Computer Vision

COMPSCI 760
2024 Semester 1

Olivier Graffeuille
ogra439@aucklanduni.ac.nz

Outline for Weeks 6-11

Week 6: Computer Vision – CNNs

Week 7: Computer Vision – Other vision tasks

Week 8: Transfer Learning

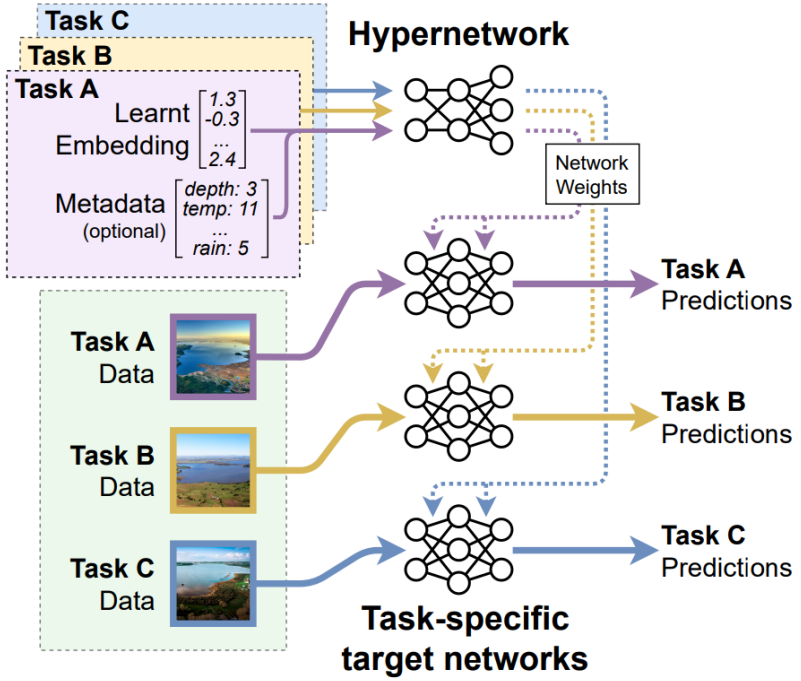
Week 9: Continual Learning

Week 10: Domain Generalisation

Week 11: Self-Supervised Learning

About me

Taken from orbit in October 2011, the worst algae bloom that Lake Erie has experienced in decades. (ITTNASA)



Disclaimer/Acknowledgment

The following slides are reusing some of the content from the Stanford CS230 and CS231n class:
https://stanford.edu/~shervine/teaching/cs-230/
https://cs231n.github.io/convolutional-networks/

Some content is also reused from the CS230 Stanford course slides created by Andrew Ng, available at
https://cs230.stanford.edu/syllabus

Outline

1. Background
2. Convolution
3. CNN components: convolutional layers, pooling layers
4. CNN architectures

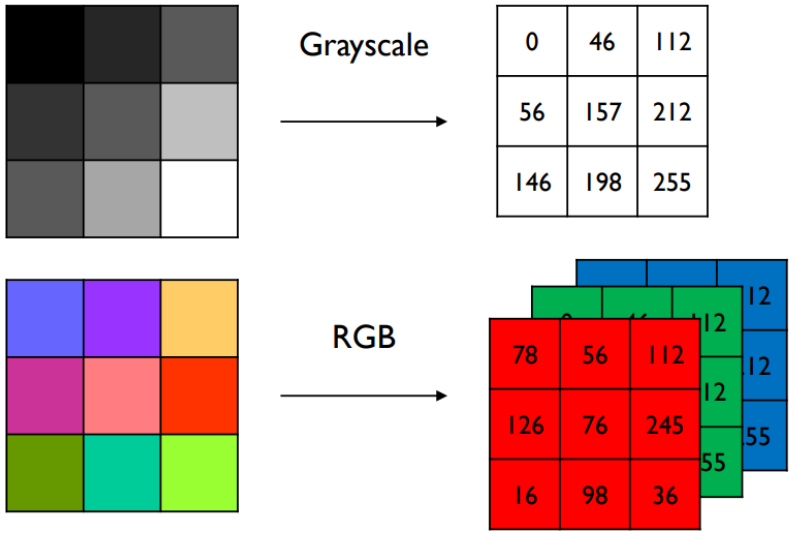
Outline

1. ***Background***
2. Convolution
3. CNN components: convolutional layers, pooling layers
4. CNN architectures

Images as vectors

Neural networks usually take a vector of values as input

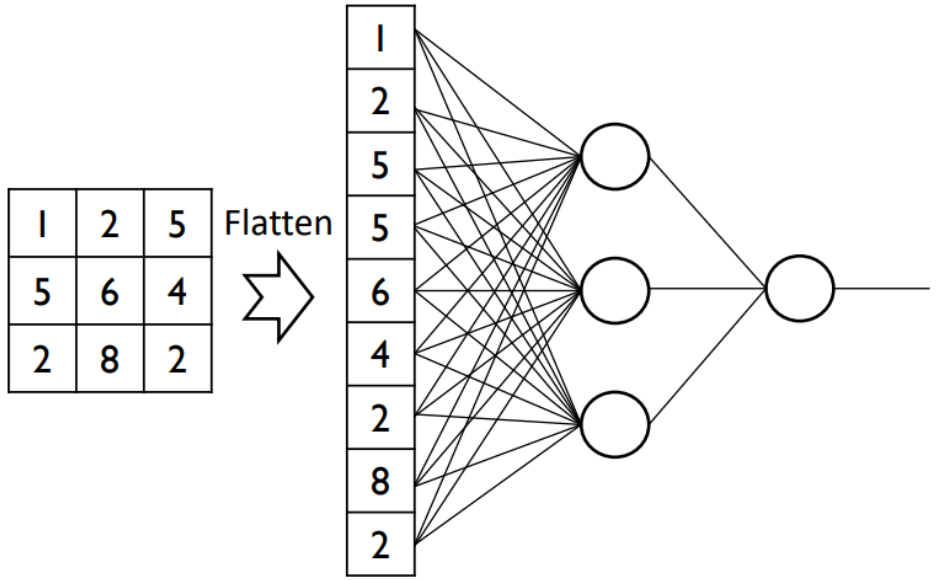
A digital image is generally represented as a 2D Matrix of pixels



1
2
5
5
6
4
2
8
2

Fully connected layers?

Why not just flatten an image into a vector, and use traditional NNs (with fully connected linear layers) to process an image?



Fully connected layers?

1. Problem 1: The number of parameters would go quickly:
    1. A 512x512x3 image has 800k values
    2. A linear layer mapping from 800k input neurons to 800k hidden neurons would have 640 billion parameters (more than GPT-4?)

Fully connected layers?

1. Problem 1: The number of parameters would go quickly:
    1. A 512x512x3 image has 800k values
    2. A linear layer mapping from 800k input neurons to 800k hidden neurons would have 640 billion parameters (more than GPT-4?)
2. Problem 2: Would lead to overfitting
    1. In a given dataset, potentially:
       if pixel #14,153 is greener than pixel #784,150 then the image is a cat
    2. The model's inductive bias is not suitable for images

Images as vectors

1. Instead, we should consider pixel locality

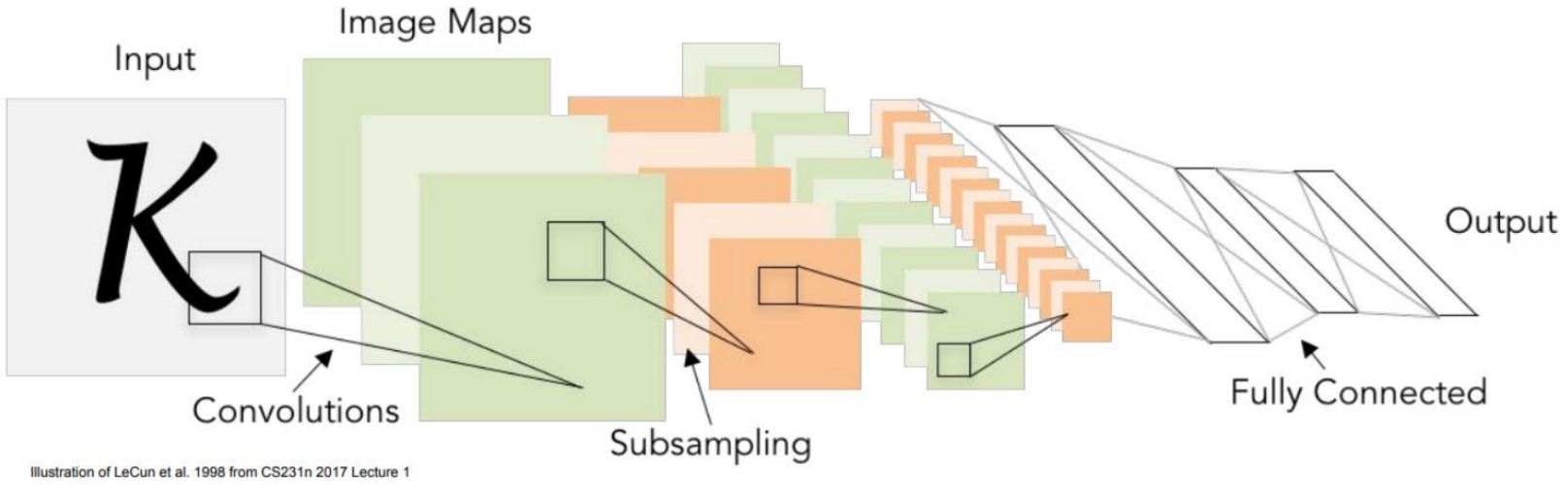


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Images as vectors

1. Main differences with a classic ANN:
    1. Explicit assumption that inputs are images.
    2. Introduce new types of layers: convolution and pooling layers

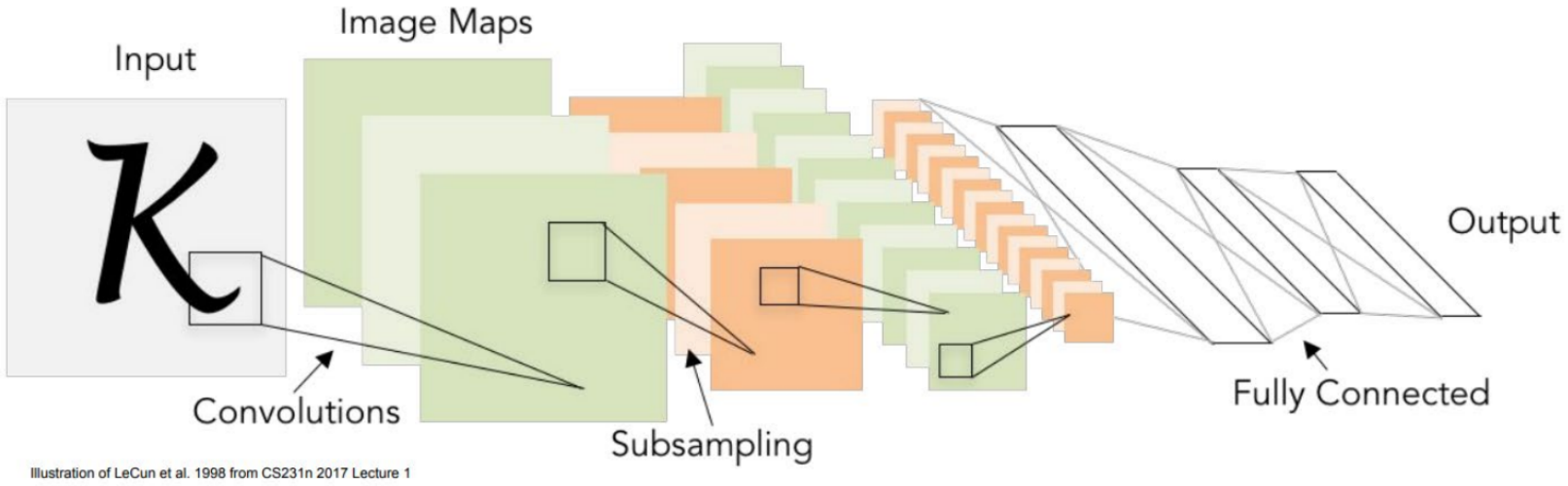


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Images as vectors

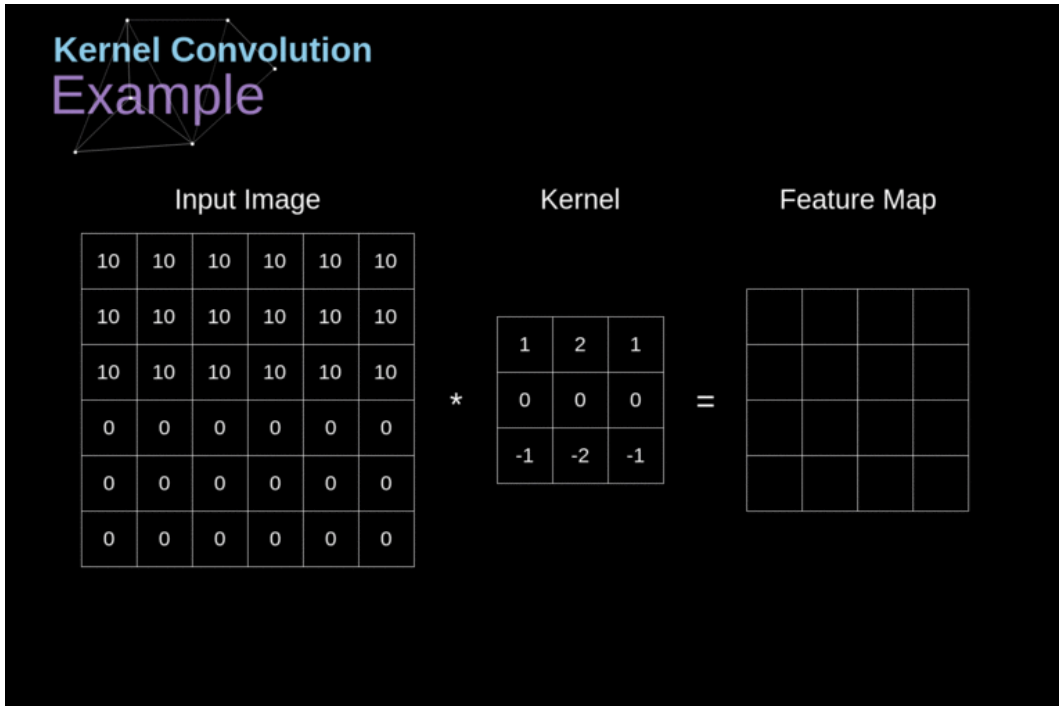
1. The benefits of CNNs:
    1. Make images easier to process
    2. Extract meaningful features from the image.
2. They are the go-to architecture for computer vision tasks*

Outline

1. Background
2. ***Convolution***
3. CNN components: convolutional layers, pooling layers
4. CNN architectures

Kernels

1. Convolution is common in image processing to filter images using **kernels**
    1. Edge detection



https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9

Kernels

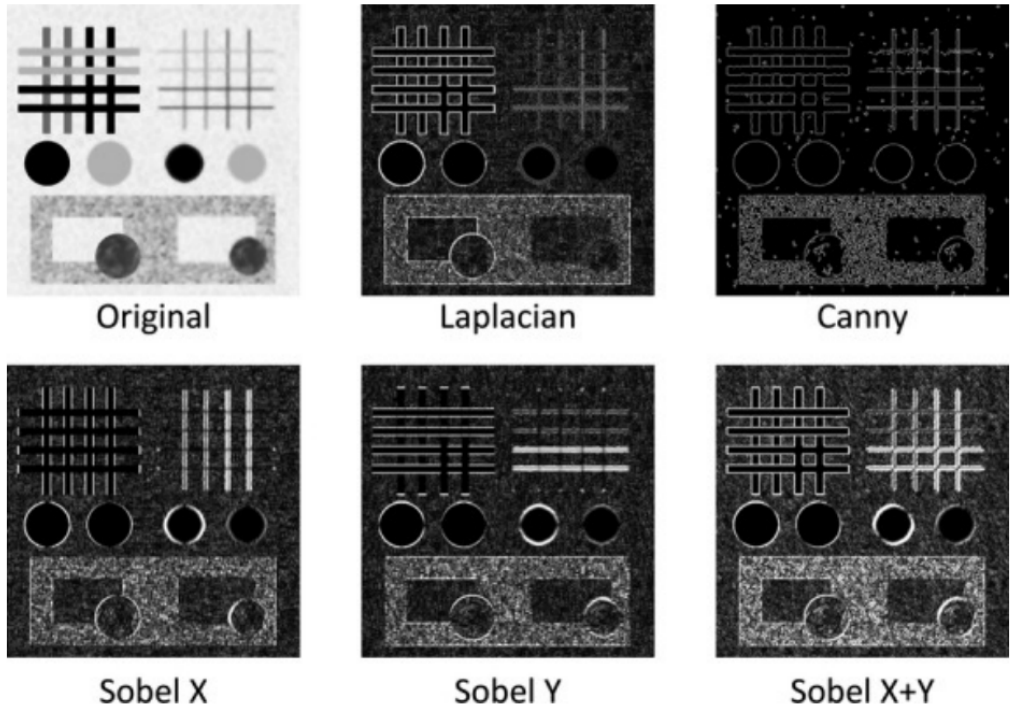
1. Convolution is common in image processing to filter images using **kernels**
    1. Edge detection
    2. Smoothing

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

https://en.wikipedia.org/wiki/Kernel_(image_processing)

Kernels

1. Convolution is common in image processing to filter images using **kernels**
    1. Edge detection
    2. Smoothing
    3. Other filters



Sobel

-1	0	1
-2	0	2
-1	0	1

Sobel X

1	2	1
0	0	0
-1	-2	-1

Sobel Y

Laplacian

0	-1	0
-1	4	-1
0	-1	0

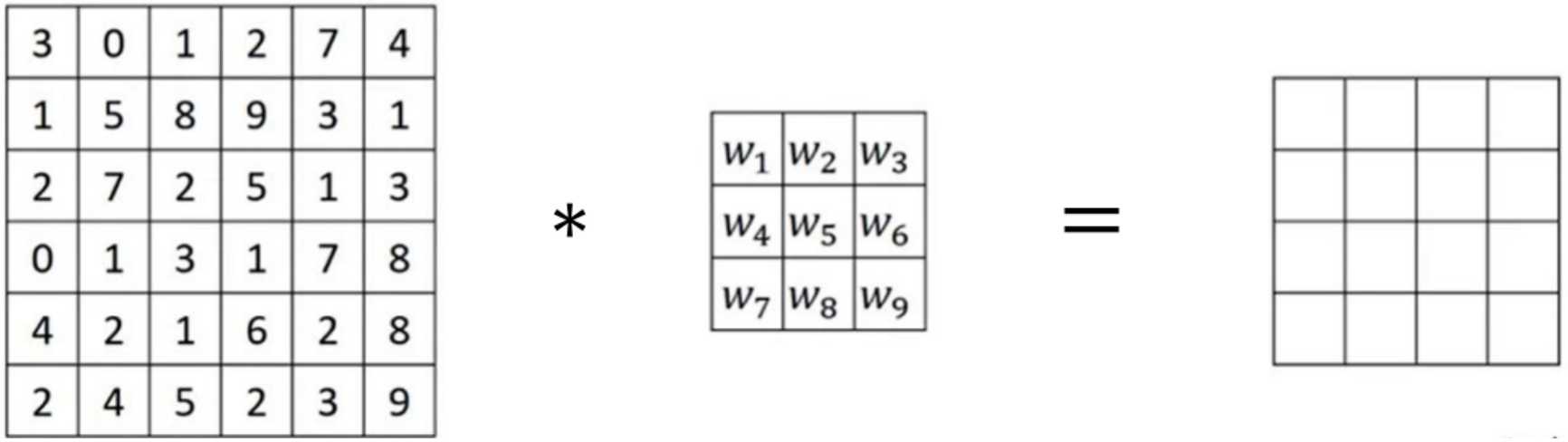
-1	-1	-1
-1	8	-1
-1	-1	-1

Uchida, S. (2013). Image processing and recognition for biological images

Kernels

What filters should we use?

