Departamento de Eletrónica, Telecomunicações e Informática

Complements of Machine Learning

LECTURE 5 : DEEP NN – MOBILE Net & Object detection

Petia Georgieva
(petia@ua.pt)

Outline

Part 1

1. **MobileNet - MobileNet v1 ; MobileNet v2**
2. **EfficientNet**

Part 2

1. **Object detection**
2. **Sliding windows detection algorithm**

Part 1: MobileNet & EfficientNet

Motivation for MobileNets

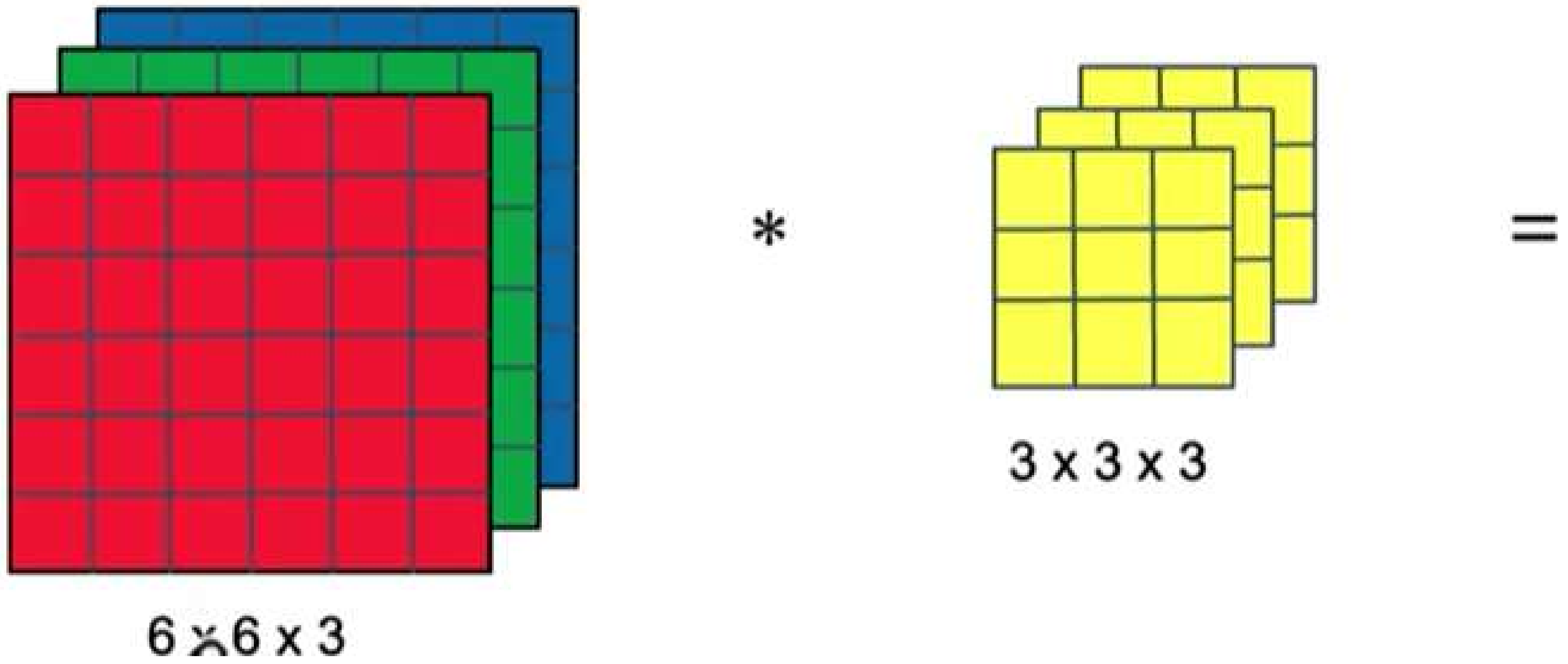
Low computational cost and low memory at deployment

Useful for mobile and embedded vision applications

Key idea: normal vs depth-wise separable convolutions

universidade de aveiro

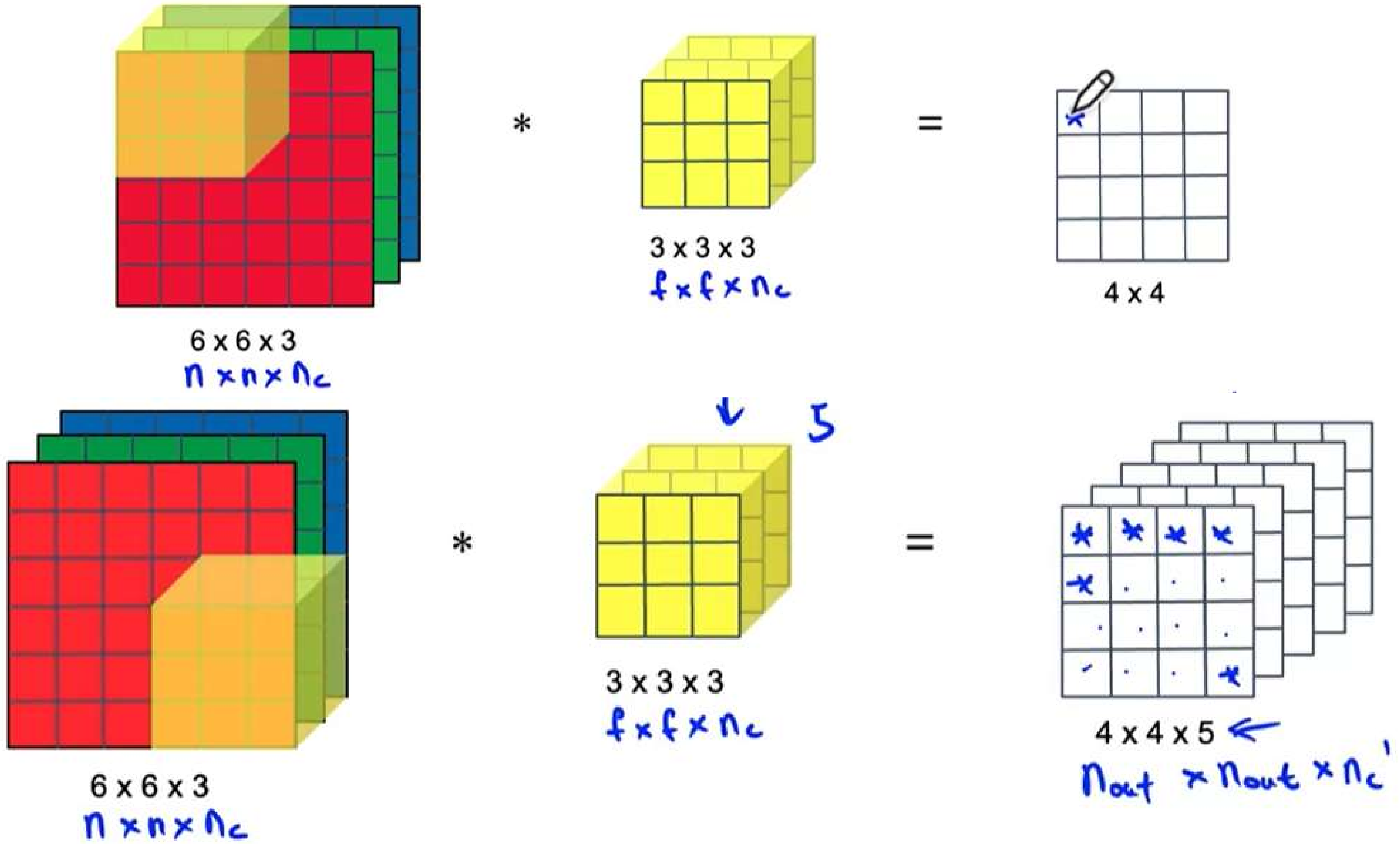
Normal Convolution



Stride =1, no padding

Computational cost for applying 5 filters = ?

Rule of Normal Convolution

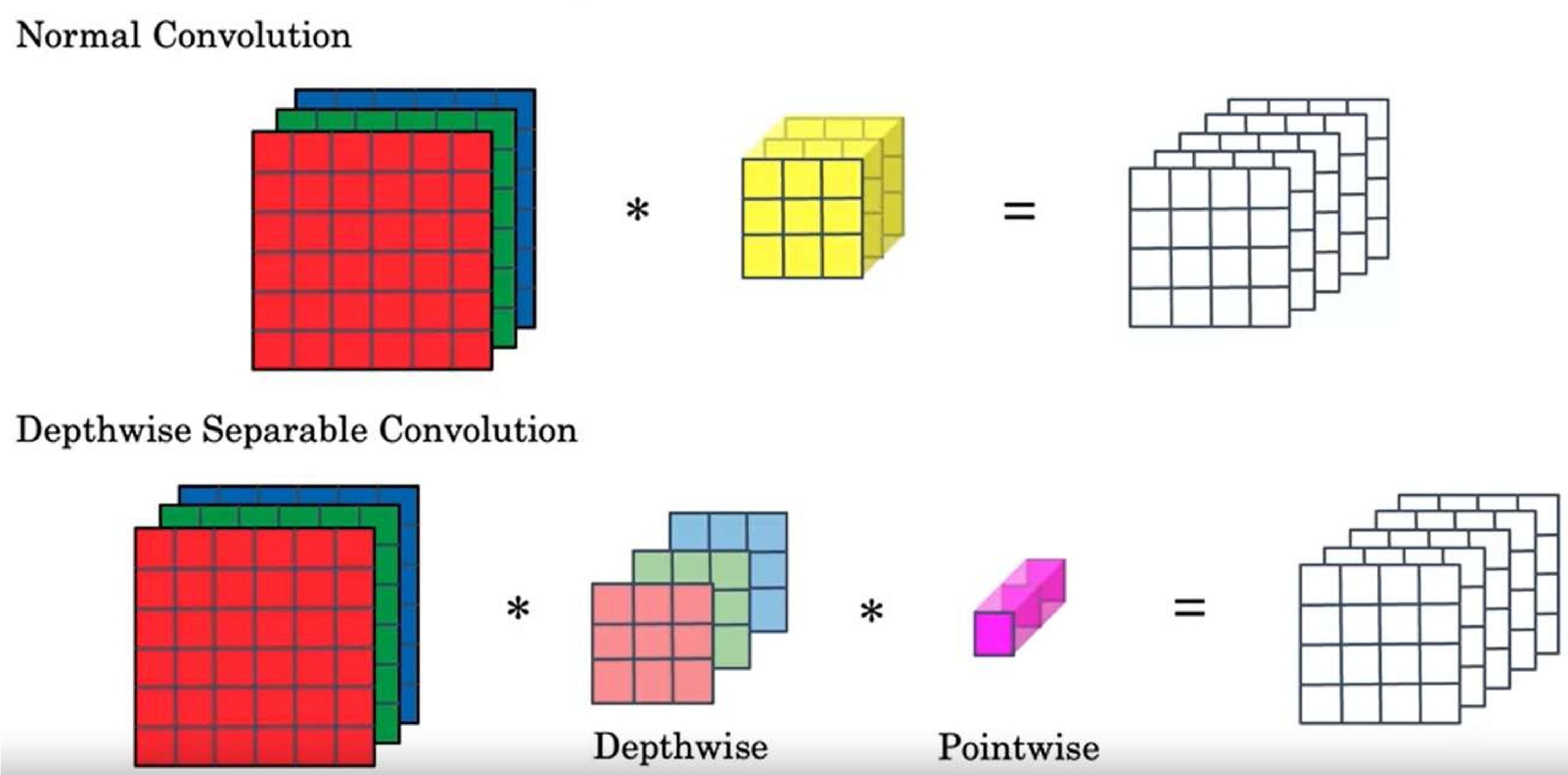


3rd filter dimension = 3rd input dimension (e.g. $\mathbf{n_c} = 3$)

