

Capstone design

Team F

2016310237 김동우
2016314577 김동한
2016311033 김영현
2017312329 최형규

Contents

1. Introduction

- AI adaptation on our project

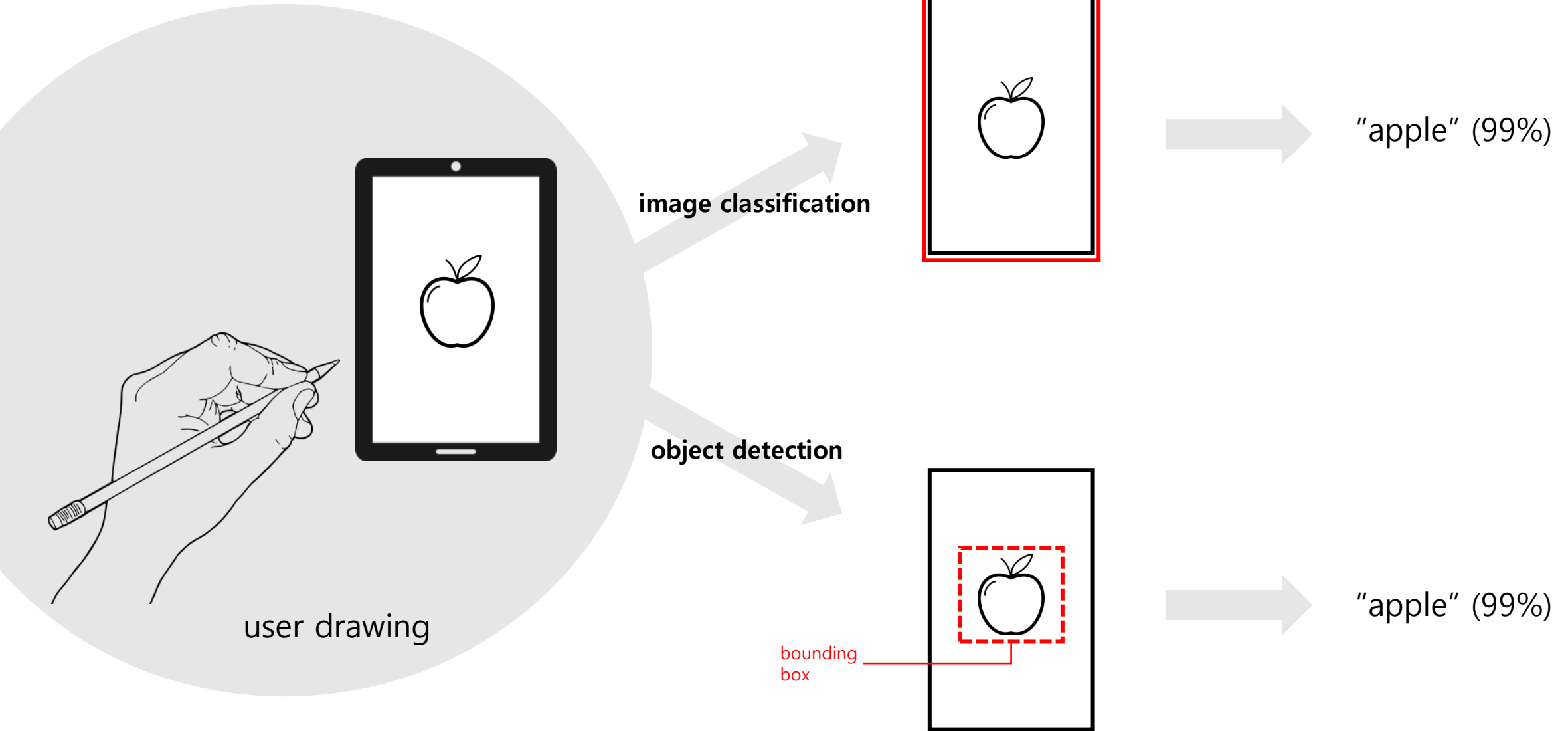
2. Image Classification

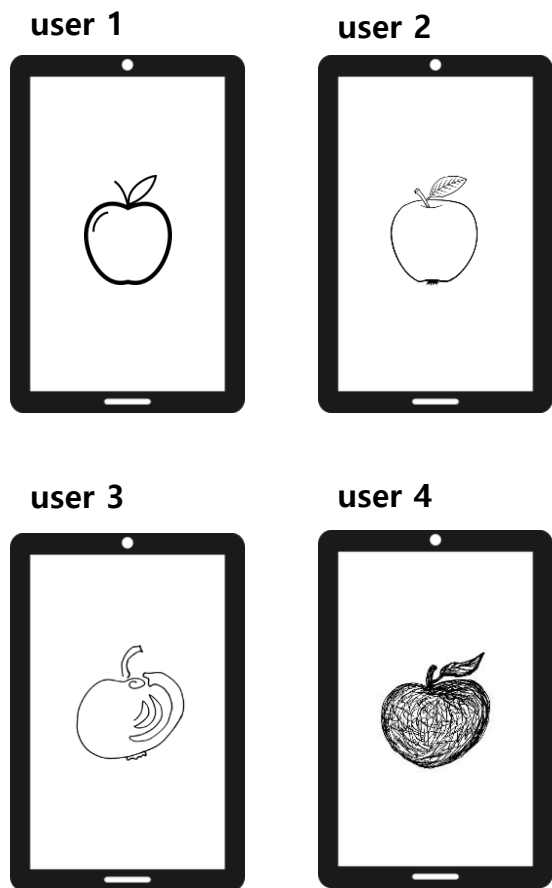
- Quick review of Image Classification
- MobileNet v1

3. Object Detection

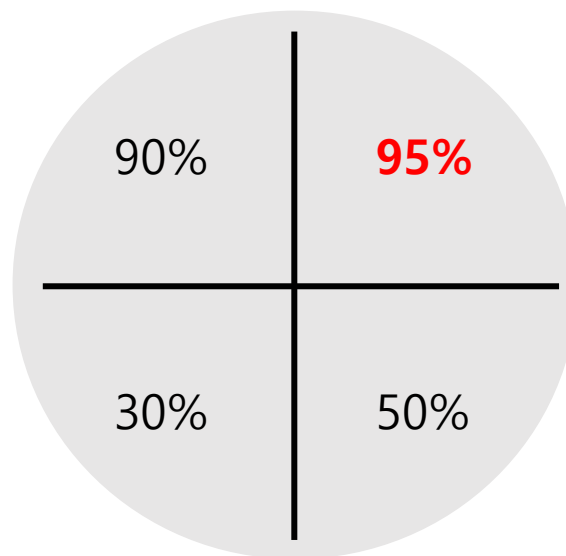
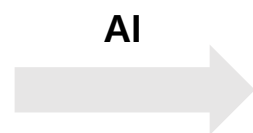
- Yolo v1
- Limitation

1. Introduction

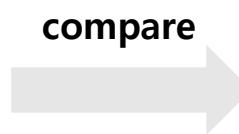




Keyword = 'apple'



Output \approx Keyword?