- Define them as neural network weights
- Learn them through backpropagation

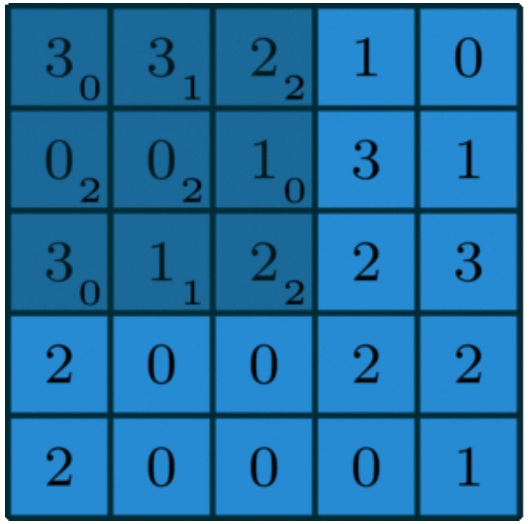


Convolutional Layer

Sliding kernel (filter) over the image

Computationally:
matrix multiplication + sum

Convolution of a *f x f* filter
with a *n x n* image
produces a *n-f+1 x n-f+1* feature map

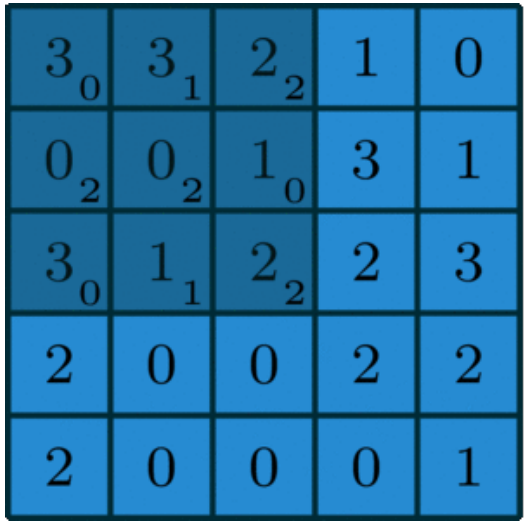


Convolutional Layer

Sliding kernel (filter) over the image

Computationally:
matrix multiplication + sum

Convolution of a *f x f* filter
with a *n x n* image
produces a *n-f+1 x n-f+1* feature map



Problem: shrinking size, and border pixels
don’t contribute much to feature map

Padding

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

padding p = 1

*

1	0	1
0	1	0
1	0	1

=

2	2	3	1	1
1	4	3	4	2
1	2	4	3	3
1	2	3	4	1
0	2	2	1	1

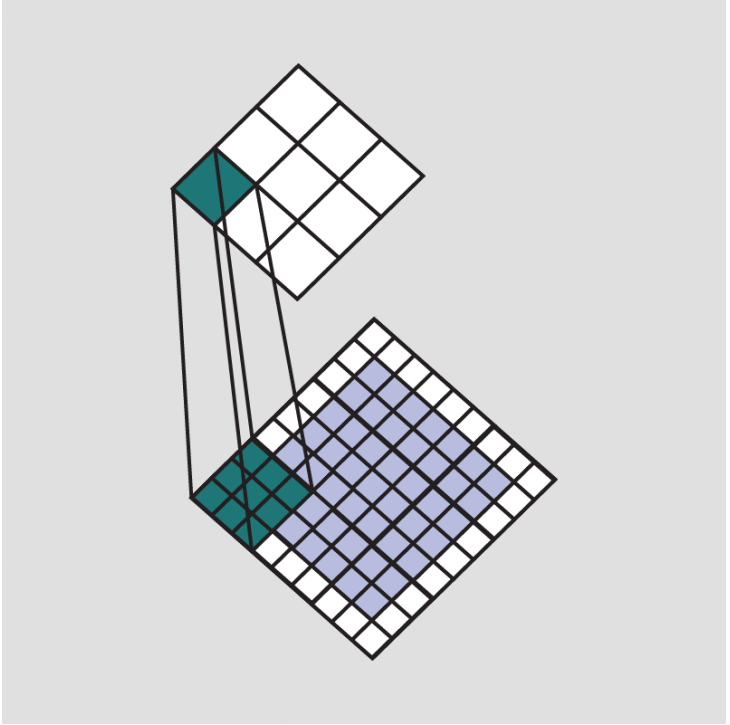
Convolution of a $f \times f$ filter with a $n \times n$ image, with p padding produces a $n+2p-f+1 \times n+2p-f+1$ feature map

Padding

- Valid padding (no padding)
  - Output: $n–f+1 \times n–f+1$ feature map
- Same padding (padding to make output feature map the same size as the input)
  - Output: $n+2p–f+1 \times n+2p–f+1$ feature map
  - What p should you use?
    $n = n+2p–f+1$
    $p = (f-1) / 2$ (depends on filter/kernel size!)
  - Filter size f is usually odd (3x3, 5x5…) to have integer p

Stride

- Strided convolutions reduce the size of the output



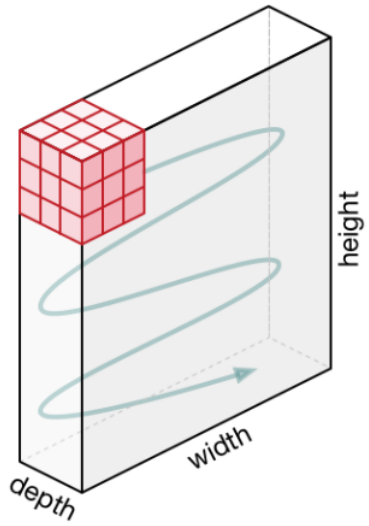
Convolution of $f \times f$ filter on a $n \times n$ image with padding p and stride s produces feature map of size $\frac{n+2p-f}{s}+1 \times \frac{n+2p-f}{s}+1$

Stride

Why use strided convolutions to reduce the size of the output?

- Reduces computational load
- Increases the receptive field of the features
- Alternative to pooling

Images are 3D tensors



0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

$308 + (-498) + 164 + 1 = -25$

Bias = 1

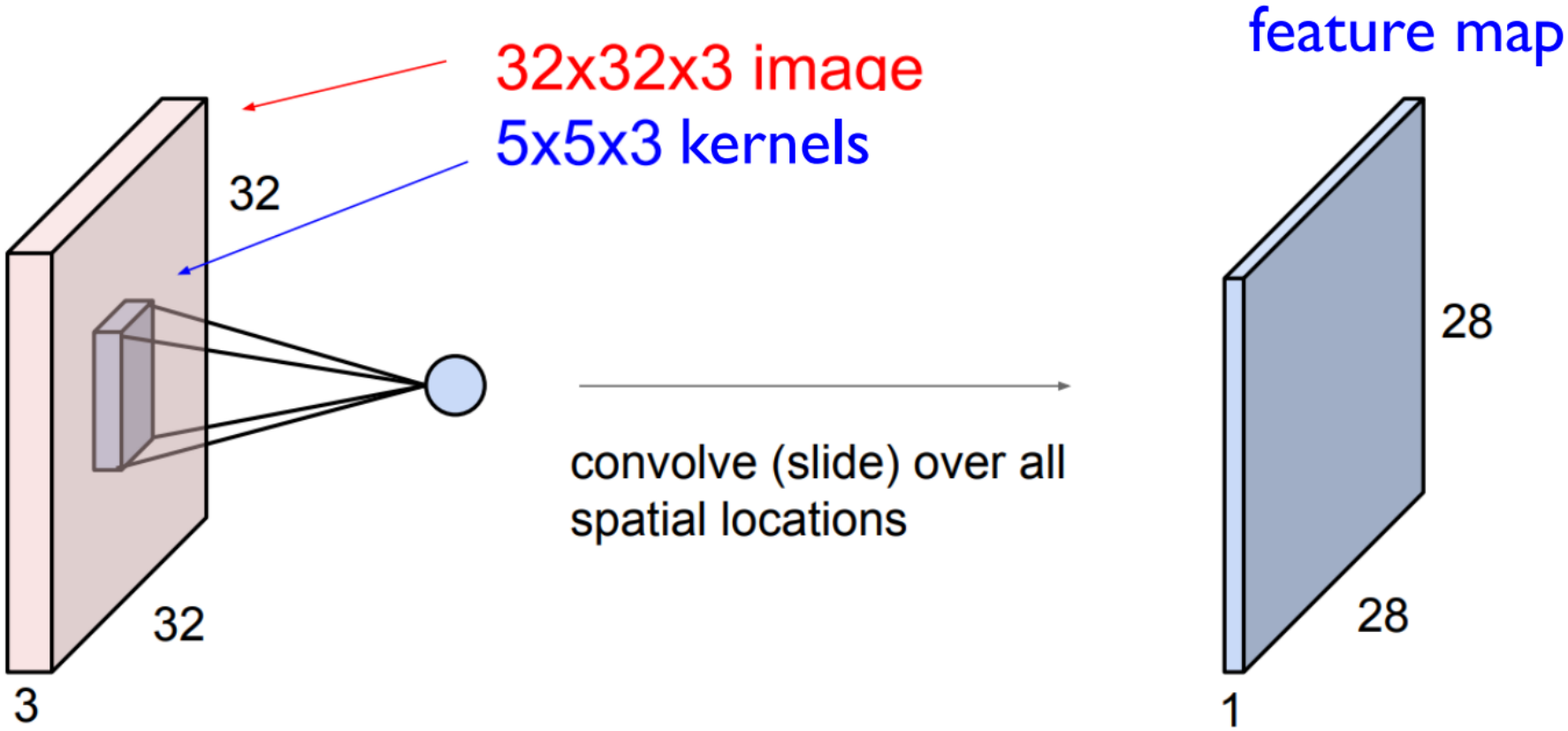
Output

-25				...
				...
				...
				...
...	...	...	...	...

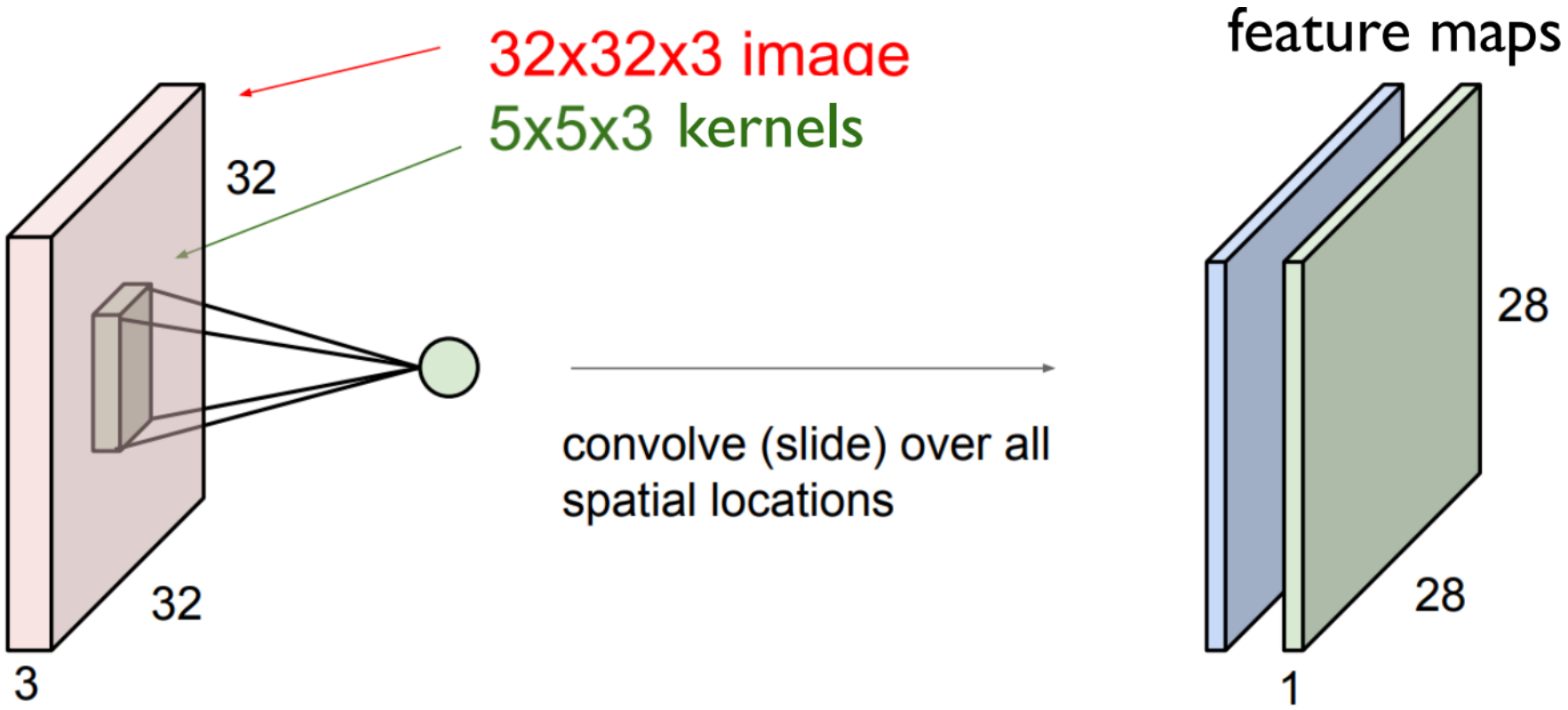
Outline

1. Background
2. Convolution
3. ***CNN components: convolutional layers, pooling layers***
4. CNN architectures

Making a convolutional layer

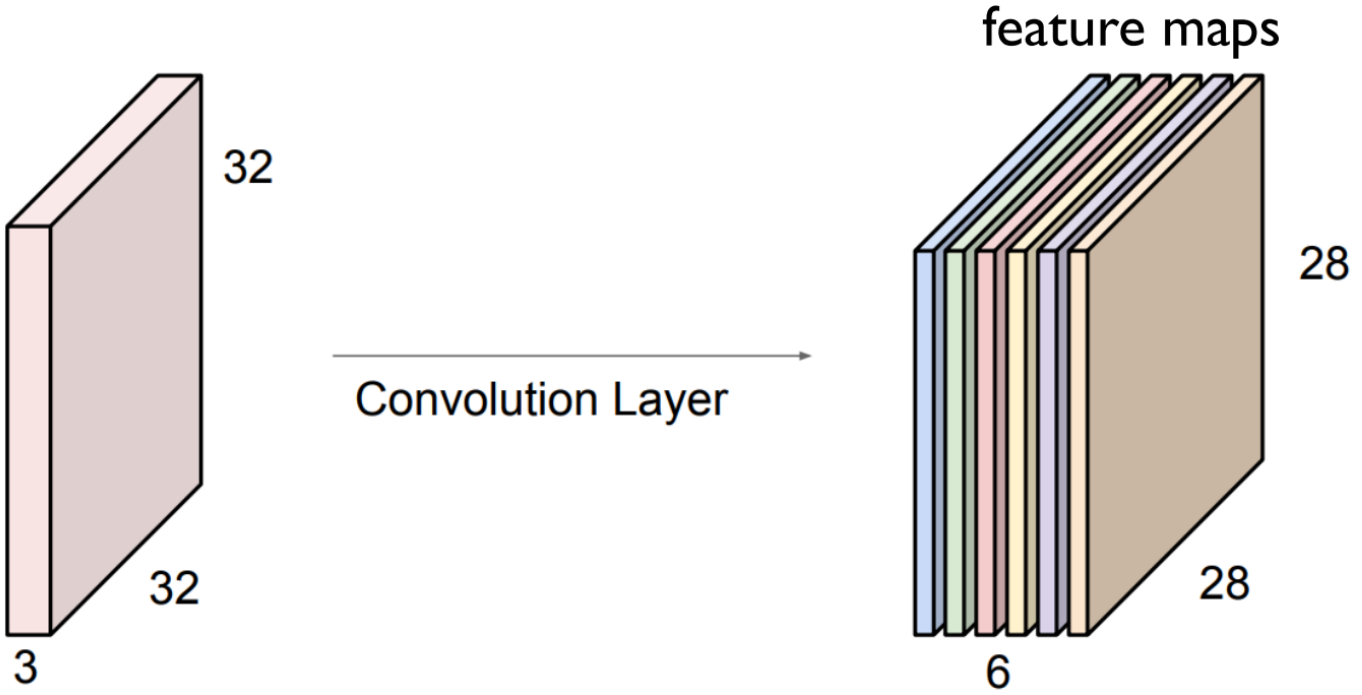


Making a convolutional layer

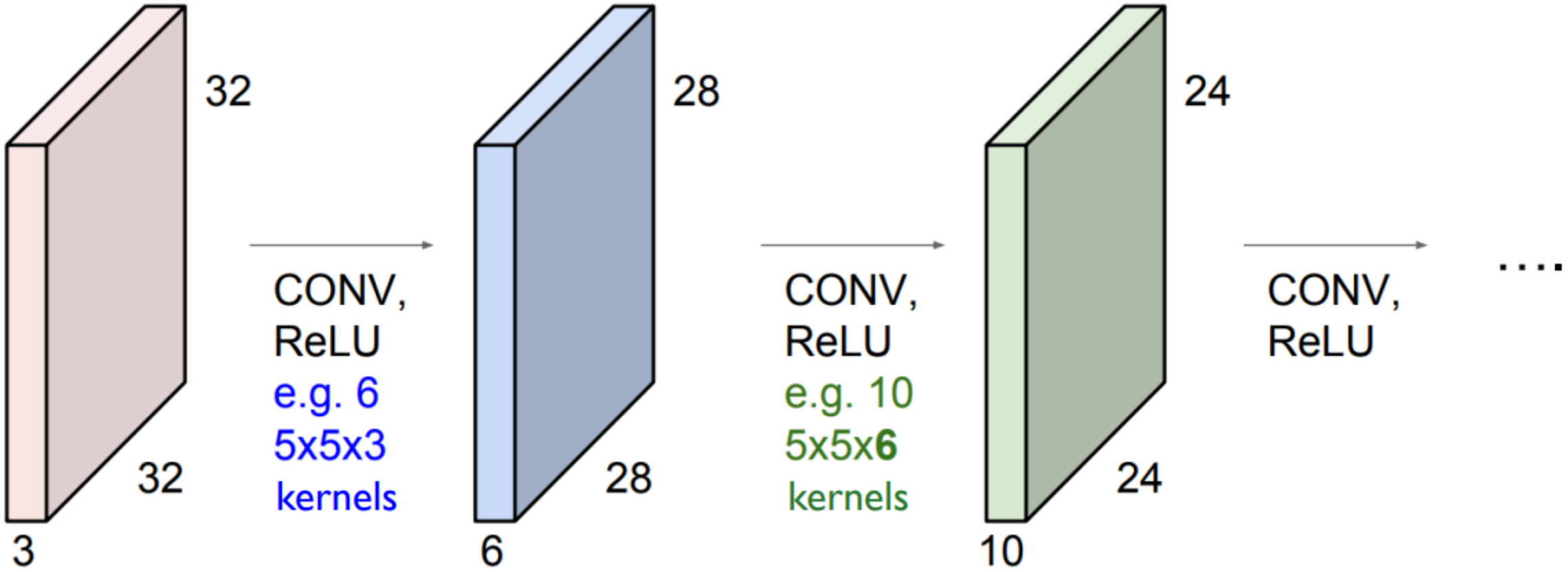


Making a convolutional layer

- Stacking multiple kernels produces new 3D "image" (feature map)



Multiple convolutional layer

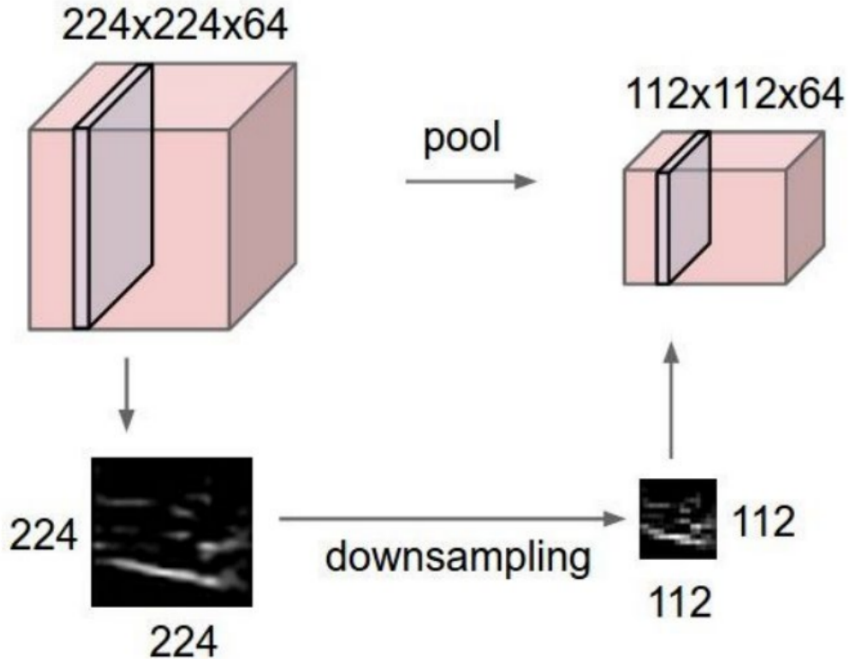


1x1 Convolutions

- Many models use 1x1 convolutions, which seems confusing
- On a 2D image, a 1x1 convolution is just a scaling factor
  out = w*in
- But since images (and feature maps) are 3D, they are a layer-wise linear transformation

Pooling

- Makes representations smaller (similar to strides)
- Applies independently on each feature map (spatial aggregation)