Computational cost = #_filter parameters * #_filters positions * #_filters

2160 = (3x3x3) * (4x4) * 5

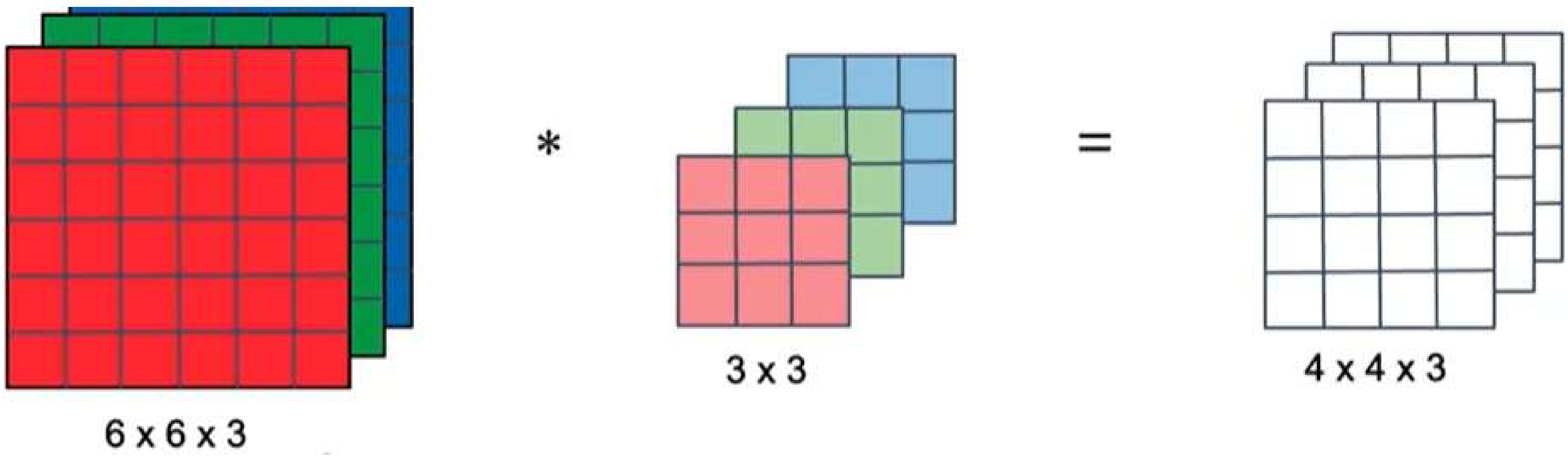
Normal vs Depthwise Separable Convolution



Depthwise separable convolution has two steps:

1) Depthwise convolutions
2) Pointwise convolutions

Step 1: Depthwise Convolution

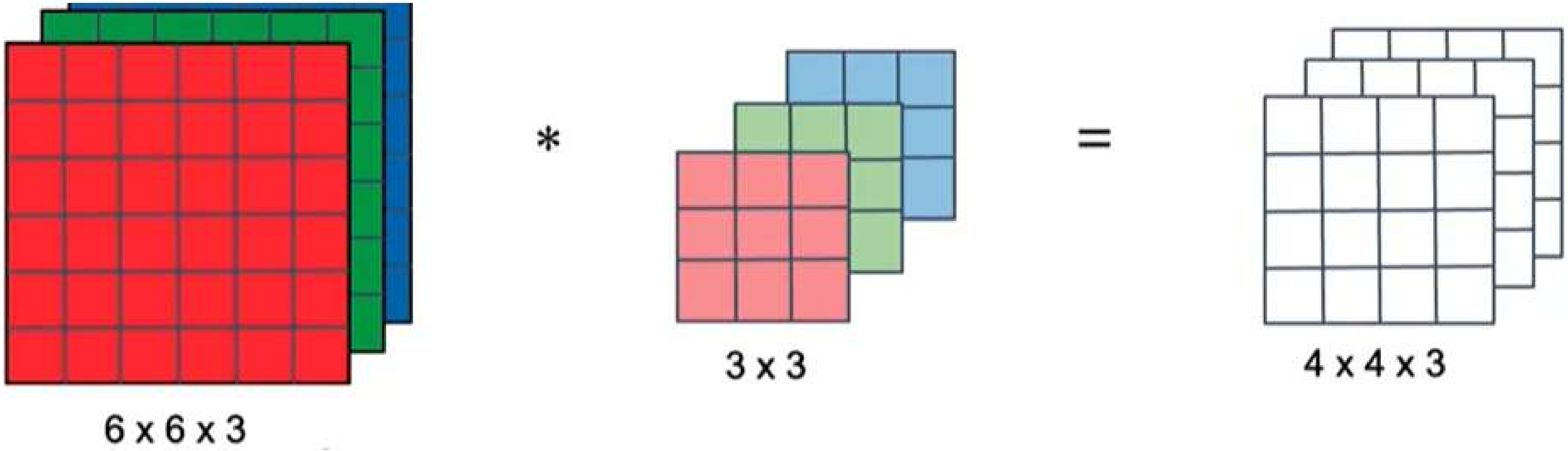


Depthwise convolution rule:

- Each filter has 2 dimensions (height and width)
- \# of filters= 3rd dimension of the input volume (input channels)

Computational cost (number of computations) = ?

Step 1: Depthwise Convolution



Depthwise convolution rule:

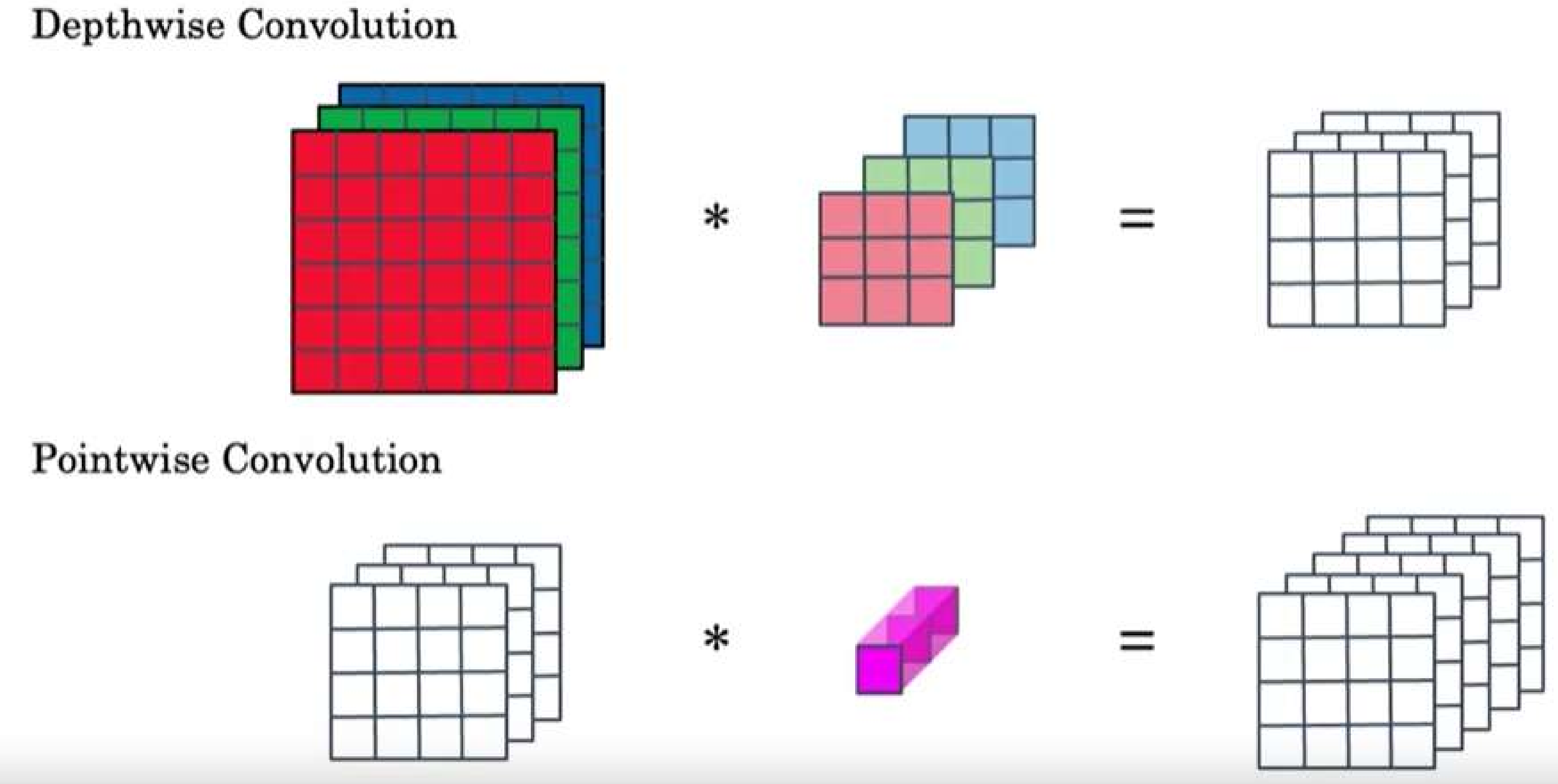
- Each filter has 2 dimensions (height and width)
- # of filters= 3rd dimension of the input (input channels)