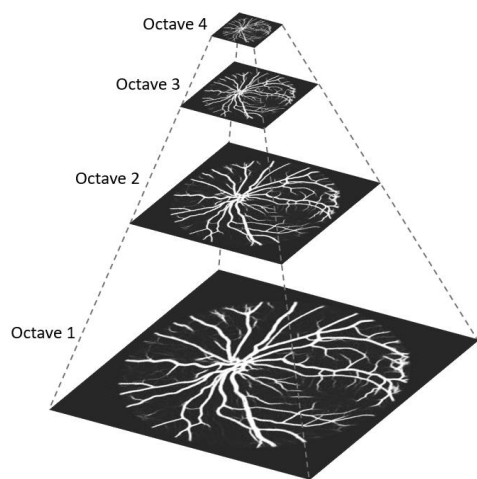


user 2
gets credit

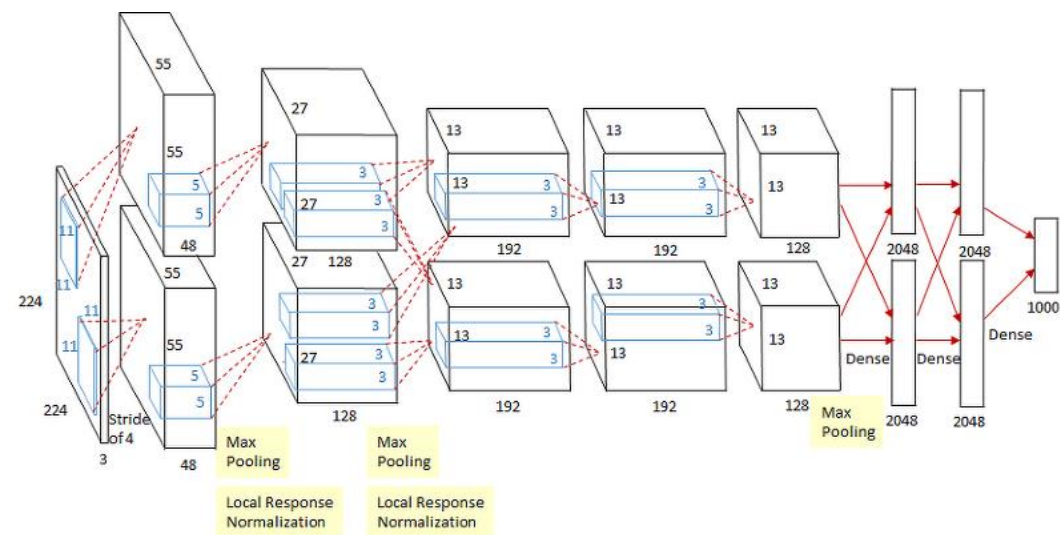
2. Image Classification



- ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
- (1000) classes + (1.35M) dataset

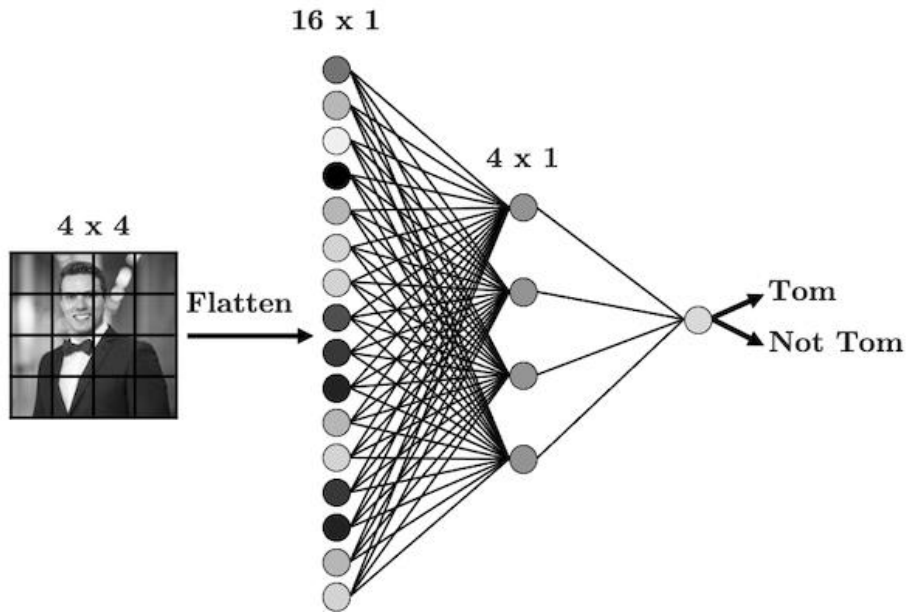


(~2011) sift/surf descriptor based algorithm



(2012~) DNN algorithm

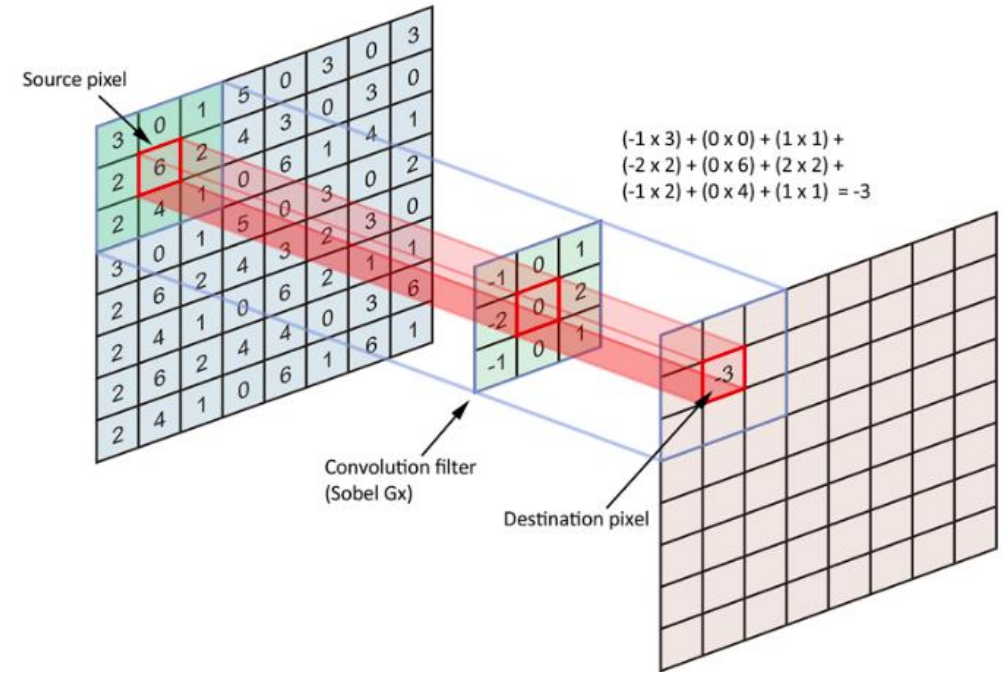
Multi-Layer Perceptron (MLP)