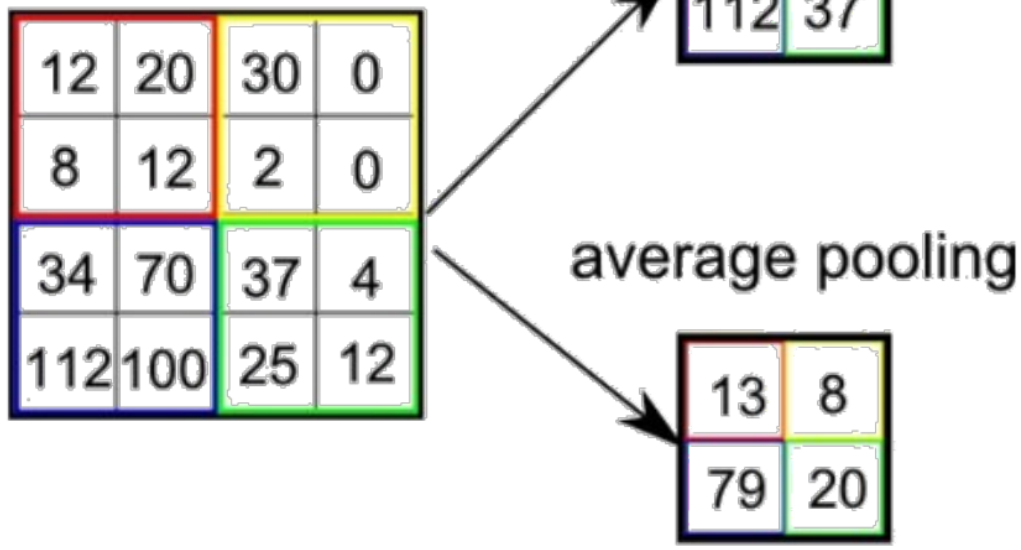


Pooling

- Two main types of pooling

kernel size f = 2

stride s = 2



- Max pooling is usually preferred (keeps the most significant features)

Hyperparameters

Convolutional layer

f : kernel size

s : stride

p : padding

n'_C : number of filters

If input size is $n_H \times n_W \times n_C$:

Output size of convolution will be:

$$\left\lfloor \frac{n_H+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_W+2p-f}{s} + 1 \right\rfloor \times n'_C$$

Pooling layer

f : kernel size

s : stride

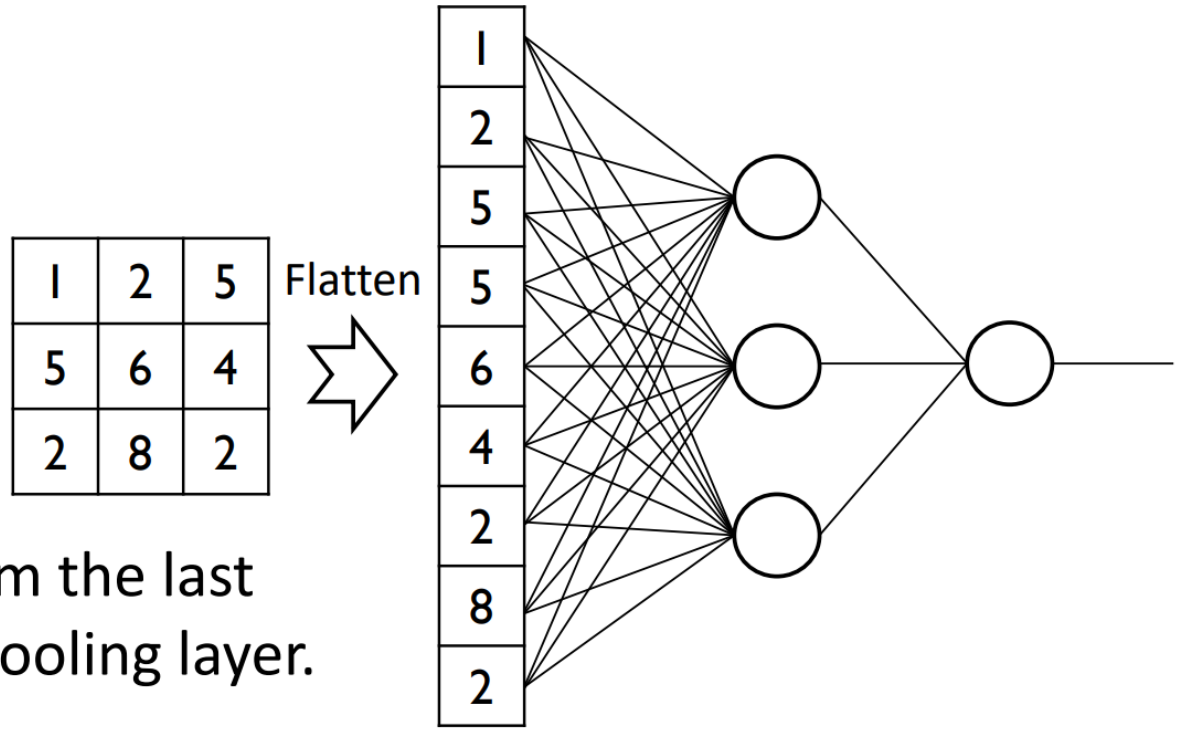
Max or average pooling

Output size of pooling will be:

$$\left\lfloor \frac{n_H-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_W-f}{s} + 1 \right\rfloor \times n_C$$

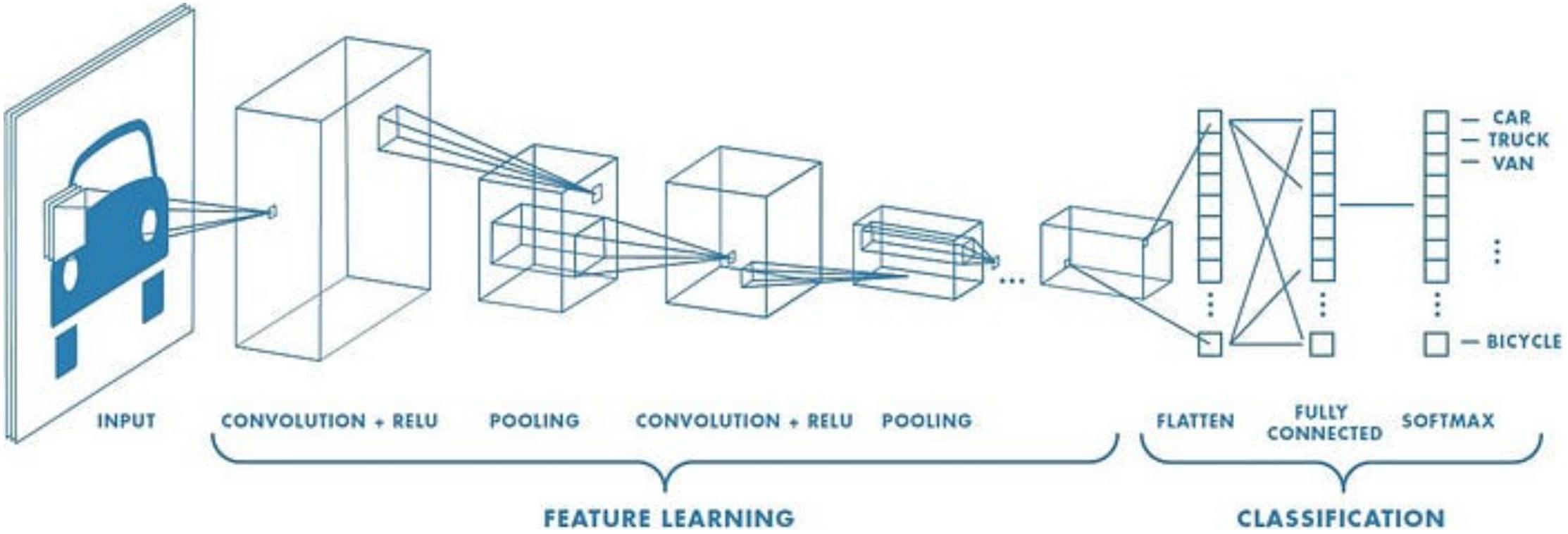
Fully Connected Layer

For Classification:



Feature map from the last convolution or pooling layer.

Putting it all Together



https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

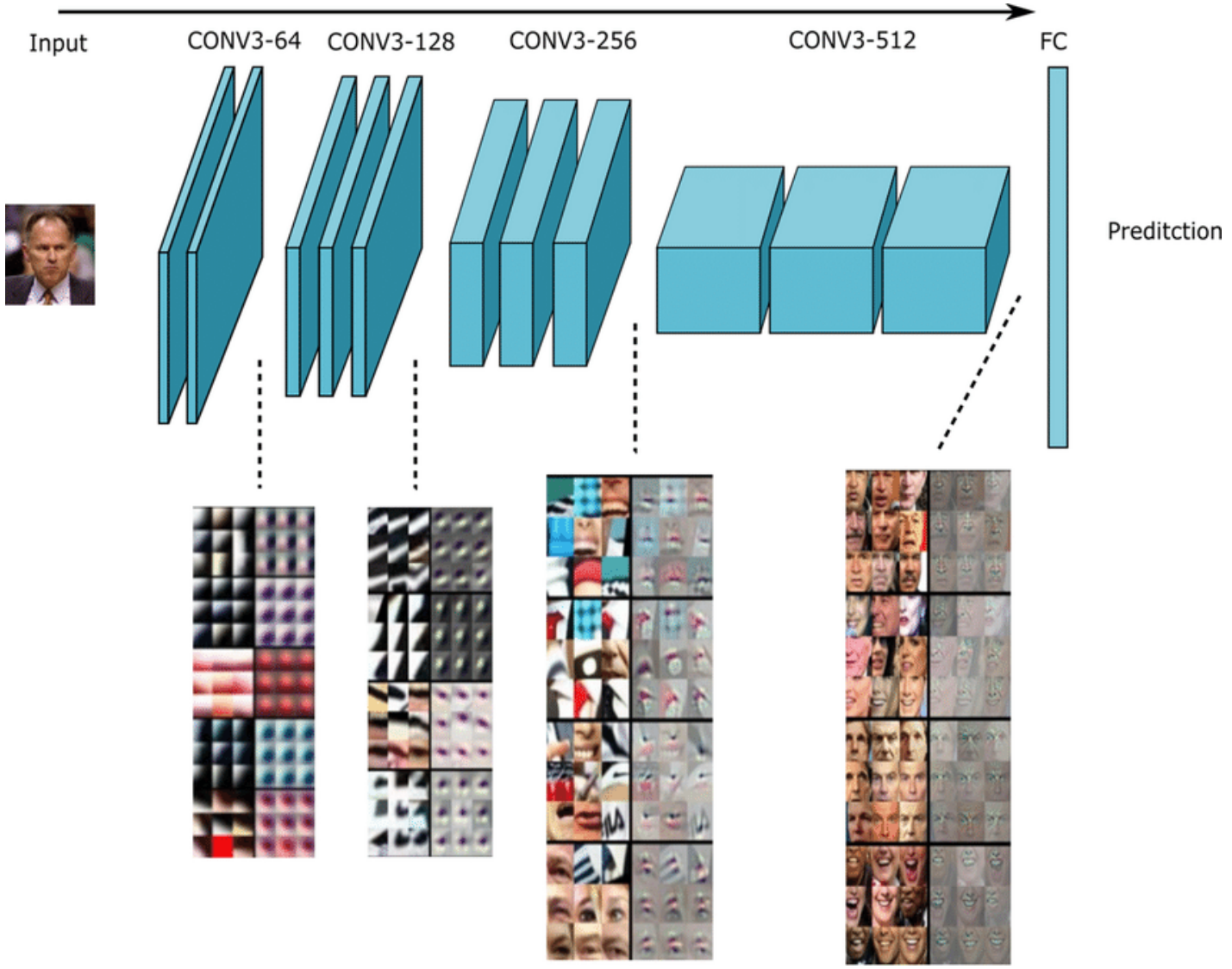
Outline

1. Background
2. Convolution
3. CNN components: convolutional layers, pooling layers
4. ***CNN architectures***

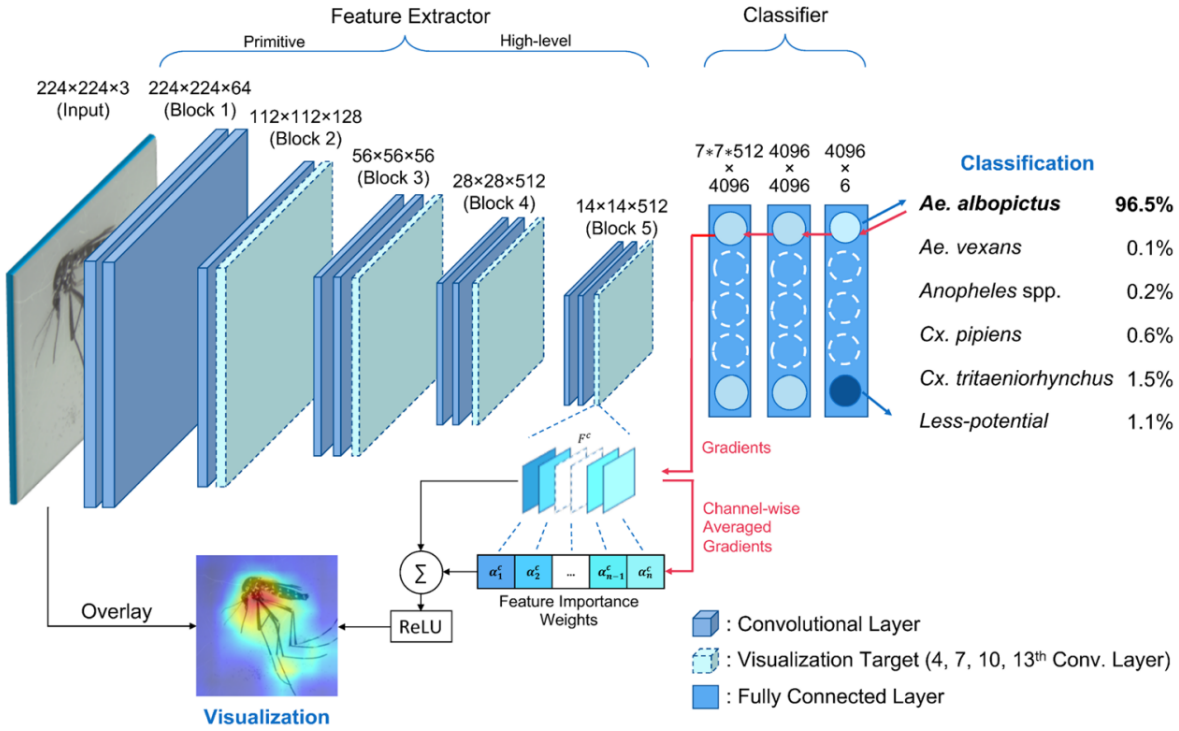
Deep Representations

Composition of convolutional layers allows these models to learn increasingly complex visual features

Biological vision systems do this



Visualising Deep Representations



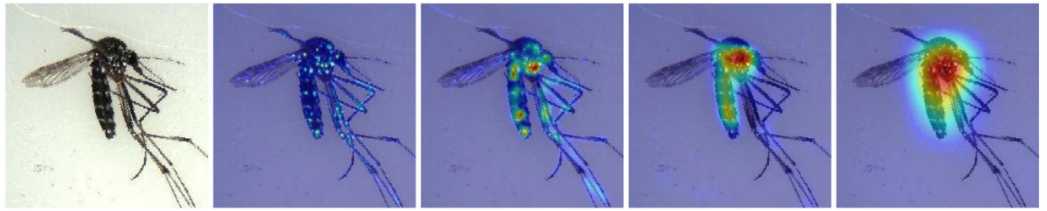


Figure 4. The visualization of discriminative regions in the 4-, 7-, 10- and 13-th convolution layers of VGG-16 when an *Aedes albopictus* input image is given.

Classifcation and Morphological Analysis of Vector Mosquitoes using Deep Convolutional Neural Networks, Park et al. 2020, Nature Scientific Reports

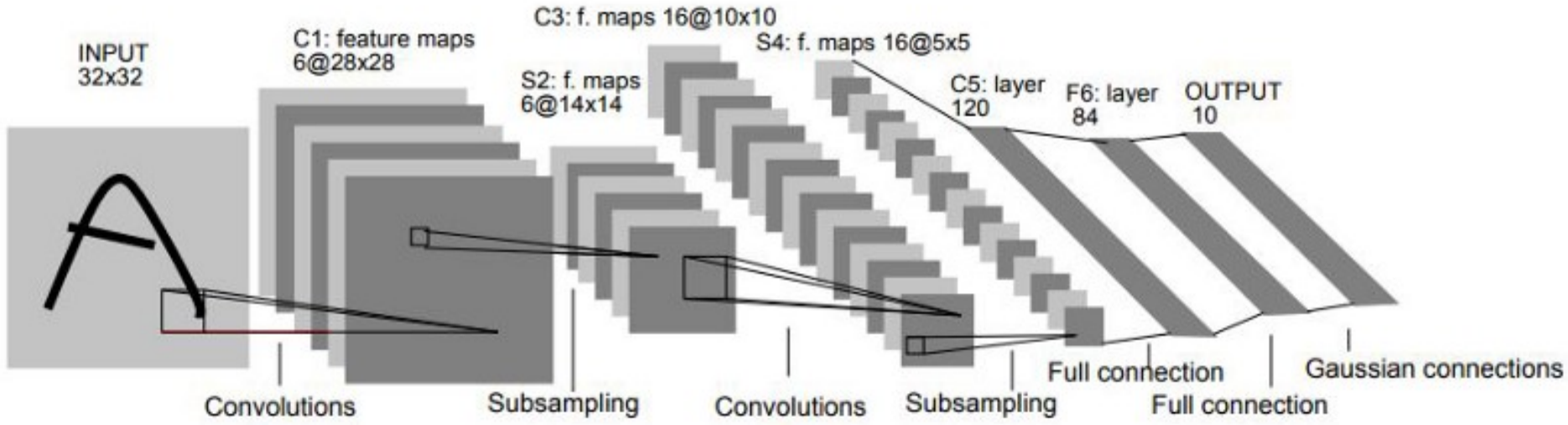
CNN architectures

- CNN architectures are effectively a combination of the previous building blocks
    - This is largely an engineering problem

- Architectures are often effective across diverse domains

LeNet-5 (1998)

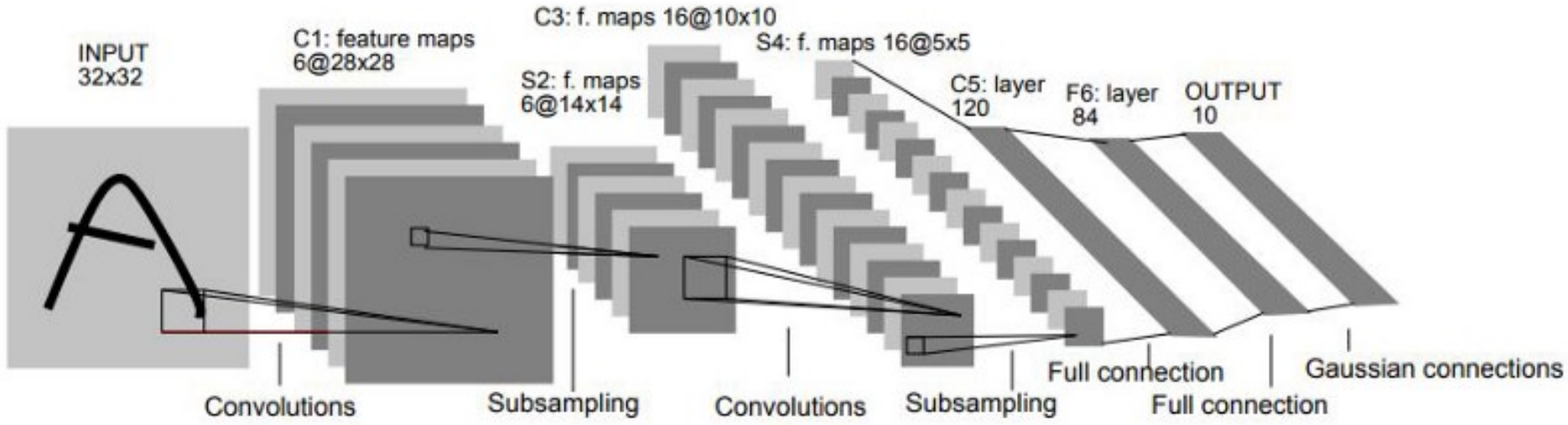
- Designed for handwritten digit recognition (MNIST)
- 60k parameters



LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

LeNet-5 (1998)

- Convolution kernels: kernel size f = 5 (5x5), stride s =1
- Pooling layers: kernel size f = 2 (2x2), stride s = 2