Computational cost 1m =

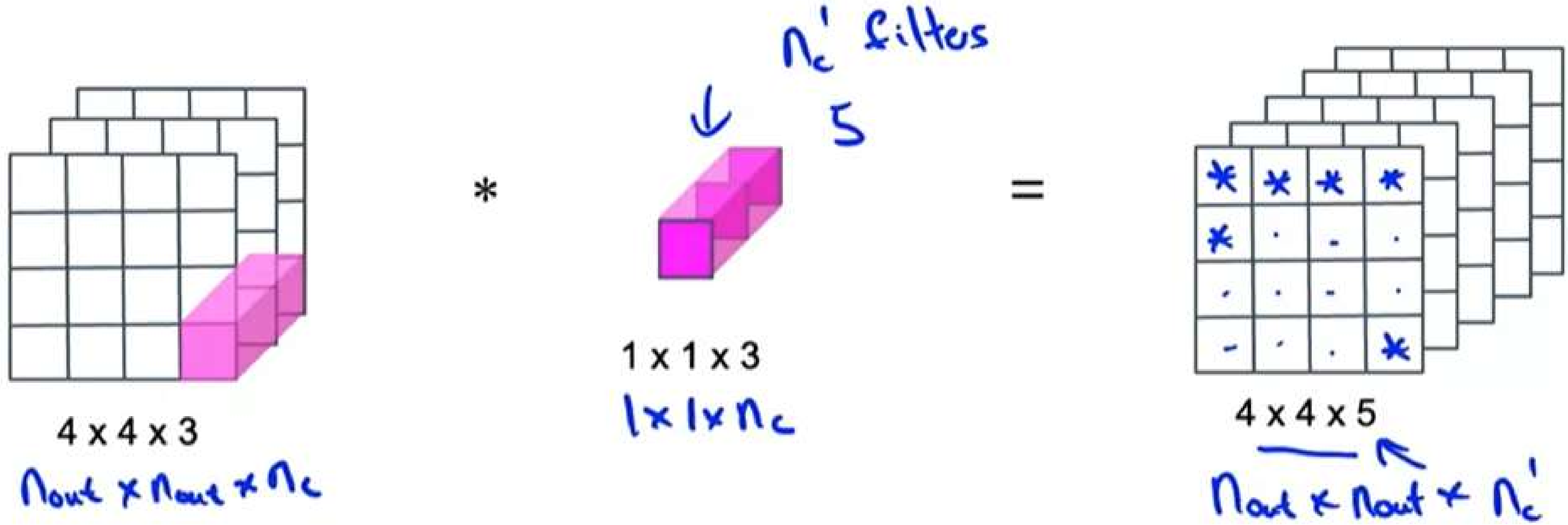
#_filter parameters * #_filters positions * #_filters

Step 1: 432 = (3x3) * (4x4) * 3

Step 2: Pointwise Convolution



Step 2: Pointwise Convolution

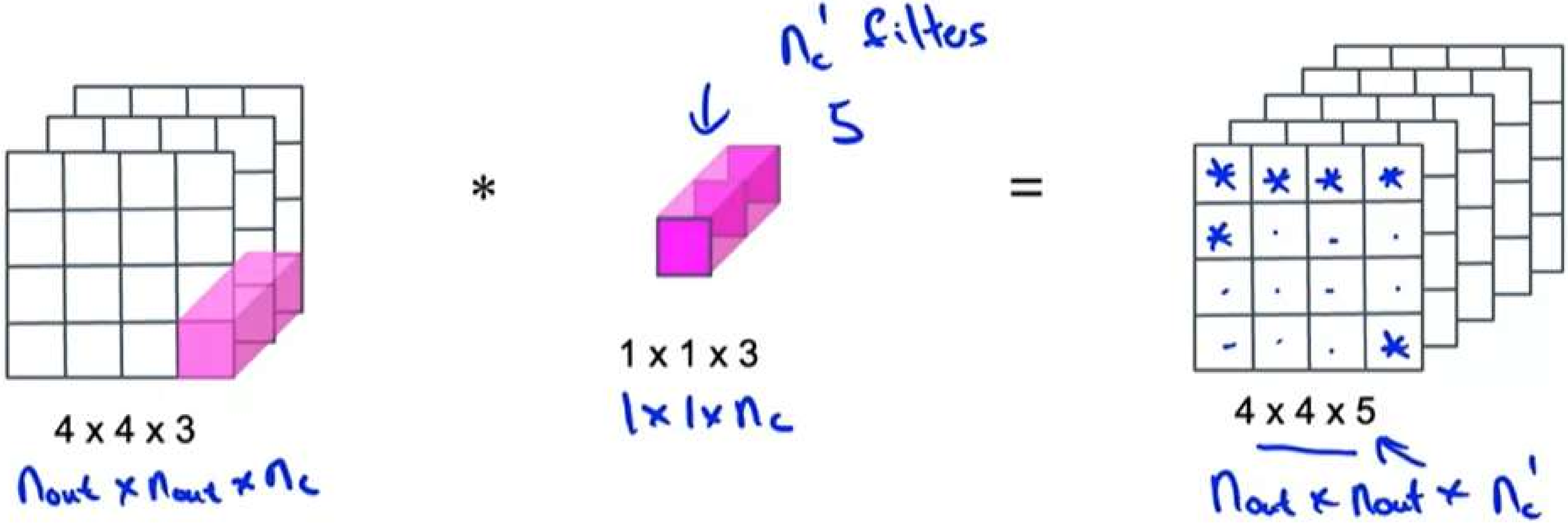


Computational cost 2 =

#_filter parameters * #_filters positions * #_filters

Computational cost 2 = ?

Step 2: Pointwise Convolution



Computational cost 2 =

#_filter parameters * #_filters positions * #_filters

Step 2: 240 = (1x1x3) * (4x4) * 5

Cost Summary

For the particular example:

Normal convolution: 2160

Depthwise separable convolution: 432+240=672

Ratio =672/2160=0.31 (31% as compared to Normal convolution)

In the paper *Andrew G. Howard at al., 2017, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.*

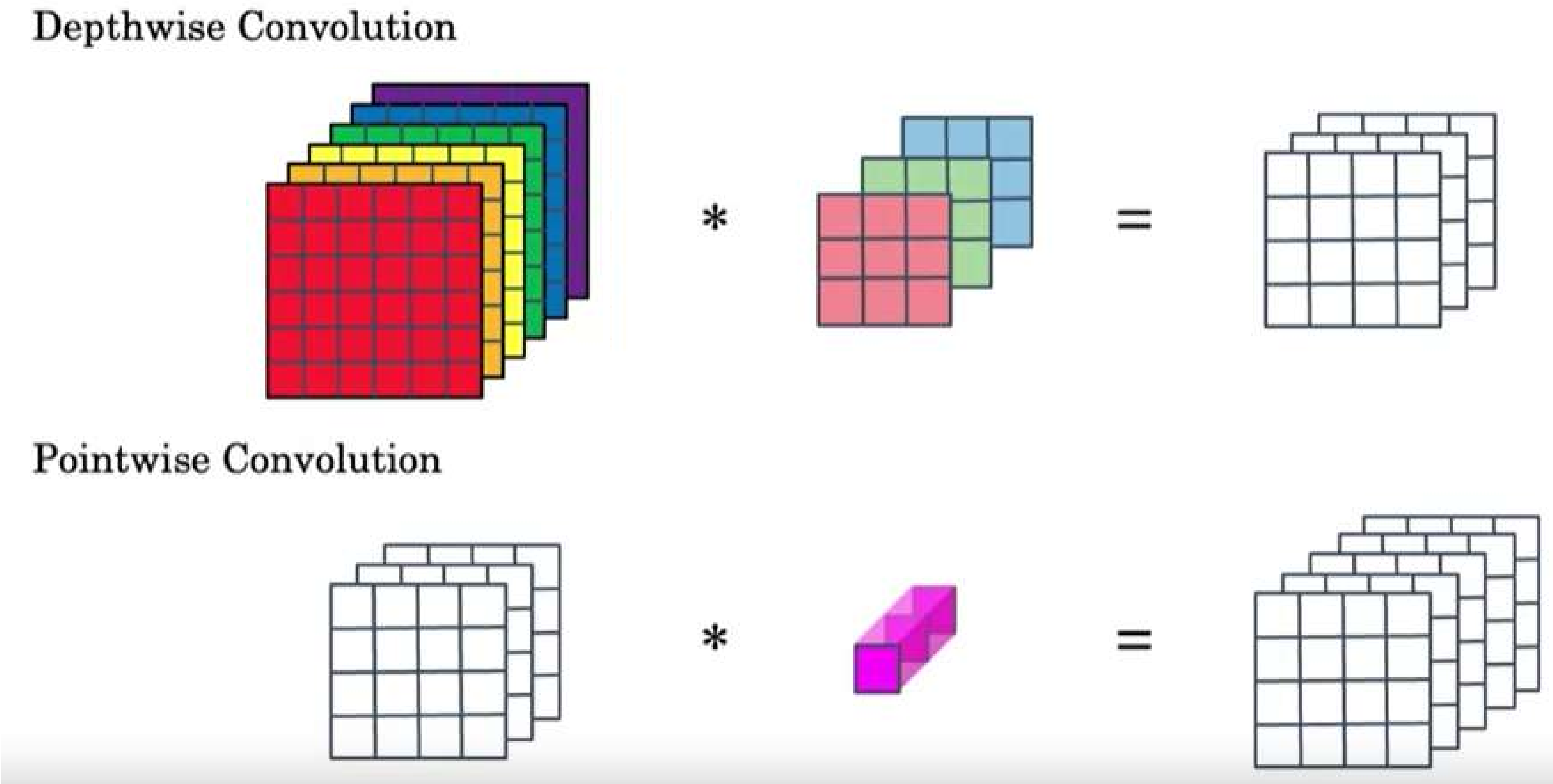
General Ratio formula = 1/output channels +1/(filter_dimension^2)

In our case: 1/5+1/(3^2)

Typical ratio: 1/512 + 1/(3^2) => 0.11

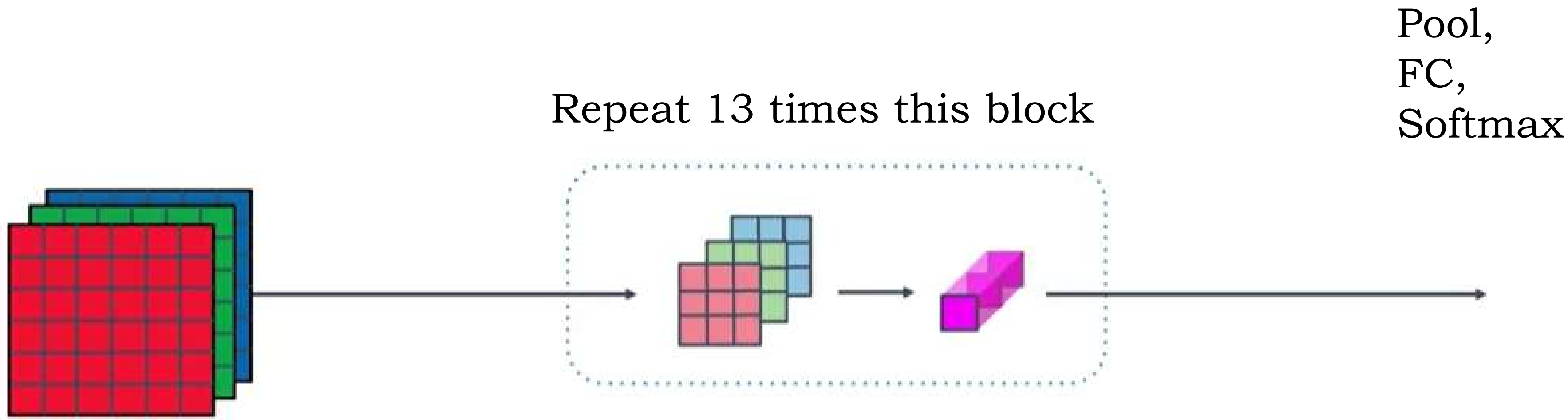
Depthwise Separable Convolution about 10 times cheaper (less computations) than Normal convolution