- flatten the input into 1-dimensional and inject into the fully connected layer
- important **spatial informations** are inherently lost

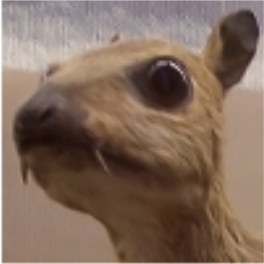
VS

convolutional Neural Network (CNN)



- more appropriate for images and video frames
- kernel slides through the entire input grid and creates a **feature map**
- keep **spatial informations**

Input image

Convolution
Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map

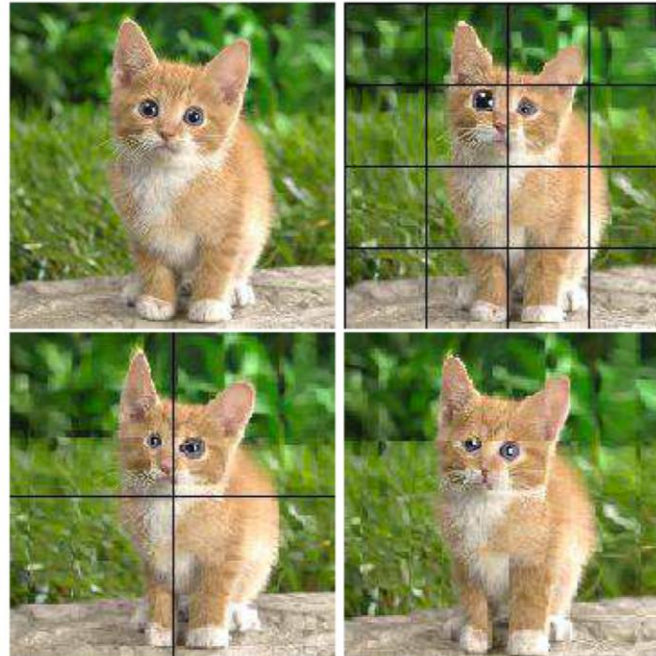


Convolution layer)

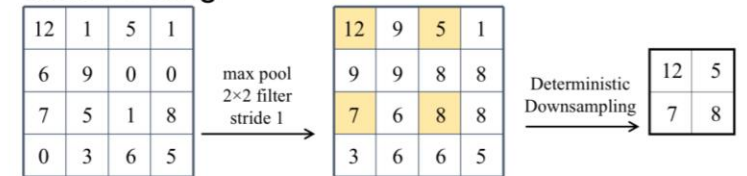
- map the input into the feature space to obtain important features of the image
- kernel size, num of kernel, stride, padding

Pooling layer)

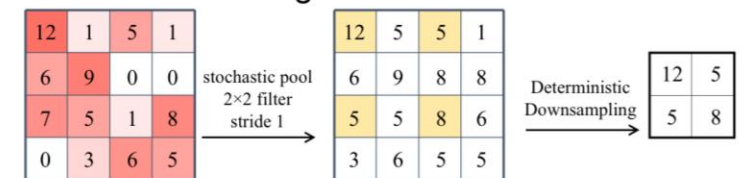
- reduce the size of the feature map
- no params for pooling layer itself
- lower params, suppress overfitting, lower computation, lower hardware resources
- Max Pooling, Stochastic Pooling..
- pooling size, stride, padding



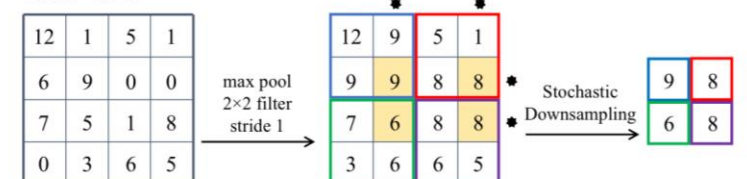
Max Pooling



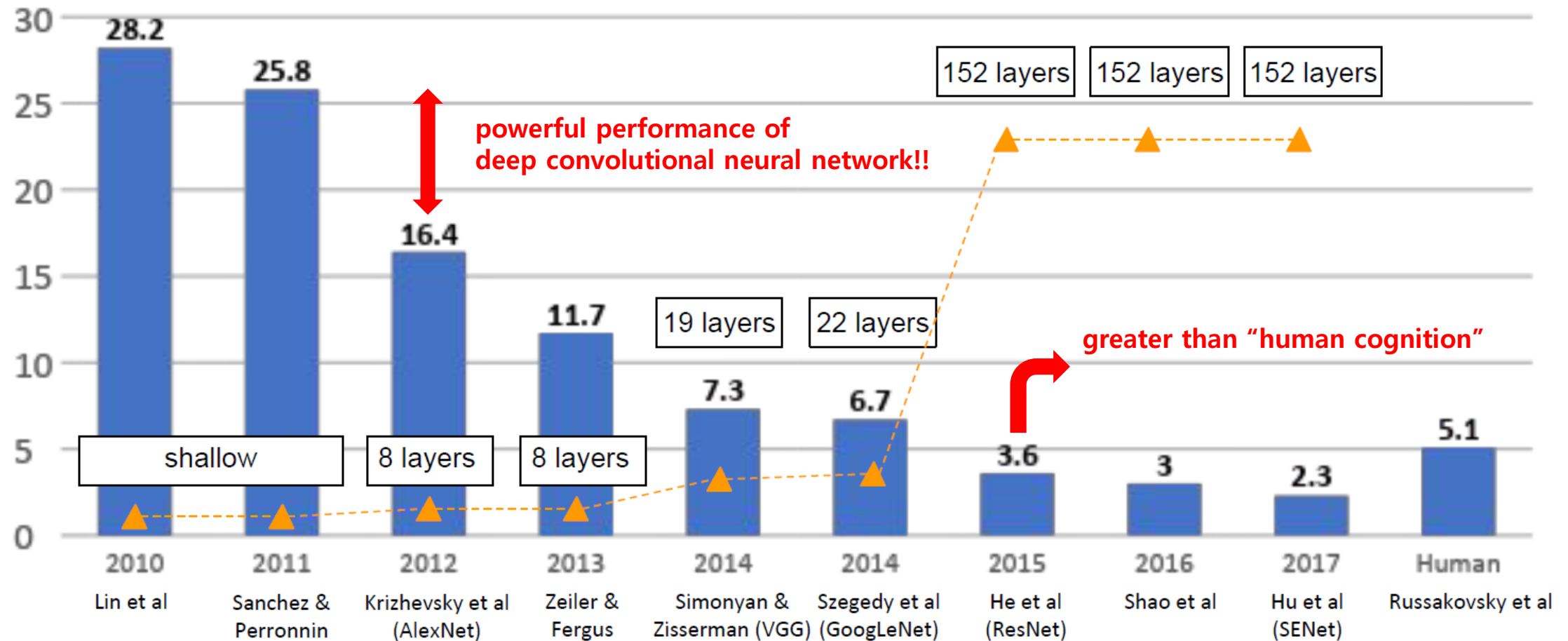
Stochastic Pooling



S3Pool



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



The diagram illustrates the VGG16 Architecture, showing the flow of data from an input image through various layers to the final classification output. The layers are represented by 3D blocks, and their dimensions are indicated by labels above them.

