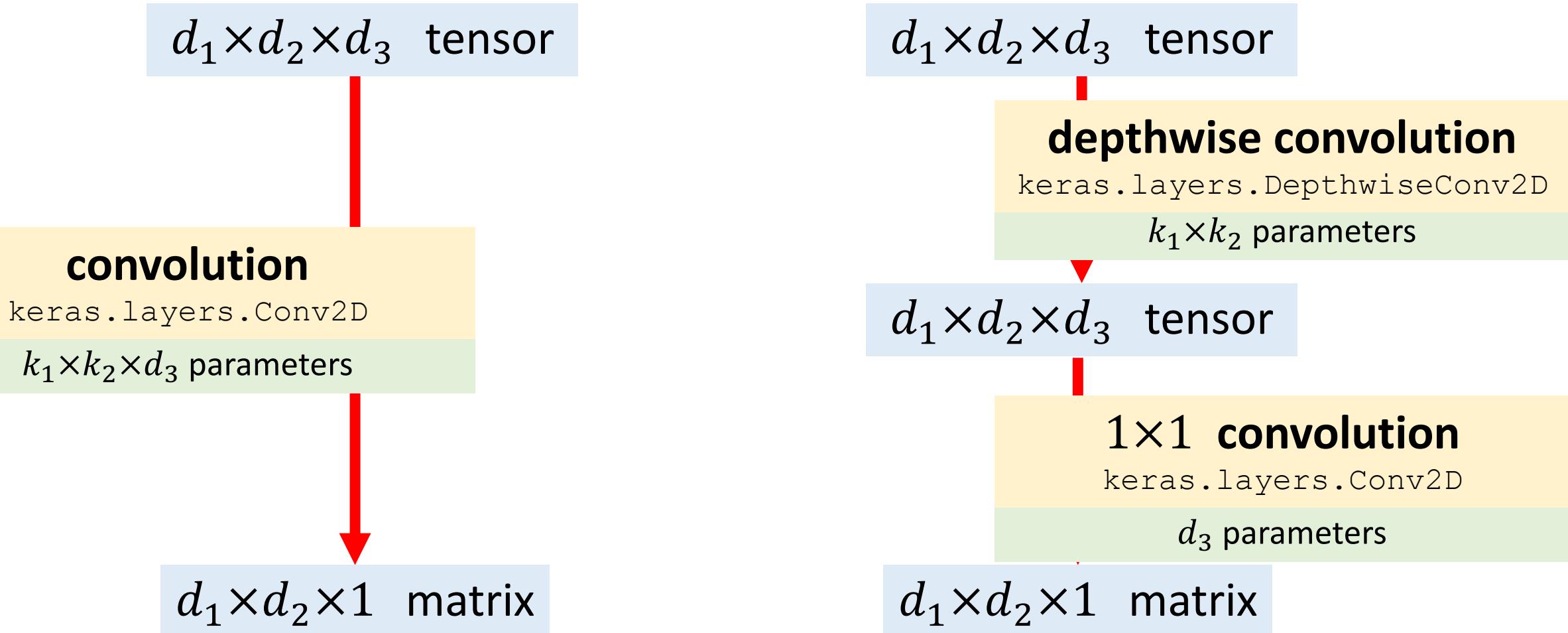
$d_1 \times d_2 \times d_3$ input

$d_1 \times d_2 \times 1$ output