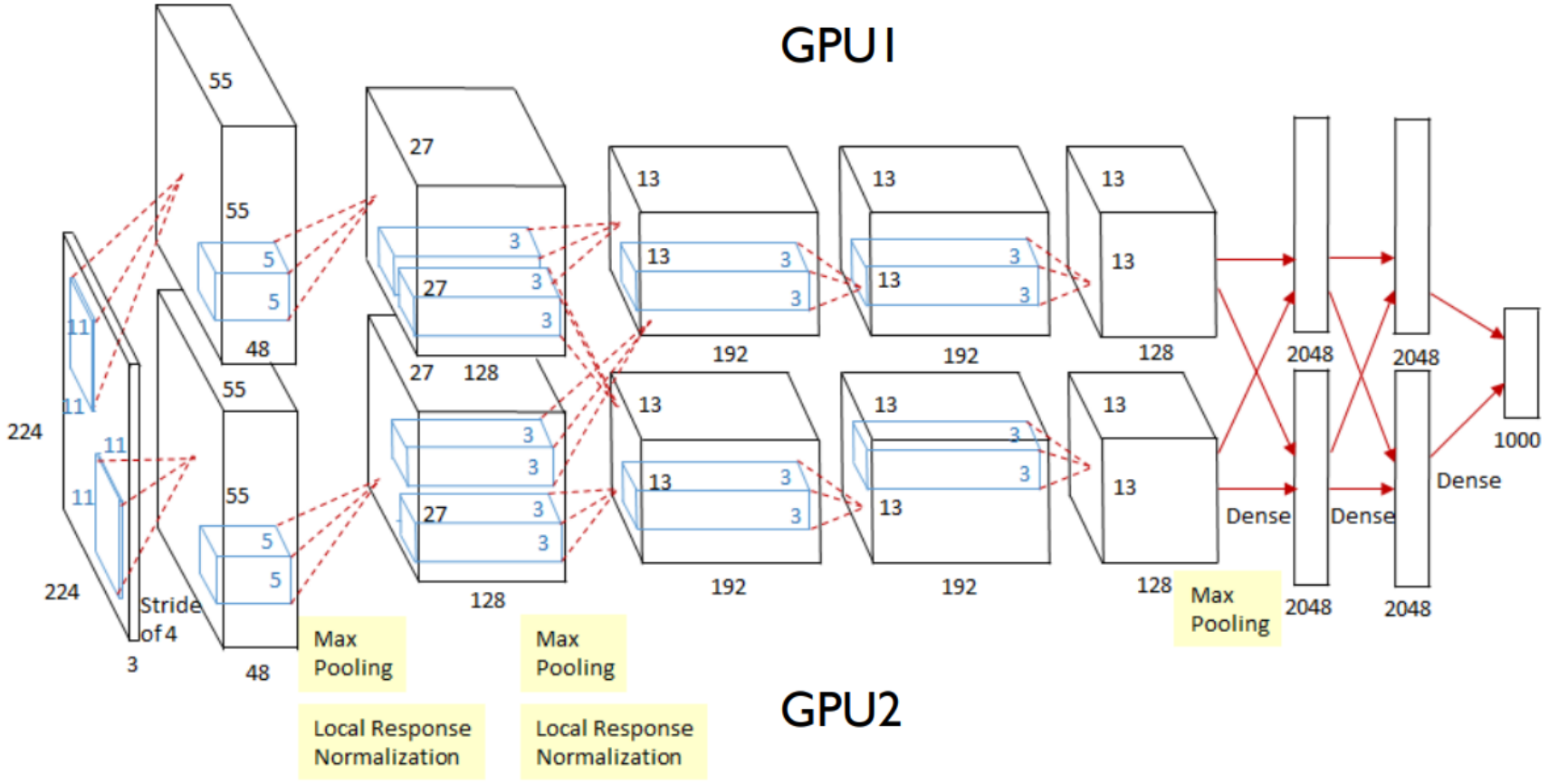


LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

AlexNet (2012)

Run on two GPUs due to memory constraints

60 million parameters



Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

AlexNet (2012)

Full (simplified) AlexNet architecture:

[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Remarks/Hyperparameters:

- First use of ReLU
- Use of normalisation layers (not common anymore)
- Heavy data augmentation (x 2048!)
- Dropout: 0.5
- Batch size: 128
- SGD Momentum: 0.9
- Learning rate: 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 regularisation weight decay: 5e-4

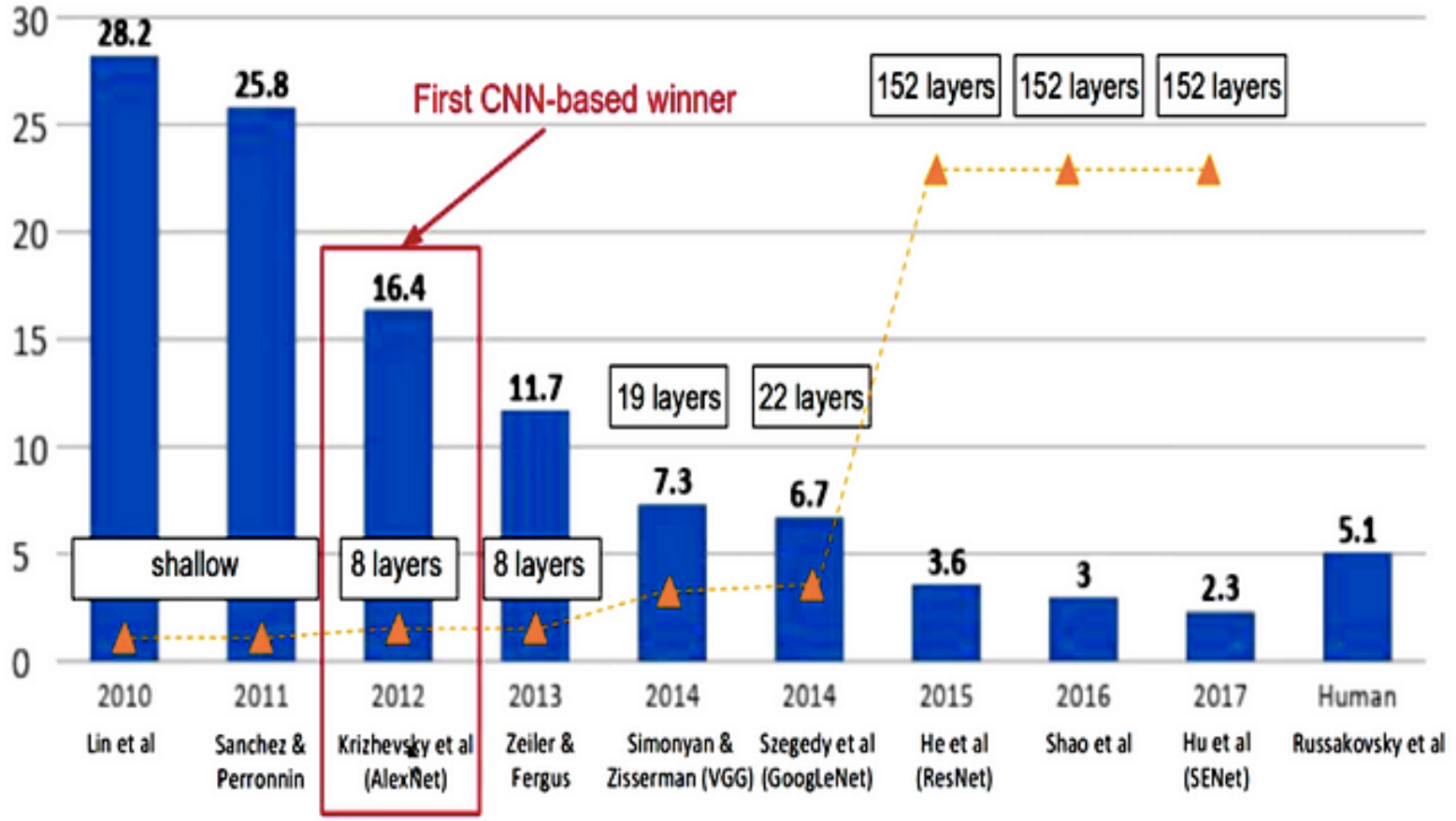
Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

ImageNet Classification Challenge

- Competition to achieve the highest accuracy on a visual classification dataset
- >1m images
- 1000 classes

https://image-net.org/challenges/LSVRC/

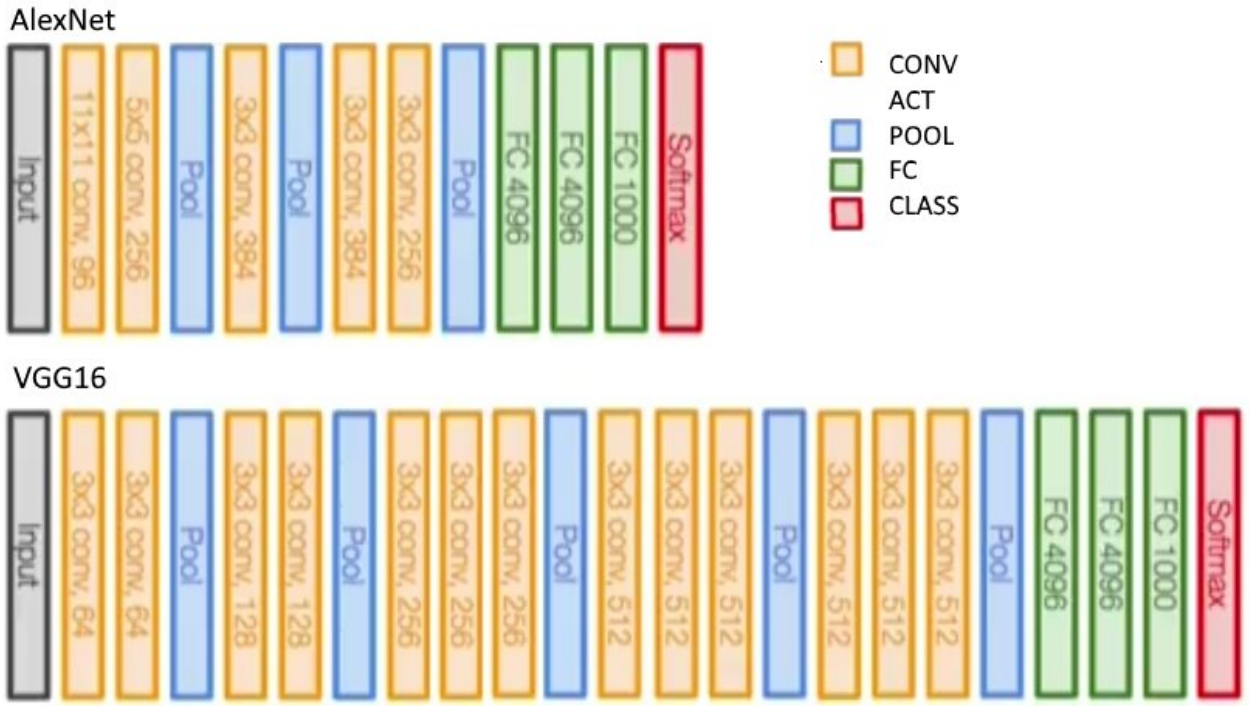
ImageNet Classification Challenge



https://medium.com/appyhigh-technology-blog/convolutional-neural-networks-a-brief-history-of-their-evolution-ee3405568597

VGGNet (2014)

- Small filters
  - Stack of small filters have the same effect than one larger one
- More layers
  - More non-linearities
  - More abstract feature representations



However, adding more layers than this would hurt performance…

ResNet (2015)

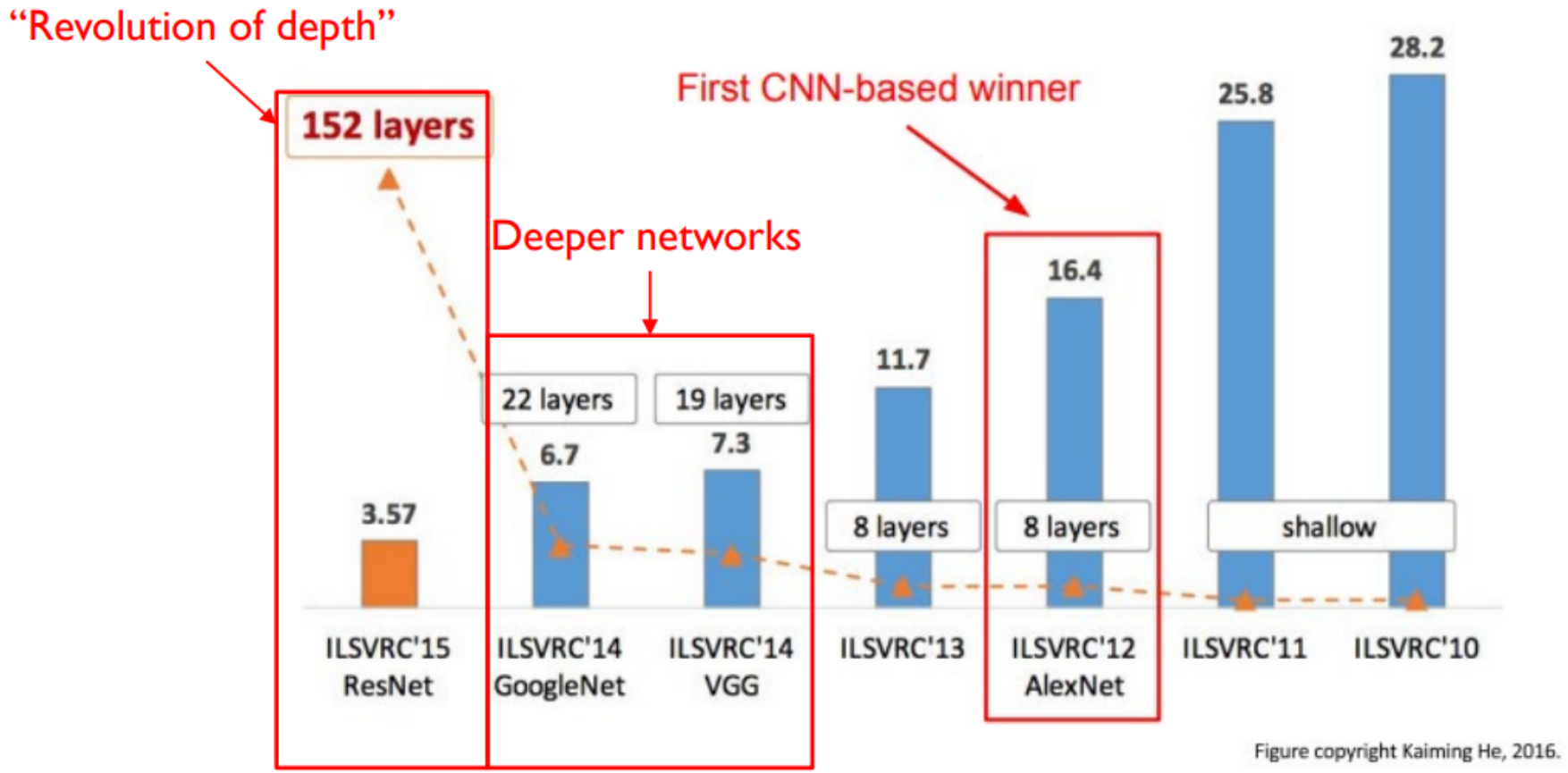
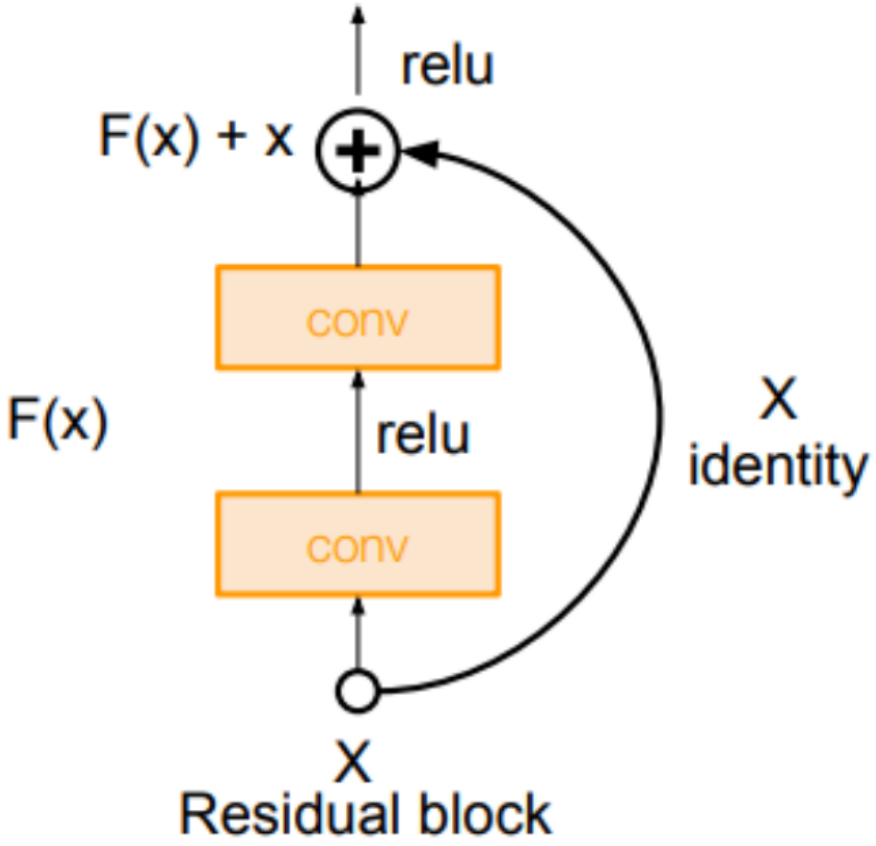


Figure copyright Kaiming He, 2016.

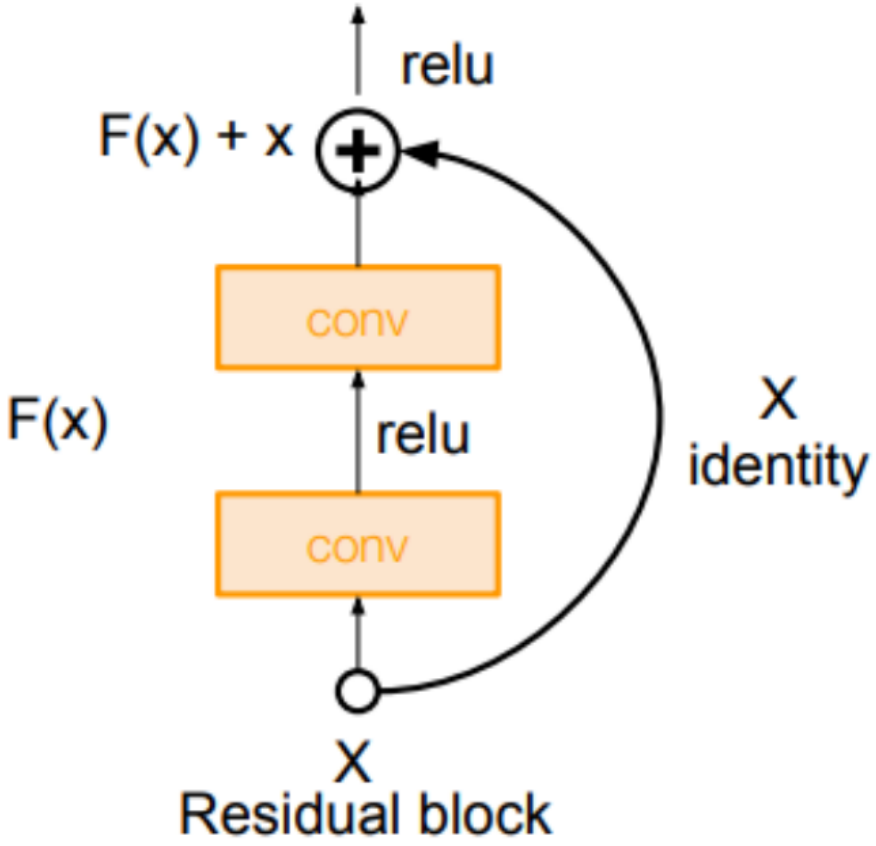
ResNet (2015)