Legend:

- convolution+ReLU
- max pooling
- fully connected+ReLU
- softmax

Architecture Details:

- Input:** A color image of a street scene.
- Layer 1:** Convolution+ReLU, dimensions $224 \times 224 \times 3$.
- Layer 2:** Convolution+ReLU, dimensions $224 \times 224 \times 64$.
- Layer 3:** Max pooling, dimensions $112 \times 112 \times 128$.
- Layer 4:** Convolution+ReLU, dimensions $56 \times 56 \times 256$.
- Layer 5:** Convolution+ReLU, dimensions $28 \times 28 \times 512$.
- Layer 6:** Max pooling, dimensions $14 \times 14 \times 512$.
- Layer 7:** Convolution+ReLU, dimensions $7 \times 7 \times 512$.
- Layer 8:** Fully connected+ReLU, dimensions $1 \times 1 \times 4096$.
- Layer 9:** Fully connected+ReLU, dimensions $1 \times 1 \times 1000$.
- Layer 10:** Softmax, dimensions $1 \times 1 \times 1000$.

VGG16 Architecture

The diagram illustrates the MobileNetV2 architecture. It starts with an **Input** of size $224 \times 224 \times 3$. This is followed by a **Convolution** layer (C1) with a 3×3 kernel, resulting in a $32 \times 112 \times 112$ feature map. The main body of the network consists of several **Depthwise separable convolution** blocks. Each block is composed of:

- Depthwise convolution**: Applying a 3×3 kernel to each channel separately, resulting in a $32 \times 112 \times 112$ feature map.
- Pointwise convolution**: A 1×1 convolution that mixes the channels, resulting in a $64 \times 112 \times 112$ feature map.
- Depthwise separable convolution**: Another 3×3 depthwise convolution, resulting in a $128 \times 56 \times 56$ feature map.

 This sequence is repeated with varying kernel sizes and channel counts (e.g., $1024 \times 7 \times 7$ for PW13 and PW14). The final feature map is processed by an **F15: layer** (1024 units), followed by **Global average pooling** and **Full connections** to produce the **Output classes**.

ResNet-50

224 × 224 × 3 224 × 224 × 64

112 × 112 × 128

56 × 56 × 256

28 × 28 × 512

14 × 14 × 512

7 × 7 × 512

1 × 1 × 4096 1 × 1 × 1000

convolution+ReLU
max pooling
fully connected+ReLU
softmax

VGG16 Architecture

The diagram illustrates the MobileNet architecture, which is designed for efficiency through depthwise separable convolutions. The process begins with an **Input** of size $224 \times 224 \times 3$. This input passes through a **Convolution** layer (kernel size 3×3) to produce a feature map of size $1 \times 112 \times 112$. This is followed by a **Depthwise convolution** layer (kernel size 3×3) to produce a feature map of size $32 \times 112 \times 112$. The architecture then enters a series of **Depthwise separable convolution** blocks, each consisting of a **Depthwise convolution** and a **Pointwise convolution**. The first block (C1) has a depthwise convolution of size $1 \times 112 \times 112$ and a pointwise convolution of size $32 \times 112 \times 112$. The second block (DW2) has a depthwise convolution of size $1 \times 112 \times 112$ and a pointwise convolution of size $64 \times 112 \times 112$. The third block (PW3) has a depthwise convolution of size $3 \times 56 \times 56$ and a pointwise convolution of size $128 \times 56 \times 56$. This is followed by a fourth block (PW13) with a depthwise convolution of size $3 \times 7 \times 7$ and a pointwise convolution of size $1024 \times 7 \times 7$. The final block (PW14) has a depthwise convolution of size $3 \times 7 \times 7$ and a pointwise convolution of size $1024 \times 7 \times 7$. The output of the final block is a feature map of size $1024 \times 7 \times 7$. This feature map is then processed by a **Global average pooling** layer, followed by a **Full connections** layer (F15: layer 1024) to produce the **Output classes**.

ResNet-50

MobileNet v1

Previous models...

- have tremendous number of parameters(weights)
- require high computational power & memories
- e.g. ALPHAGO used 1202 CPUs and 176 GPUs

However in real world...