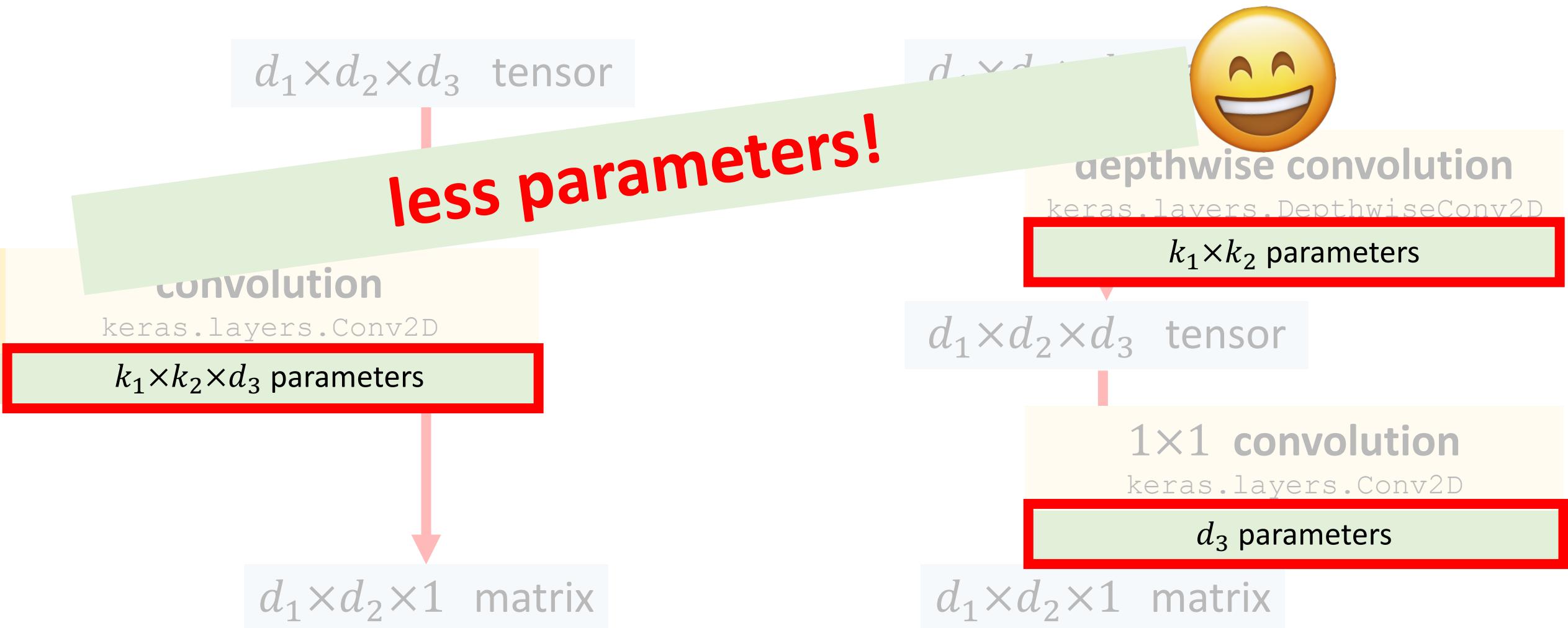
MobileNet: Depthwise Separable Conv + 1×1 Conv



