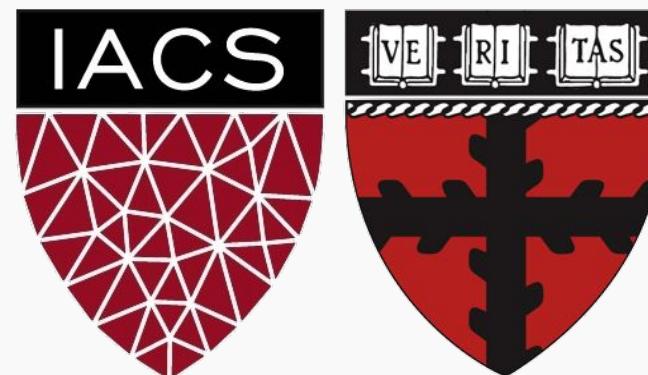


# Lecture 11: Convolutional Neural Networks 2

CS109B Data Science 2

Pavlos Protopapas, Mark Glickman and Chris Tanner



# Outline

---

1. Review from last lecture
2. Training CNNs
3. BackProp of MaxPooling layer
4. Layers Receptive Field
5. Saliency maps
6. Transfer Learning
7. A bit of history

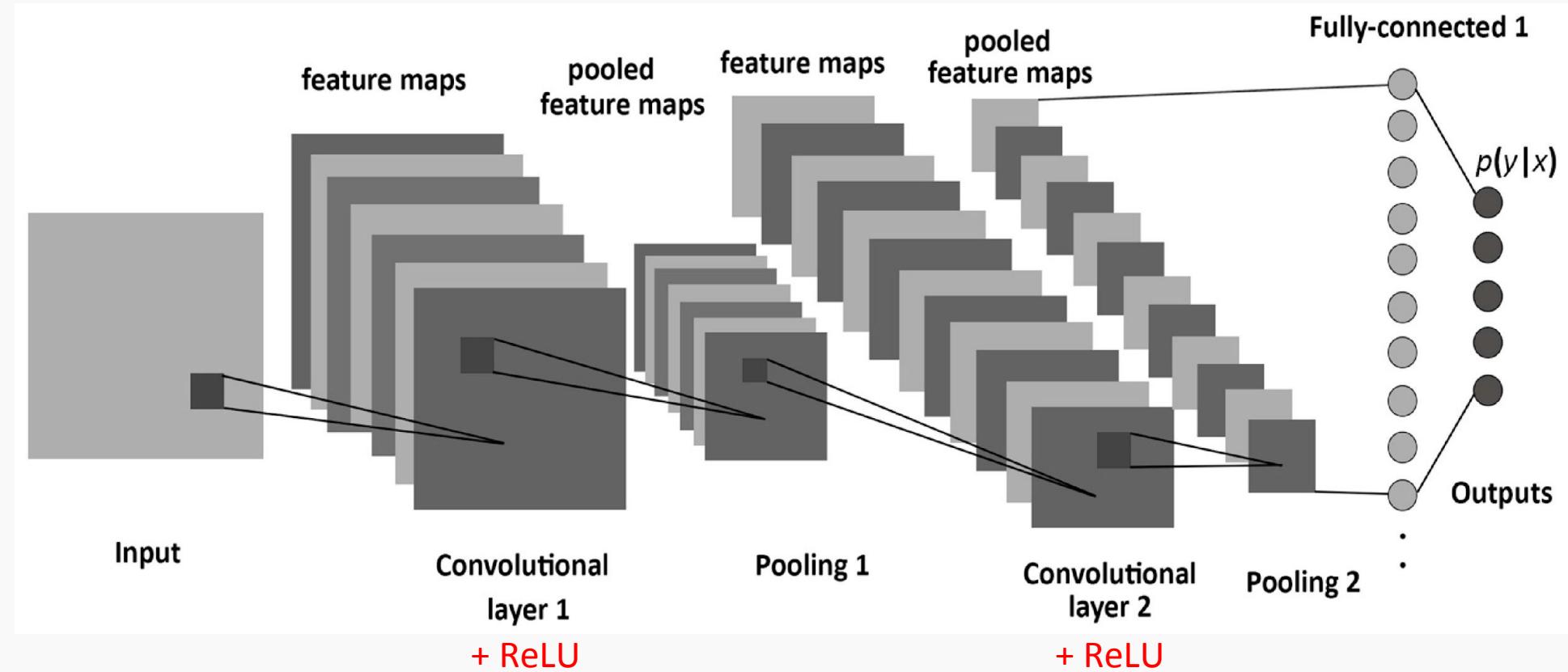


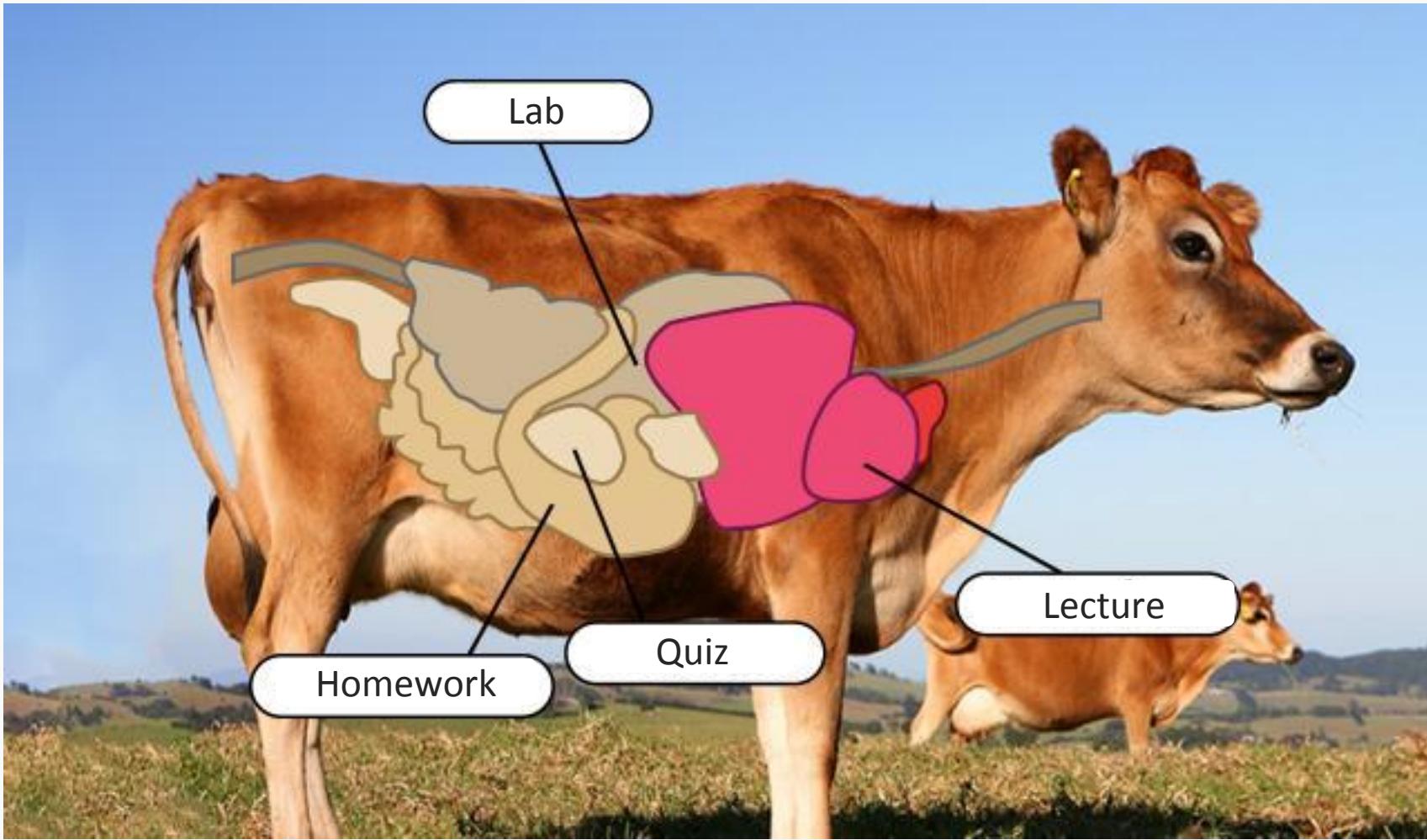
# Outline

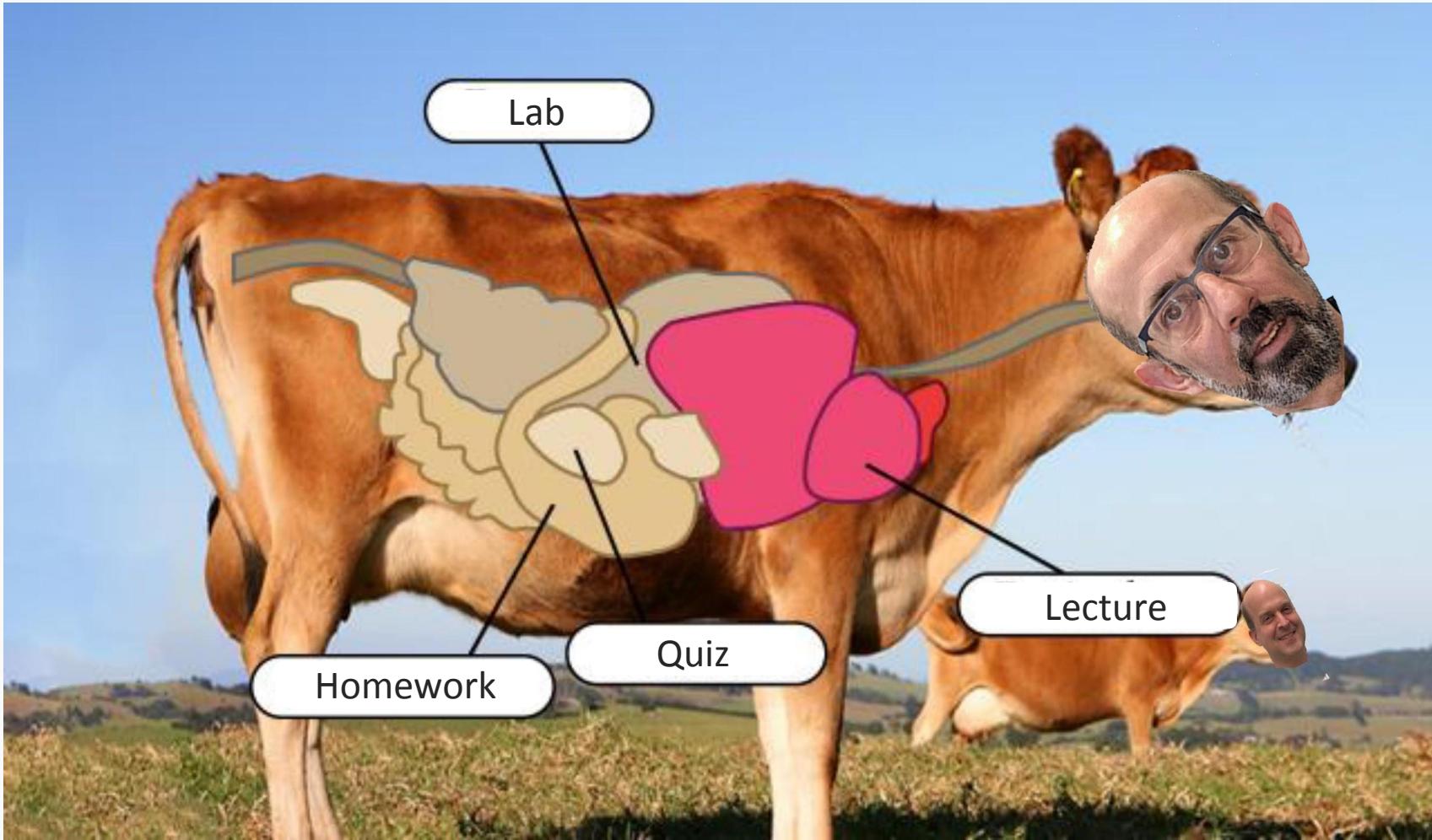
---

1. Review from last lecture
2. Training CNNs
3. BackProp of MaxPooling layer
4. Layers Receptive Field
5. Saliency maps
6. Transfer Learning
7. A bit of history

# From last lecture

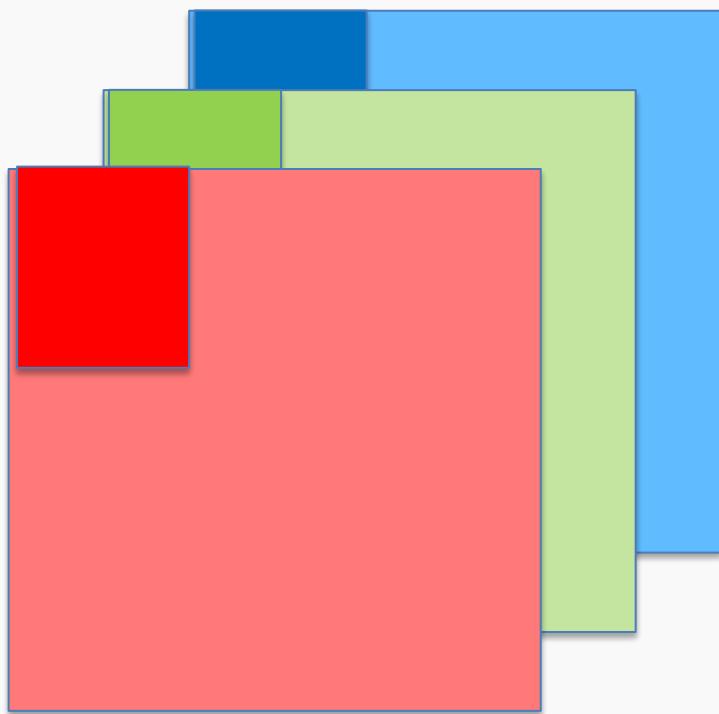






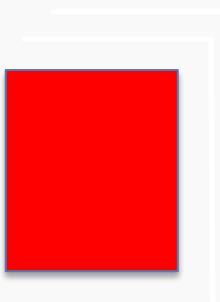
## Input

(size=32X32,  
channels=3)



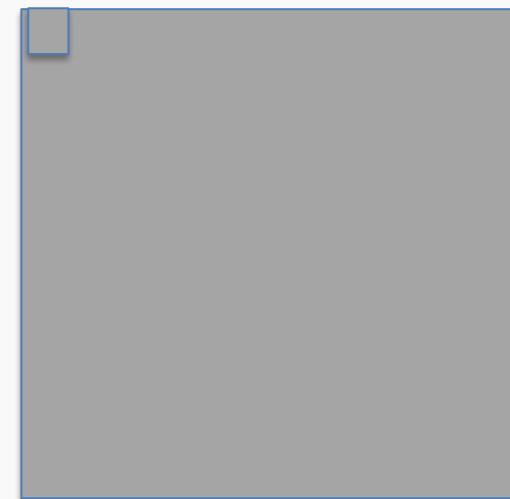
## 1 Filter

(size=3x3X3,  
stride = 1,  
padding = same)



## Output

(size=32X32)



How many parameters does the layer have?