- numerous environments s.t...
 - no GPUs
 - only one CPU, lack of memories

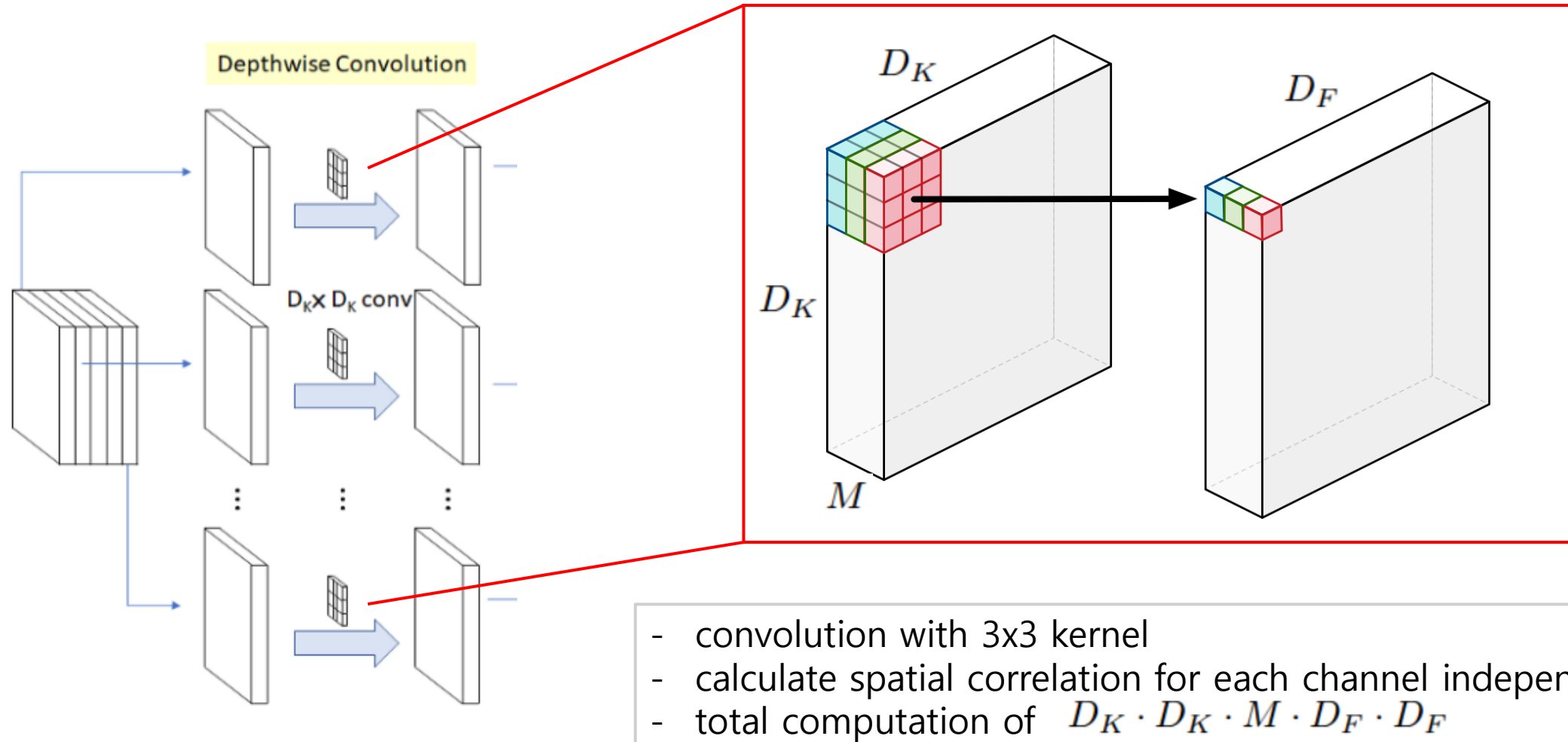


Figure 1. MobileNet models can be applied to various recognition tasks for efficient on device intelligence.

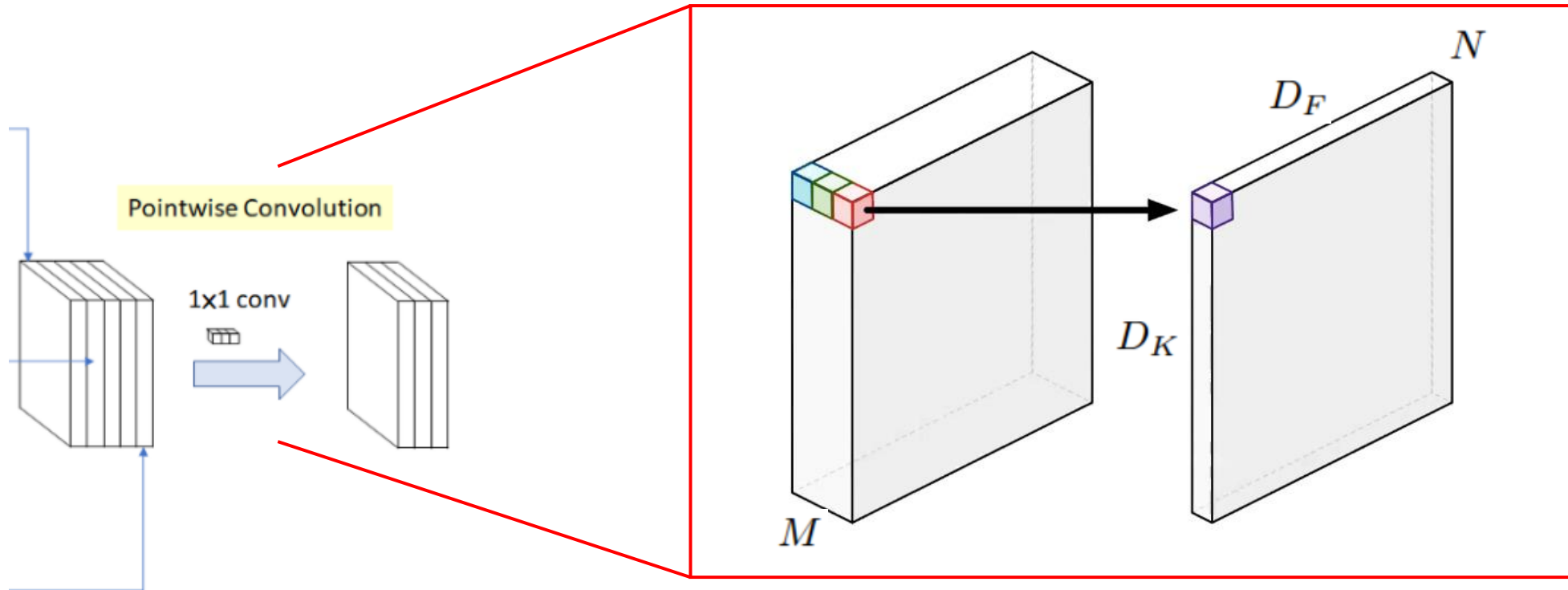
MobileNet is exactly for these situations

- small num of parameters
- model with less complexity
- low latency & high speed

1) Depthwise convolution



2) Pointwise convolution



- convolution with 1×1 kernel \Rightarrow control the size of out-channel
- calculate cross-channel correlation
- total computation of $M \cdot N \cdot D_F \cdot D_F$

3) Depthwise separable convolution

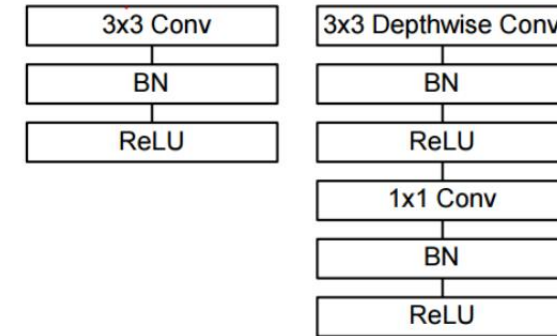
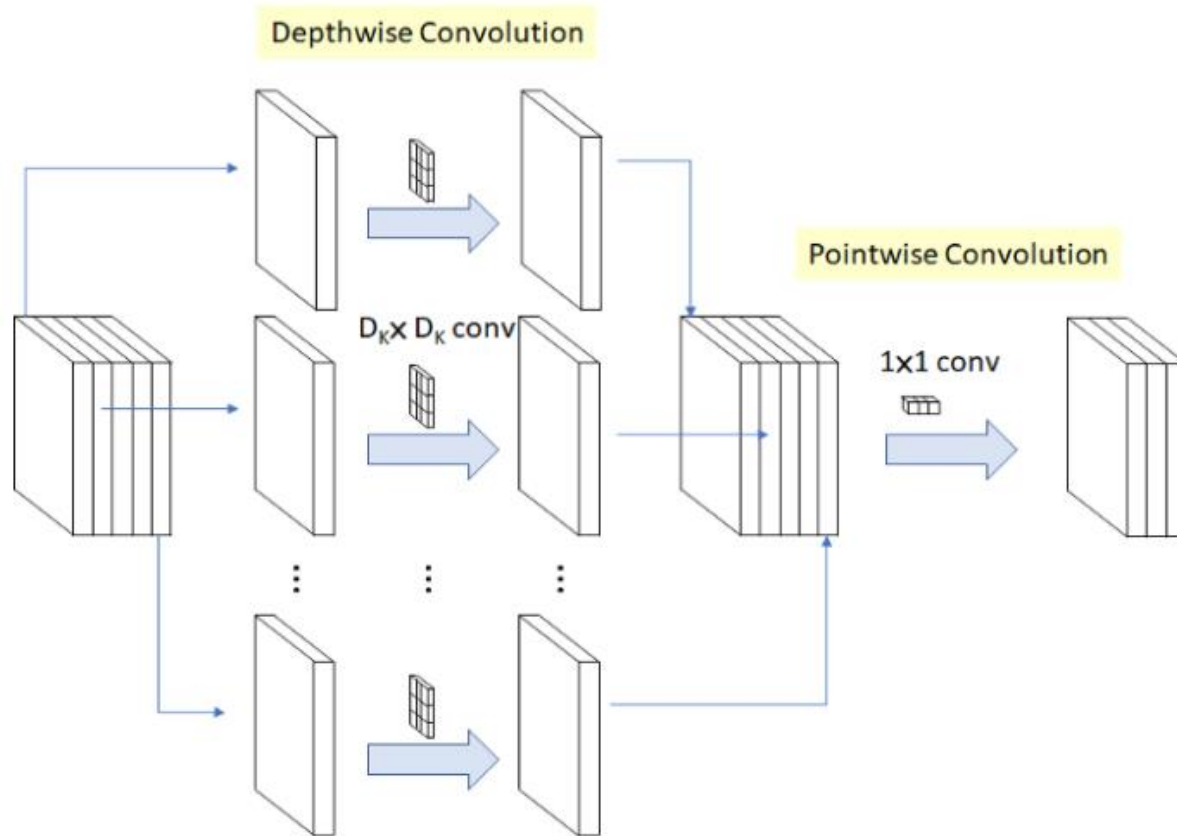