- Main novelty: residual connections
- Architecture consists of residual blocks

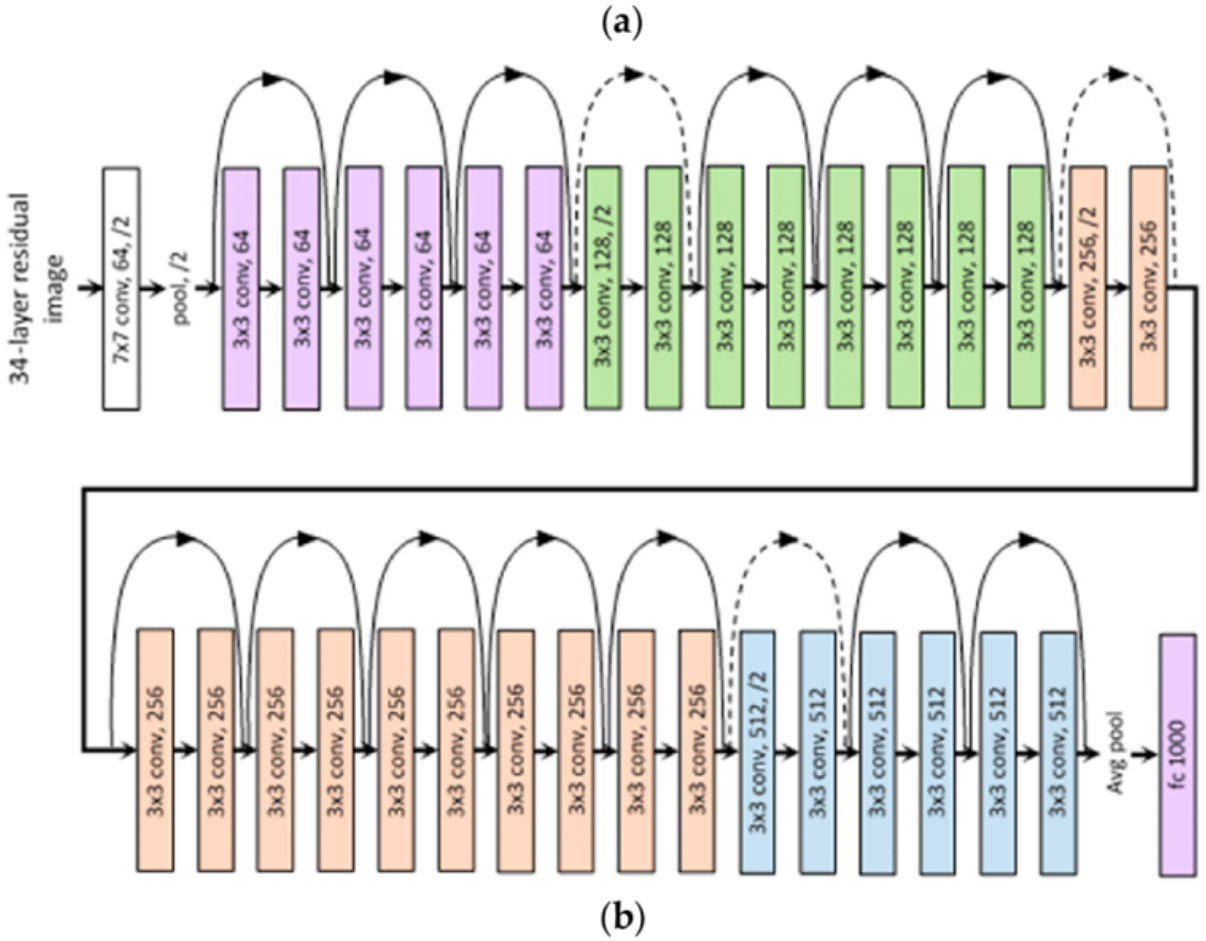


ResNet (2015)

- Why? Deeper networks!
- Training deep networks is hard
  - Vanishing gradient problem
  - Composition of many functions is difficult to optimise
- Residual blocks offer a skip/shortcut
  - We only need to learn deviation from the identi
- Commonly used in most deep architectures

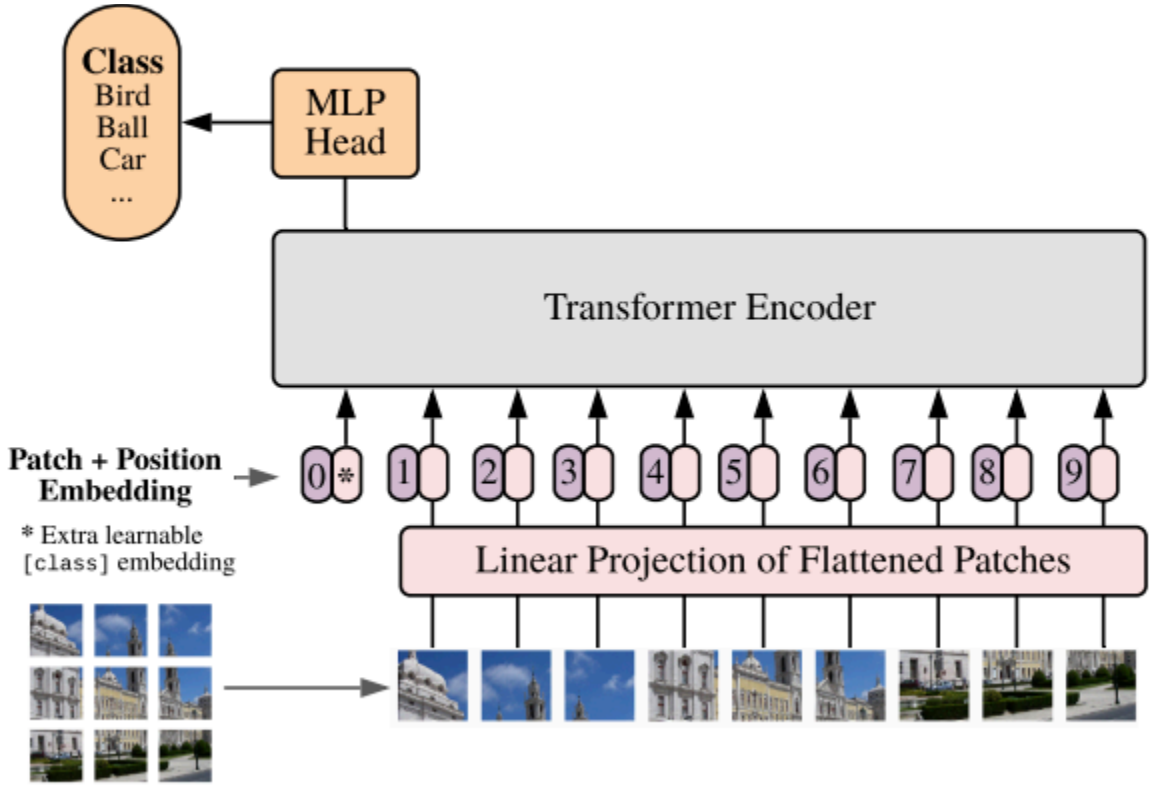


ResNet (2015)


(a)
34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
Avg pool
fc 1000
(b)

Vision Transformers (2020)

- Not a CNN:
  Original implementation avoids convolutions
- Attention and positional encoding allows interactions across entire image, without needing to build a receptive field like CNNs



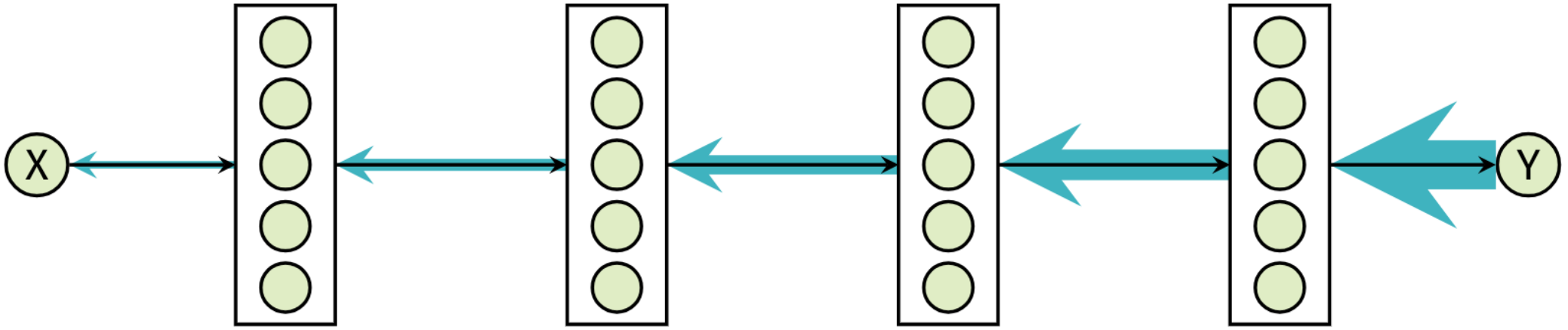
Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Computer Vision 2

COMPSCI 760
2024 Semester 1

Olivier Graffeuille
ogra439@aucklanduni.ac.nz

Vanishing Gradients

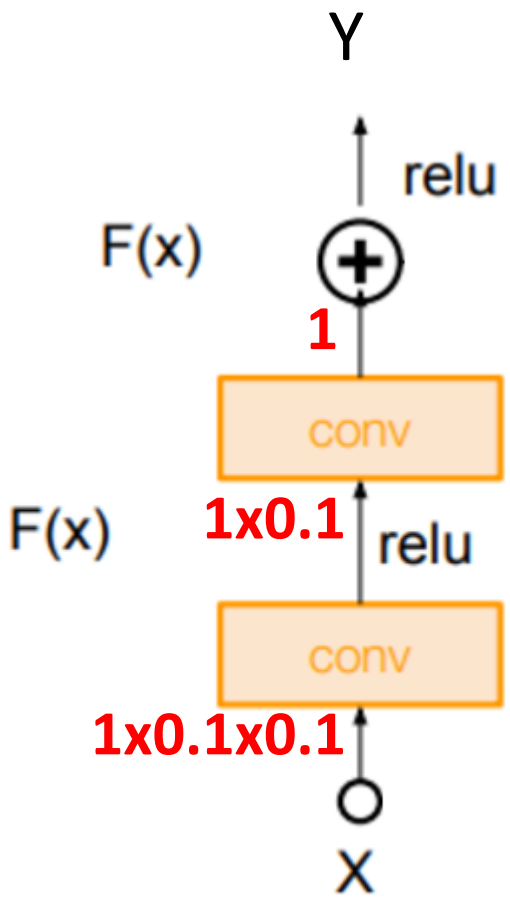


$$\delta^{(1)} = \varphi'(A^{(1)}) \circ W^{(0)T}\left(\varphi'(A^{(2)}) \circ W^{(1)T}\left(\varphi'(A^{(3)}) \circ W^{(2)T}\left(\varphi'(A^{(4)}) \circ (Y^{(3)} - Y)\right)\right)\right)$$

- A long sequence of mathematical operations is unstable

Clarifications from Last Lecture

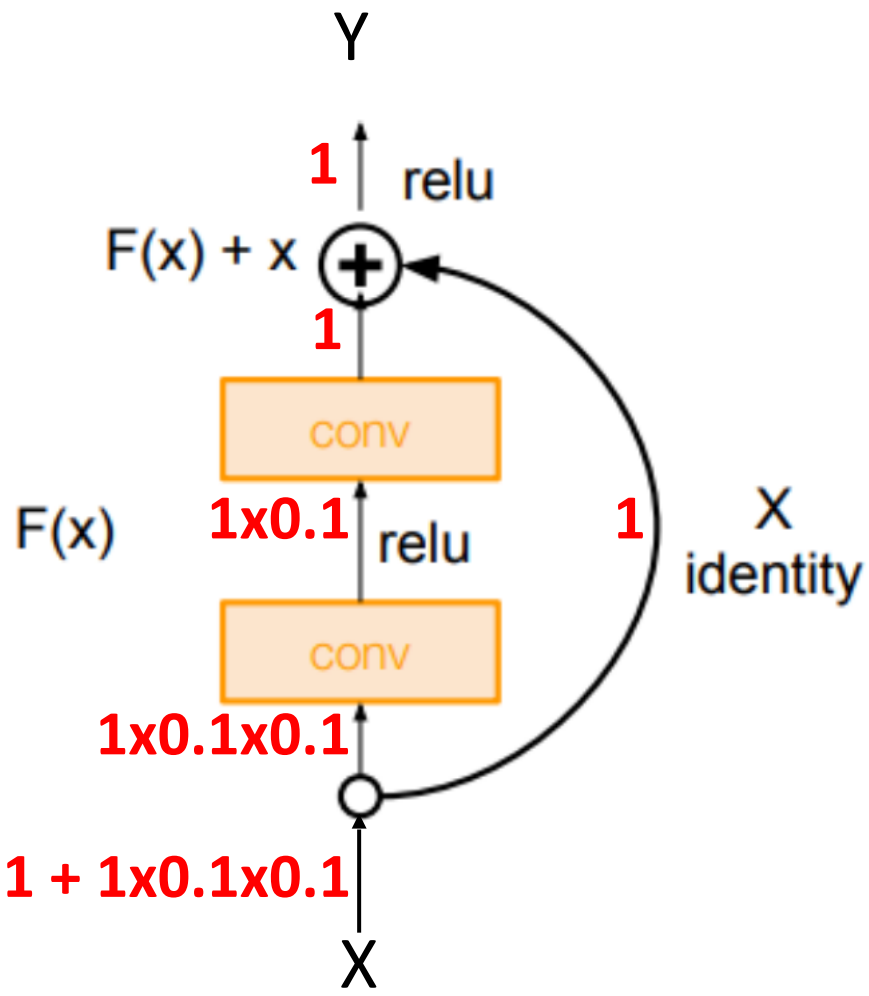
Regular Deep Networks



Clarifications from Last Lecture

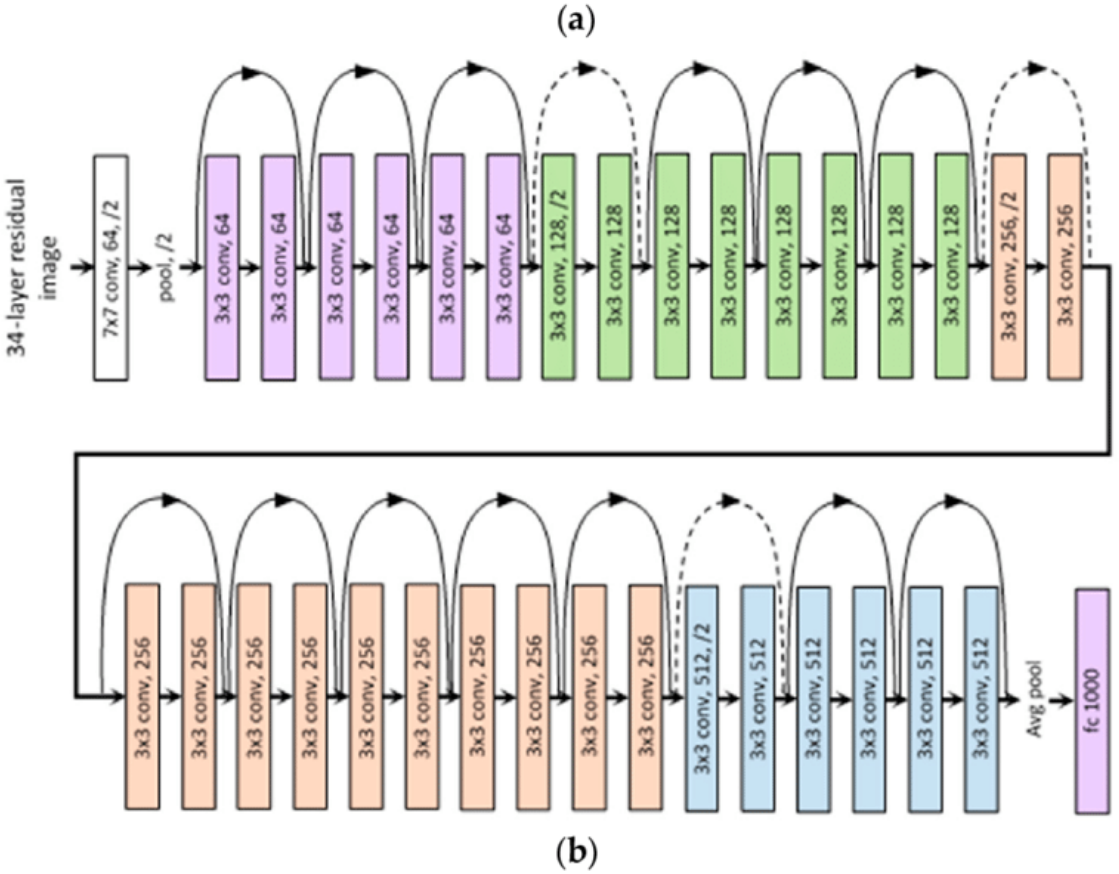
Residual Networks

- Avoids vanishing gradients

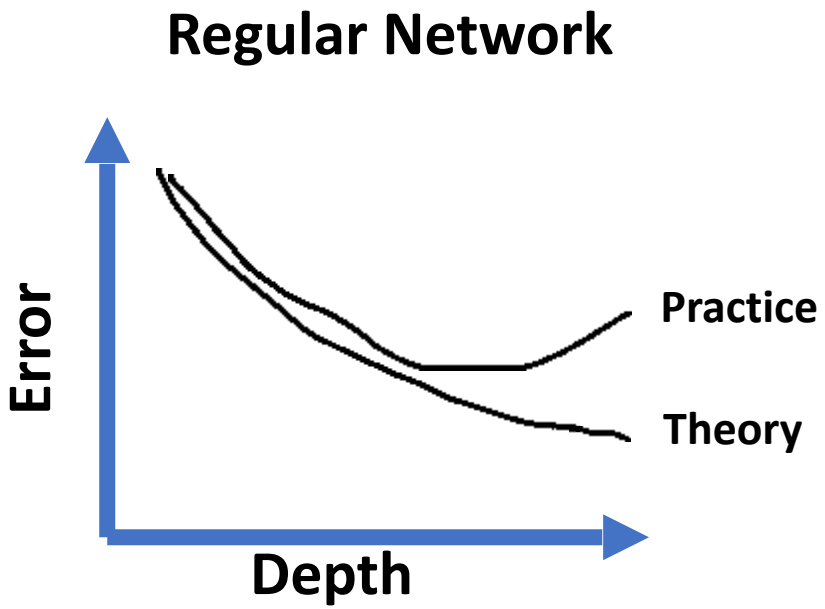


ResNet (2015)

- Adding new layers, at worst, is like adding the identity function (no change)
  - These new layers can learn a function which is better than the identity



Depth vs. Performance



- Regular CNNs have enough ***capacity*** to learn accurate models
- However, we aren't able to train them accurately
  - Vanishing gradients
  - Identity function start

Vision Transformers vs. CNNs

CNNs

- Requires large datasets
- Similar to human vision

Vision transformers

- Requires HUGE datasets
- Very different from human vision
- Useful for vision/language interaction (foundation VLMs)

Outline

- Computer vision tasks
  - Classification, localisation, landmark detection
- Object Detection
  - Sliding window method
  - Bounding box prediction (YOLO)
  - Anchor boxes
  - Putting it all together

Outline

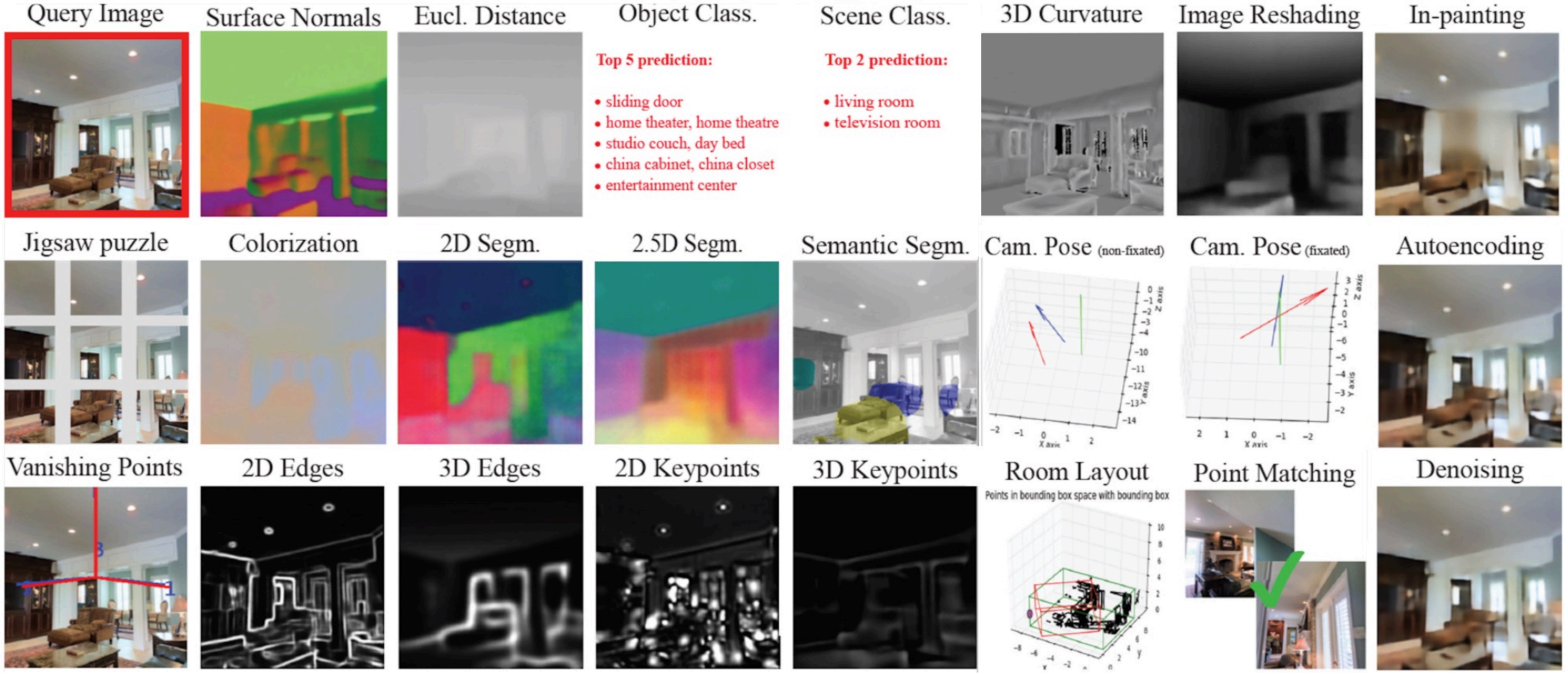
- ***Computer vision tasks***
  - ***Classification, localisation, landmark detection***
- Object Detection
  - Sliding window method
  - Bounding box prediction (YOLO)
  - Anchor boxes
  - Putting it all together

Acknowledgement/Disclaimer

The following slides are taken from Andrew's Ng Coursera Convolutional Neural Networks lectures, part of the Deep Learning Specialization.