n\_filter x filter\_volume + biases = total number of params

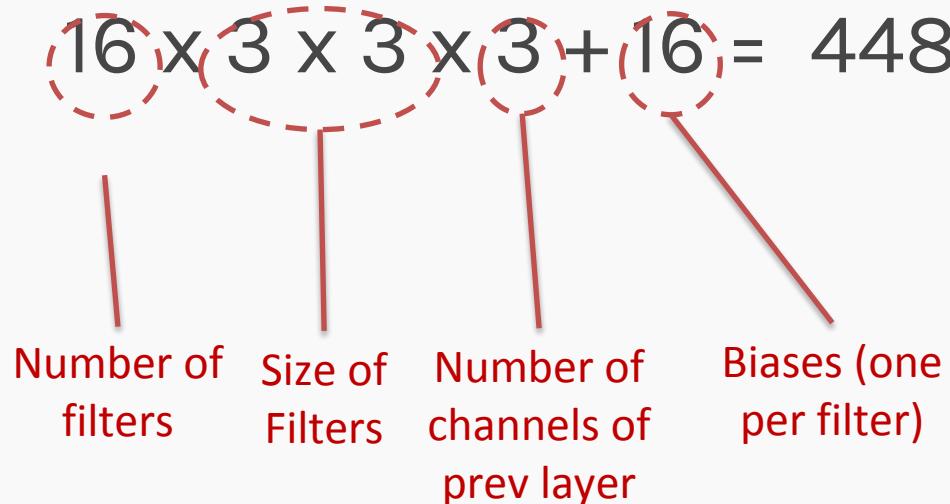
$$1 \times (3 \times 3 \times 3) + 1 = 28$$

# Examples

- I have a convolutional layer with 16 3x3 filters that takes an RGB image as input.
  - How many parameters does the layer have?

$$16 \times 3 \times 3 \times 3 + 16 = 448$$

Number of filters    Size of Filters    Number of channels of prev layer    Biases (one per filter)



# Examples

- Let C be a CNN with the following disposition:
  - Input: 32x32x3 images
  - Conv1: 8 3x3 filters, stride 1, padding=same
  - Conv2: 16 5x5 filters, stride 2, padding=same
  - Flatten layer
  - Dense1: 512 nodes
  - Dense2: 4 nodes
- How many parameters does this network have?

$$(8 \times 3 \times 3 \times 3 + 8) + (16 \times 5 \times 5 \times 8 + 16) + (16 \times 16 \times 16 \times 512 + 512) + (512 \times 4 + 4)$$

Conv1

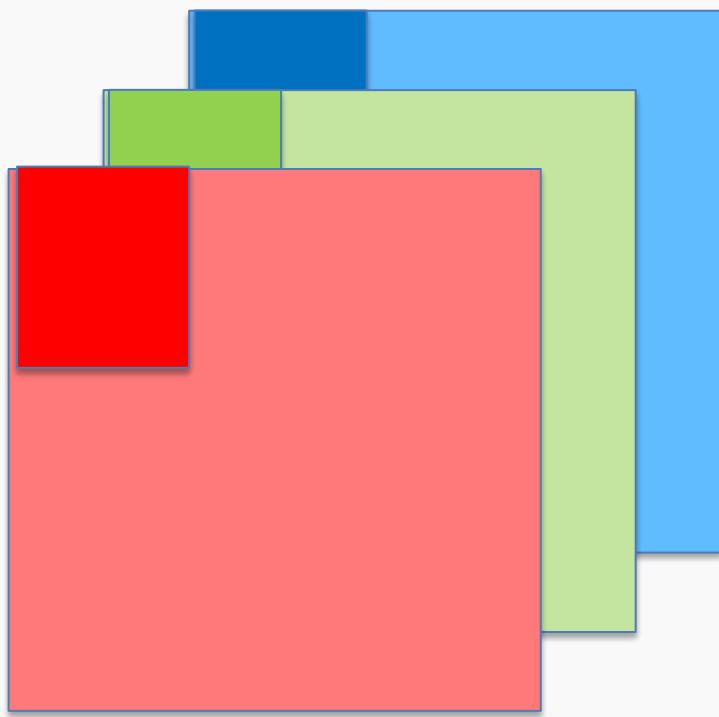
Conv2

Dense1

Dense2

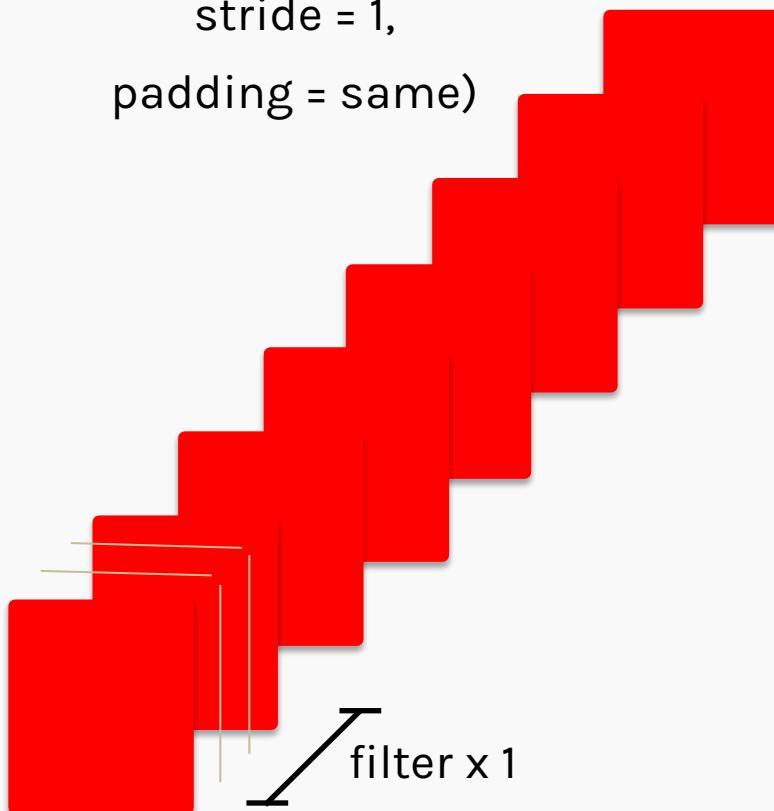
## Input

(size=32X32,  
channels=3)



## Filter

8 x (size=3X3x3,  
stride = 1,  
padding = same)



## Output

(size=32X32,  
channels = 8)



**How many parameters does the layer have if I want to use 8 filters?**

n\_filters x filter\_volume + biases = total number of params

$$8 \times (3 \times 3 \times 3) + 8 = 224$$

## Input

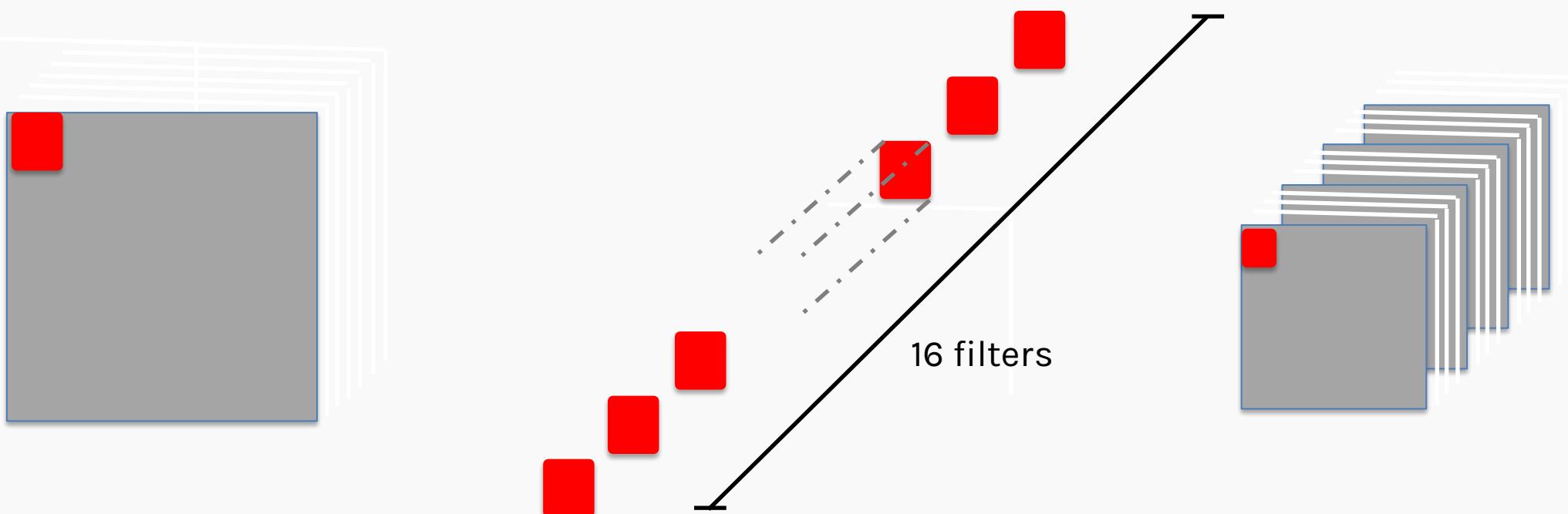
(size=32X32,  
channels=8)

## Filter

16 x (size=5X5X8,  
stride = 2,  
padding = same)

## Output

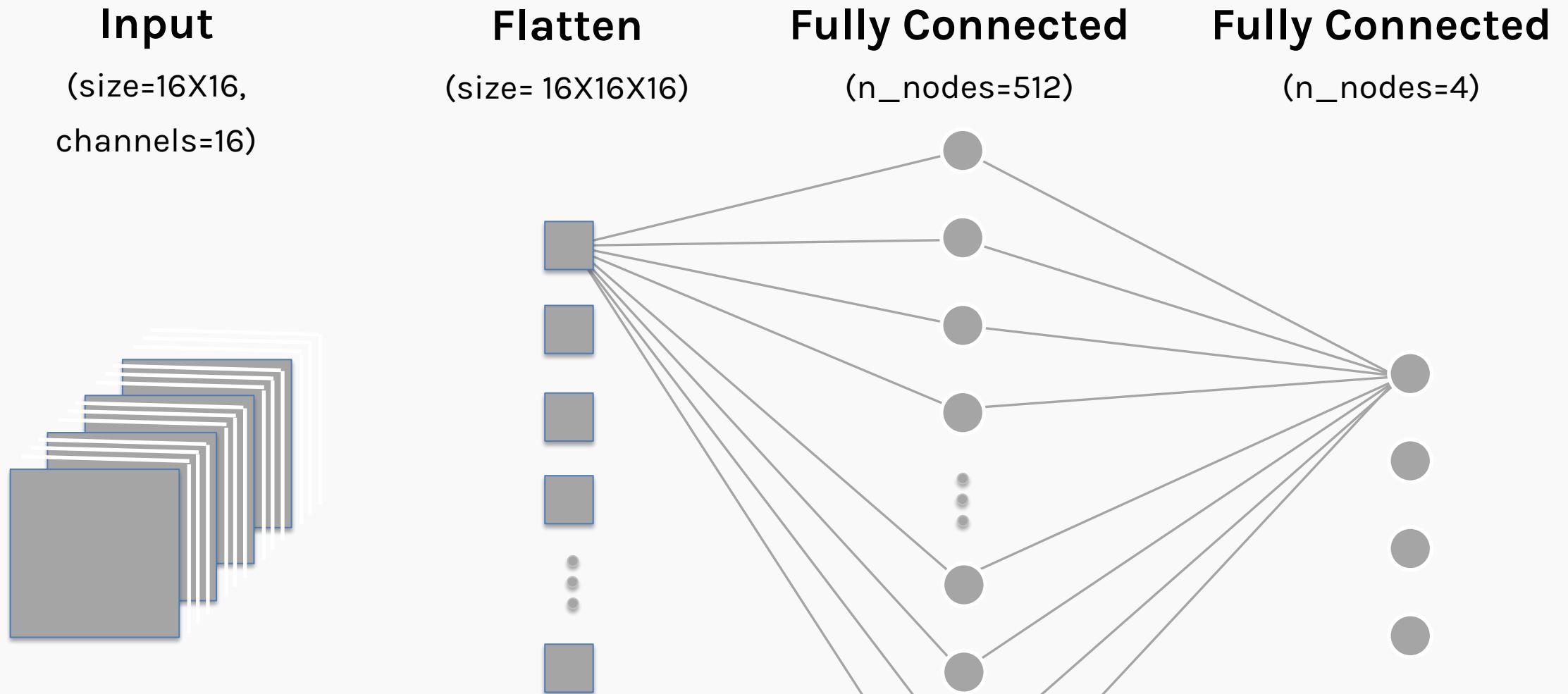
(size=16X16,  
channels=16)



**How many parameters does the layer have if I want to use 16 filters?**

$$\text{n\_filters} \times \text{filter\_volume} + \text{biases} = \text{total number of params}$$

$$16 \times (5 \times 5 \times 8) + 16 = 3216$$



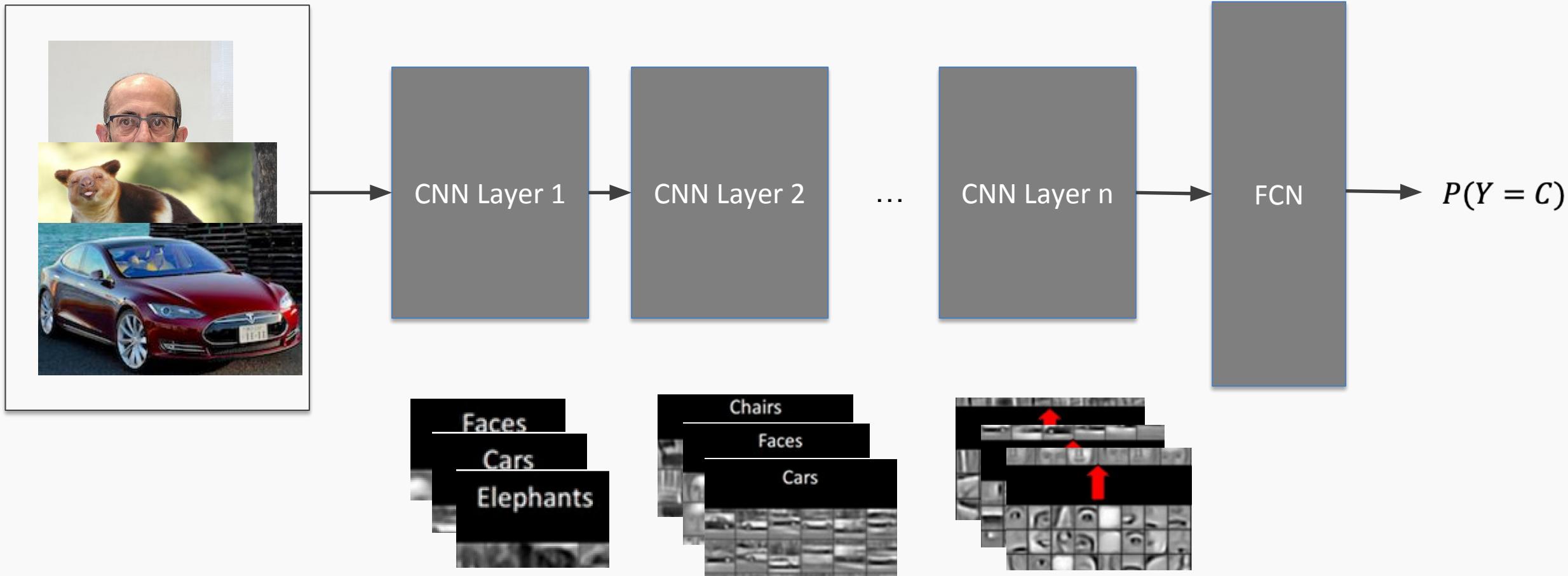
**How many parameters ... ?**

input x FC1\_nodes + FC2\_nodes = total number of params

$$(16 \times 16 \times 16) \times 512 + 512 + 512 \times 4 + 4 = 2,099,716$$

# Representation Learning

Task: classify cars, people, animals and objects



# What do CNN layers learn?

---

- Each CNN layer learns filters of increasing complexity.
- The first layers learn **basic feature detection filters**: edges, corners, etc.
- The middle layers learn filters that detect **parts of objects**.  
For faces, they might learn to respond to eyes, noses, etc.
- The last layers have higher representations: they learn to **recognize full objects**, in different shapes and positions.