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

- total calculation for depthwise separable convolution:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$$

- total calculation for traditional convolution:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

- more than 8~9 times less computation

4) parameters for latency and accuracy

- 1) width multiplier: **latency**(α) (default=1.0, range=0.0~1.0)
- control the 'width' of the layer
 - total computation of $D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F$

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

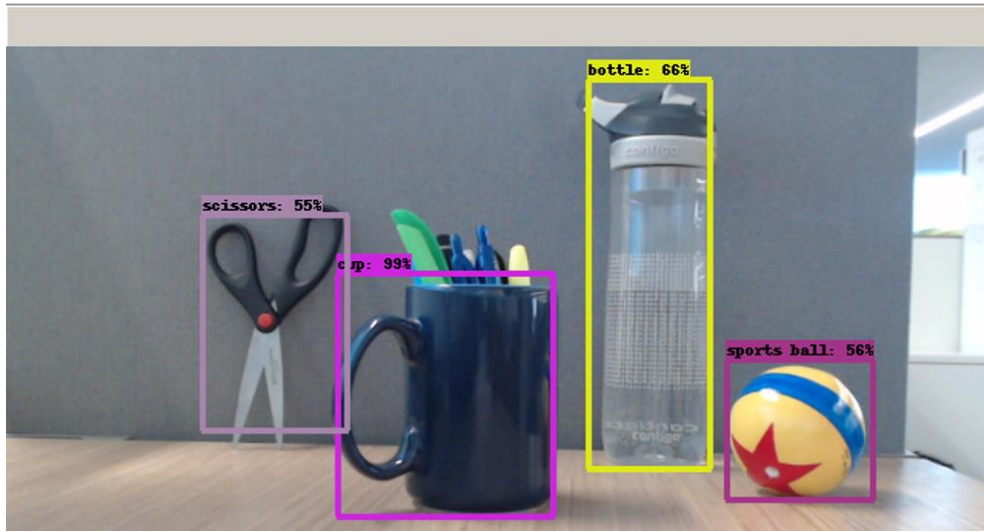
Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

- 2) resolution multiplier: **accuracy**(ρ) (default=1.0, range=0.0~1.0)
- control the 'resolution' of the image
 - in paper, author tests for image size of 224, 192, 169, 128

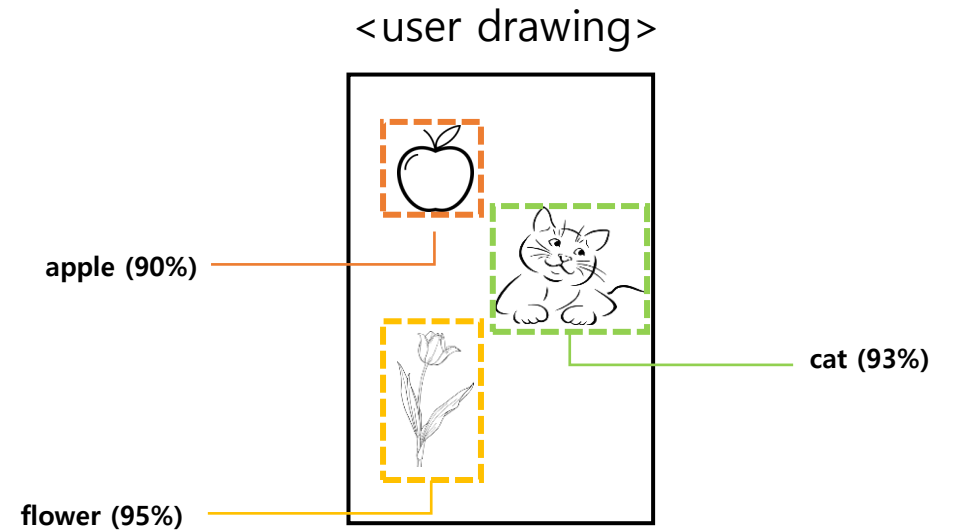
3. Object Detection



- Image classification : **one** class per **one** image
- *what if multiple classes in one image?*



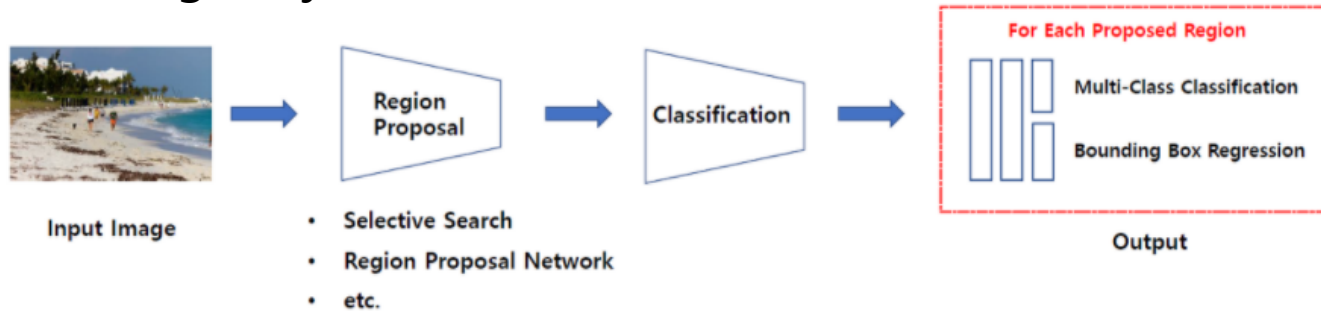
(object detection)