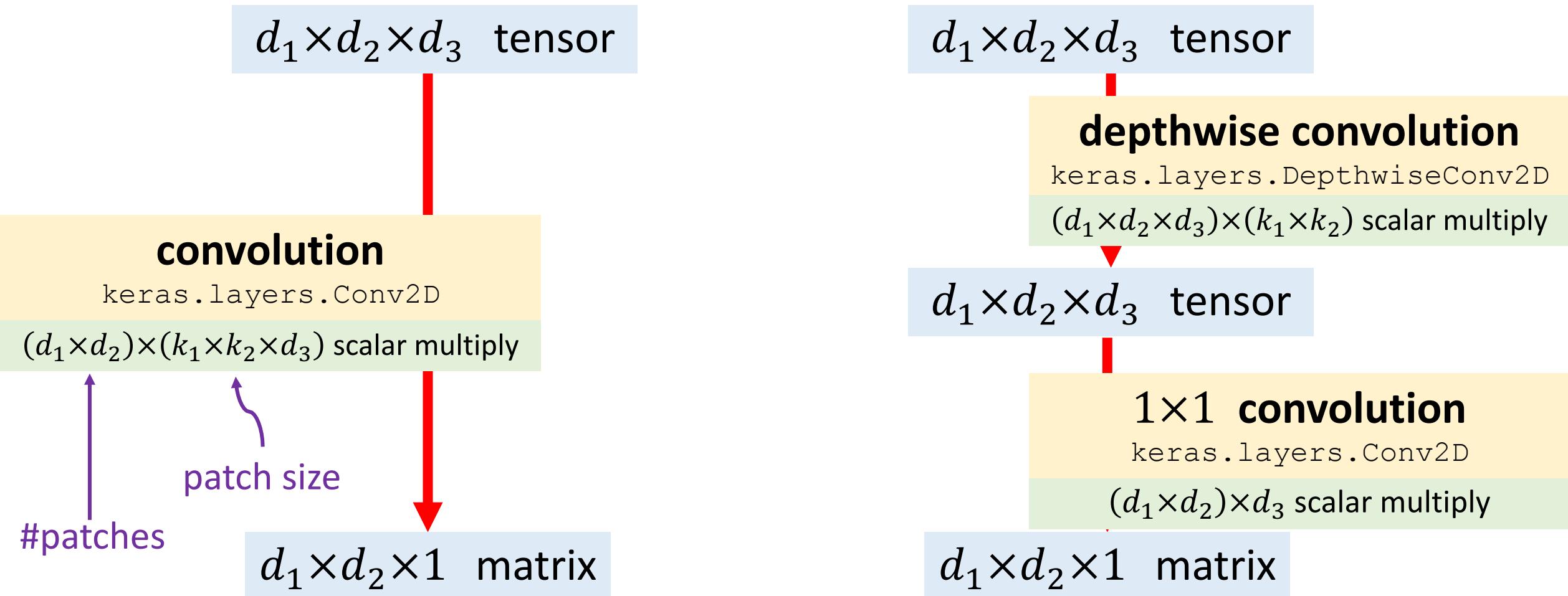
MobileNet: Depthwise Separable Conv + 1×1 Conv



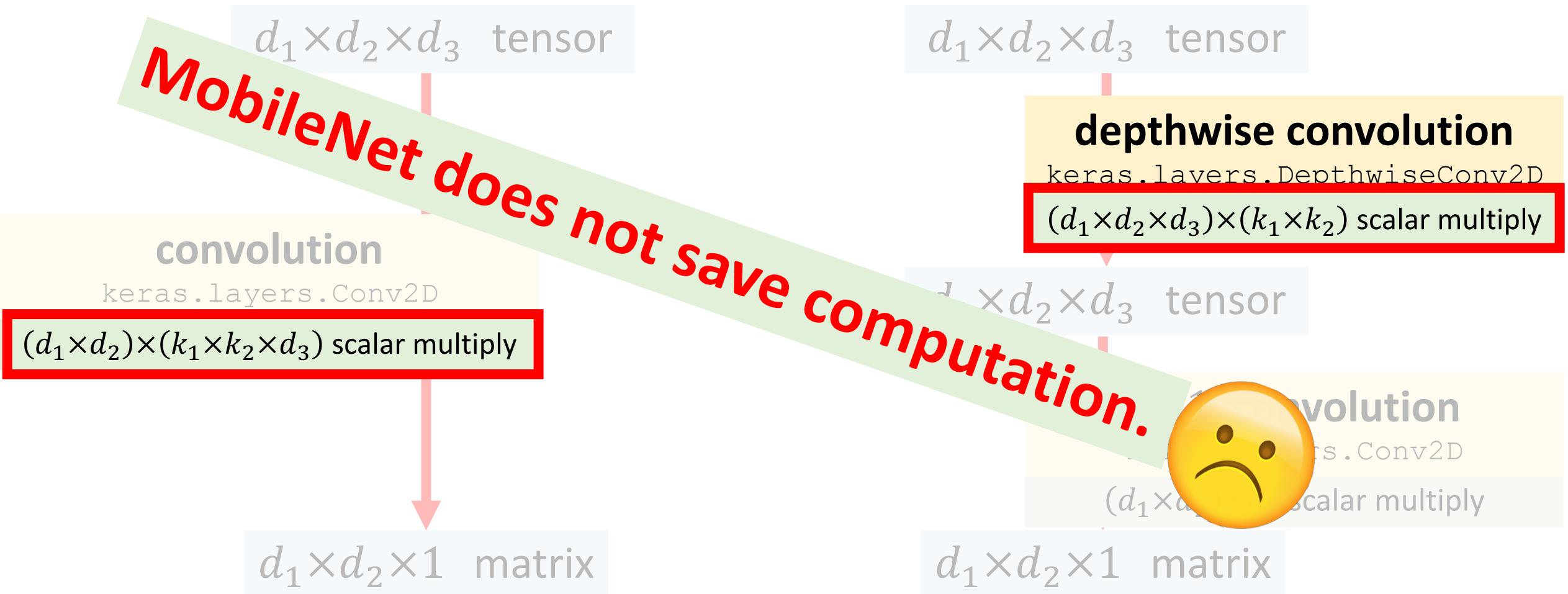
MobileNet: Depthwise Separable Conv + 1×1 Conv