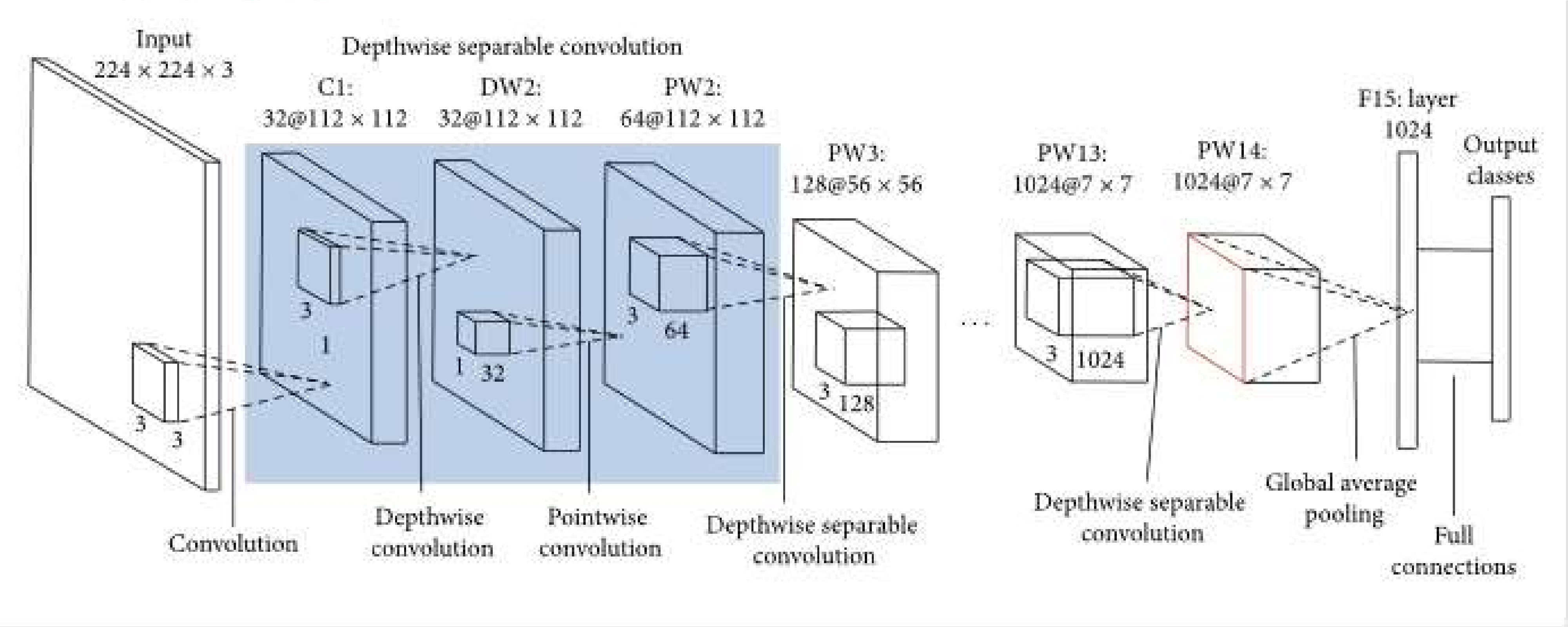
Depthwise Separable Convolution



For more than 3 input channels (*n_input_channels*) we still use the same depthwise convolution icon, but will apply *n_input_channels* filters.

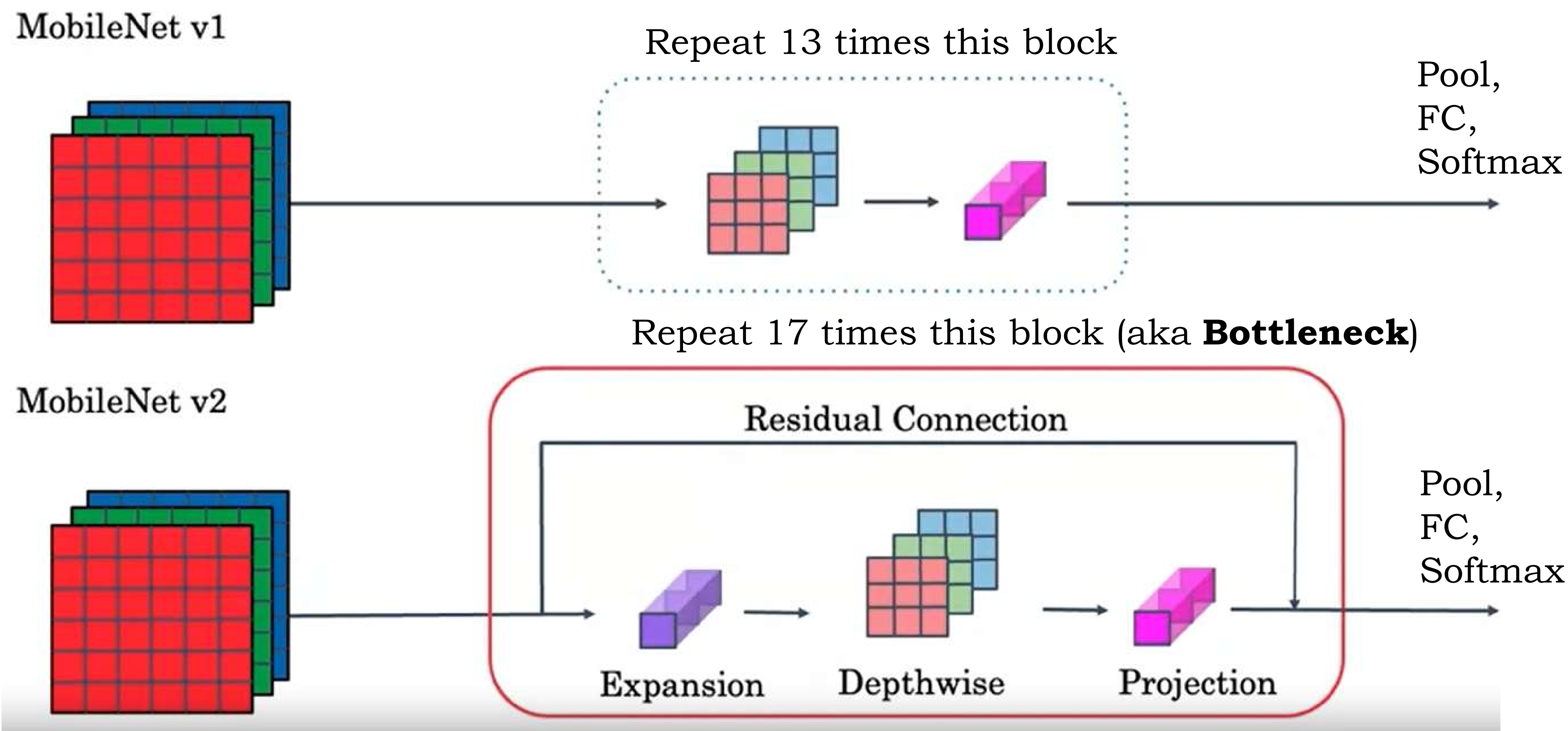
MobileNet v1





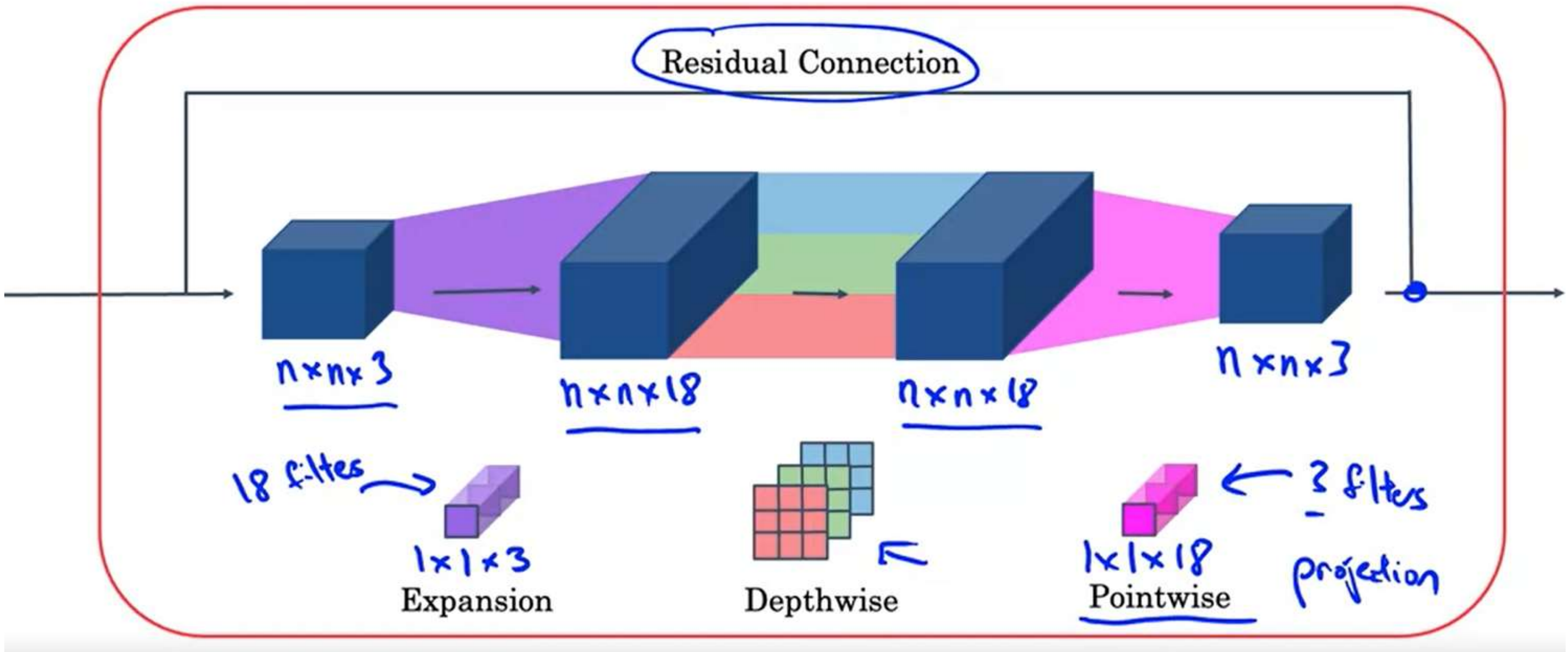
Andrew G. Howard at al., 2017, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.