(our usage of object detection)

- Object detection: localization + image classification
- Use **Object detection** for input with multiple classes of doodling

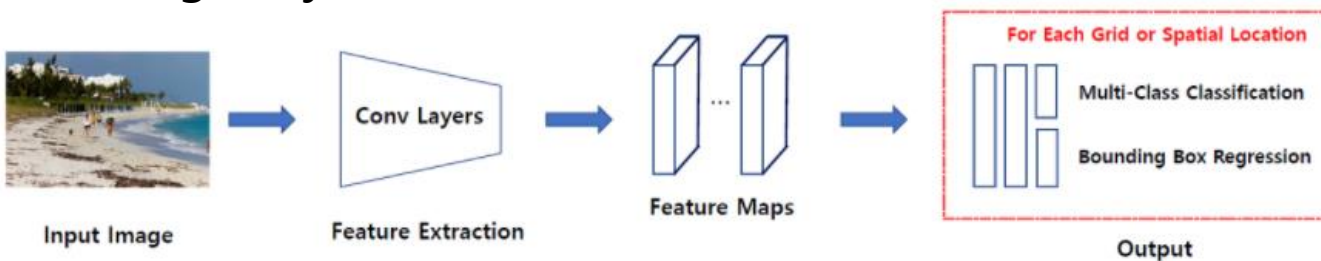
two-stage object detection



- localization & classification in different step
- relatively high accuracy, low speed
- e.g. R-CNN, Fast R-CNN...

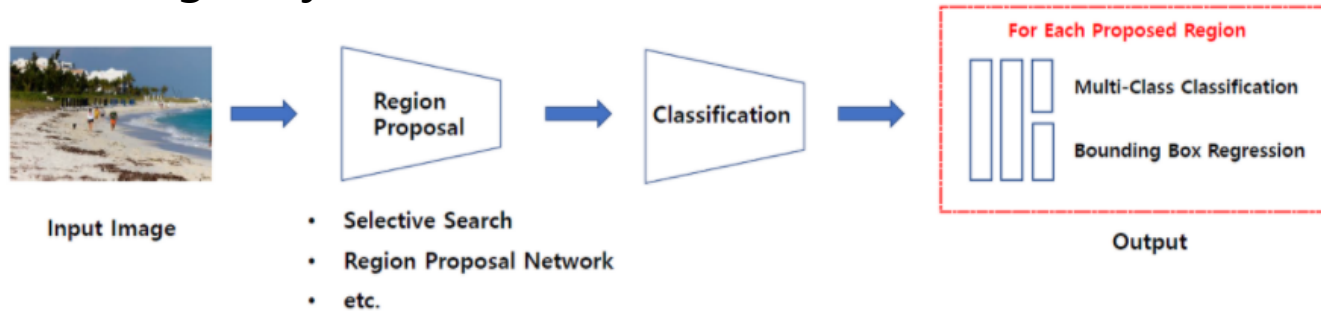
VS

one-stage object detection



- localization & classification in same step
- relatively low accuracy, high speed
- e.g. Yolo, SSD...

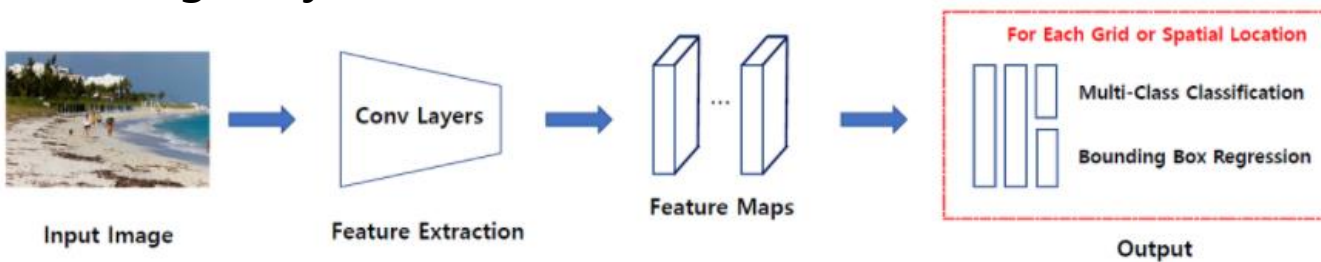
two-stage object detection



- localization & classification in different step
- relatively high accuracy, low speed
- e.g. R-CNN, Fast R-CNN...

VS

one-stage object detection



- localization & classification in same step
- relatively low accuracy, high speed
- e.g. **Yolo** SSD...

↑
today's topic

Yolo v1

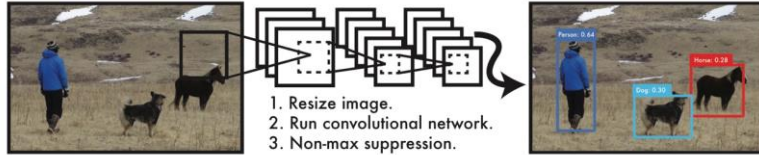
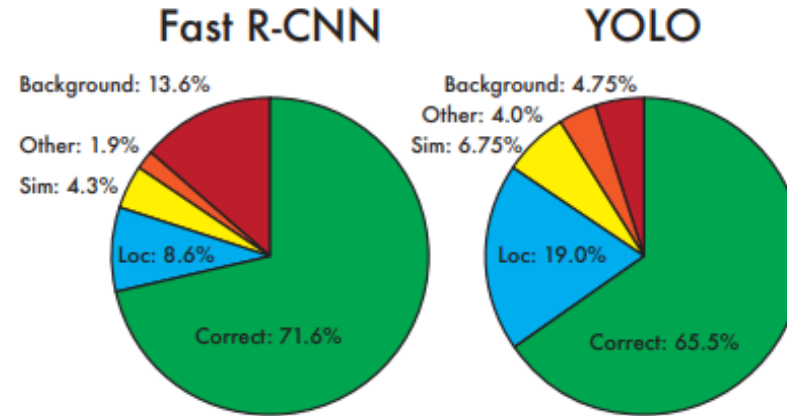


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448×448 , (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

(one-stage object detection)



(low background err)



(performance on artwork)

Yolo)

- One-stage detection

Strength of Yolo)

- Extremely fast : real-time detection (45 fps)
- Low background error than state-of-the-art (fast r-cnn)
- Learns generalizable representations of object