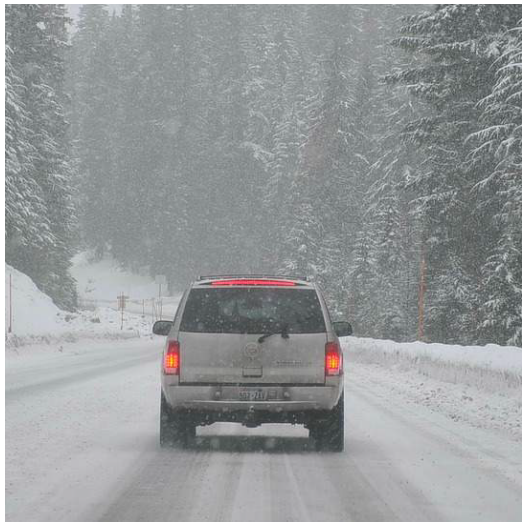
The source of the slides comes from DeepLearning.AI

Computer Vision Tasks



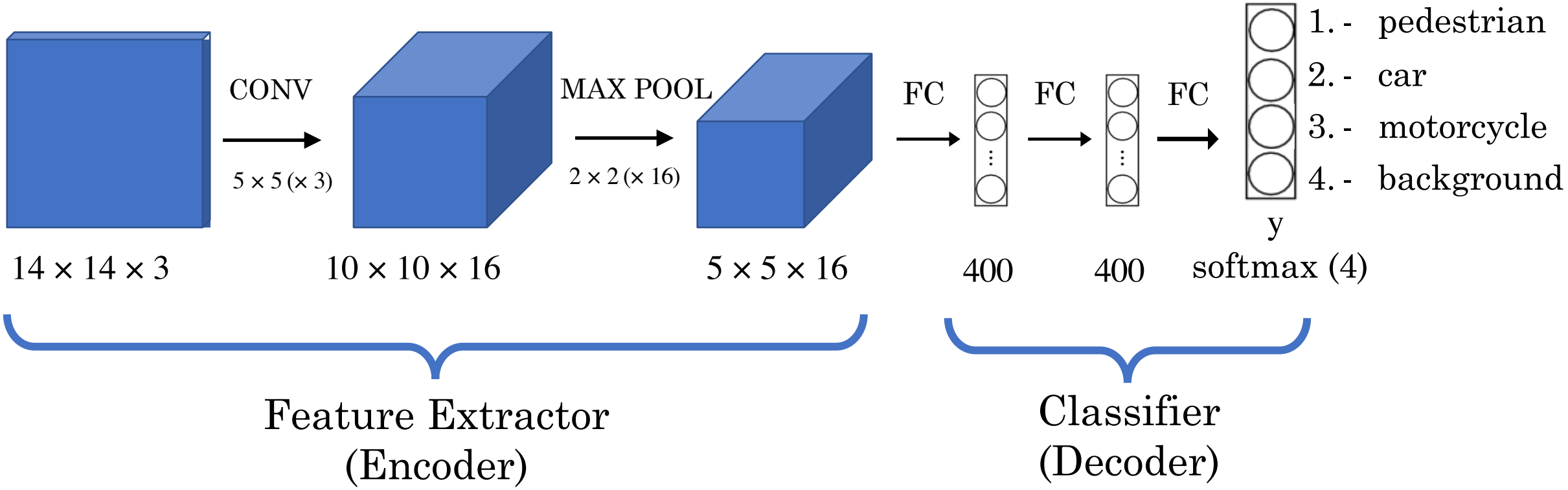
Classification

Image classification



« Car »

Classification



Classification + Localisation

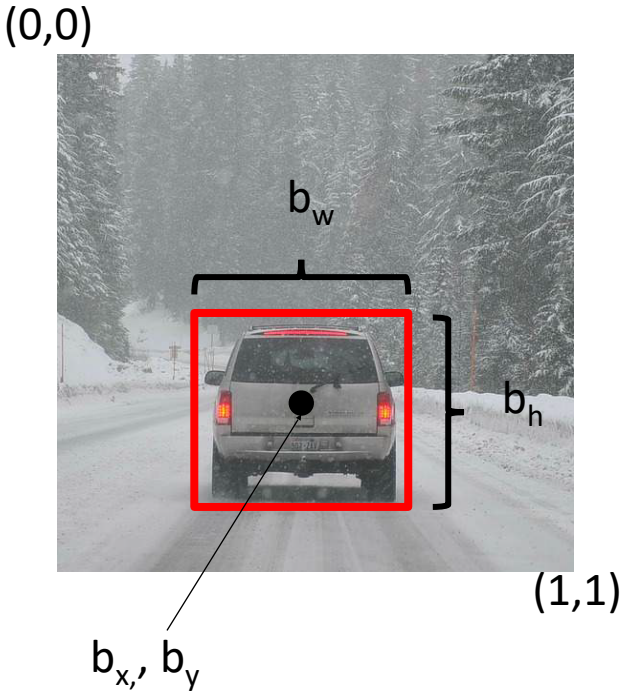
Image classification

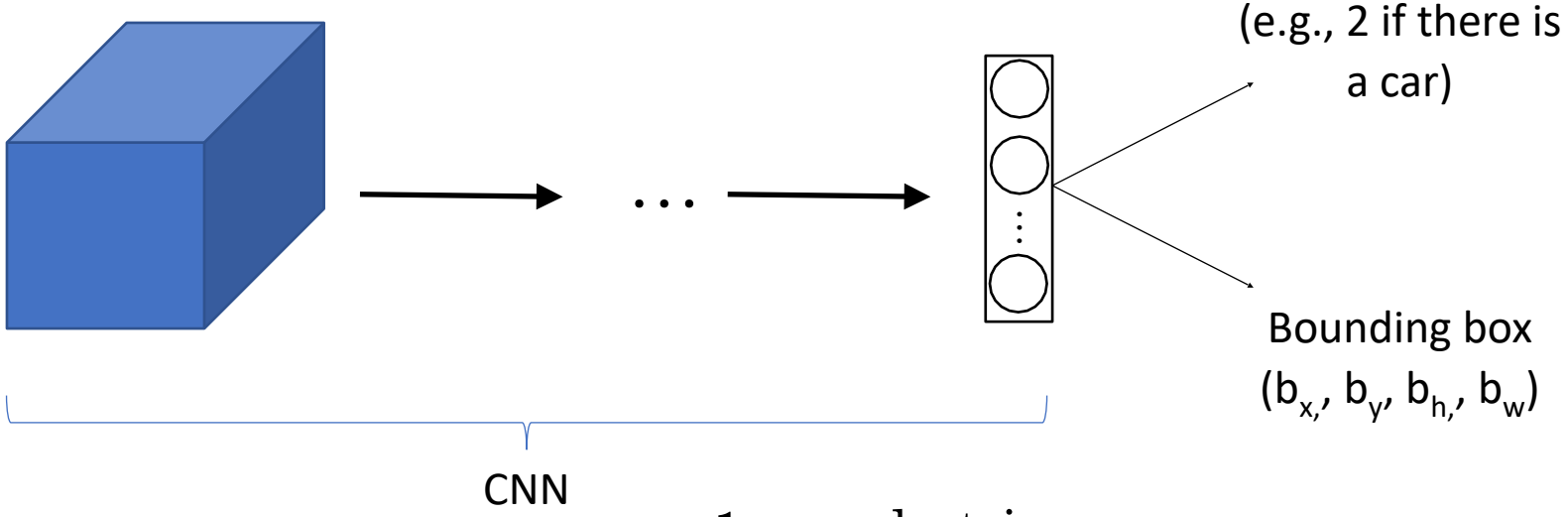
« Car »

Classification with localization

« Car » + localisation
(e.g., ***bounding box***)

Classification + Localisation





1. - pedestrian
2. - car
3. - motorcycle
4. - background

Computer Vision Tasks

Need to output b_x, b_y, b_h, b_w, class label (1-4)

1.- pedestrian
2.- car
3.- motorcycle
4.- background

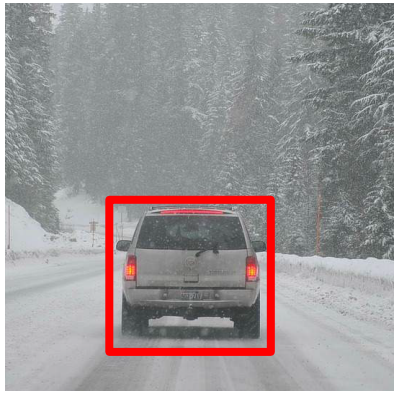
Is there any object? (1 or 0 if none of the classes)

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

If $p_c = 1$

Bounding box coordinates

Class



$$\begin{bmatrix} 1 \\ 0.5 \\ 0.7 \\ 0.3 \\ 0.4 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

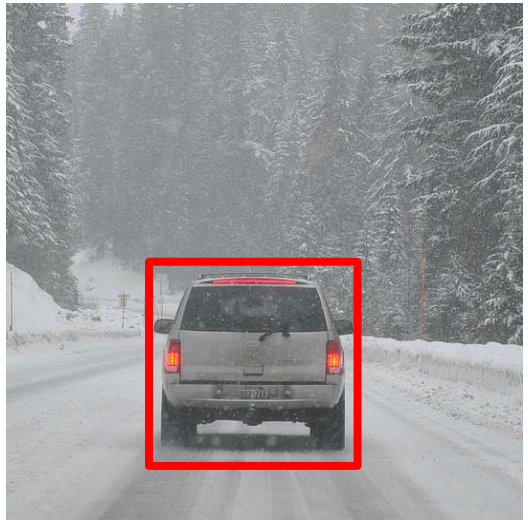
Do not care

Classification + Localisation

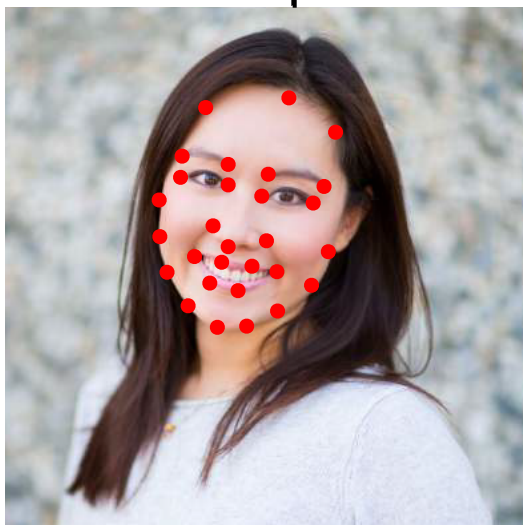
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

$$L(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \ldots + (\hat{y}_8 - y_8)^2 & if\ p_c = 1 \\ (\hat{y}_1 - y_1)^2 & if\ p_c = 0 \end{cases}$$

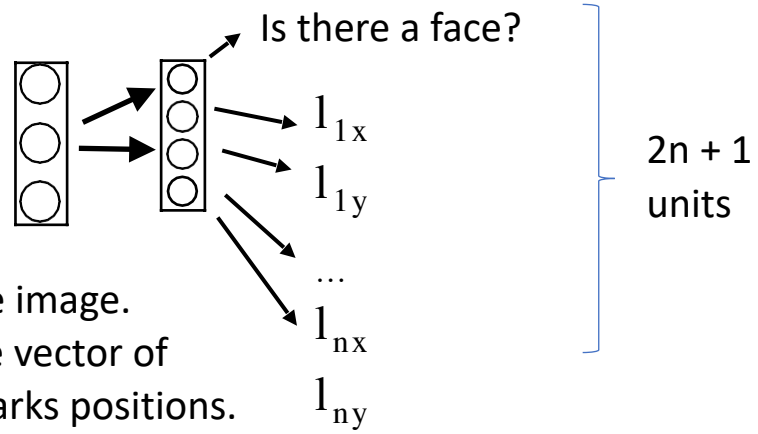
Landmark Detection



b_x, b_y, b_h, b_w



l_{1x}, l_{1y}
l_{2x}, l_{2y}
l_{1x}, l_{1y}
l_{2x}, l_{2y}
…



X is the image.
Y is the vector of landmarks positions.

Can also be used for pose recognition.

Object Detection

Image classification

« Car »

Classification with localization

« Car » + localisation (e.g., bounding box)

1 object

Object Detection

Multiple objects + localisations (e.g., bounding boxes)

Outline

- Computer vision tasks
  - Classification, localisation, landmark detection
- ***Object Detection***
  - ***Sliding window method***
  - Bounding box prediction (YOLO)
  - Anchor boxes
  - Putting it all together

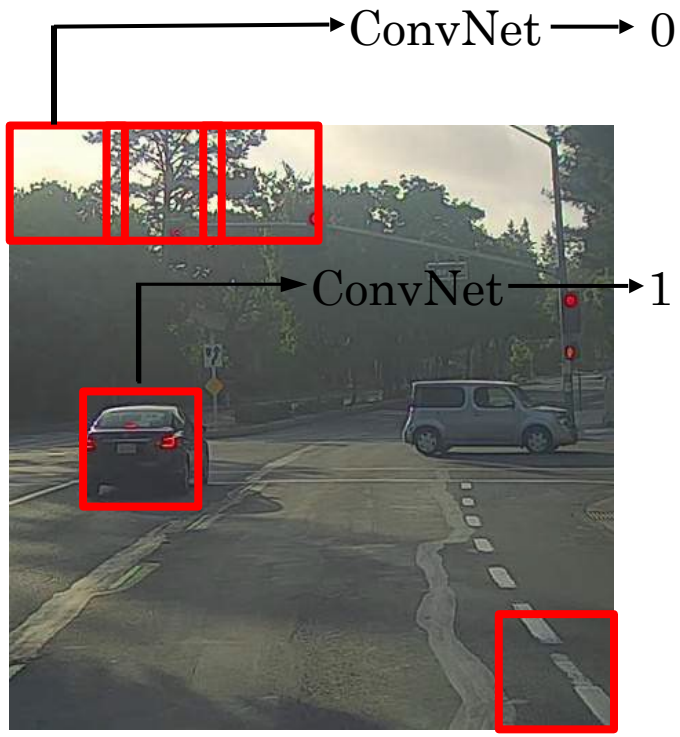
Can we just use classification?

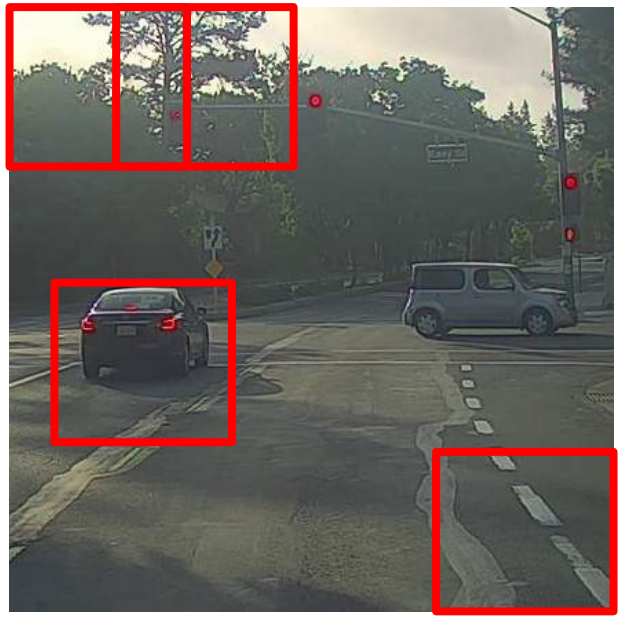
Training set:

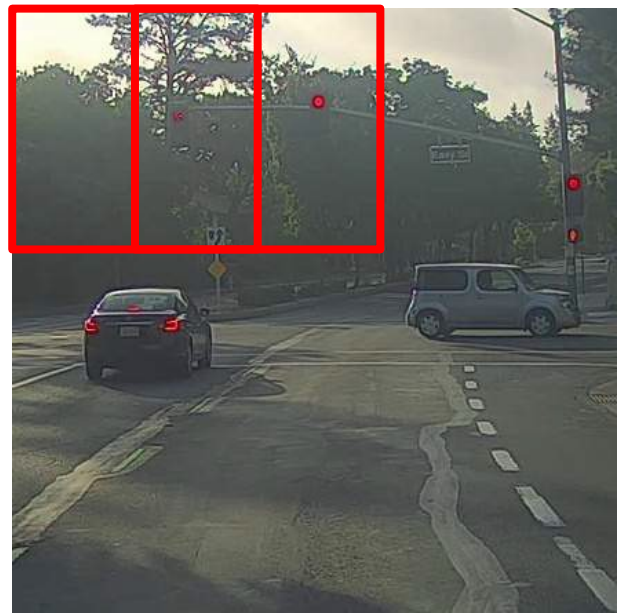
x	y
	1
	1
	0
	0

 → ConvNet → 1

Sliding Window Method

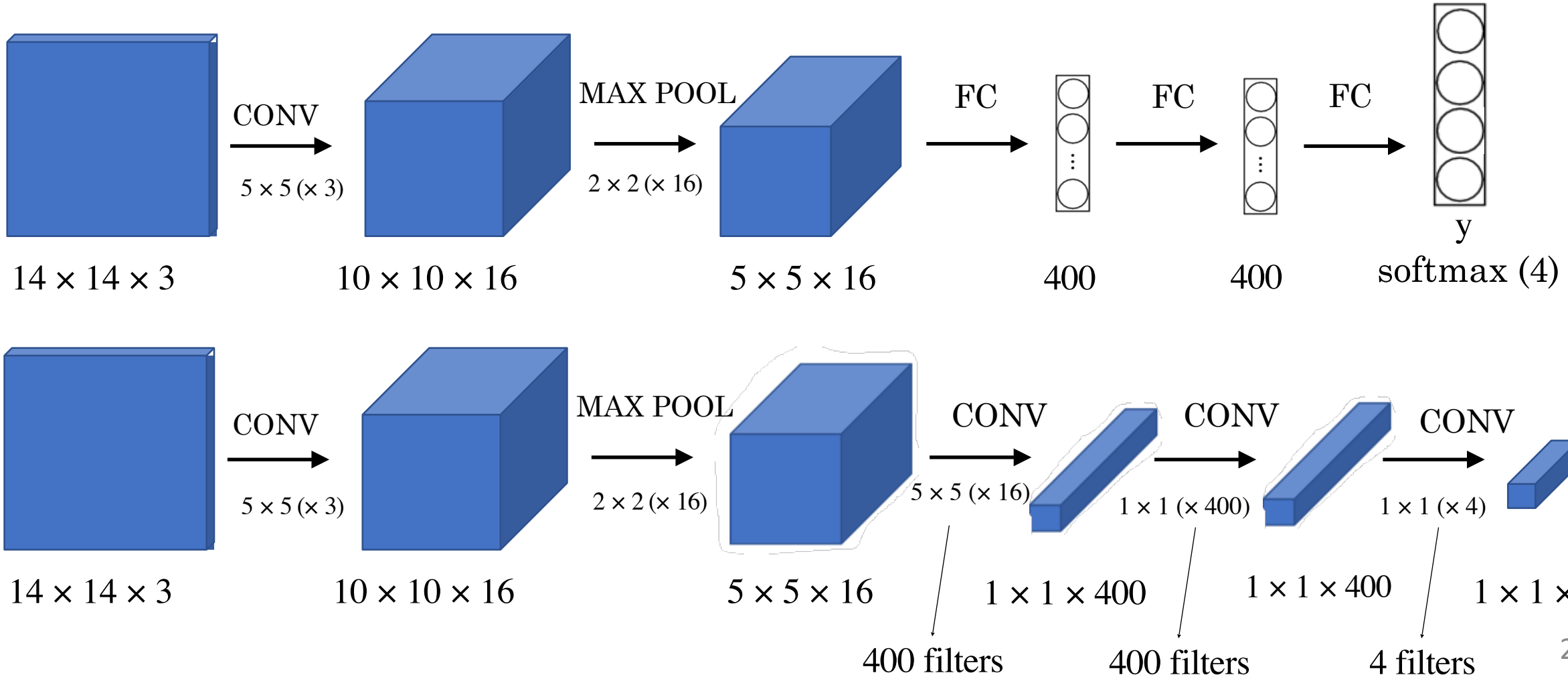




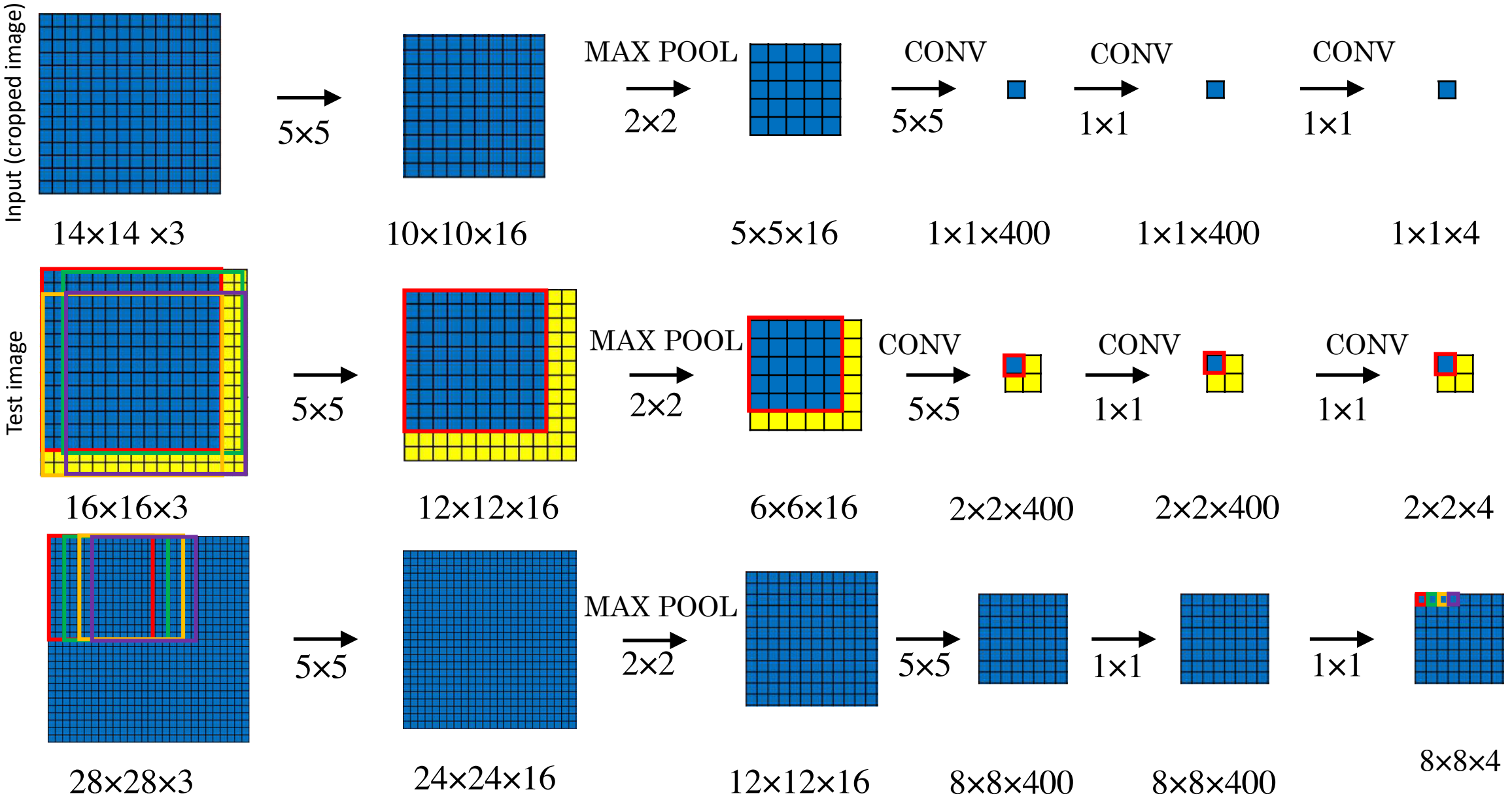


What are the problems with this approach?

Linear Layers as Convolutions

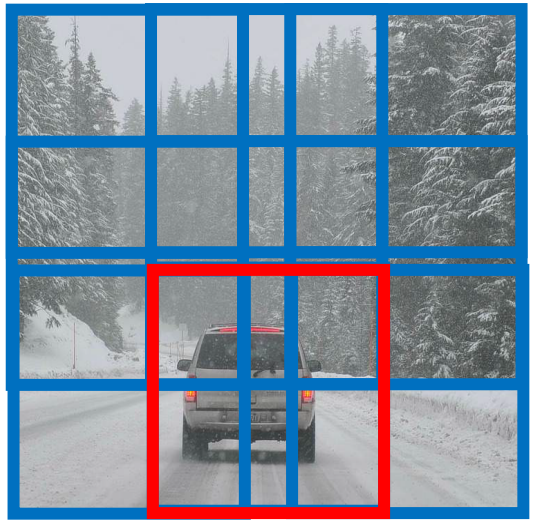


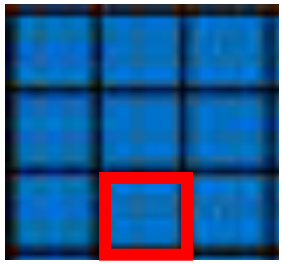
Convolution implementation of sliding windows



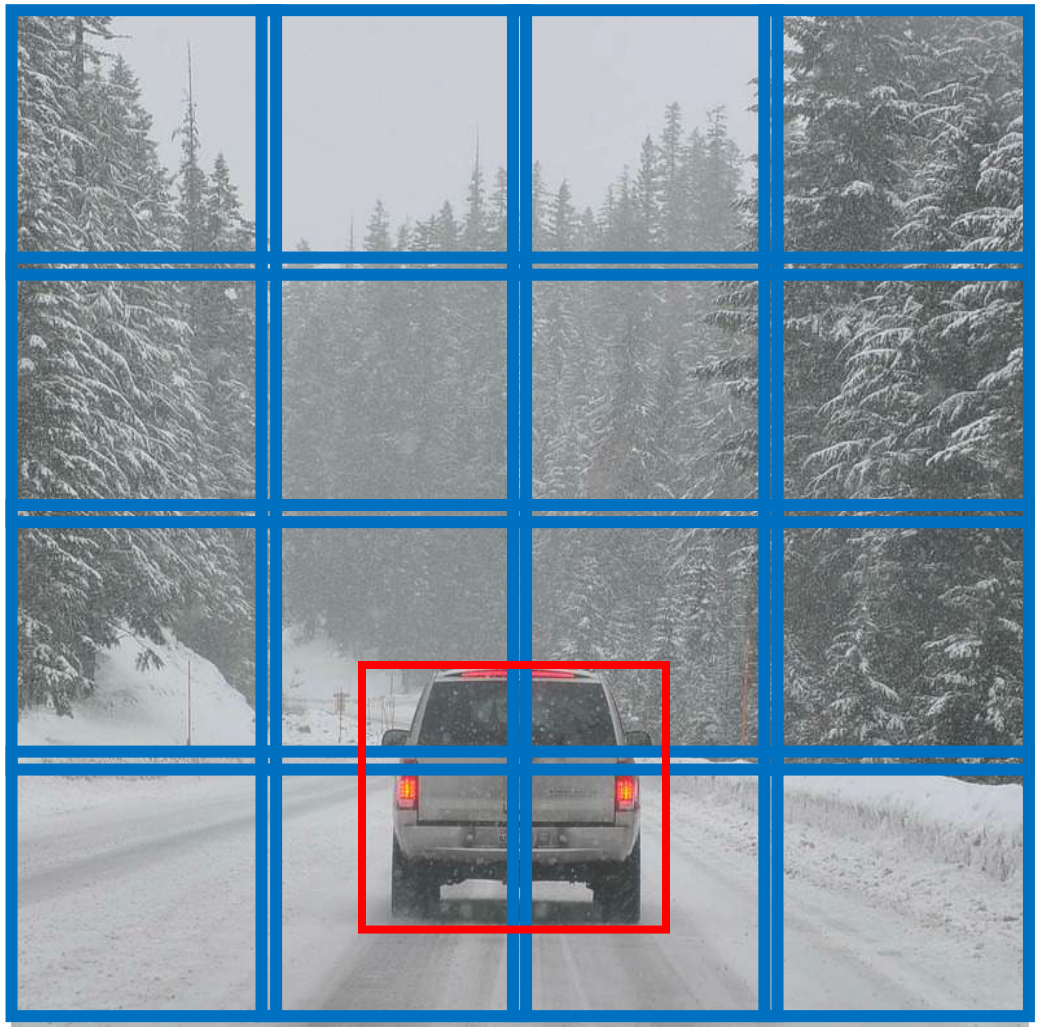
[Sermanet et al., 2014, OverFeat: Integrated recognition, localization and detection using convolutional networks]

Convolution implementation of sliding windows





Problem: Inaccurate estimation



Outline

- Computer vision tasks
  - Classification, localisation, landmark detection
- Object Detection
  - Sliding window method
  - ***Bounding box prediction (YOLO)***
  - Anchor boxes
  - Putting it all together

YOLO: You Only Look Once (2015)

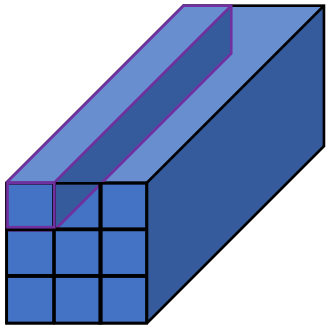
- This paper revolutionised object detection
- What do these two images have in common?

[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

YOLO: You Only Look Once

What size is the output?

Target output: 3×3×8



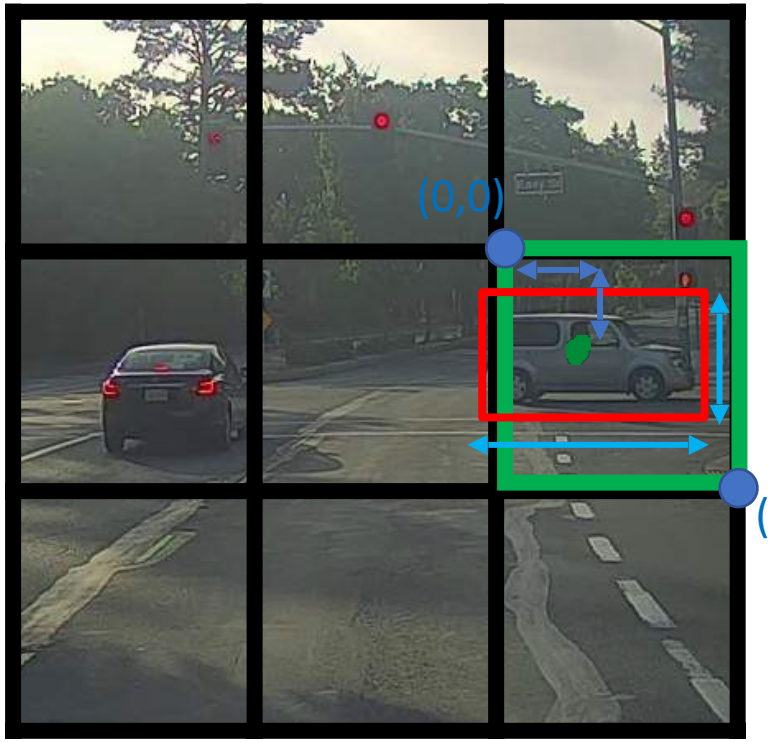
Labels for training

For each grid cell:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

Bounding Boxes for Object Detection



$$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

- b_x → x position of middle of bb to cell (0,0) = 0.4
- b_y → y position of middle of bb to cell (0,0) = 0.3

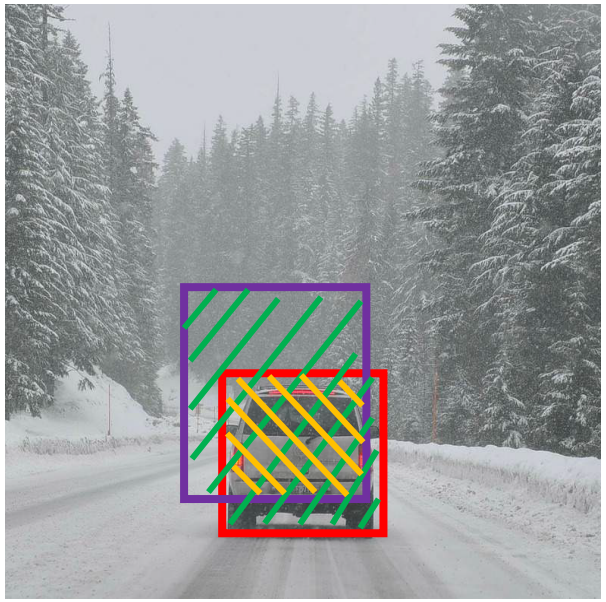
Between 0 and 1

- b_h → heigth of bb as a fraction of cell size = 0.5
- b_w → width of bb as a fraction of cell size = 0.9

Can be > 1

[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

Bounding Boxes: Evaluation



Intersection over union (IoU):

$$= \frac{size\ of}{size\ of}$$

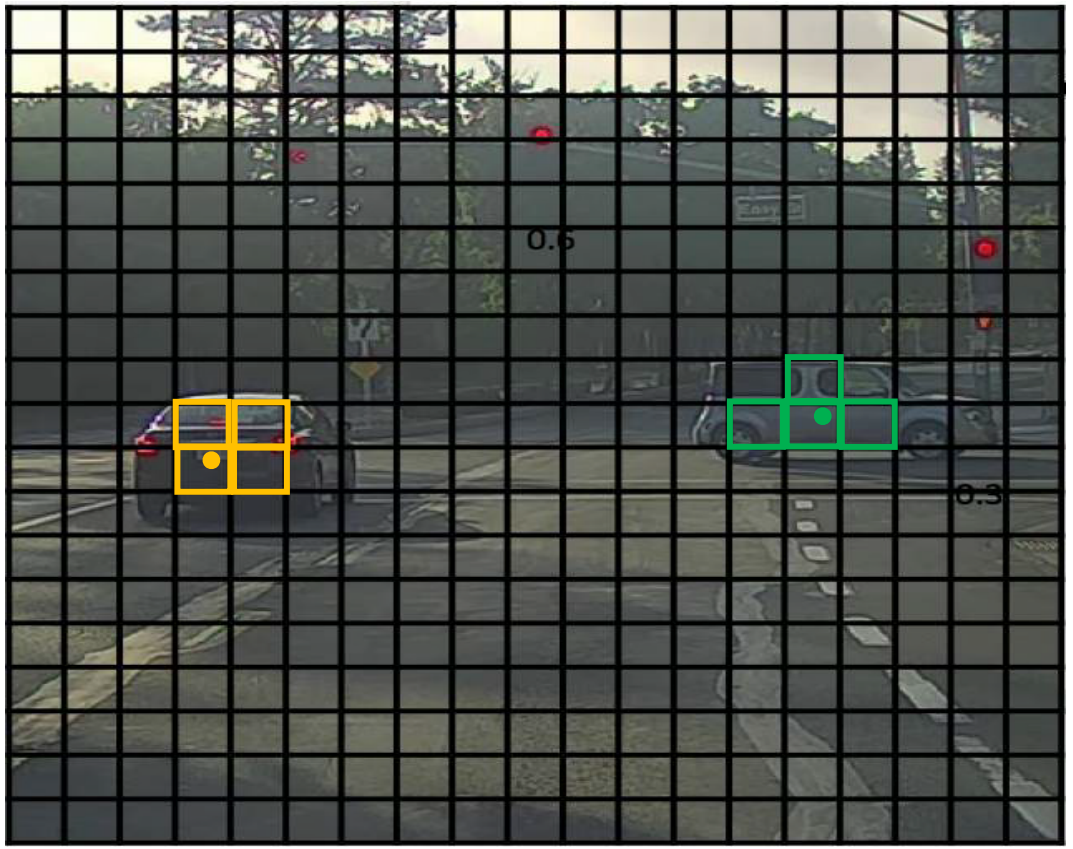
"Correct" if IoU ≥ 0.5

More generally, IoU is a measure of the overlap between two bounding boxes.

[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

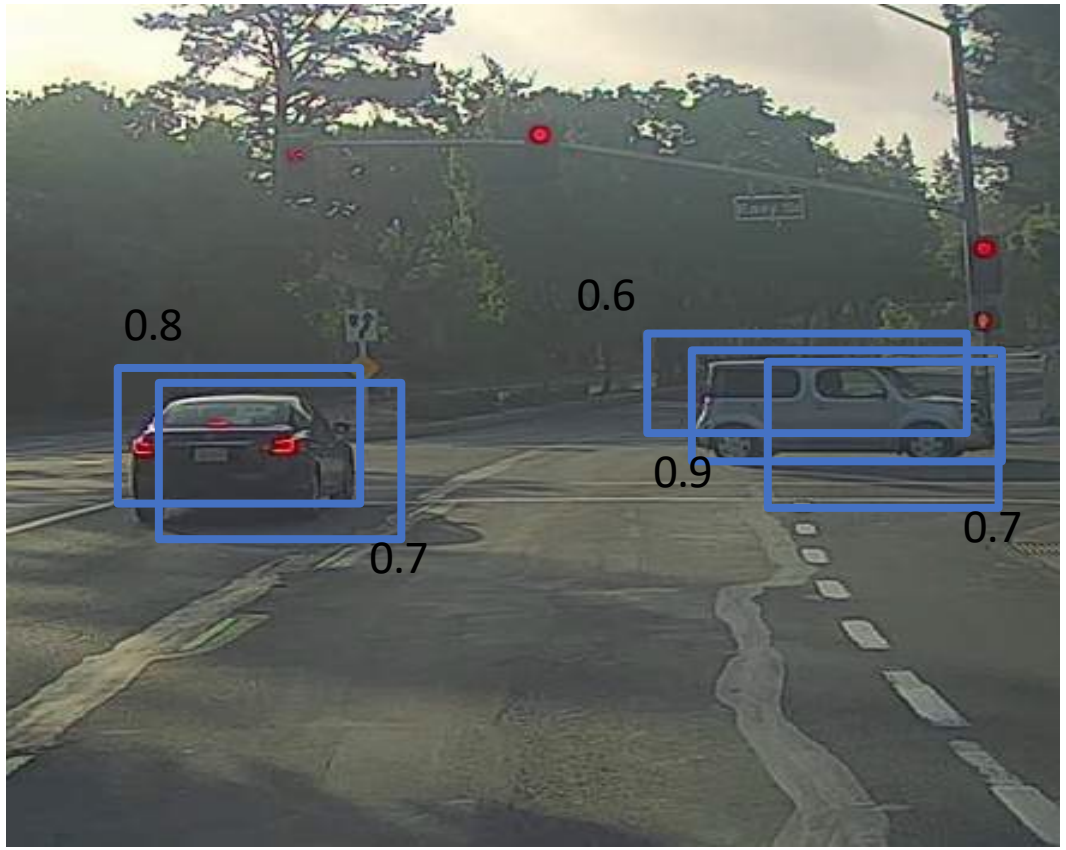
Bounding Boxes: Duplicates

Bounding Boxes: Duplicates



19×19

Non-Max Suppression



Each output prediction is:

$$\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$$

Discard all boxes with $p_c \leq 0.6$

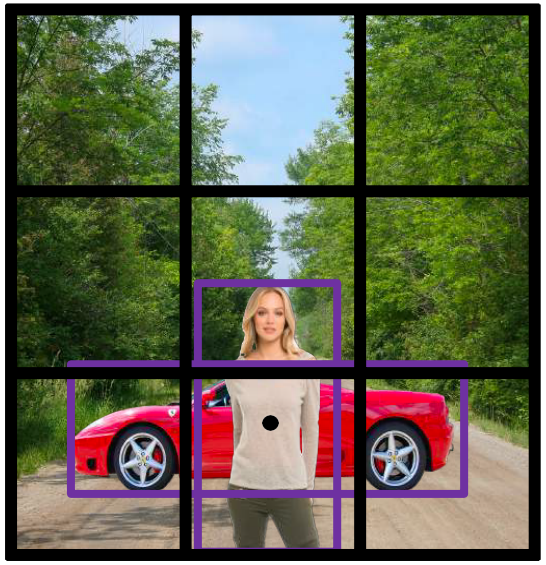
While there are any remaining boxes:

- Pick the box with the largest p_c. Output that as a prediction.
- Discard any remaining box with IoU ≥ 0.5 with the box output in the previous step.