---

## 3D visualization of networks in action

<http://scs.ryerson.ca/~aharley/vis/conv/>

<https://www.youtube.com/watch?v=3JQ3hYko51Y>



# Outline

---

1. Review from last lecture
2. Training CNNs
3. BackProp of MaxPooling layer
4. Layers Receptive Field
5. Saliency maps
6. Transfer Learning
7. A bit of history



**NOTHING**

**THAT'S ALL I HAVE TO SAY ABOUT  
THAT**

# Outline

---

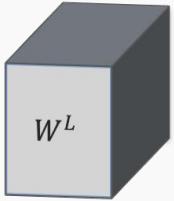
1. Review from last lecture
2. Training CNNs
- 3. BackProp of MaxPooling layer**
4. Layers Receptive Field
5. Saliency maps
6. Transfer Learning
7. A bit of history



# Backward propagation of Maximum Pooling Layer

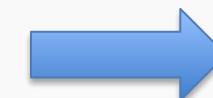
Forward mode, 3x3 stride 1

rest of the network



Activation of layer L

2	4	8	3	6
9	3	4	2	5
5	4	6	3	1
2	3	1	3	4
2	7	4	5	7



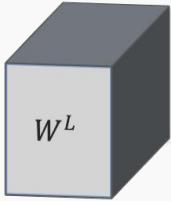



rest of the network

# Backward propagation of Maximum Pooling Layer

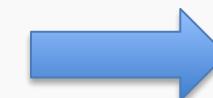
Forward mode, 3x3 stride 1

rest of the network



Activation of layer L

2	4	8	3	6
9	3	4	2	5
5	4	6	3	1
2	3	1	3	4
2	7	4	5	7



9		

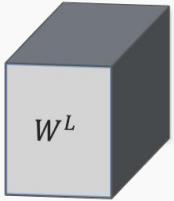


rest of the network

# Backward propagation of Maximum Pooling Layer

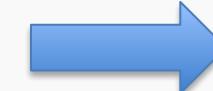
Forward mode, 3x3 stride 1

rest of the network



Activation of layer L

2	4	8	3	6
9	3	4	2	5
5	4	6	3	1
2	3	1	3	4
2	7	4	5	7



9	8	

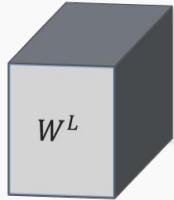


rest of the network

# Backward propagation of Maximum Pooling Layer

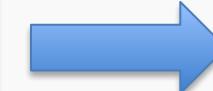
Forward mode, 3x3 stride 1

rest of the network



Activation of layer L

2	4	8	3	6
9	3	4	2	5
5	4	6	3	1
2	3	1	3	4
2	7	4	5	7



9	8	8

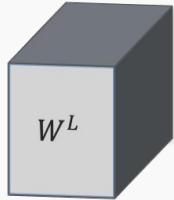


rest of the network

# Backward propagation of Maximum Pooling Layer

Forward mode, 3x3 stride 1

rest of the network



Activation of layer L

2	4	8	3	6
9	3	4	2	5
5	4	6	3	1
2	3	1	3	4
2	7	4	5	7



9	8	8
9	6	6
7	7	7



rest of the network

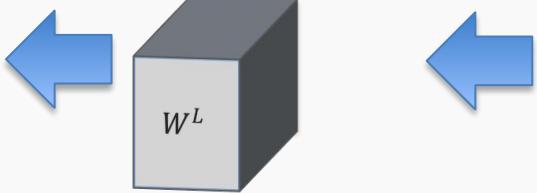
# Backward propagation of Maximum Pooling Layer

Backward mode. Large fonts represents the values of the derivatives of the current layer (max-pool) and small font the corresponding value of the previous layer.

rest of the network

Activation of layer L

2	4	8	3	6
9	3	4	2	5
5	4	6	3	1
2	3	1	3	4
2	7	4	5	7



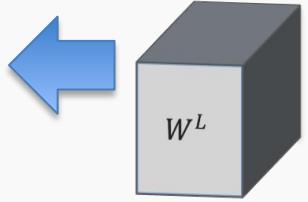
rest of the network

# Backward propagation of Maximum Pooling Layer

Backward mode. Large fonts represents the values of the derivatives of the current layer (max-pool) and small font the corresponding value of the previous layer.

rest of the network

Activation of layer L



2	4	8	3	6
9	3	4	2	5
5	4	6	3	1
2	3	1	3	4
2	7	4	5	7



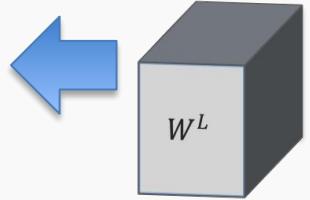
rest of the network

# Backward propagation of Maximum Pooling Layer

Backward mode. Large fonts represents the values of the derivatives of the current layer (max-pool) and small font the corresponding value of the previous layer.

rest of the network

Activation of layer L



2	4	8	3	6
9	3	4	2	5
5	4	6	3	1
2	3	1	3	4
2	7	4	5	7



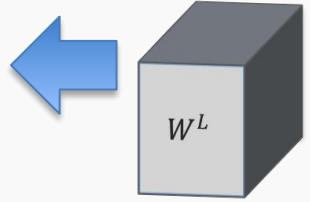
rest of the network

# Backward propagation of Maximum Pooling Layer

Backward mode. Large fonts represents the values of the derivatives of the current layer (max-pool) and small font the corresponding value of the previous layer.

rest of the network

Activation of layer L



2	4	8	3	6
+1 9	3	4	2	5
5	4	6	3	1
2	3	1	3	4
2	7	4	5	7



1 9	3 8	1 8
1 9	4 6	2 6
6 7	2 7	1 7



rest of the network

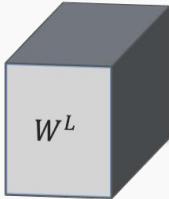
# Backward propagation of Maximum Pooling Layer

Backward mode. Large fonts represents the values of the derivatives of the current layer (max-pool) and small font the corresponding value of the previous layer.

rest of the network

Activation of layer L

2	4	8	3	6
+1 9	3	4	2	5
5	4	6	3	1
2	3	1	3	4
2	7	4	5	7



1 <sub>9</sub>	3 <sub>8</sub>	1 <sub>8</sub>
1 <sub>9</sub>	4 <sub>6</sub>	2 <sub>6</sub>
6 <sub>7</sub>	2 <sub>7</sub>	1 <sub>7</sub>



rest of the network

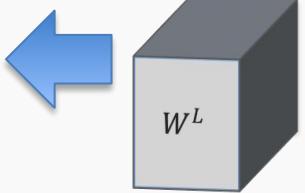
# Backward propagation of Maximum Pooling Layer

Backward mode. Large fonts represents the values of the derivatives of the current layer (max-pool) and small font the corresponding value of the previous layer.

rest of the network

Activation of layer L

2	4	+3 8	3	6
+1 9	3	4	2	5
5	4	6	3	1
2	3	1	3	4
2	7	4	5	7



1 9	3 8	1 8
1 9	4 6	2 6
6 7	2 7	1 7



rest of the network

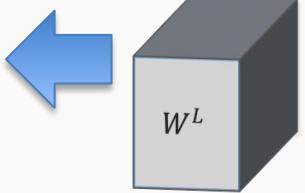
# Backward propagation of Maximum Pooling Layer

Backward mode. Large fonts represents the values of the derivatives of the current layer (max-pool) and small font the corresponding value of the previous layer.

rest of the network

Activation of layer L

2	4	+3 8	3	6
+1 9	3	4	2	5
5	4	6	3	1
2	3	1	3	4
2	7	4	5	7



1 <sub>9</sub>	3 <sub>8</sub>	1 <sub>8</sub>
1 <sub>9</sub>	4 <sub>6</sub>	2 <sub>6</sub>
6 <sub>7</sub>	2 <sub>7</sub>	1 <sub>7</sub>



rest of the network

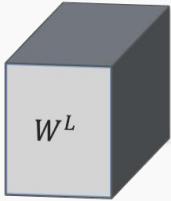
# Backward propagation of Maximum Pooling Layer

Backward mode. Large fonts represents the values of the derivatives of the current layer (max-pool) and small font the corresponding value of the previous layer.

rest of the network

Activation of layer L

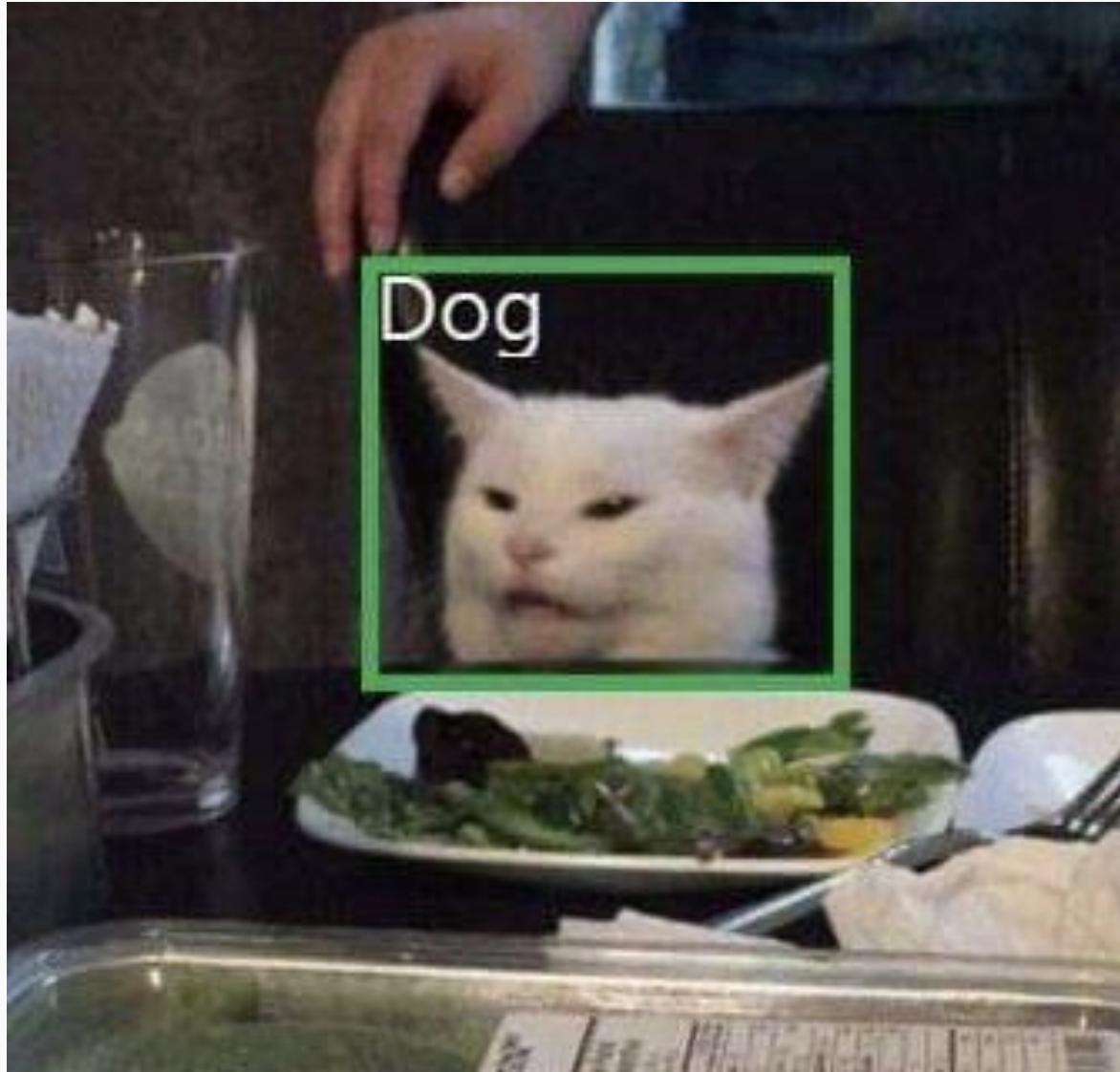
2	4	+4 8	3	6
+1 9	3	4	2	5
5	4	6	3	1
2	3	1	3	4
2	7	4	5	7



1 9	3 8	1 8
1 9	4 6	2 6
6 7	2 7	1 7



rest of the network



# Outline

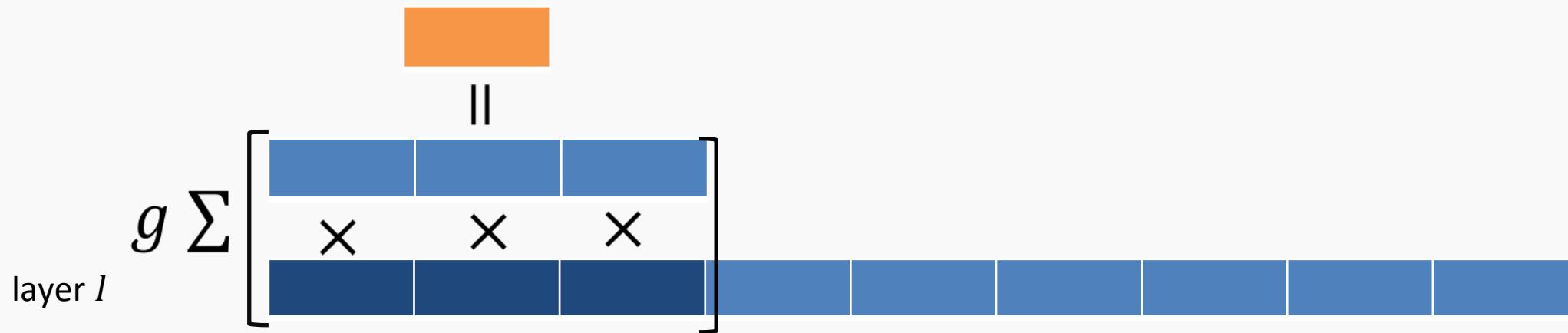
---

1. Review from last lecture
2. Training CNNs
3. BackProp of MaxPooling layer
4. **Layers Receptive Field, dilated CNNs**
5. Saliency maps
6. Transfer Learning
7. A bit of history



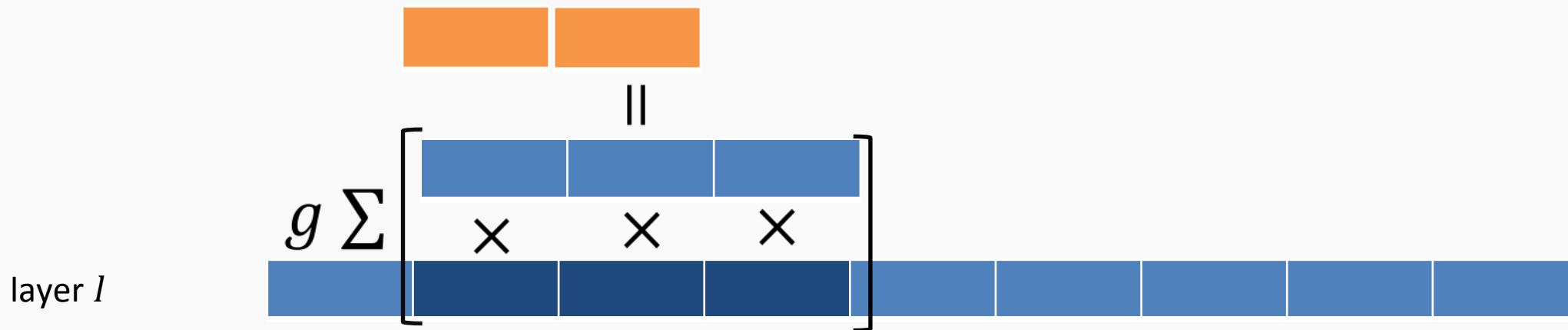
# Layers Receptive Field

Let's look at the receptive field again in 1D, no padding, stride 1 and kernel 3x1