MobileNet v1 and v2 architectures



Mark Sandler, Andrew Howard, et al. 2019, MobileNetV2: Inverted Residuals and Linear Bottlenecks.

MobileNet v2 Bottleneck



Expansion: The input of the bottleneck block is a small volume (nxnx3) , apply 18 filters (i.e. typical factor of expansion 6) , we get big (expanded) volume (nxnx18).

Depthwise: Keep the dimension of the volumes by adding padding.

Pointwise: convolve 3 (1x1x18) dimensional filters, end up with nxnx3 output volume. In this last step, we project from big volume (nxnx18) down to smaller volume (nxnx3).

MobileNet v2 – advantages

The bottleneck block with the expansion learns richer and more complex functions, while the size of the activations to pass from layer to layer is kept small => relatively low memory required.

MobileNet v2 can get a better performance than MobileNet v1, and still use a modest amount of computing and memory resources.

Mark Sandler, Andrew Howard, et al. 2019, MobileNetV2: Inverted Residuals and Linear Bottlenecks.

Tan & Le, 2020, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

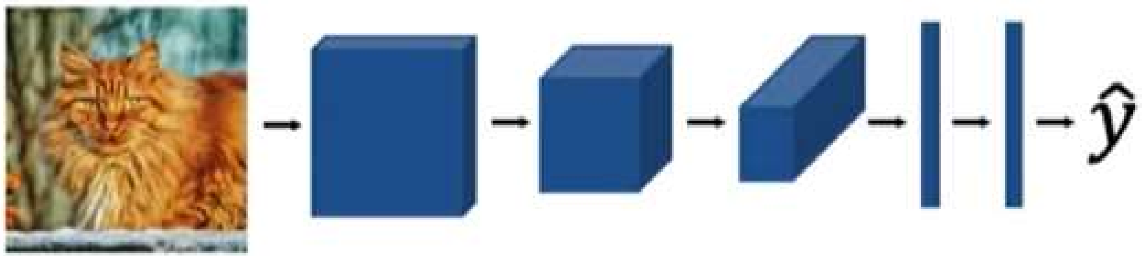
Efficient Net

Major parameters to vary (up or down) to satisfy computacional limits:

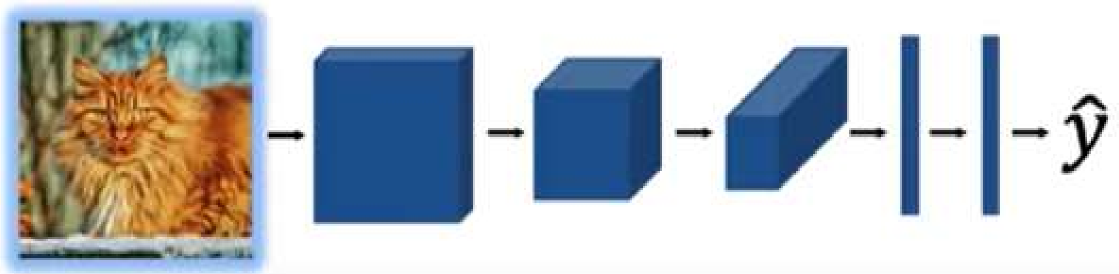
1) Image resolution (r)

2) NN Depth (d)

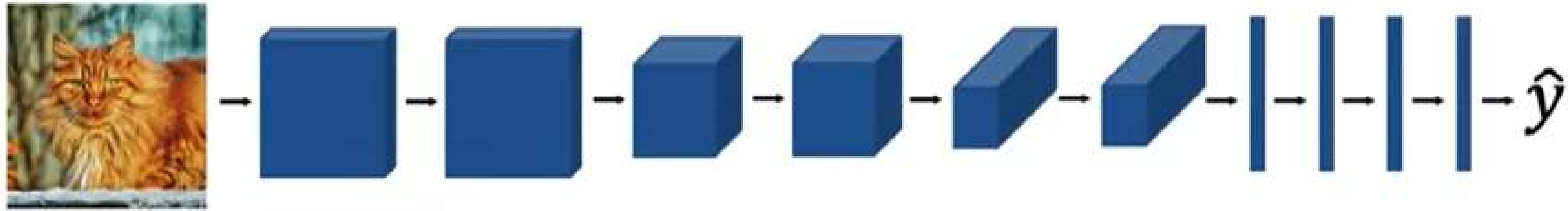
Baseline



Higher Resolution



Deeper

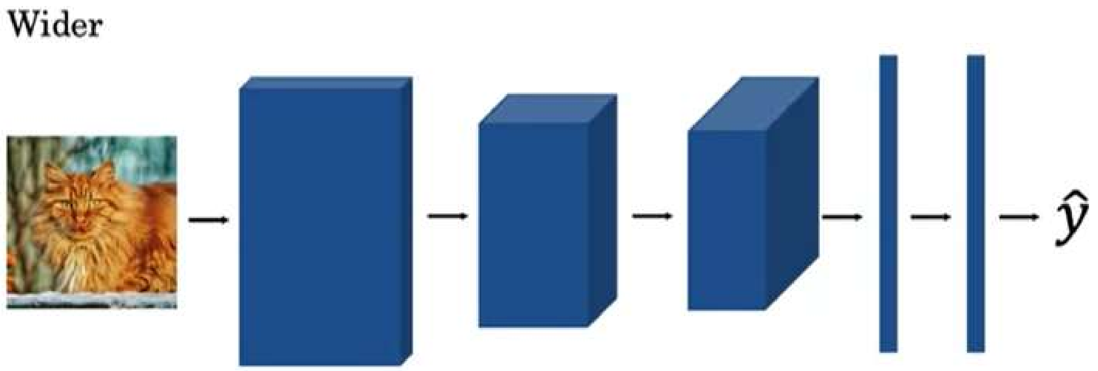


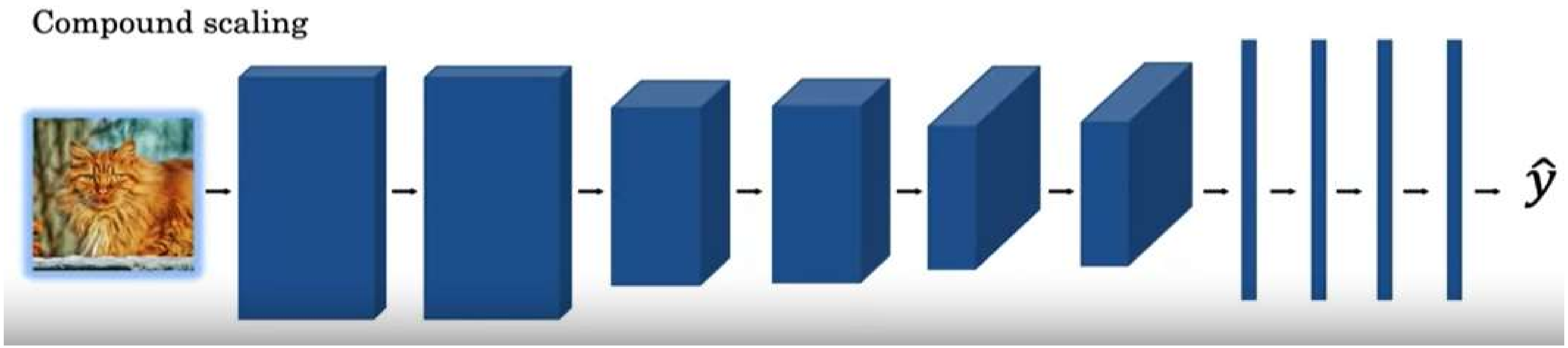
Efficient Net

Major parameters to vary (up or down) to satisfy computacional limits:

3**) Width of the layers (w)**

4) Compound scaling: Simultaneously scale up or down image resolution (r), depth (d), width (w).