MobileNet: Depthwise Separable Conv + 1×1 Conv



MobileNet: Depthwise Separable Conv + 1×1 Conv



One Actual Block of MobileNet

$d_1 \times d_2 \times d_3$ tensor

Repeat the following for $2d_3$ times:

- Depthwise convolution:

DepthwiseConv2D (3×3), stride=2

- 1×1 (standard) convolution:

Conv2D (1×1), stride=1



One Actual Block of MobileNet

$d_1 \times d_2 \times d_3$ tensor

Repeat the following for $2d_3$ times:

- Depthwise convolution:

DepthwiseConv2D (3×3), stride=2

- 1×1 (standard) convolution:

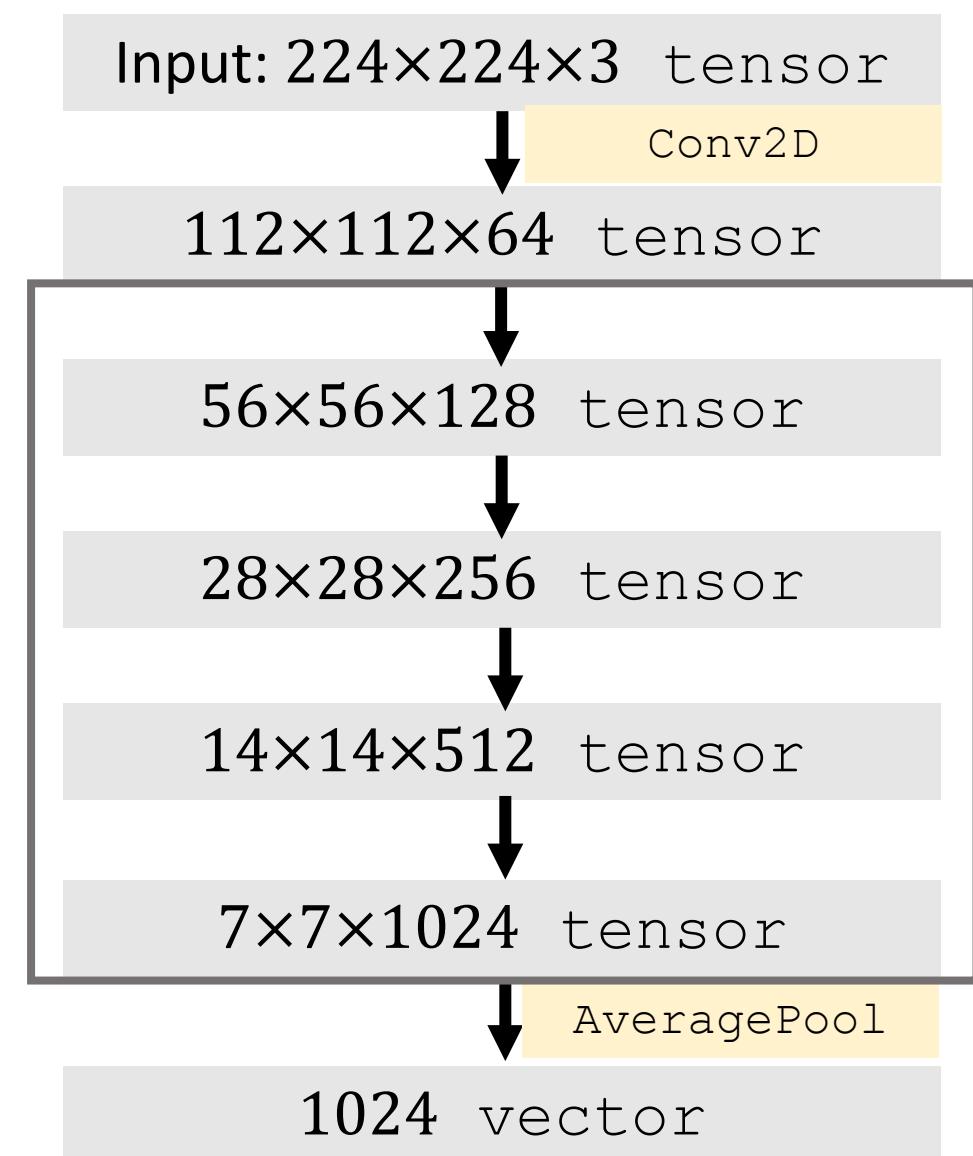
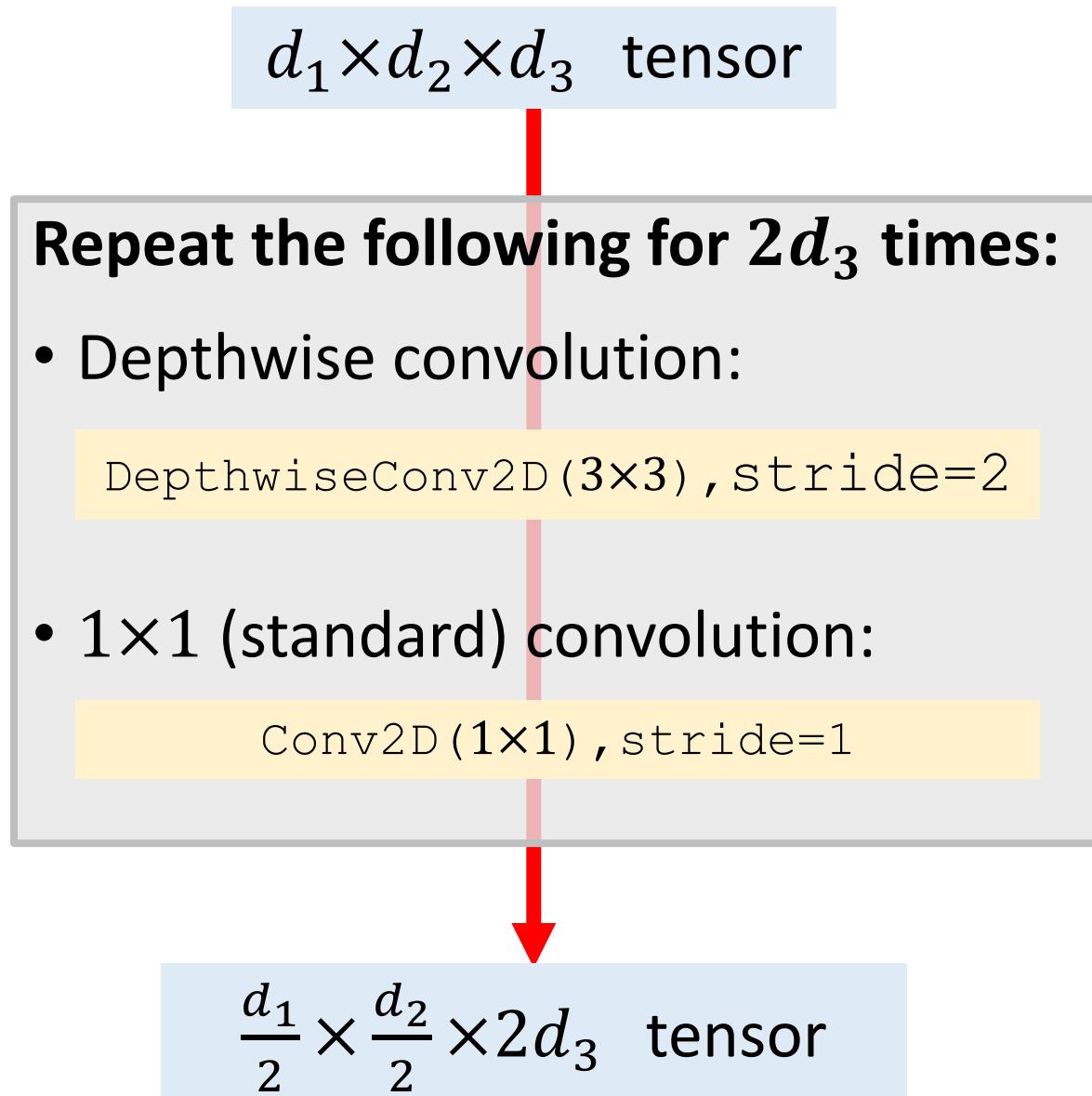
Conv2D (1×1), stride=1