Outline

- Computer vision tasks
  - Classification, localisation, landmark detection
- Object Detection
  - Sliding window method
  - Bounding box prediction (YOLO)
  - ***Anchor boxes***
  - Putting it all together

Anchor Box Method



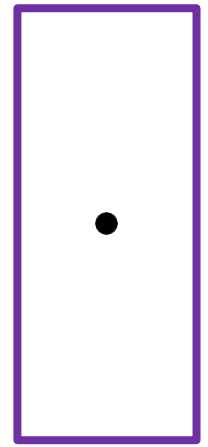
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \end{bmatrix}$$

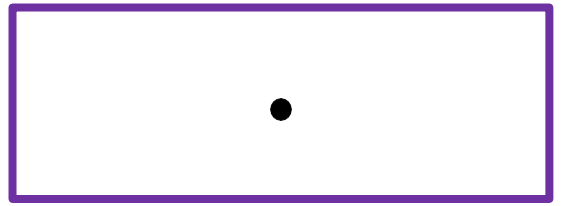
Anchor box 1 (more similar to pedestrian's shape)

Anchor box 2 (more similar to car's shape)

Anchor box 1:

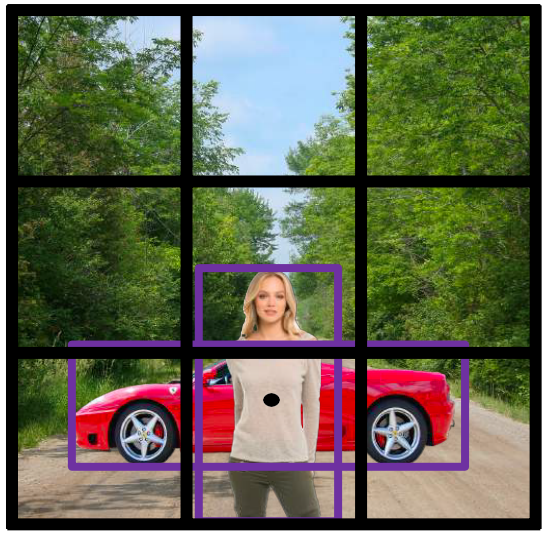


Anchor box 2:

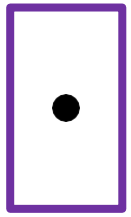


36

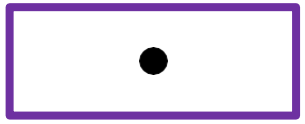
Anchor Box: Example



Anchor box 1



Anchor box 2



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \end{bmatrix} \quad \begin{matrix} \text{Car only?} \\ \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \end{matrix}$$

Anchor box 1

Anchor box 2

Anchor Box Method

Previously:

Each object in training image is assigned to a grid cell that contains that object's midpoint.

Output (for each grid cell and 3 classes)

y: 3×3×8

With two anchor boxes:

Each object in training image is assigned to grid a cell that contains that object's midpoint, and anchor box for the grid cell with highest IoU.

Output (for each grid cell, 3 classes, and 2 anchor boxes)

y: 3×3×16 (3×3×2×8)

Anchor Box Method

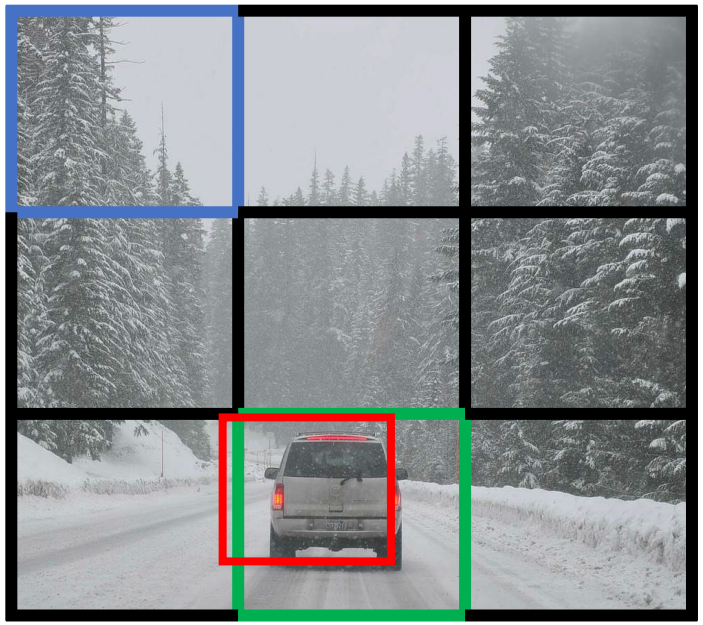
- Can detect multiple objects within a cell, but:
  - Only one object related to each anchor box per cell (usually enough)
- Allows the network to specialise into different object types

Outline

- Computer vision tasks
  - Classification, localisation, landmark detection
- Object Detection
  - Sliding window method
  - Bounding box prediction (YOLO)
  - Anchor boxes
  - ***Putting it all together***

YOLO: You Only Look Once

Training YOLO

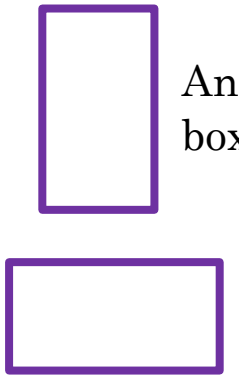


3×3×16

y is 3×3×2×8

5 + 3 classes

anchor boxes

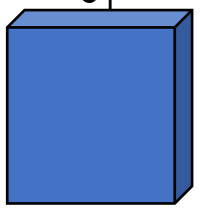


$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \end{bmatrix} \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 0 \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

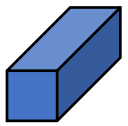
Anchor box 2 (bounding box for car is a bit wider than higher)

x

100 × 100 × 3



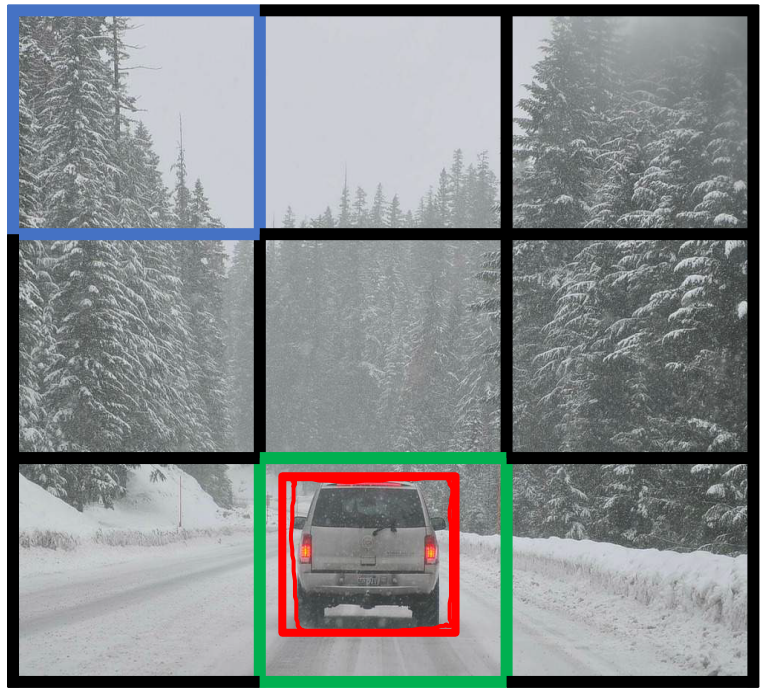
ConvNet

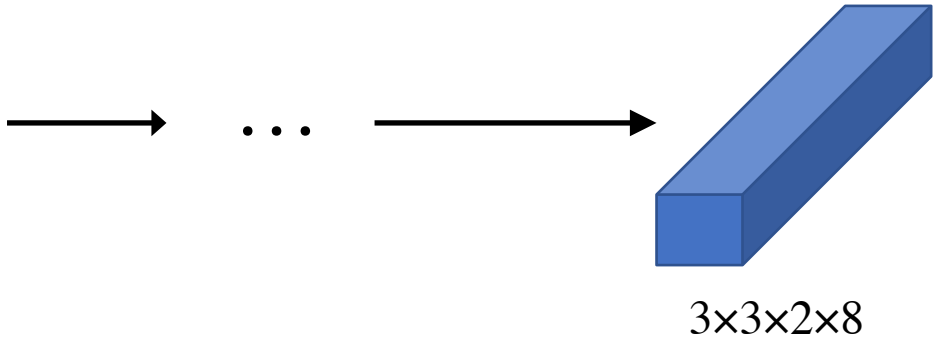


y

3 × 3 × 16

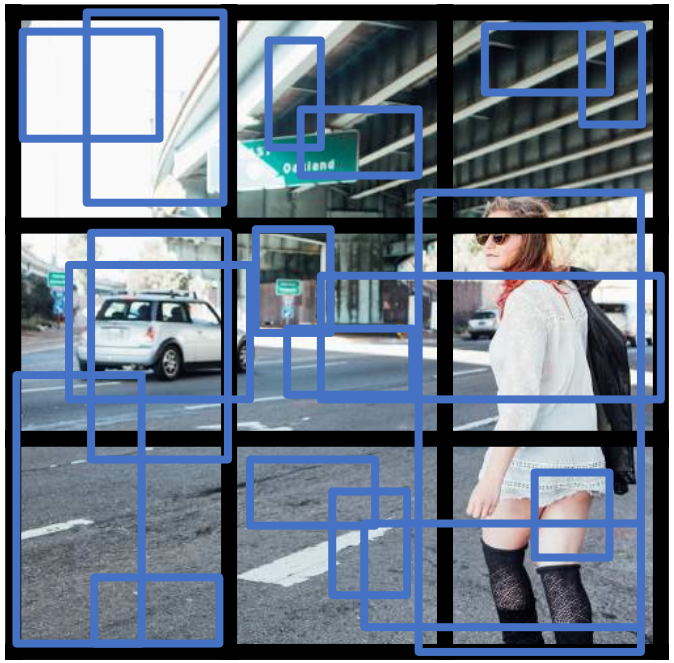
Predicting with YOLO





$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \end{bmatrix} \begin{bmatrix} 0 \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ 0 \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \end{bmatrix} \begin{bmatrix} 0 \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Predicting with YOLO: Non-Max Suppression



- For each grid call, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, motorcycle) independently run non-max suppression to generate final predictions.

YOLO Summary

- Object detection by estimating object position in a grid of cells
- Convolutional implementation for computational speed
- Anchor boxes to predict multiple objects per cell
- Non-max suppression to remove duplicate predictions

YOLO Versions

- **YOLOv1**: Introduced a novel real-time object detection system that predicts bounding boxes and class probabilities directly from full images in one evaluation, significantly speeding up detection speeds.
- **YOLOv2:** Improved accuracy with anchor boxes, finer-grained features using a passthrough layer from earlier layers, and the ability to detect over 9000 object categories using a joint training method on both ImageNet and COCO datasets.
- **YOLOv3**: Enhanced detection at various scales by incorporating three different scales for prediction, and used a deeper network backbone (Darknet-53) for improved feature extraction.
- **YOLOv4**: Boosted accuracy and efficiency by integrating advanced techniques such as Mish activation, Cross-Stage Partial connections, and Self-Adversarial Training, while still maintaining real-time performance.
- **YOLOv8…**

An interesting history

- The main author, Joseph Redmon, was a computer vision PhD student
- The first paper's name "YOLO" is a reference to a meme at the time
- The second paper was called "YOLO9000" and promised to predict "over 9000" classes, a reference to dragon ball Z

An interesting history

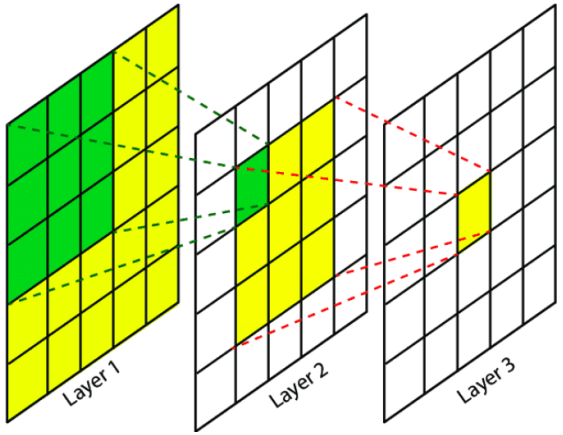
- Joseph stopped working on computer vision for ethical reasons, *e.g.* military drones

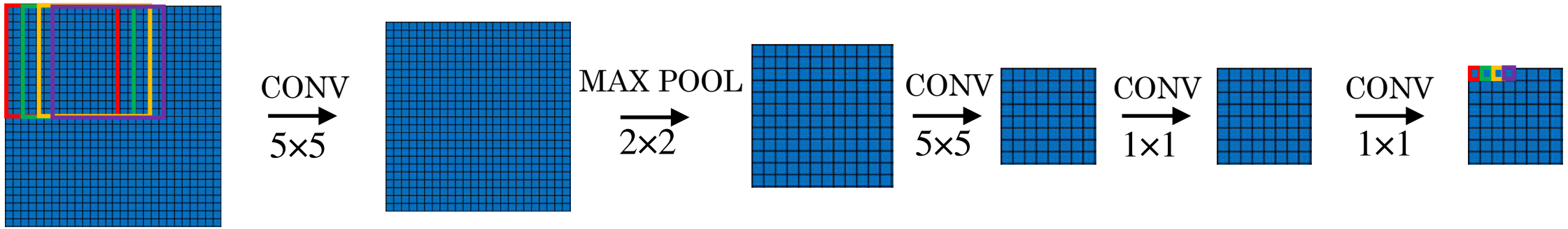
Joe Redmon
@pjreddie

I stopped doing CV research because I saw the impact my work was having. I loved the work but the military applications and privacy concerns eventually became impossible to ignore.twitter.com/RogerGrosse/st…

Receptive Field

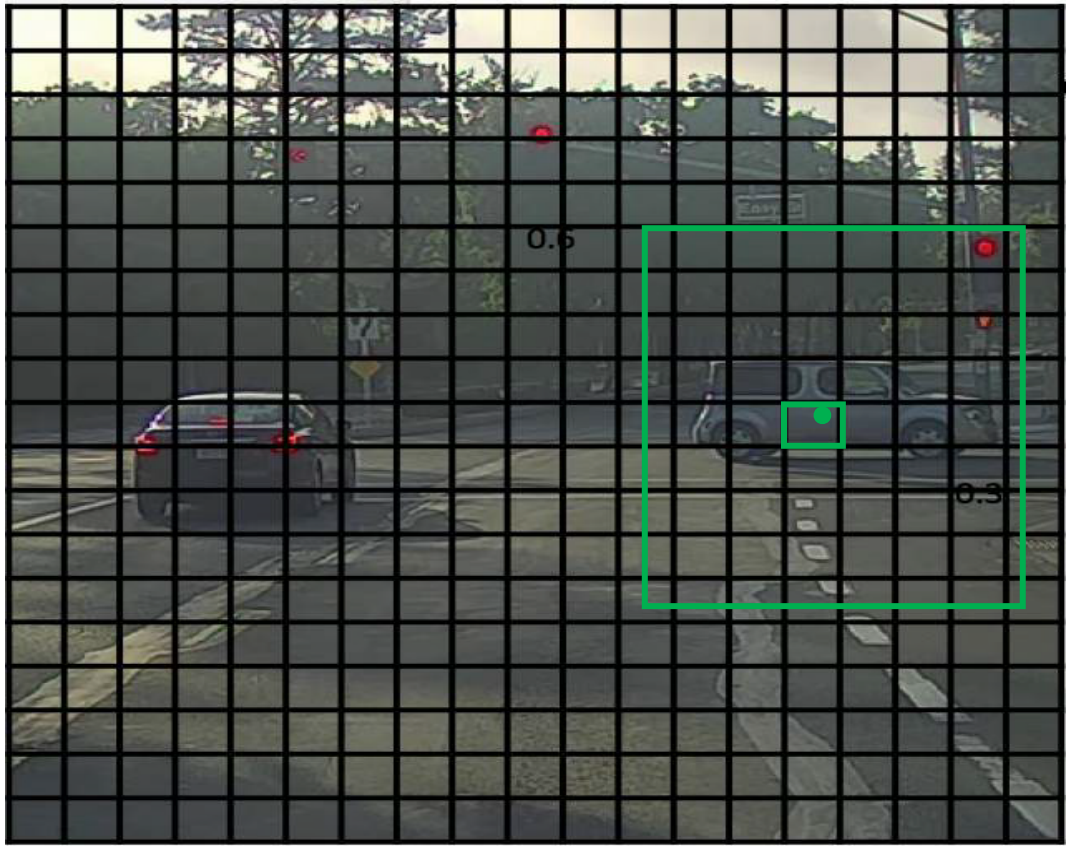
- The prediction within each feature/cell is the function of a larger patch of the input image aka
  - This is due to convolutional and pooling layer





Cells in YOLO

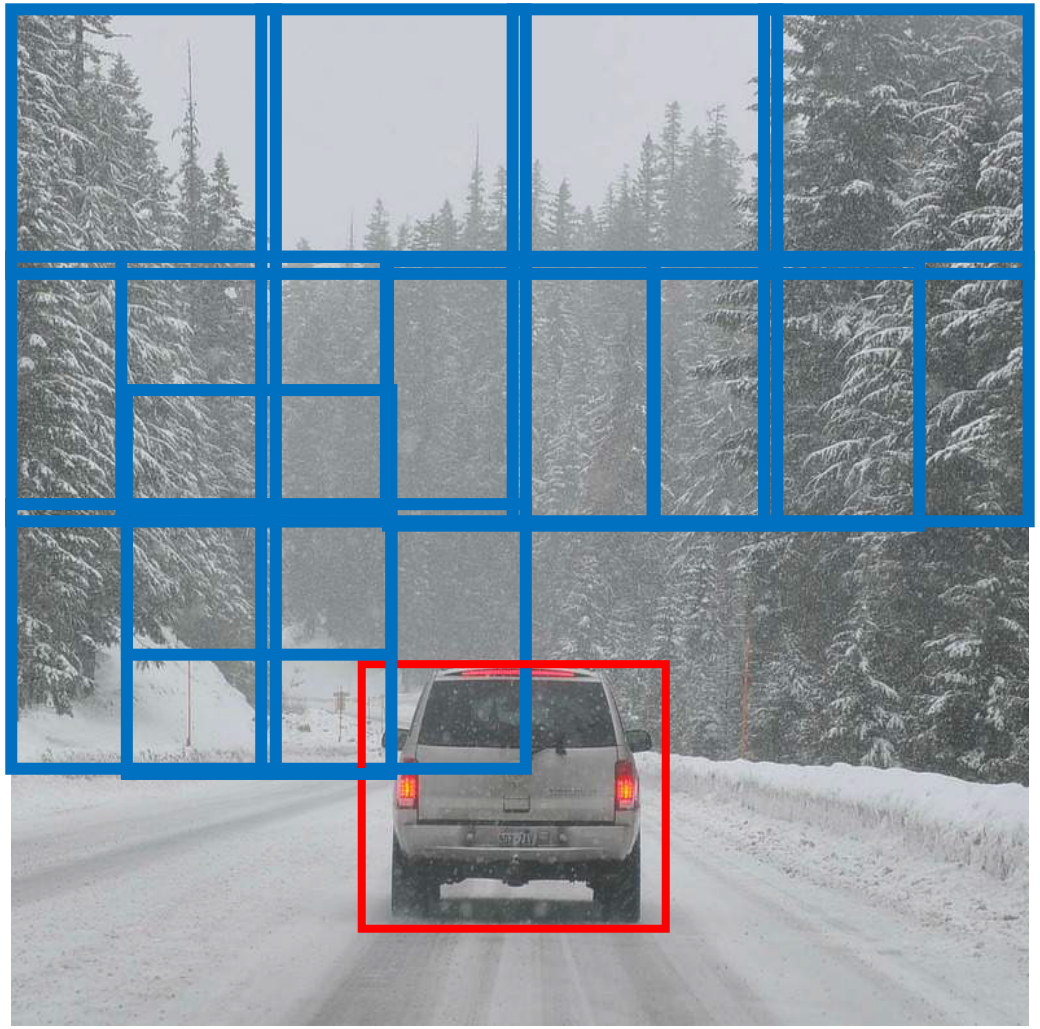
- YOLO splits the image into cells
- Cells are a way of organising labels (object positions)
- The CNN can "see" more than the contents of a cell for each cell prediction



19×19

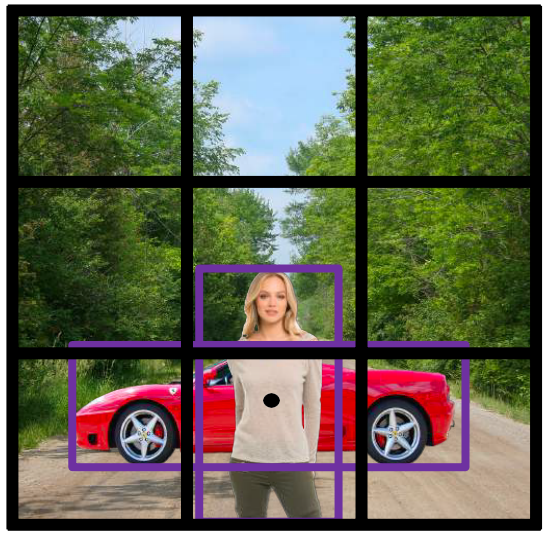
Cells in YOLO

- Patches can overlap



Anchor Boxes in YOLO

- They allow us to predict multiple objects per cell



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \end{bmatrix}$$

Which is correct?

$$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \end{bmatrix}$$

Anchor Boxes in YOLO

- Anchor boxes are set before training
  - Manually or with clustering over data
- They serve as "reference point" shapes
  - Not directly linked to classes
    e.g. bounding box of a person will be different if they are standing vs. they are lying down