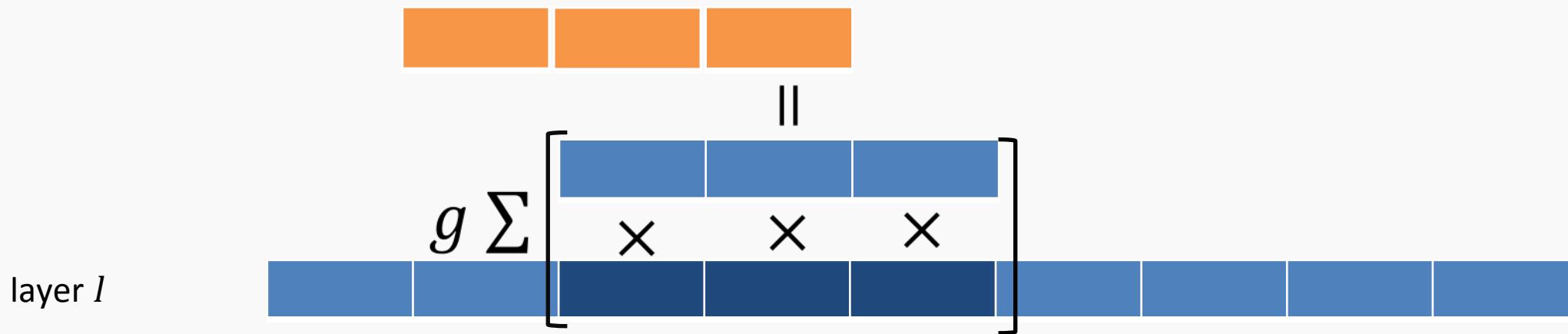
# Layers Receptive Field

Let's look at the receptive field again in 1D, no padding, stride 1 and kernel 3x1



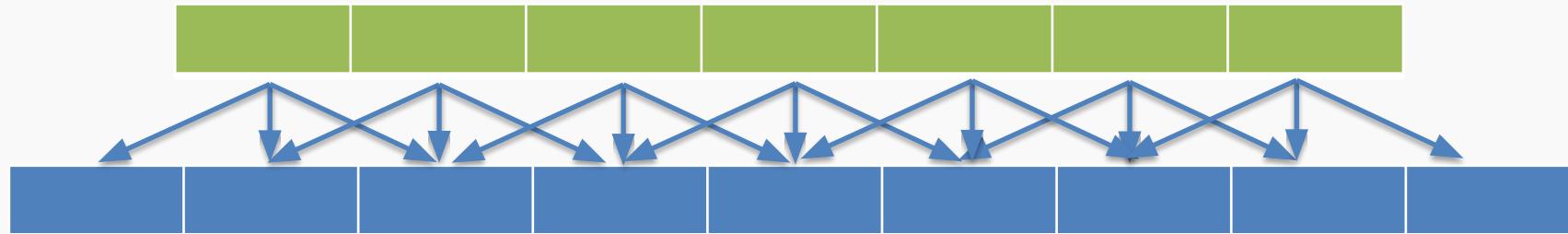
# Layers Receptive Field

Let's look at the receptive field again in 1D, no padding, stride 1 and kernel 3x1



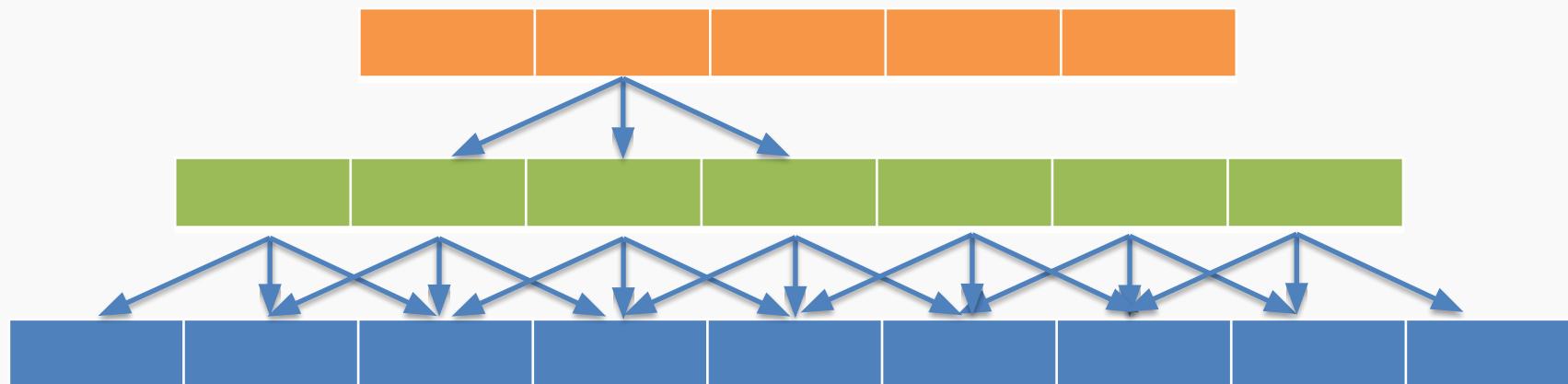
# Layers Receptive Field

Let's look at the receptive field again in 1D, no padding, stride 1 and kernel 3x1



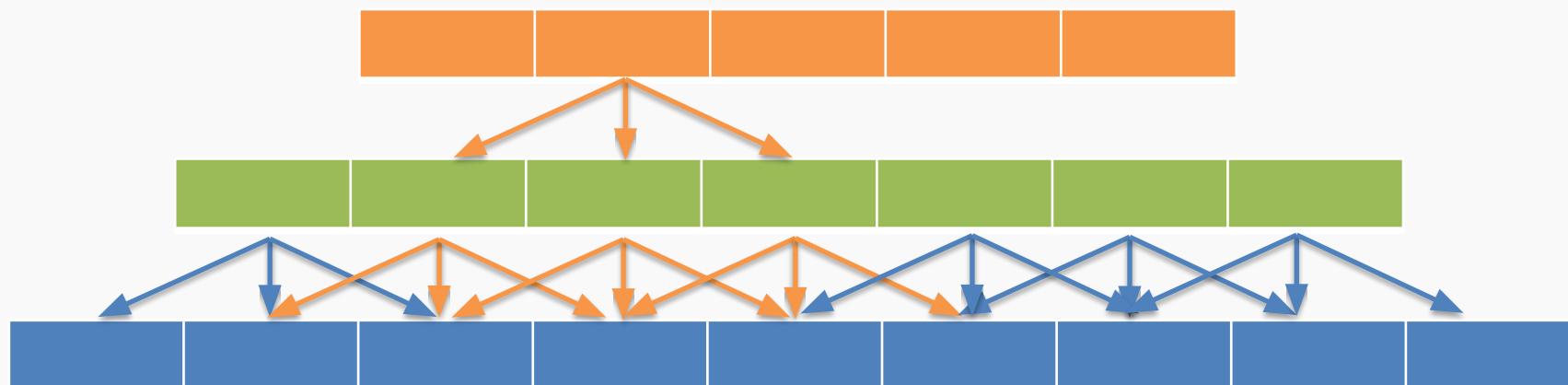
# Layers Receptive Field

Let's look at the receptive field again in 1D, no padding, stride 1 and kernel 3x1



# Layers Receptive Field

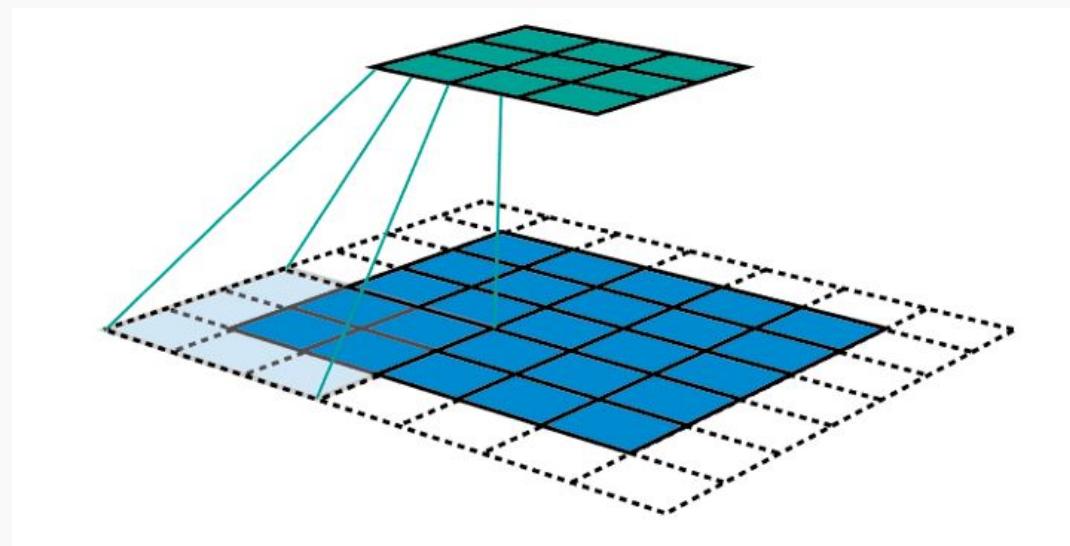
Let's look at the receptive field again in 1D, no padding, stride 1 and kernel 3x1



# Layer's Receptive Field

The **receptive field** is defined as the region in the input space that a particular CNN's feature is looking at (i.e. be affected by).

Apply a convolution C with kernel size  **$k = 3 \times 3$** , padding size  **$p = 1 \times 1$** , stride  **$s = 2 \times 2$**  on an input map  **$5 \times 5$** , we will get an output feature map  **$3 \times 3$**  (green map).

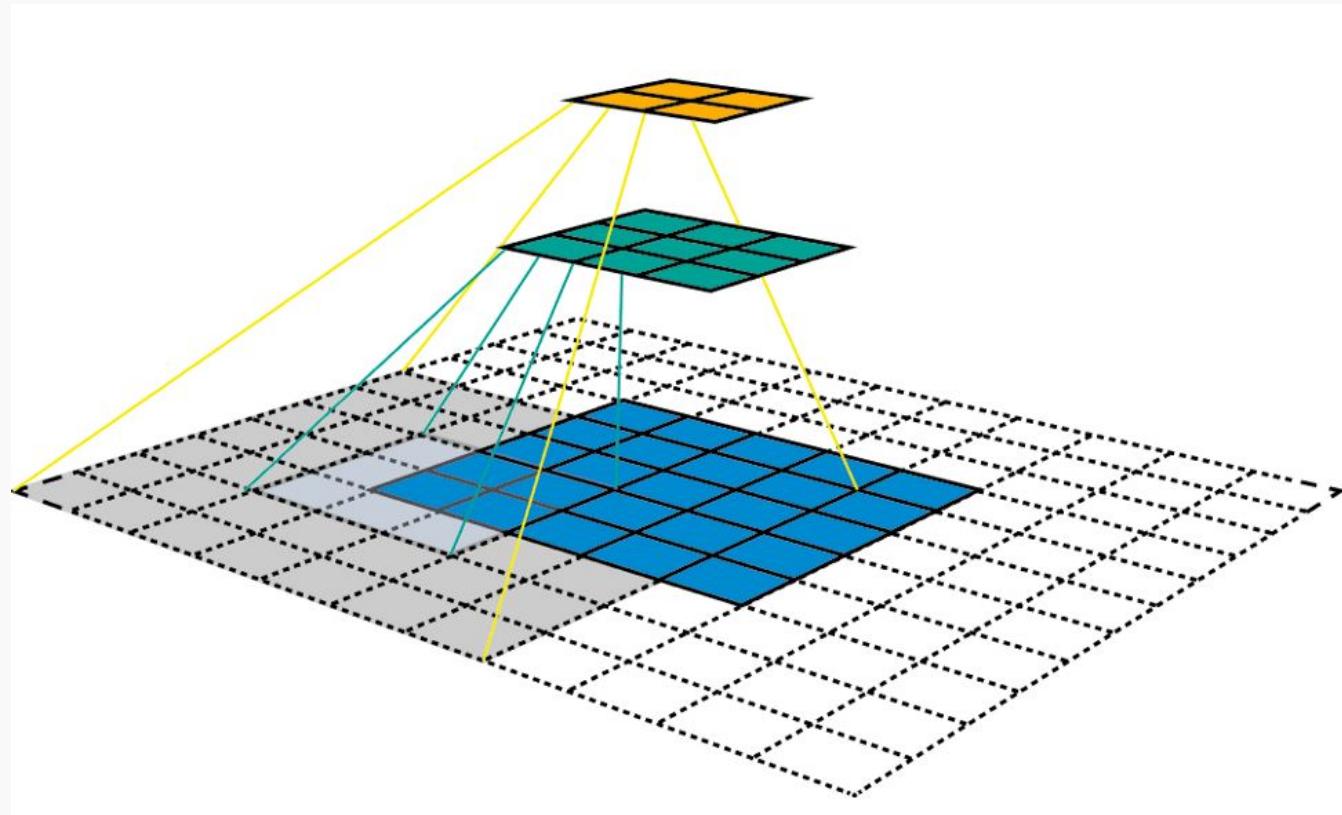


$$n_{out} = \left\lceil \frac{n_{in} + 2p - k}{s} \right\rceil + 1$$

- $n_{in}$ : number of input features  
 $n_{out}$ : number of output features  
 $k$ : convolution kernel size  
 $p$ : convolution padding size  
 $s$ : convolution stride size

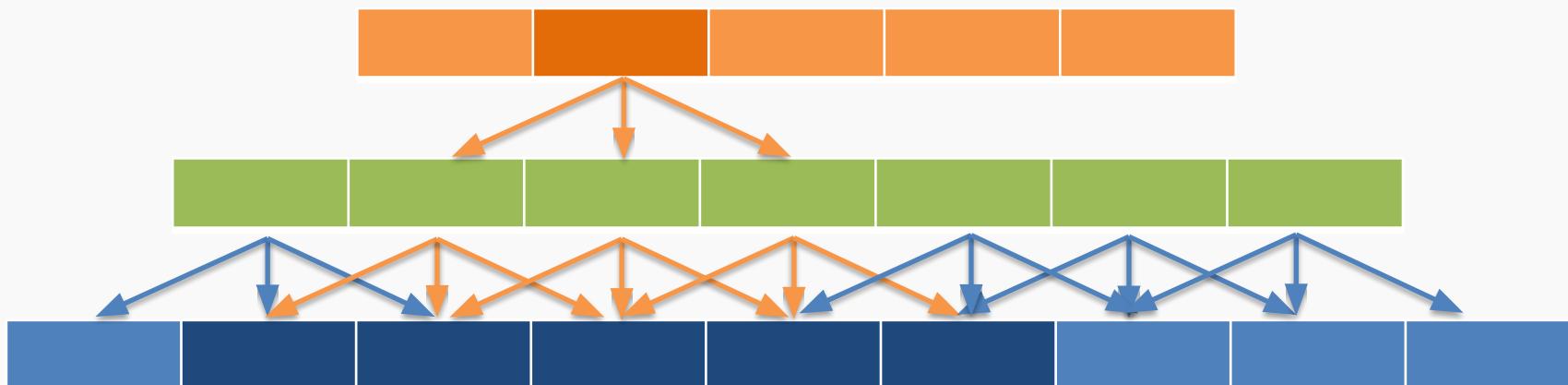
# Layer's Receptive Field

Applying the same convolution on top of the  $3 \times 3$  feature map, we will get a  **$2 \times 2$**  feature map (orange map)