Double the depth

Half the width and length.

$\frac{d_1}{2} \times \frac{d_2}{2} \times 2d_3$ tensor

One Actual Block of MobileNet



Implementation in Keras

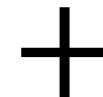
separable convolution

`keras.layers.SeparableConv2D`

- In Keras, you can directly use SeparableConv2D to build MobileNet.
- $\text{SeparableConv2D} = \text{DepthwiseConv2D} + \text{Conv2D}(1 \times 1)$

depthwise convolution

`keras.layers.DepthwiseConv2D`



1×1 convolution

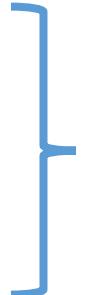
`keras.layers.Conv2D`

Implementation in Keras

```
import keras
model = keras.applications.mobilenet.MobileNet(input_shape=None,
                                                 alpha=1.0, depth_multiplier=1, dropout=1e-3,
                                                 include_top=True, weights='imagenet',
                                                 input_tensor=None, pooling=None, classes=1000)
model.summary()
```

Summary

CNN Architectures

- LeNet-5: 2 Conv + 2 FC layers.
 - AlexNet: 5 Conv + 3 FC layers.
 - VGG16: 13 Conv + 2 FC layers.
- 
- classic architectures (sequential)

CNN Architectures

- LeNet-5: 2 Conv + 2 FC layers.
 - AlexNet: 5 Conv + 3 FC layers.
 - VGG16: 13 Conv + 2 FC layers.
- 
- classic architectures (sequential)

Question: Can the classic architectures go deeper?

Answer: No.

- Deeper nets have worse training and test errors.
- Vanishing gradient.

CNN Architectures

- LeNet-5: 2 Conv + 2 FC layers.
- AlexNet: 5 Conv + 3 FC layers.
- VGG16: 13 Conv + 2 FC layers.

classic architectures (sequential)

- Inception: 21 Conv + 1 FC layers.
- ResNet: Up to 151 Conv + 1 FC layers.

modern architectures

CNN Architectures

- LeNet-5: 2 Conv + 2 FC layers.
- AlexNet: 5 Conv + 3 FC layers.
- VGG16: 13 Conv + 2 FC layers.



classic architectures (sequential)

- Inception: 21 Conv + 1 FC layers.
- ResNet: Up to 151 Conv + 1 FC layers.



modern architectures

Tricks:

- Inception: auxiliary output (gradient injection).
- ResNet: skip connection.

Reduce Memory and Computation

- Deploy a deep neural network to a smart device.
- Challenges:
 - Computation → Power consumption.
 - Parameters → Storage and memory.

Reduce Memory and Computation

- Deploy a deep neural network to a smart device.
- Challenges:
 - Computation → Power consumption.
 - Parameters → Storage and memory.
- MobileNet's solutions:
 - Separable Convolution instead of Tensor Convolution.
 - AveragePool Layer instead of Flatten Layer.