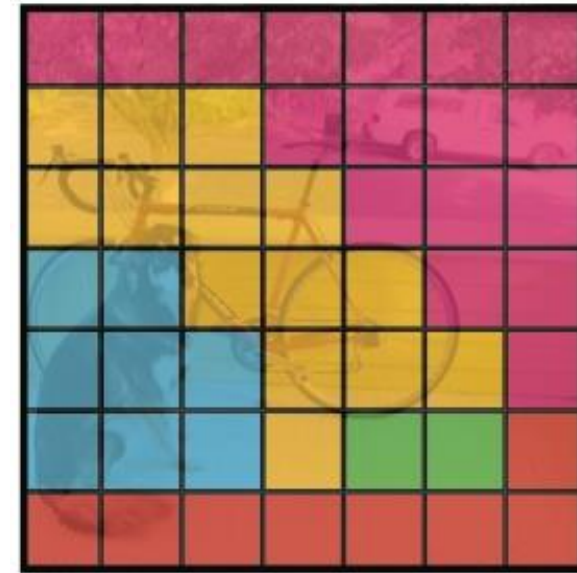
$S \times S$ grid on input

- divides the input image into an $S \times S$ grid
- If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object
- Each grid cell predicts B bounding boxes & confidence score($\Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}}$)



bounding boxes+confidence

- each bounding box predicts
 $x, y, w, h, \text{confidence}(\Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}})$

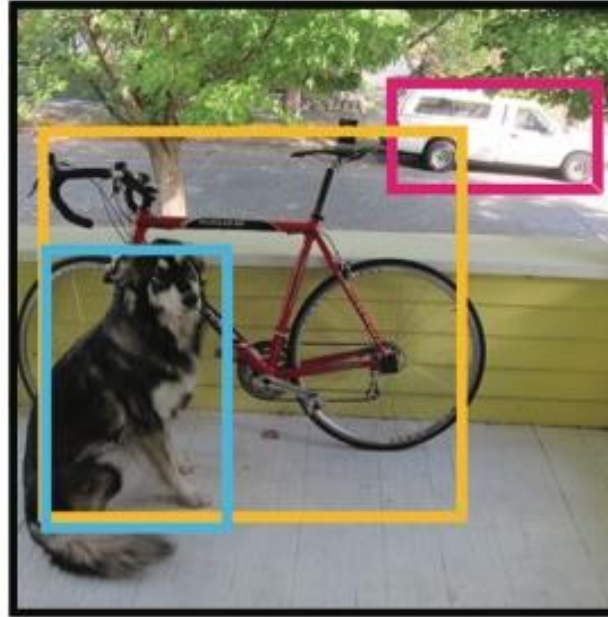


class probability map

- each grid cell predicts C conditional probabilities $\Pr(\text{Class}_i | \text{Object})$

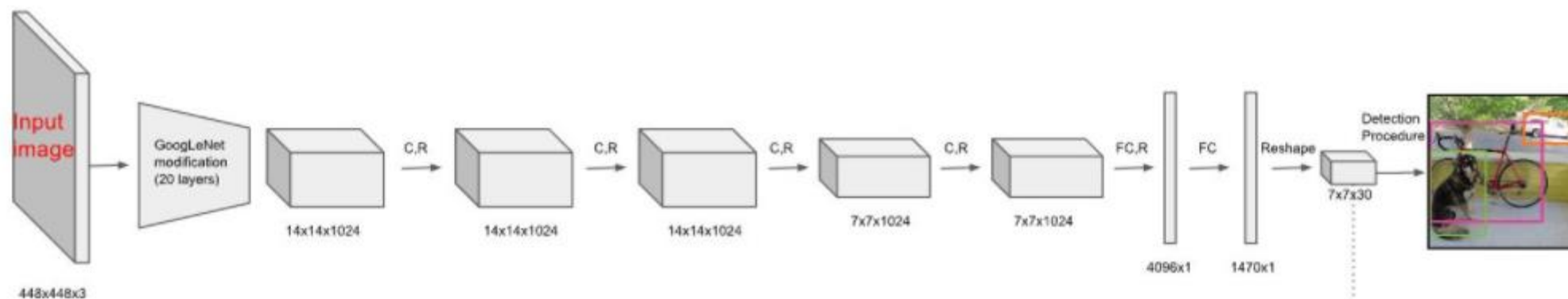
<class-specific confidence score>

$$\Pr(\text{Class}_i | \text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}}$$

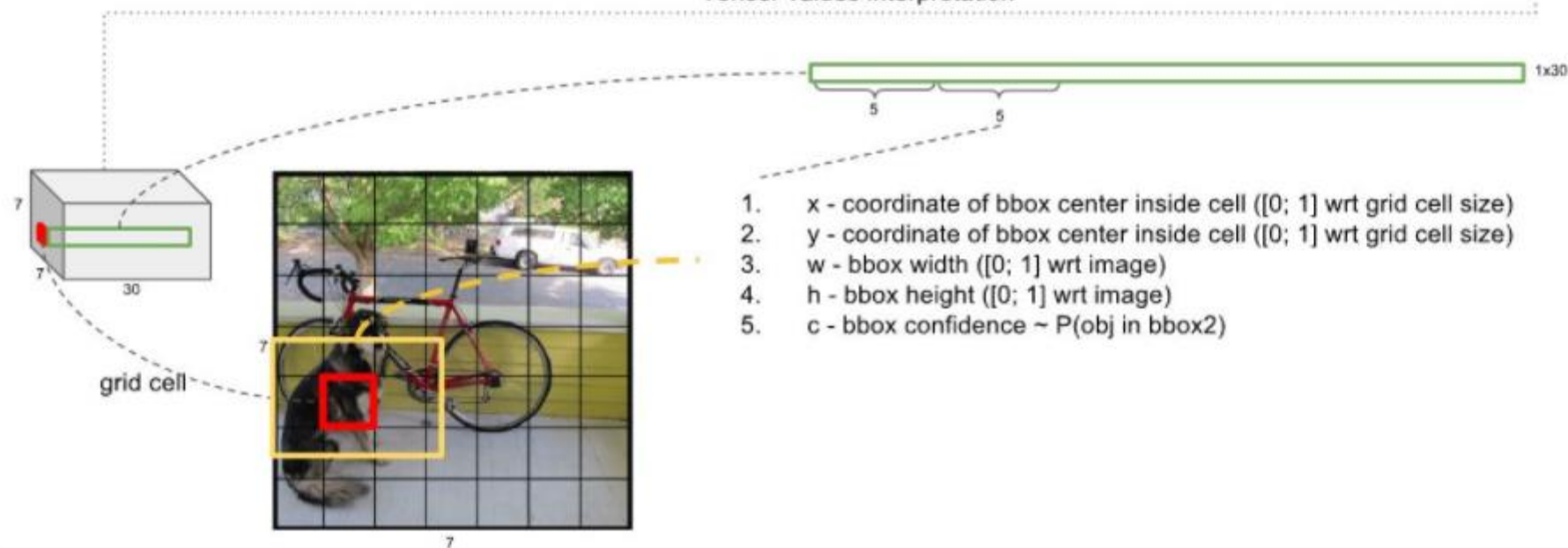


final detections

- obtain **class-specific confidence scores** from each box
- class-specific confidence score encode both the probability of that class appearing in the box & how well the predicted box fits the object



Tensor values interpretation



Thank you