# Layers Receptive Field

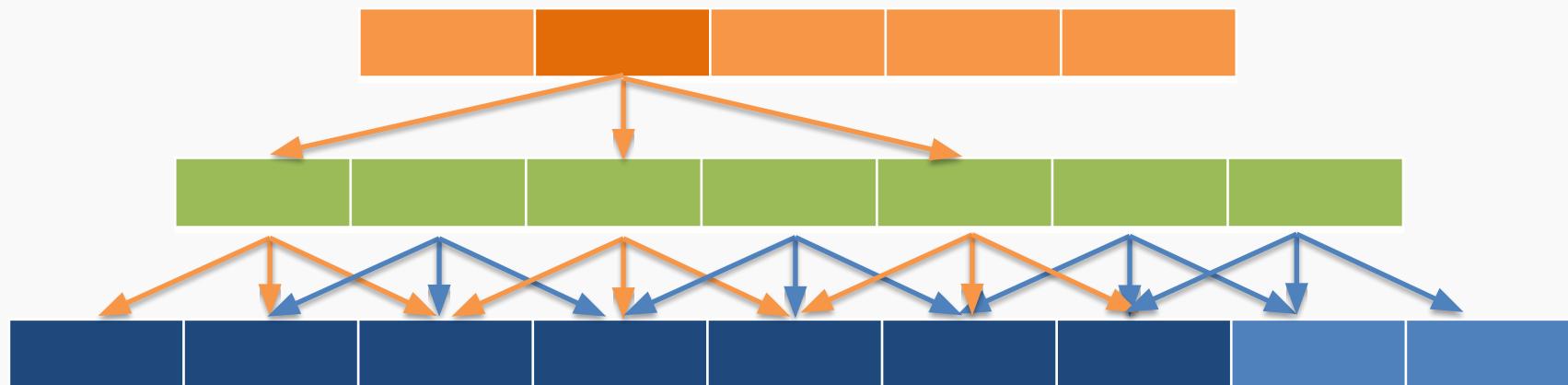
Let's look at the receptive field again in 1D, no padding, stride 1 and kernel 3x1



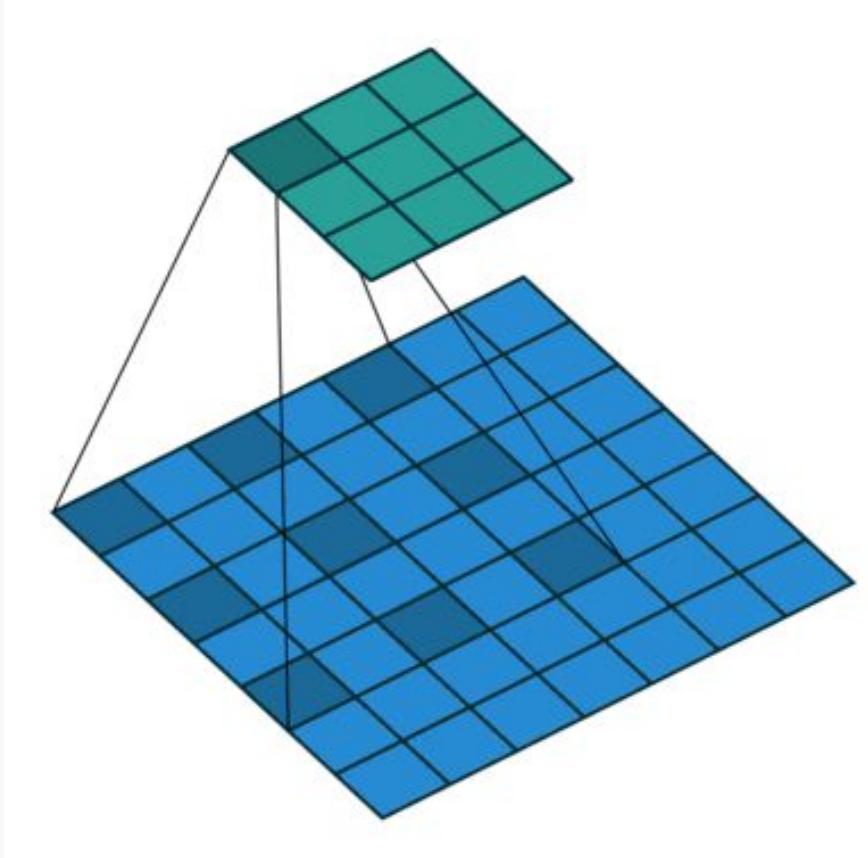
# Dilated CNNs

Let's look at the receptive field again in 1D, no padding, stride 1 and kernel 3x1.

**Skip some of the connections**



# Dilated CNNs



# Outline

---

1. Review from last lecture
2. Training CNNs
3. BackProp of MaxPooling layer
4. Layers Receptive Field
5. **Saliency maps**
6. Transfer Learning
7. A bit of history



# Saliency maps (cont)

If you are given an image of a **dog** and asked to classify it. Most probably you will answer immediately - **Dog!** But your Deep Learning Network might not be as smart as you. It might classify it as a cat, a lion or Pavlos!

What are the reasons for that?

- bias in training data
- no regularization
- or your network has seen too many celebrities



# Saliency maps (cont)

---

We want to understand what made my network give a certain class as output?

**Saliency Maps**, they are a way to measure the spatial support of a particular class in a given image.

*“Find me pixels responsible for the class C having score  $S(C)$  when the image  $I$  is passed through my network”.*

# Salience maps (cont)

---

**Question:** How do we do that?

**We differentiate!**



# Salience maps (cont)

---

**Question:** Easy Peasy?

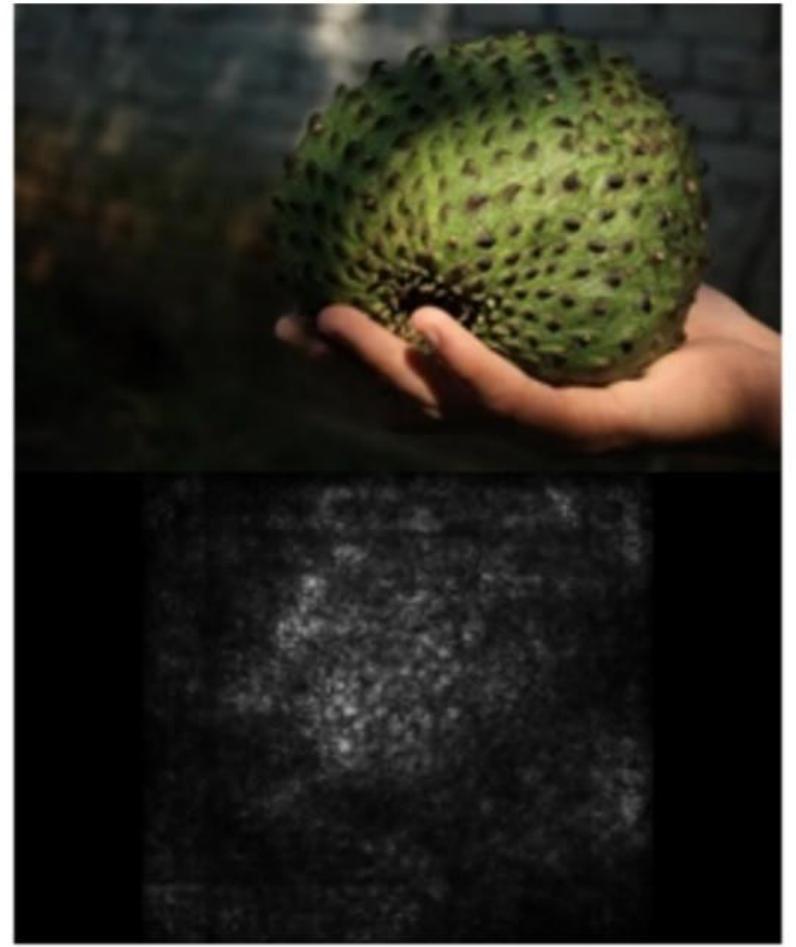
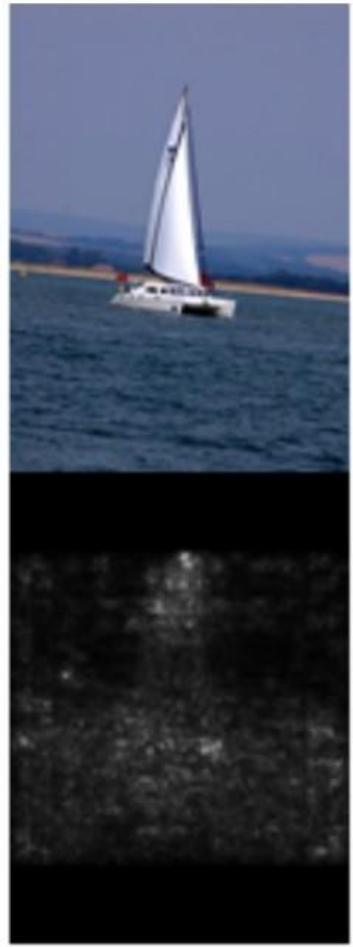
Sort of! Auto-grad can do this!

1. Forward pass of the image through the network.
2. Calculate the scores for every class.
3. Enforce derivative of score  $S$  at last layer for all classes except class C to be 0. For C, set it to 1
4. Backpropagate this derivative till the start
5. Render them and you have your Saliency Map!

Note: On step #2. Instead of doing softmax, we turn it to linear and use the logits.



# Salience maps (cont)



# Salience maps (cont)

**Question:** What do we do with color images?

Take the saliency map for each channel and either take the max or average or use all 3 channels.

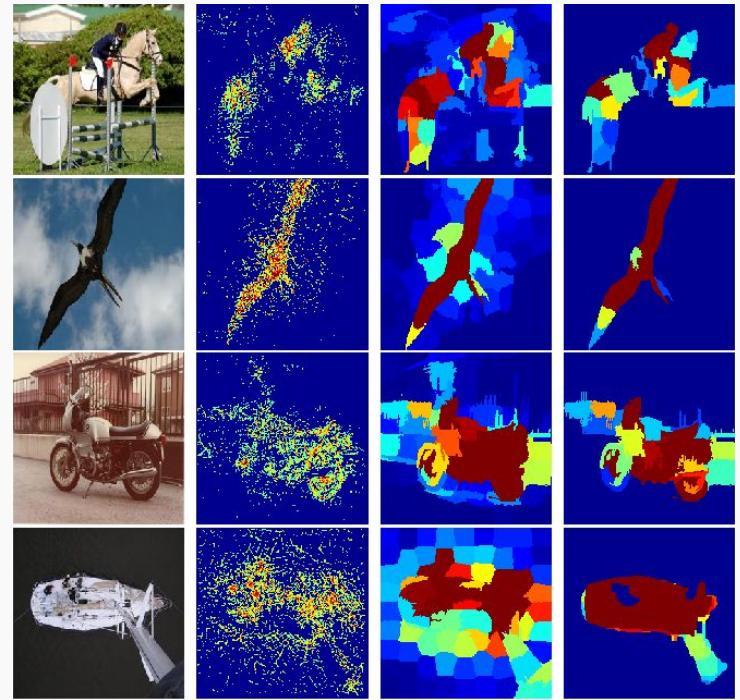
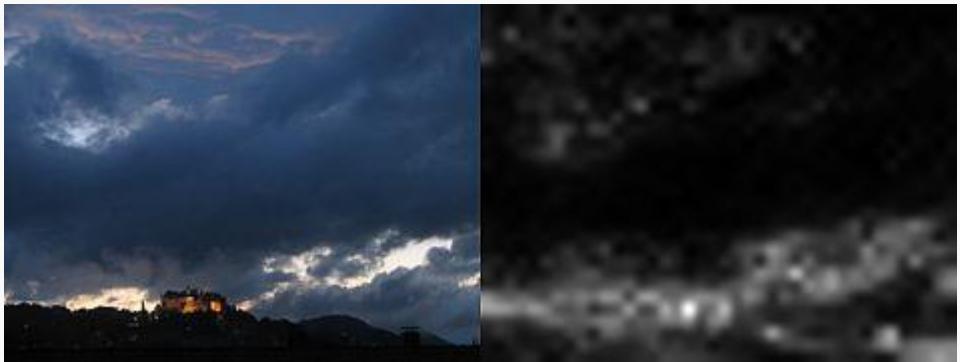


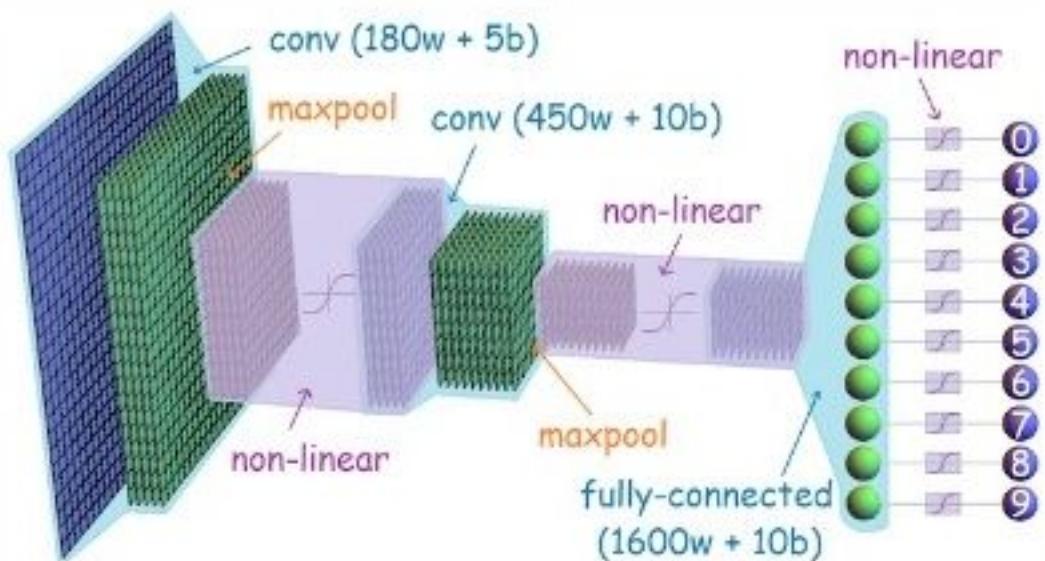
Figure 2. From left to right: original images, raw saliency maps,

- [1]: [Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps](#)
- [2]: [Attention-based Extraction of Structured Information from Street View Imagery](#)

# WHO WOULD WIN?

---

**AN INCREDIBLY COMPLEX  
MULTI-LAYER CONVOLUTIONAL  
NEURAL NETWORK**



**ONE NAIVE BOI**



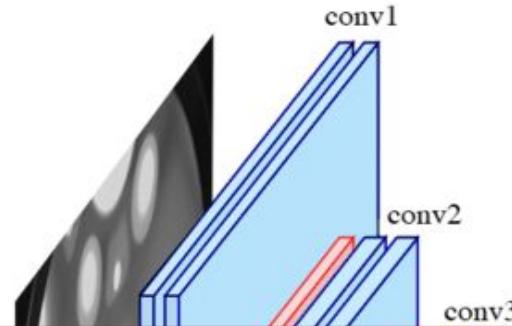
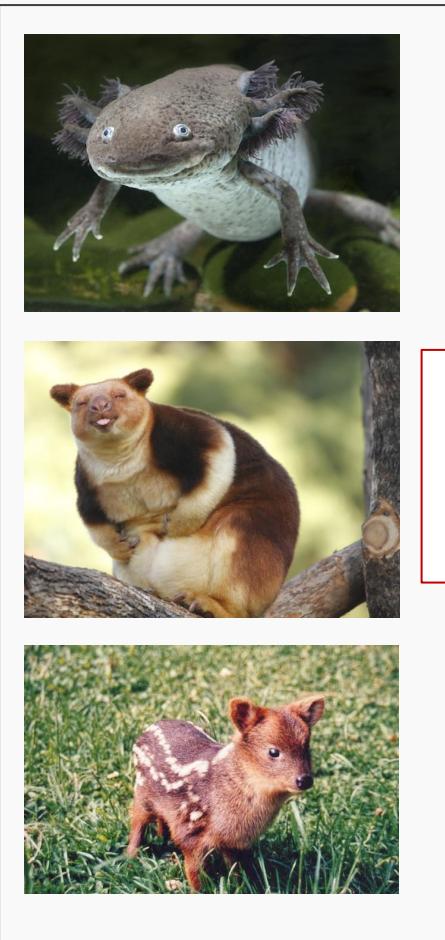
# Outline