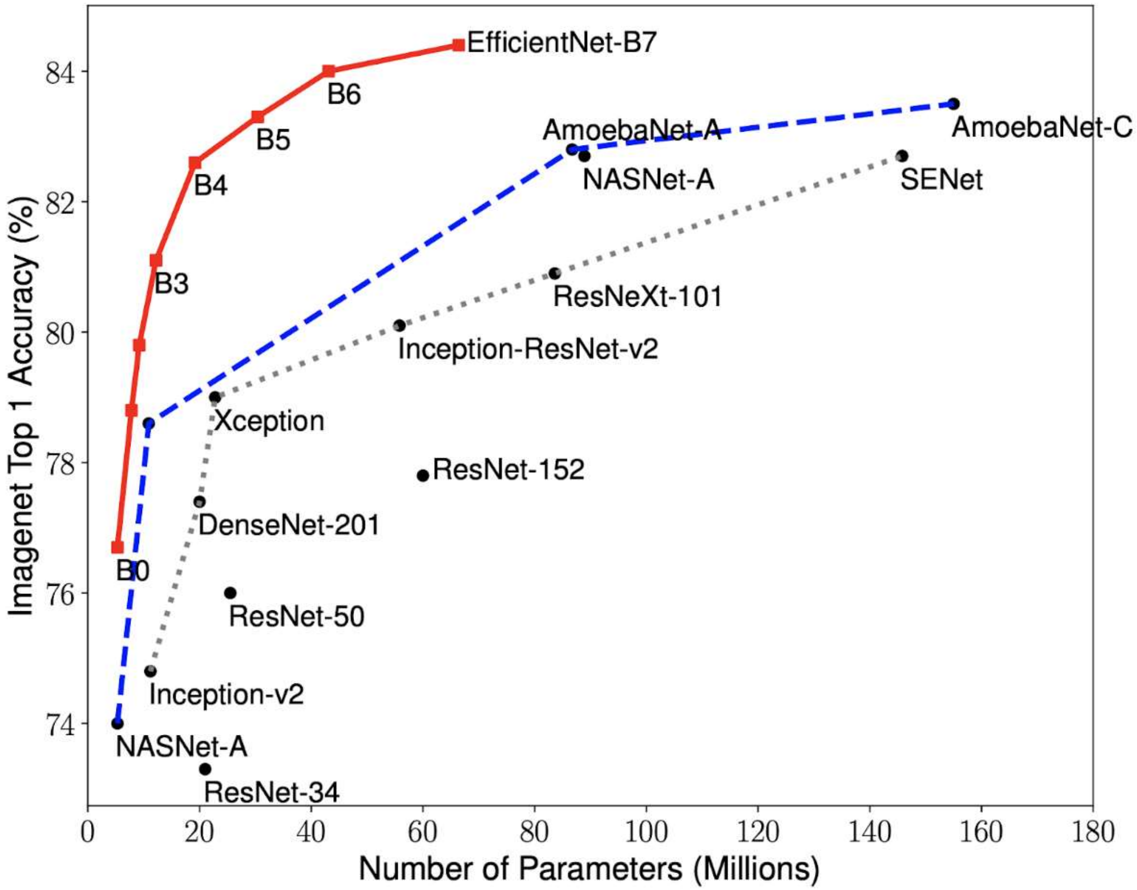




Efficient Net

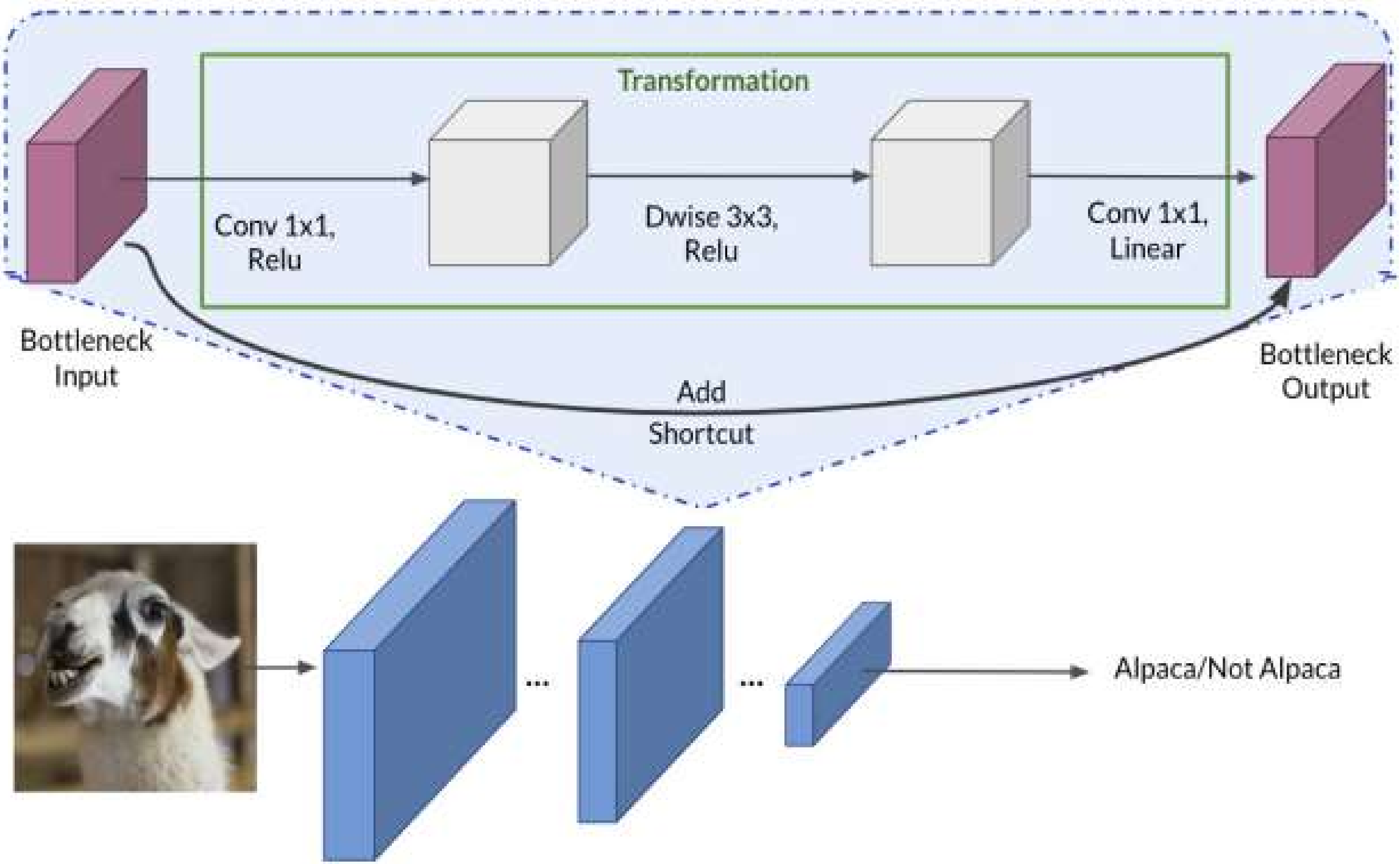
Suggestion: look at EfficientNet implementations, to choose a trade-off between r, d, w.

Unlike conventional practice that arbitrary scales these factors, the EfficientNet uniformly scales network width, depth, and resolution with a set of fixed scaling coeff.



EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling
Tan, and Le, Google AI, 2019

Lab: MobileNetV2 for Transfer Learning

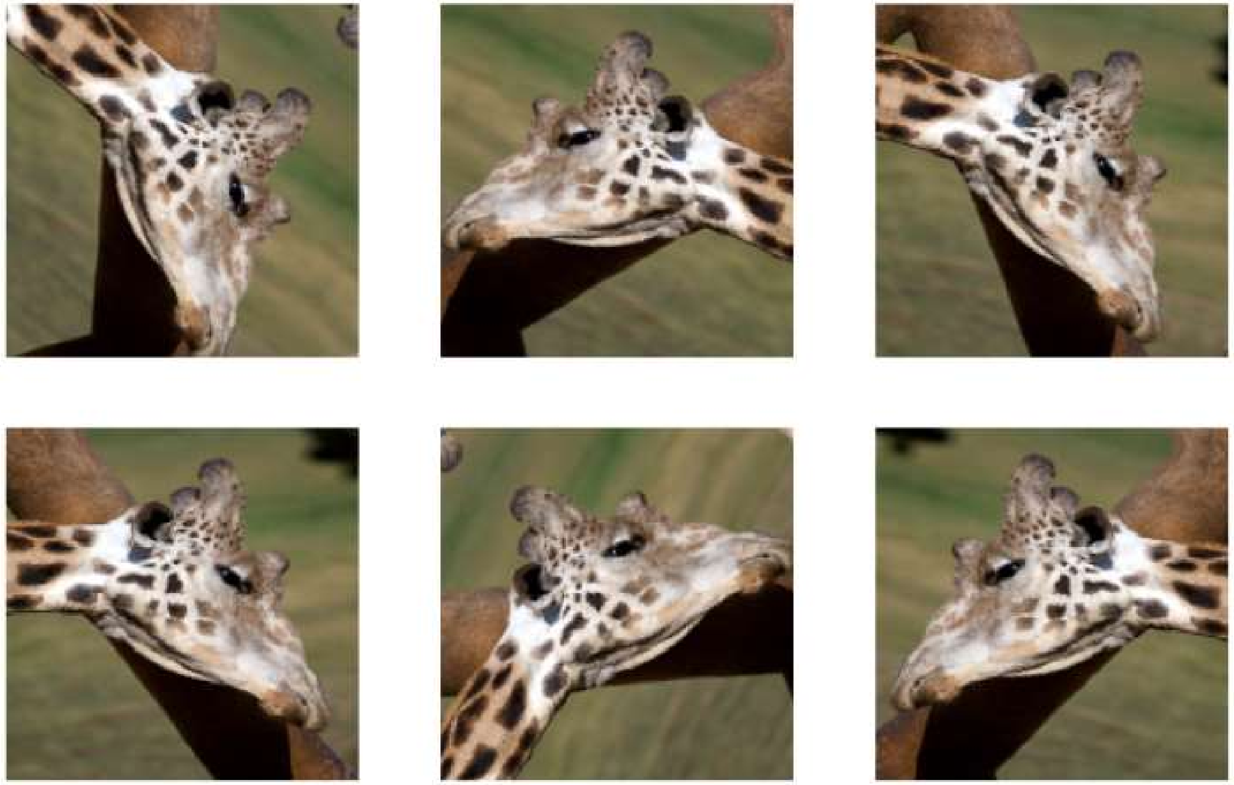


Use a pre-trained MobileNetV2 to build a binary classifier (Image of Alpaca (Lama) /Not Alpaca animal).

Lab Data

262 training images; 65 validation images

Lab: On-line Data Augmentation



Data Prefetch : CPU loads stream of images (e.g. 32) coming from the hard disc and generate distortions to form mini-batches that are passed to the training algorithm.
The two processes (data augmentation and training run in parallel.

```
Create a Sequential model composed of 2 layers
'''
data_augmentation = tf.keras.Sequential()
data_augmentation.add(RandomFlip("horizontal"))
data_augmentation.add(RandomRotation(0.2))
```

Part 2: Object Detection

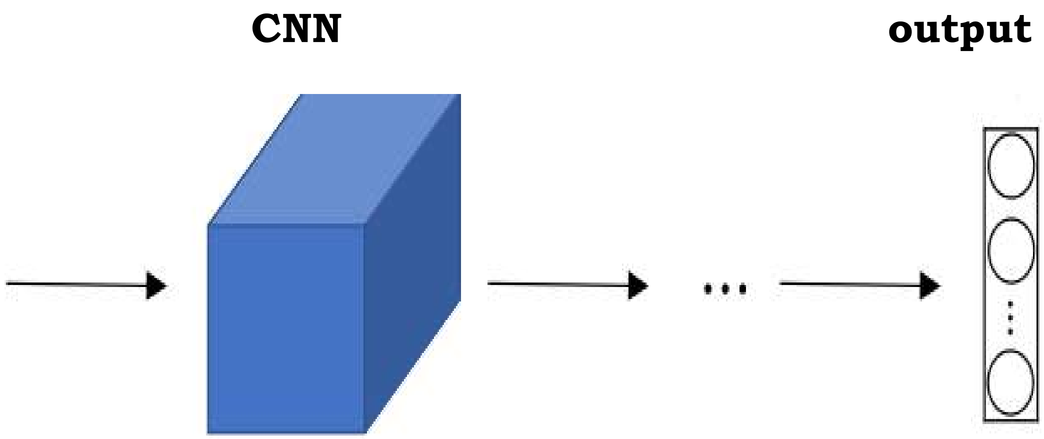
Image classification/localization/detection

Image classification	Classification & Localization	Detection
		
	b_x, b_y, b_h, b_w	

car **car + bounding box** **many objects+ bounding boxes**

Image classification: input a picture to CNN and the output is a class label (e.g. person, bike, car, background, etc.)
Classification with localization: the algorithm gives not only the class label of the object but also draws a bounding box (the coordinates) of its position in the image.
Standard notation: (0,0) as the upper-left corner and (1,1) to be the lower-right corner.
(b_x , b_y , b_h , b_w) describes the bounding box.