---

1. Review from last lecture
2. Training CNNs
3. BackProp of MaxPooling layer
4. Layers Receptive Field, dilated CNNs
5. Saliency maps
6. Transfer Learning
7. A bit of history

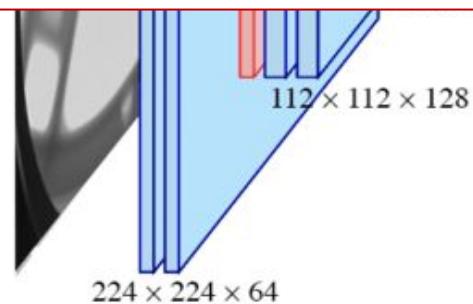


# Classify Rarest Animals



VGG16

NOT ENOUGH DATA

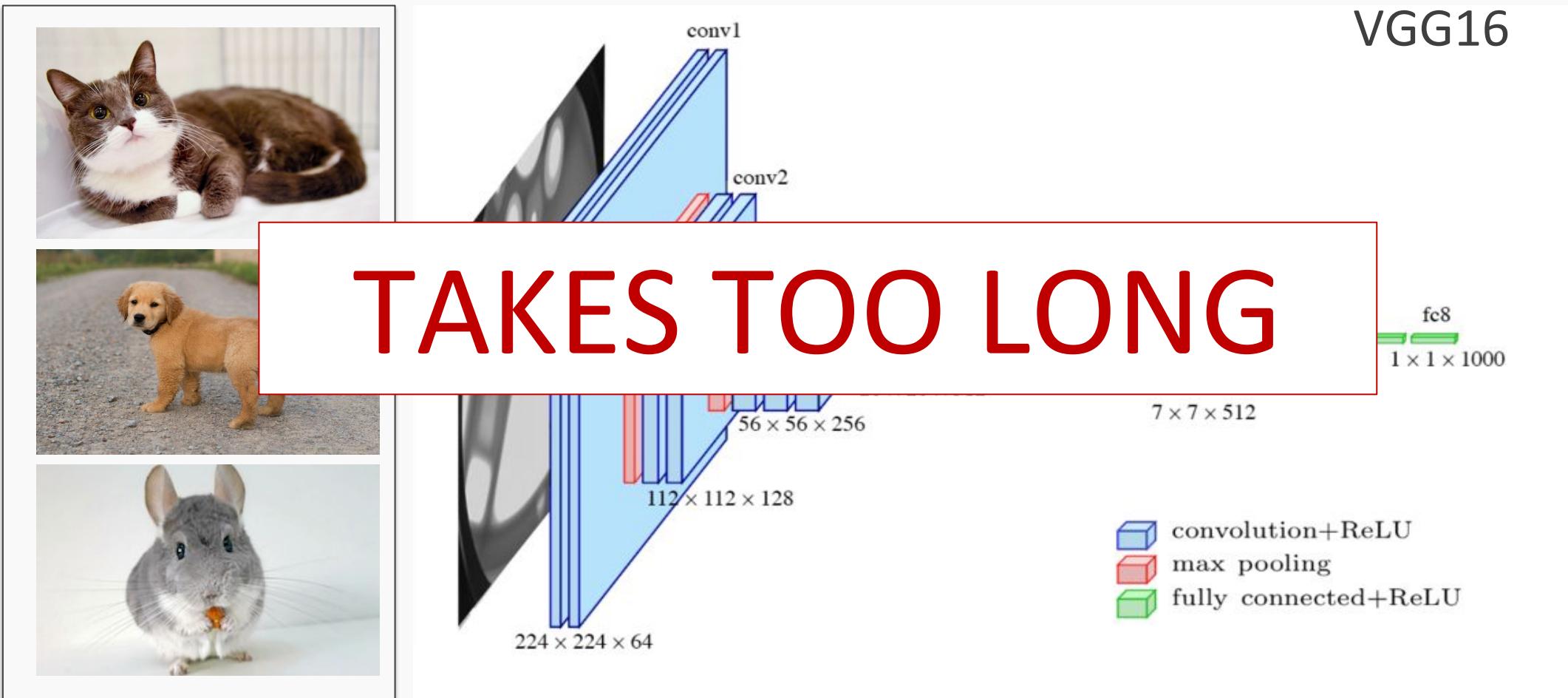


- convolution+ReLU
- max pooling
- fully connected+ReLU

Number of parameters: 134,268,737  
Data Set: Few hundred images

CS109B, PROTOPAPAS, GLICKMAN AND TANNER

# Classify Cats, Dogs, Chinchillas etc



Number of parameters: 134,268,737

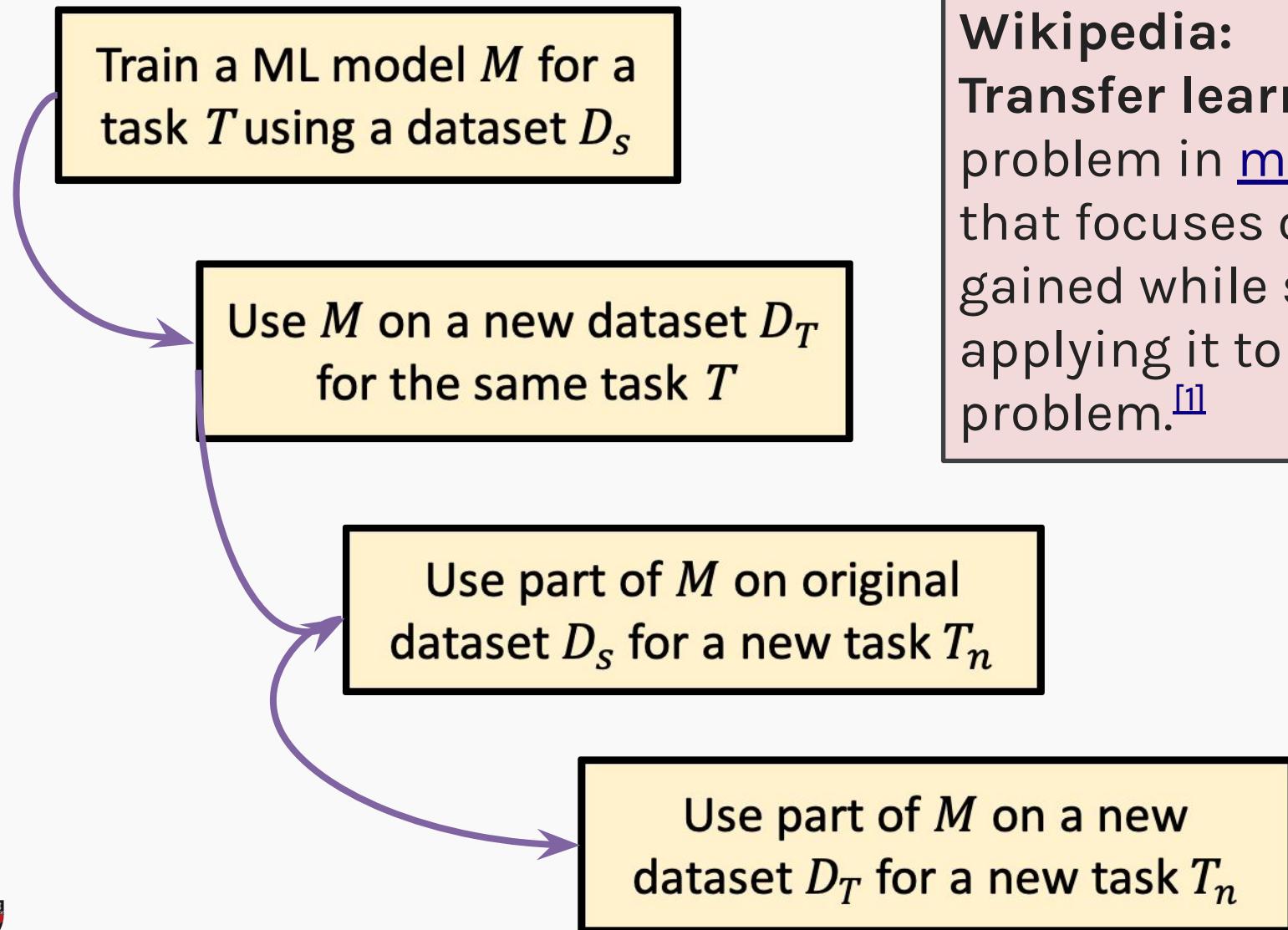
Enough training data. ImageNet approximate 1.2M

# Transfer Learning To The Rescue

How do you build an image classifier that can be trained in a few minutes on a CPU with very little data?



# Basic idea of Transfer Learning



## Wikipedia:

**Transfer learning (TL)** is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.<sup>[1]</sup>

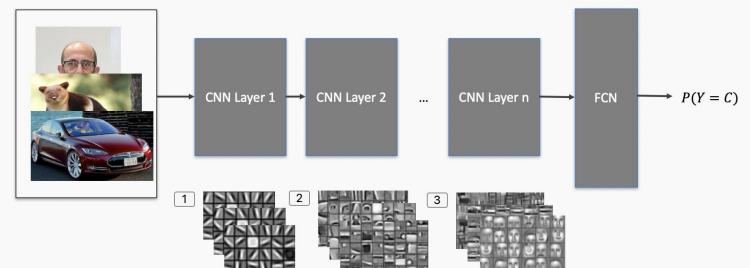
# Transfer Learning To The Rescue

How do you make **an image classifier** that can be **trained in a few minutes** on a CPU with very little data?

**Use pre-trained models**, i.e., models with known weights.

**Main Idea:** earlier layers of a network learn low level features, which can be adapted to new domains by changing weights at later and fully-connected layers.

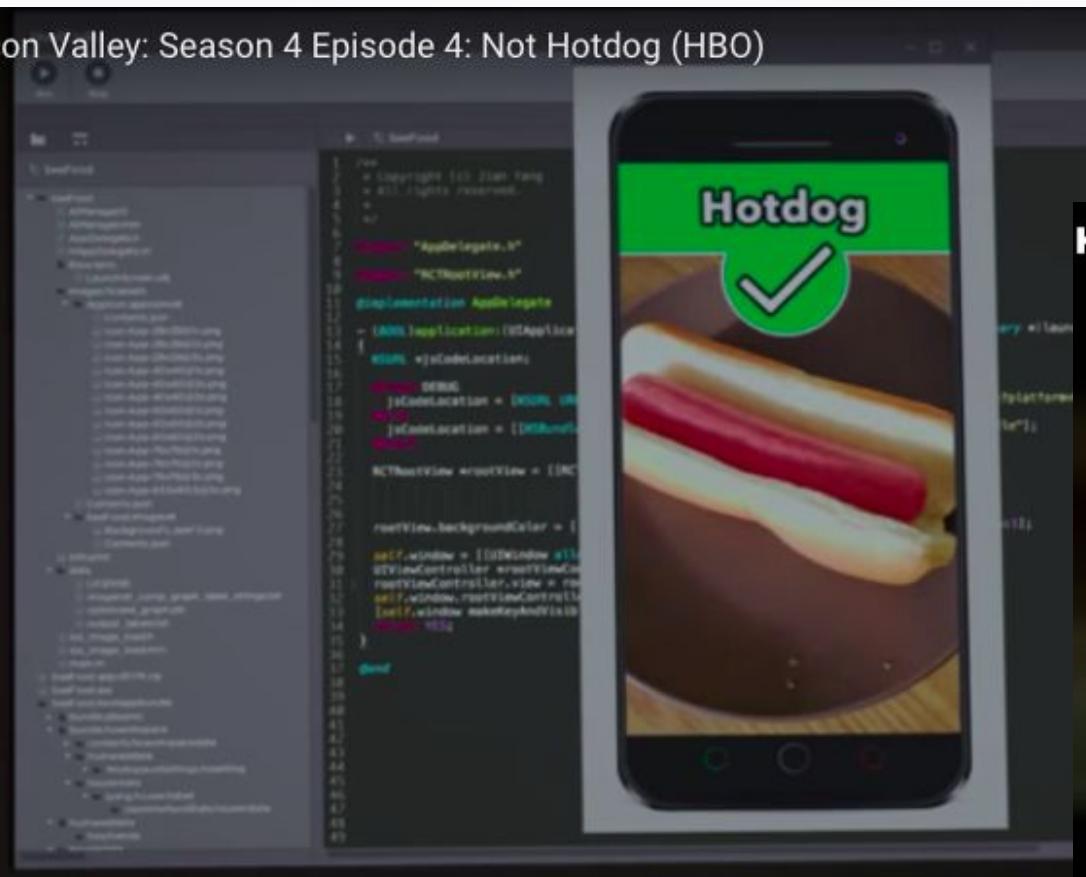
**Example:** use ImageNet trained with any sophisticated huge network. Then retrain it on a few images



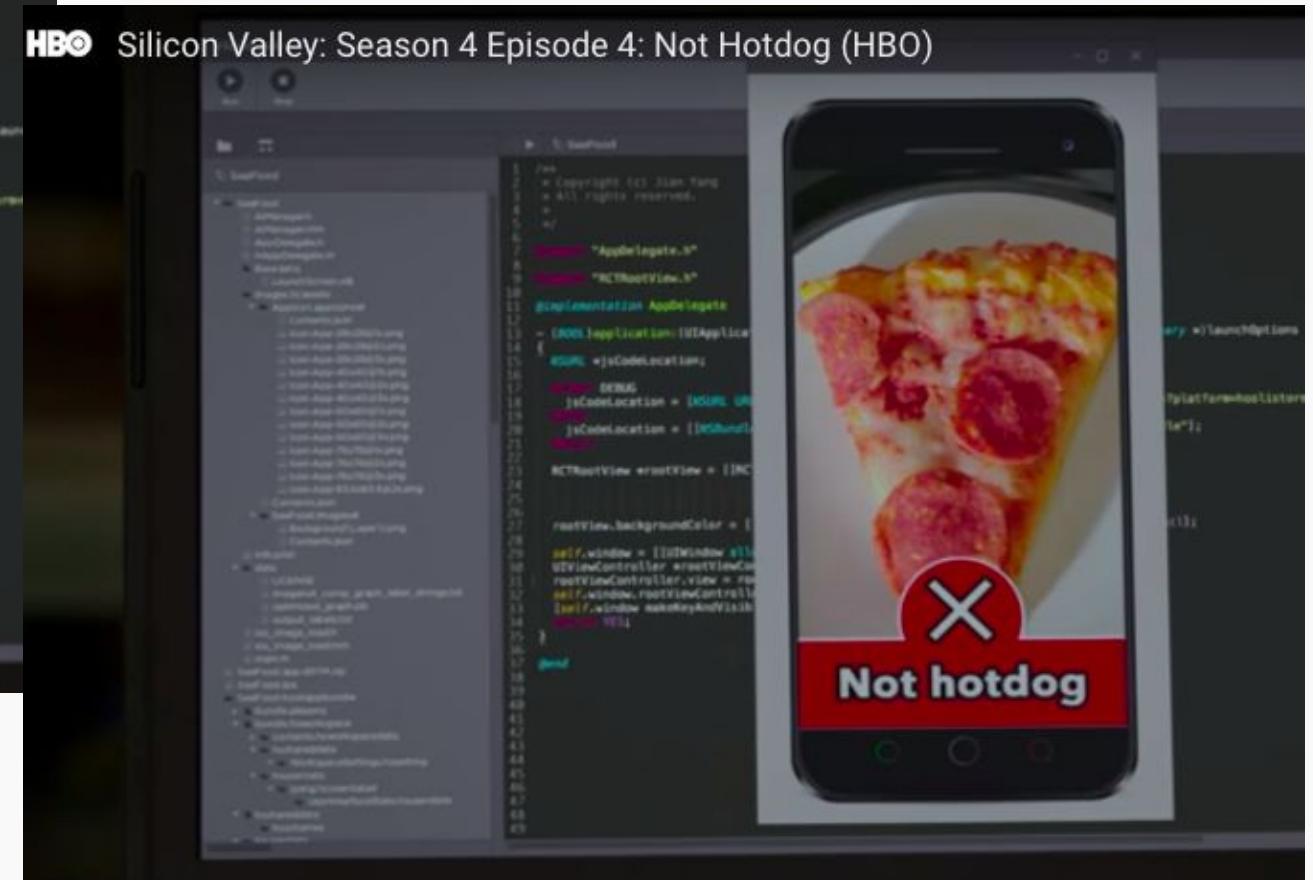
# Hotdog or NotHotDog: <https://youtu.be/ACmydtFDTGs> (offensive language and tropes alert)



Silicon Valley: Season 4 Episode 4: Not Hotdog (HBO)



HBO Silicon Valley: Season 4 Episode 4: Not Hotdog (HBO)

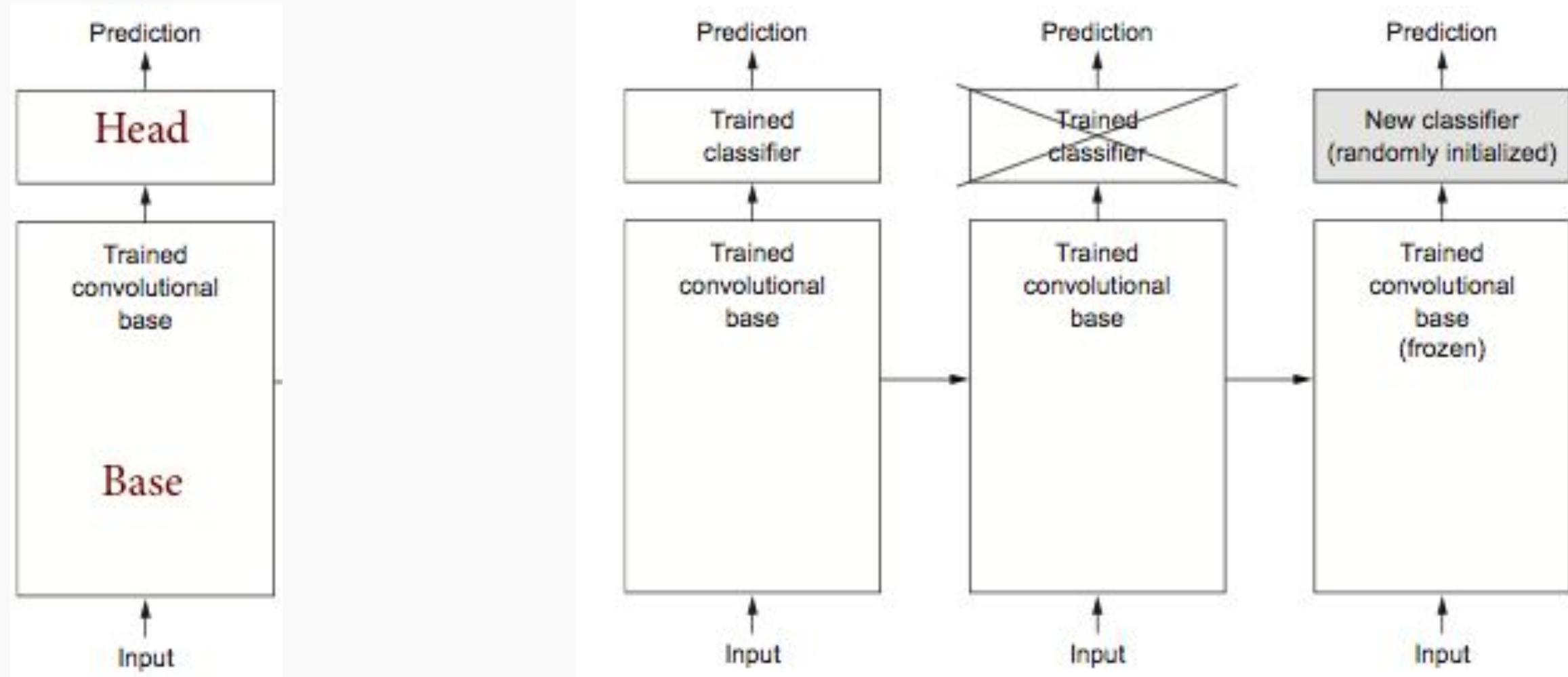


# Transfer Learning (cont)

---

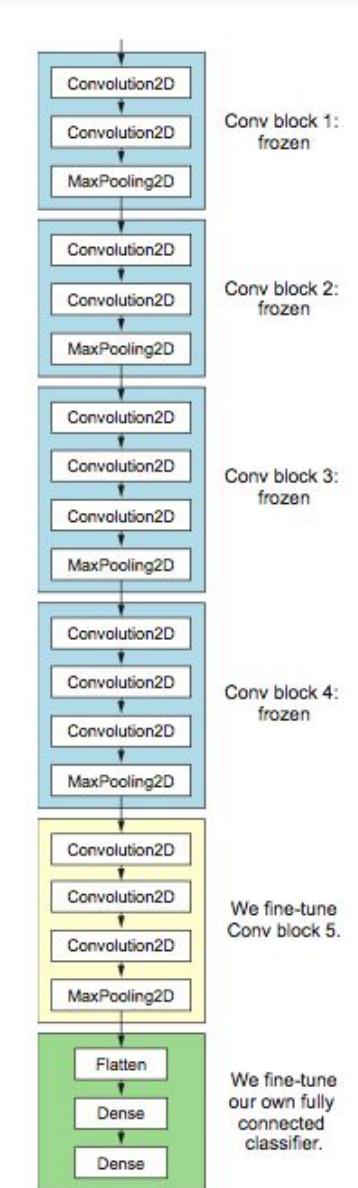
- train on a big "**source**" data set, with a big model, on one particular downstream tasks (say classification). Do it once and save the parameters. This is called a **pre-trained model**.
- use these parameters for other smaller "**target**" datasets, say, for classification on new images (possibly different **domain**, or training distribution), or for image segmentation on old images (new **task**), or new images (new task and new domain).
- less helpful if you have a large target dataset with many **labels**.
- will fail if source domain (where you trained big model) has nothing in common with target domain (that you want to train on smaller data set).

# Transfer Learning (cont)



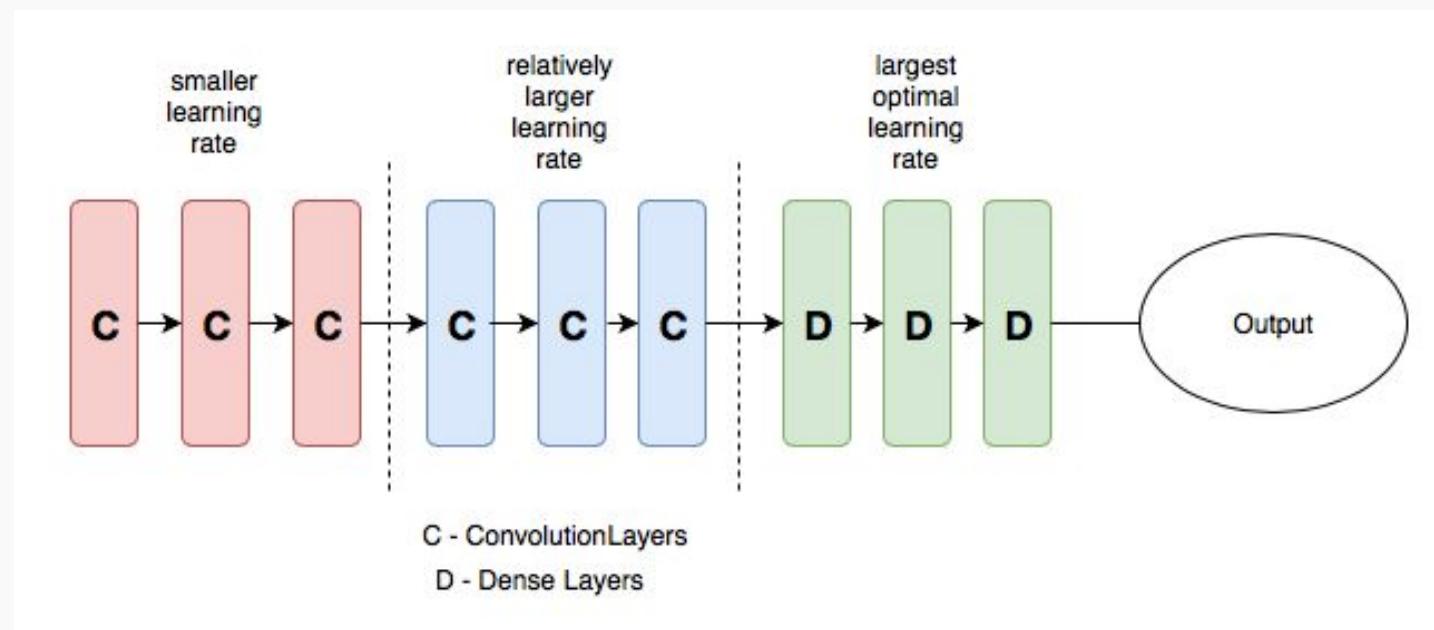
# Transfer Learning: Fine-tuning

- Up to now we have frozen the entire convolutional base.
- Remember that earlier layers learn highly generic feature maps (edges, colors, textures).
- Later layers learn abstract concepts (dog's ear).
- To particularize the model to our task, its often worth tuning the later layers as well.



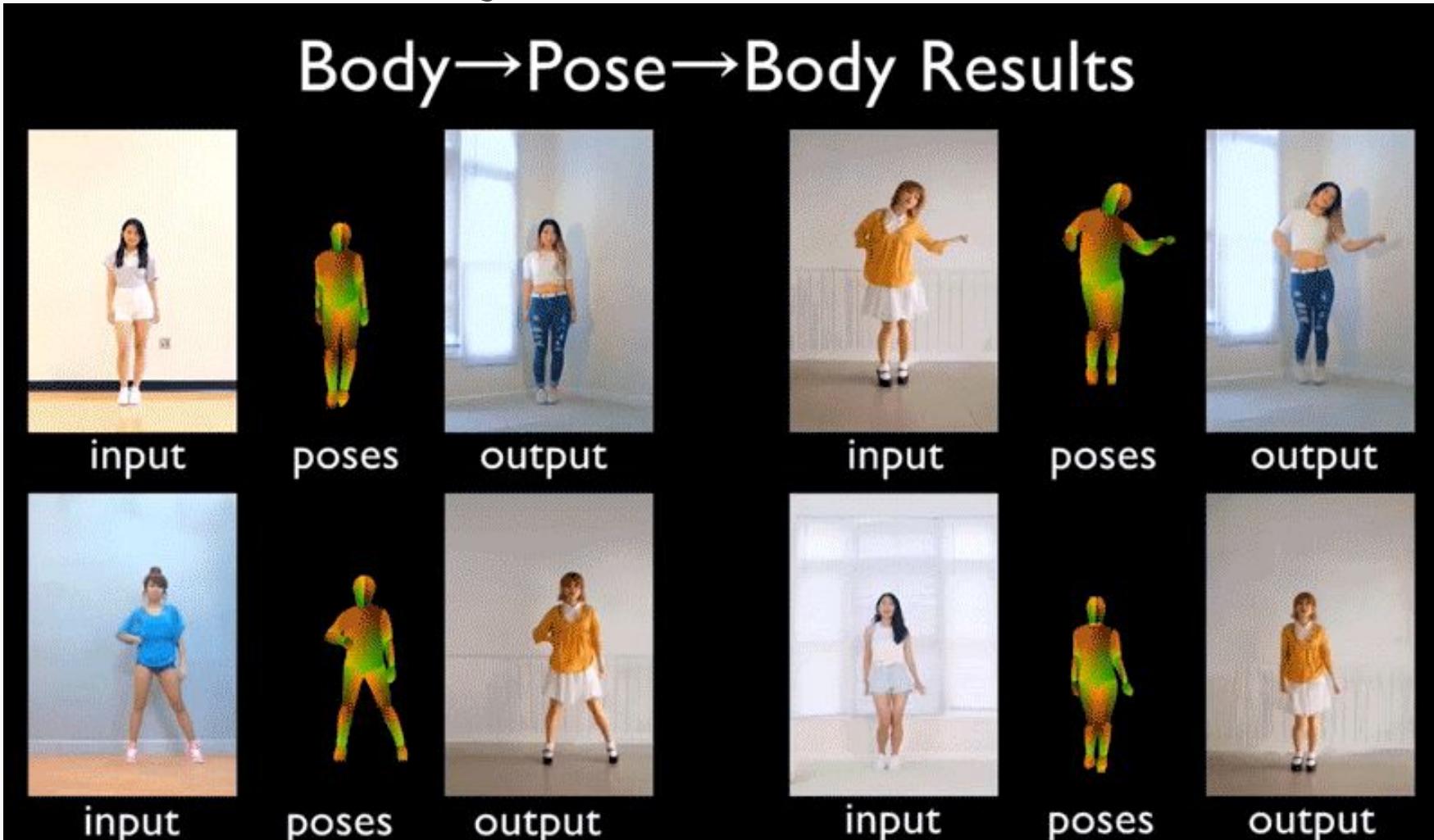
# Transfer Learning: Fine-tuning

- A low learning rate can take a lot of time to train on the "later" layers. Since we trained the FC head earlier, we could probably retrain them at a higher learning rate.
- General Idea: Train different layers at different rates.
- Each "earlier" layer or layer group (the color-coded layers in the image) can be trained at 3x-10x smaller learning rate than the next "later" one.
- One could even train the entire network again this way until we overfit and then step back some epochs.