Object classification with localization



Classes (e.g.):
1. person
2. Bikes
3. Car
4. Background (no object)

b_x, b_y, b_h, b_w

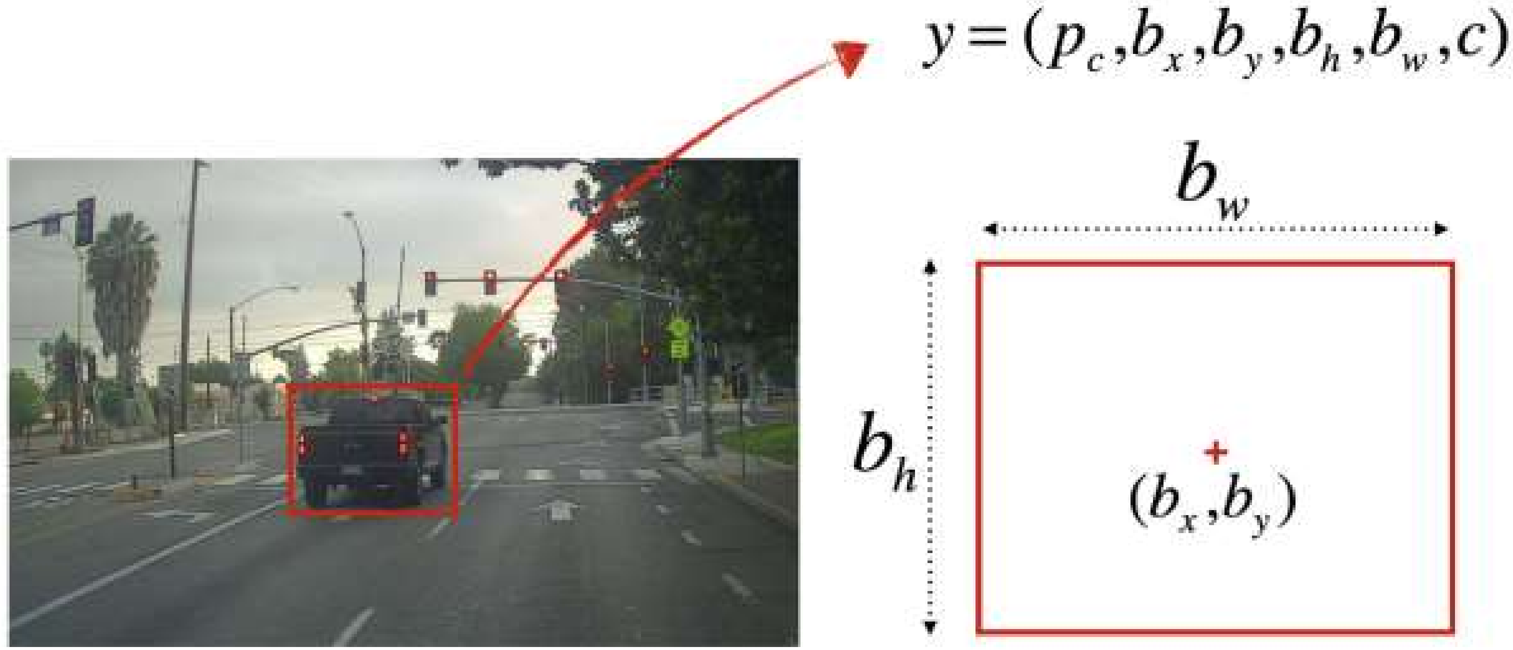
Output: (p_c , b_x , b_y , b_h , b_w , c= [c_1, c_2….c_{end}])
p_c – is there an object or not (1/0)

<= Image label: [1 , b_x , b_y , b_h , b_w , 0, 0, 1]

<= Image label: [0 , ? , ? , ? , ? . ?, ?; ?]
? – “don’t care”

universidade de aveiro

Output/label vector



$p_c = 1$: confidence of an object being present in the bounding box

$p_c \quad b_x \quad b_y \quad b_h \quad b_w$ ⟵ 80 class probabilities ⟶

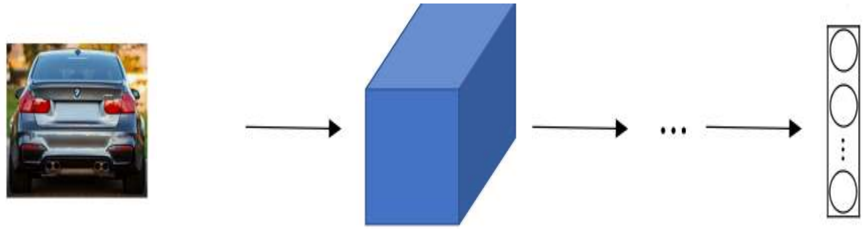
Landmark Detection

The idea of bounding boxes (b_x, b_y, b_h, b_w) inspired the use of DL for e.g. emotion recognition from faces, person's pose detection.
CNN can output the important coordinates (called landmarks) of targeted objects in the image. For example 64 chosen points on the face, or the body (key positions in the persons pose, e.g. the mid point of the chess, left/right shoulder, etc.).
Need to manually label all landmarks in the training data !!!
Consistent annotation over several images.

b_x, b_y, b_h, b_w

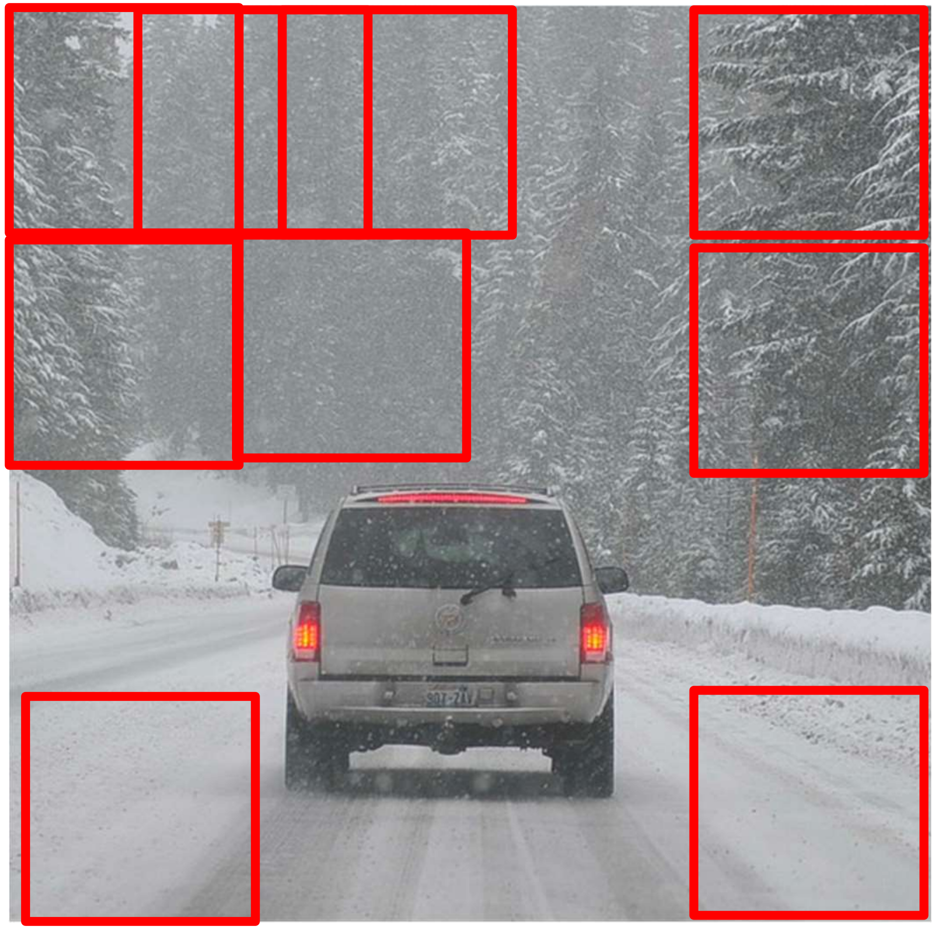
$$\text{Image} \longrightarrow \text{ConvNet} \longrightarrow \underbrace{\begin{bmatrix} \circ \\ \circ \\ \circ \end{bmatrix} \text{Output: face}, l_{1x}, l_{1y}, \ldots, l_{64x}, l_{64y}}_{\text{total of 129 outputs}}$$

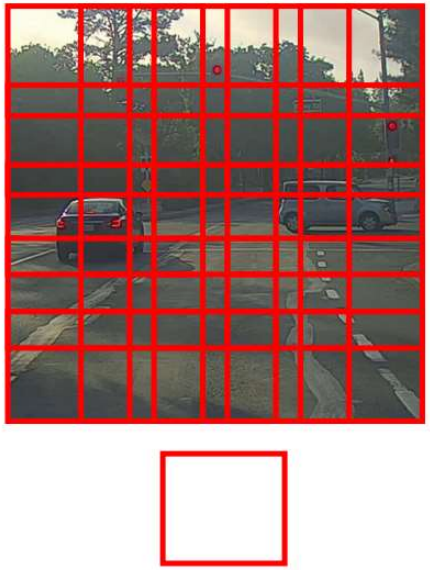
Object Detection algorithm

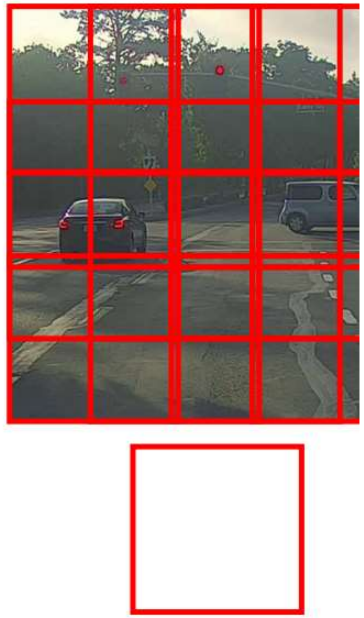


Let's say we want to build a car detection algorithm.
First create a labelled training set.
Take a picture, cut out anything else that's not part of a car and get the car centred in pretty much the entire image (cropped examples of cars).
Train a CNN to output y (0 or 1, is there a car or not).
The trained CNN is used in the Sliding Windows detection algorithm.

Sliding windows detection algorithm



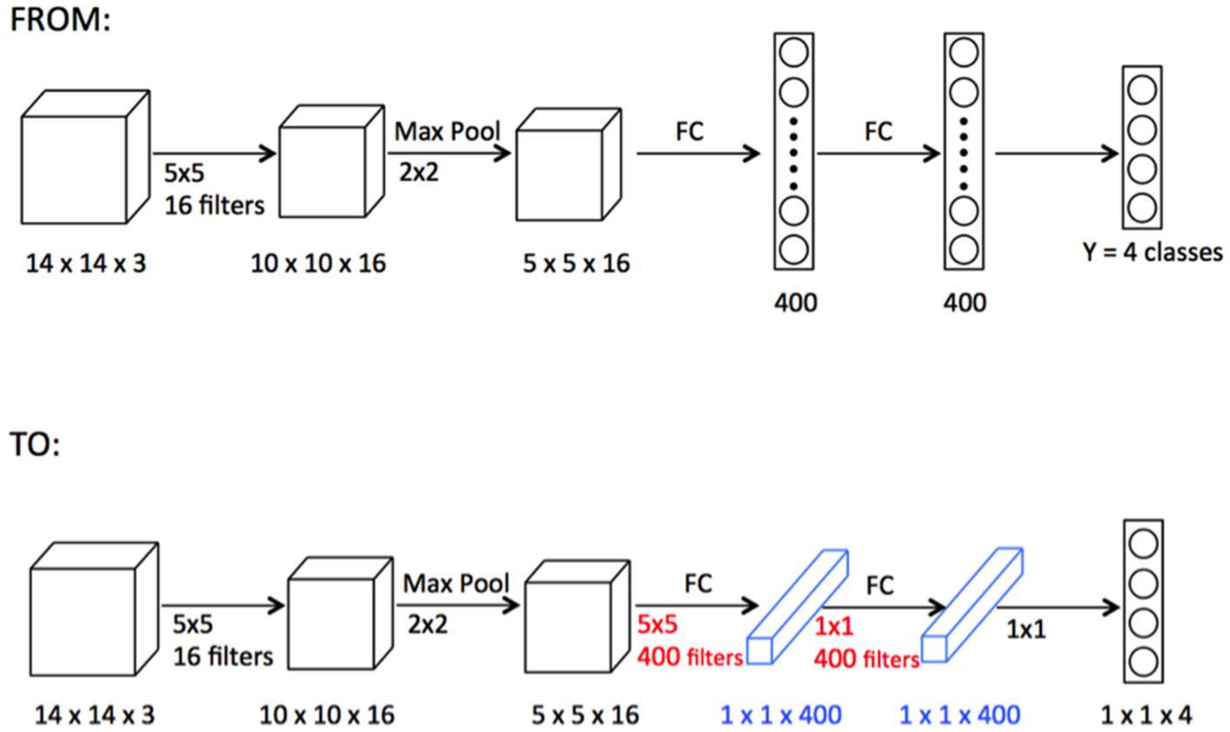




Pick a certain window size, input this sub-picture into <u>already trained CNN to detect objects</u>. Then shift the detection window to the right with one step (stride) and feed the new sub-picture into ConvNet. Go through every region (the stride needs to be small for the detection algorithm to run smoothly). Repeat the same with different sizes of the detection window in order to detect objects with different sizes of the picture.
The Sliding Windows object detection algorithm has **infeasibly <u>high computationally cost.</u>**

<u>Solution: Convolutional Implementation of sliding windows</u>

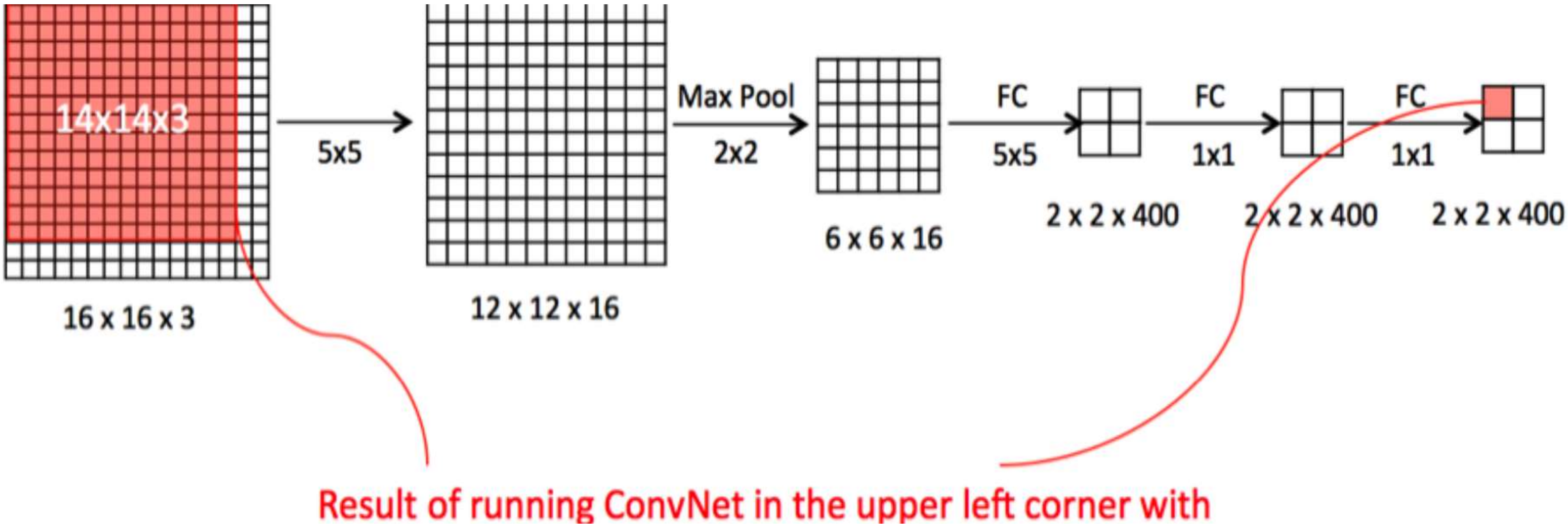
Turning FC layers into convolutional layers



The conv layers are the same in both implementations.
The difference is that the Fully Connected (FC) layers are implemented as convolutional layers.

Ex. _The first FC layer:_ Let the input volume is 5x5x16, convolve it with 5x5x16 filter.
The outputs will be 1x1. Add 400 of these 5x5x16 filters, then the output dimension will be 1x1x400. Rather than seeing the FC as a set of 400 nodes, we view it as a 1x1x400 volume.
The second FC layer: implemented as convolution with 400 1x1 filters and will output 1x1x400 volume.
The last layer: is implemented as convolution with 4 (if we have 4 classes) 1x1 filters, followed by a softmax activation.

Convolutional Implementation of Sliding Windows



The original sliding windows algorithm, takes one window (e.g. 14x14x3) and run it through the CNN, then take the next region and so on.
With the convolutional implementation instead of doing it sequentially, the entire image (16x16x3) is input and convolutionally make all the predictions at the same time by one forward pass through the CNN.

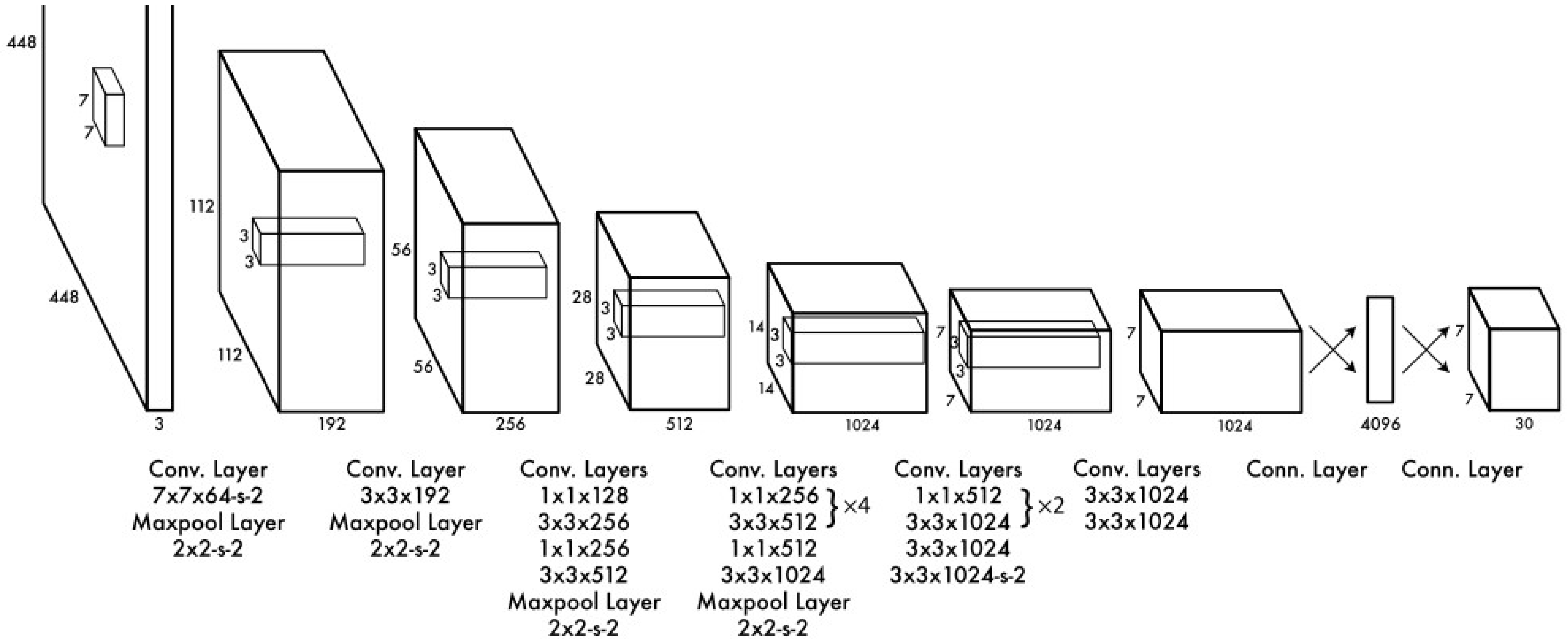
Pierre Sermanet et al. , 2014, OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks

Bounding Box Predictions

With the Sliding Windows algorithm, it may happen that none of the boxes really match perfectly with the position of the target object.
For example, we want to detect the car in this picture. The sliding windows (since we initially set the windows sizes and we don’t not know how big the car is) may match only part of the car.
In some cases, the object may look like a rectangle instead of square.

Solution: YOLO (You Only Look Once) algorithm is a way to output more accurate bounding boxes.

YOLO (You Only Look Once)



YOLO - CNN network for both classification and localising the object using bounding boxes.
24 convolutional layers + 2 fully connected layers.
Conv layers pretrained on ImageNet dataset.

*Redmon et al, 2015, “You Only Look Once: Unified, Real-Time Object Detection” (https://arxiv.org/abs/1506.02640)
Redmon & Farhadi, 2016 (https://arxiv.org/abs/1612.08242).