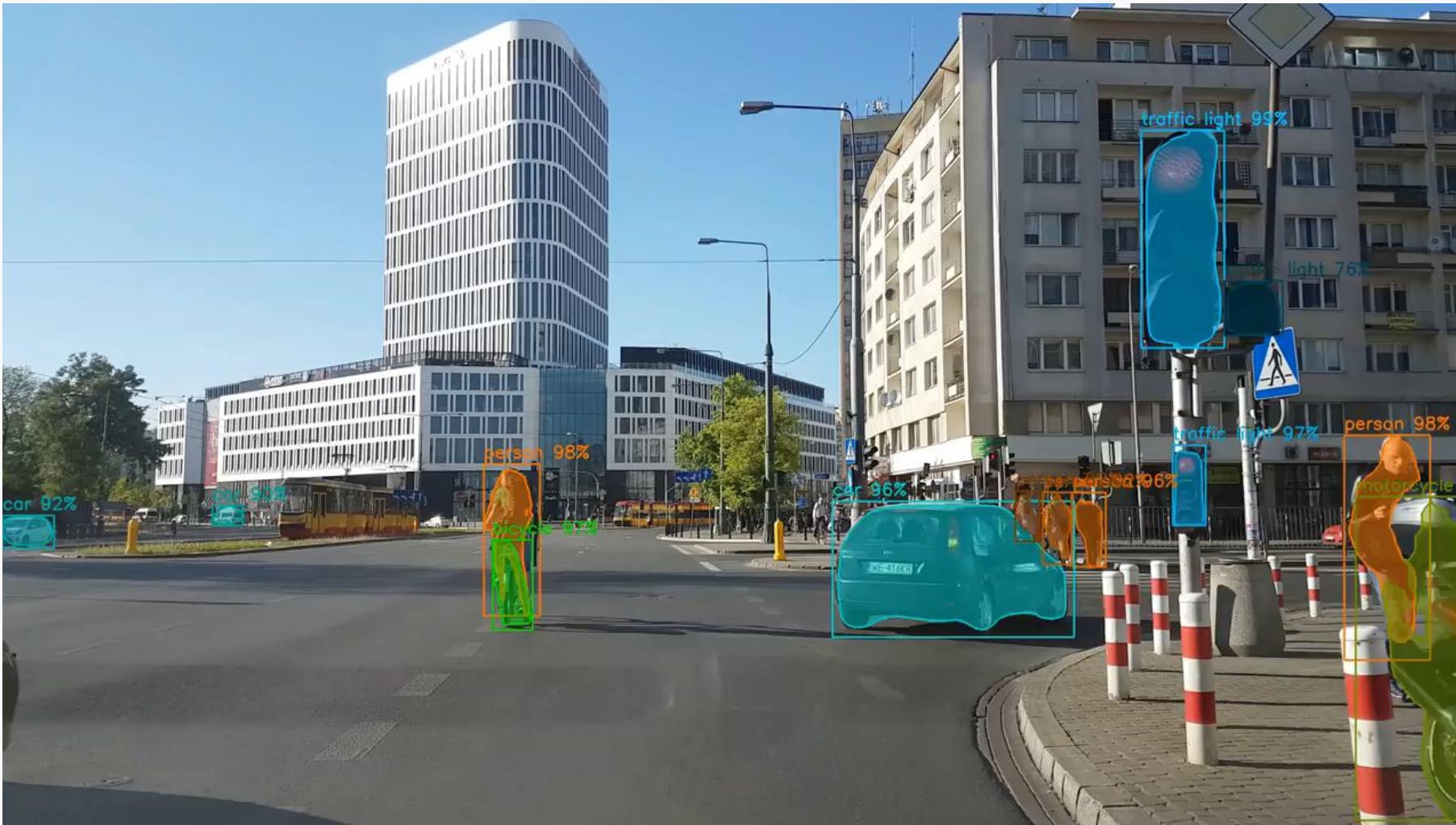
# Cool Transfer learning application

NVIDIA Video to Video Synthesis - 2018



# Latest events on Image Recognition

Mask- RCNN - 2017



# Outline

---

1. Review from last lecture
2. Training CNNs
3. BackProp of MaxPooling layer
4. Layers Receptive Field
5. Saliency maps - more graphics
6. Transfer Learning. - AC295
7. Segmentation
8. A bit of history and SOTA



# Initial ideas

---

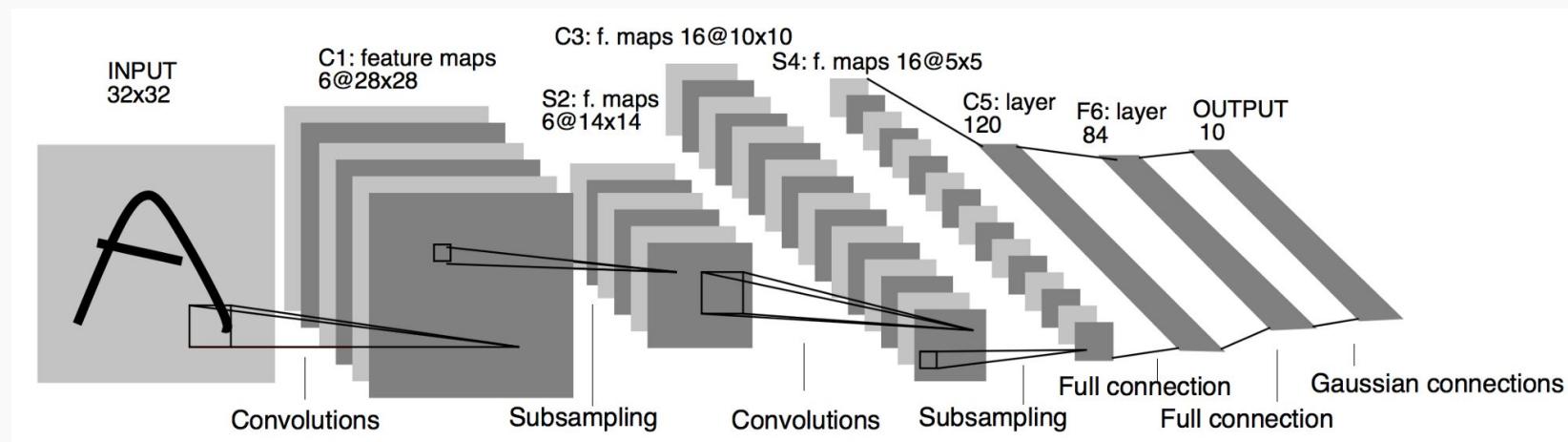
- The first piece of research proposing something similar to a Convolutional Neural Network was authored by Kunihiko Fukushima in 1980, and was called the **NeoCognitron**<sup>1</sup>.
- Inspired by discoveries on visual cortex of mammals.
- Fukushima applied the NeoCognitron to hand-written character recognition.
- End of the 80's: several papers advanced the field
  - **Backpropagation** published in French by Yann LeCun in 1985 (independently discovered by other researchers as well)
  - TDNN by Waibel et al., 1989 - **Convolutional-like** network trained with backprop.
  - Backpropagation applied to handwritten zip code recognition by LeCun et al., 1989

---

<sup>1</sup> K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36(4): 93-202, 1980.

# LeNet

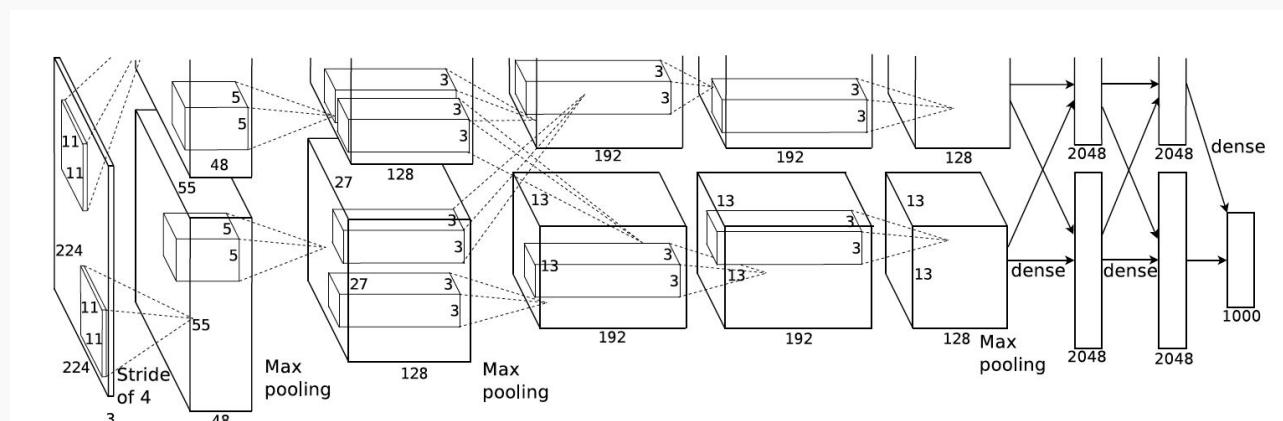
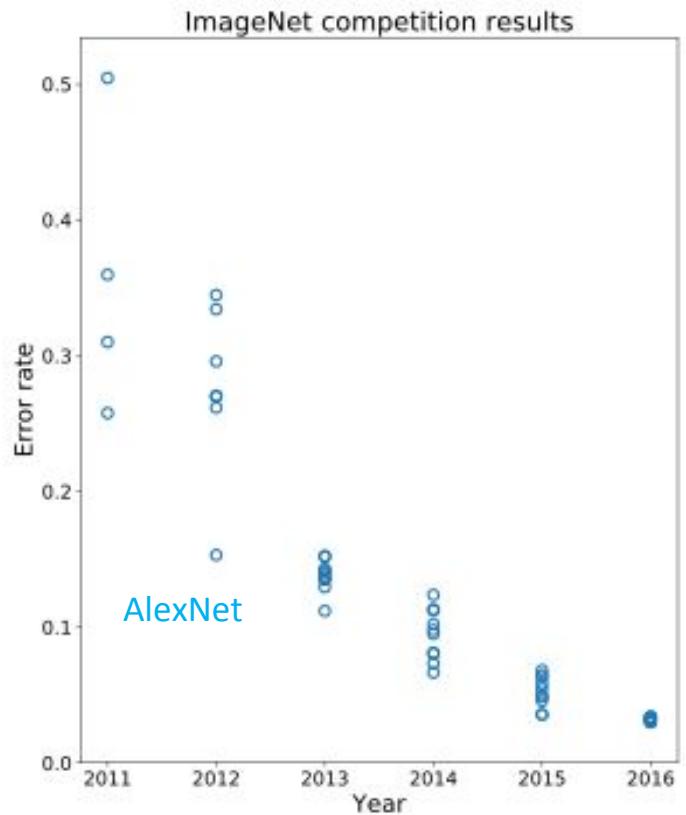
- November 1998: LeCun publishes one of his most recognized papers describing a “modern” CNN architecture for document recognition, called LeNet<sup>1</sup>.
- Not his first iteration, this was in fact LeNet-5, but this paper is the commonly cited publication when talking about LeNet.



<sup>1</sup> LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

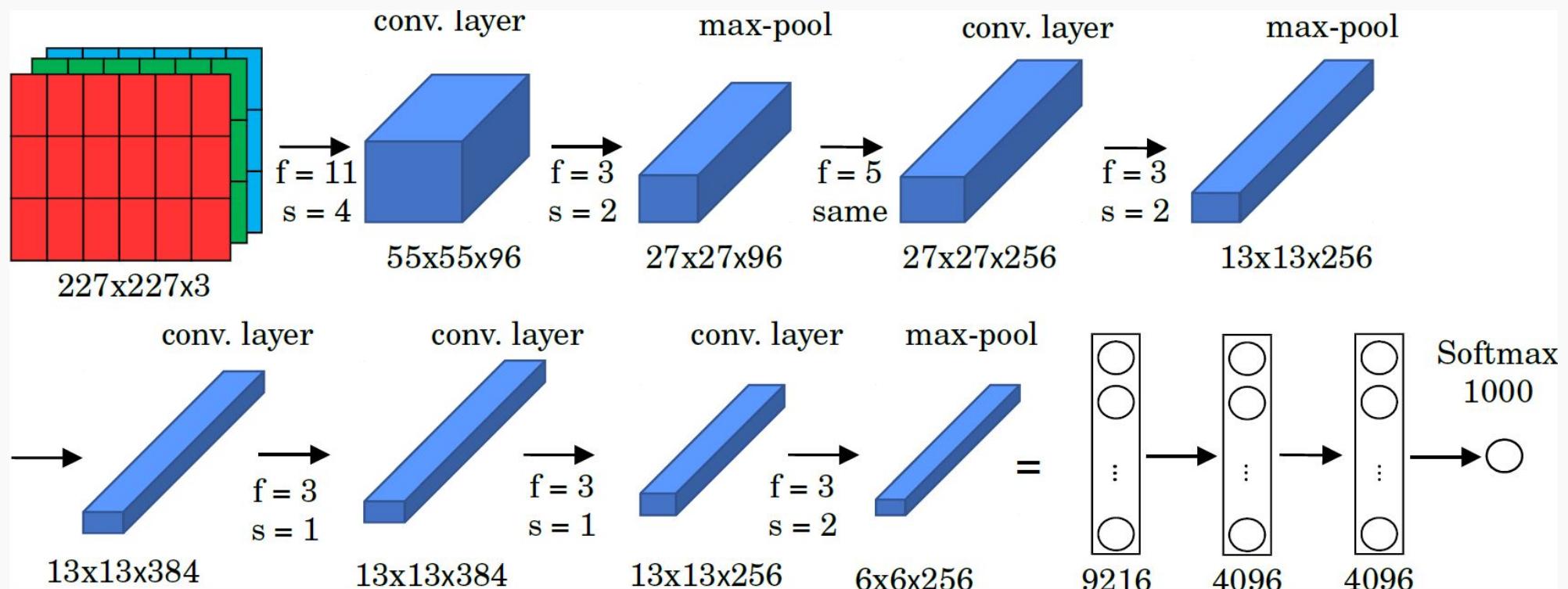
# AlexNet

- Developed by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton at Utoronto in 2012. More than 25000 citations.
- Destroyed the competition in the 2012 **ImageNet Large Scale Visual Recognition Challenge**. Showed benefits of CNNs and kickstarted AI revolution.
- top-5 error of 15.3%, more than 10.8 percentage points lower than runner-up.
- Main contributions:
  - Trained on ImageNet with data augmentation
  - Increased depth of model, GPU training (*five to six days*)
  - Smart optimizer and Dropout layers
  - ReLU activation!



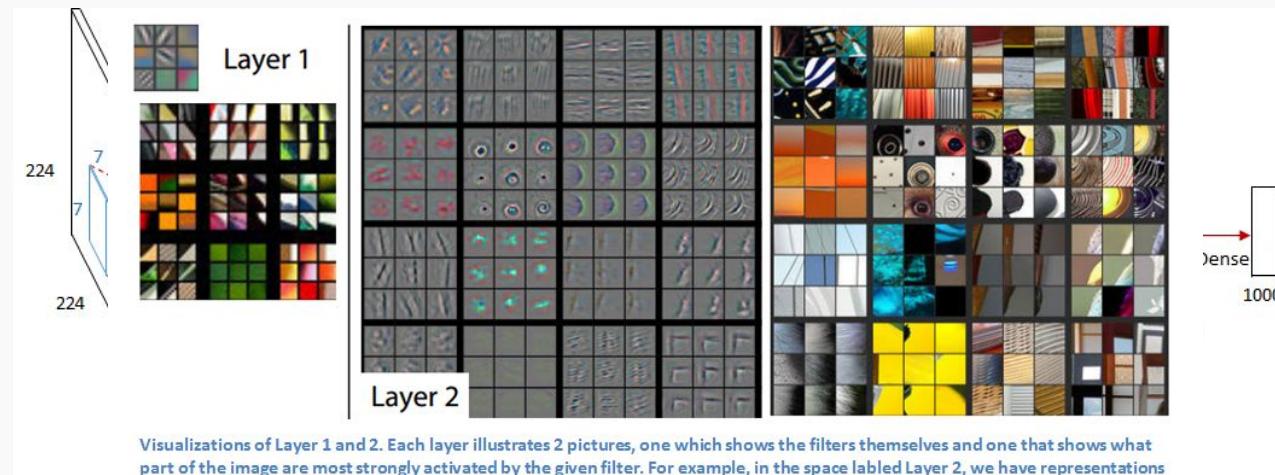
# AlexNet

- 1.2 million high-resolution ( $227 \times 227 \times 3$ ) images in the ImageNet 2010 contest;
- 1000 different classes, NN with 60 million parameters to optimize (~ 255 MB);
- Uses ReLu activation functions; GPUs for training, 12 layers.



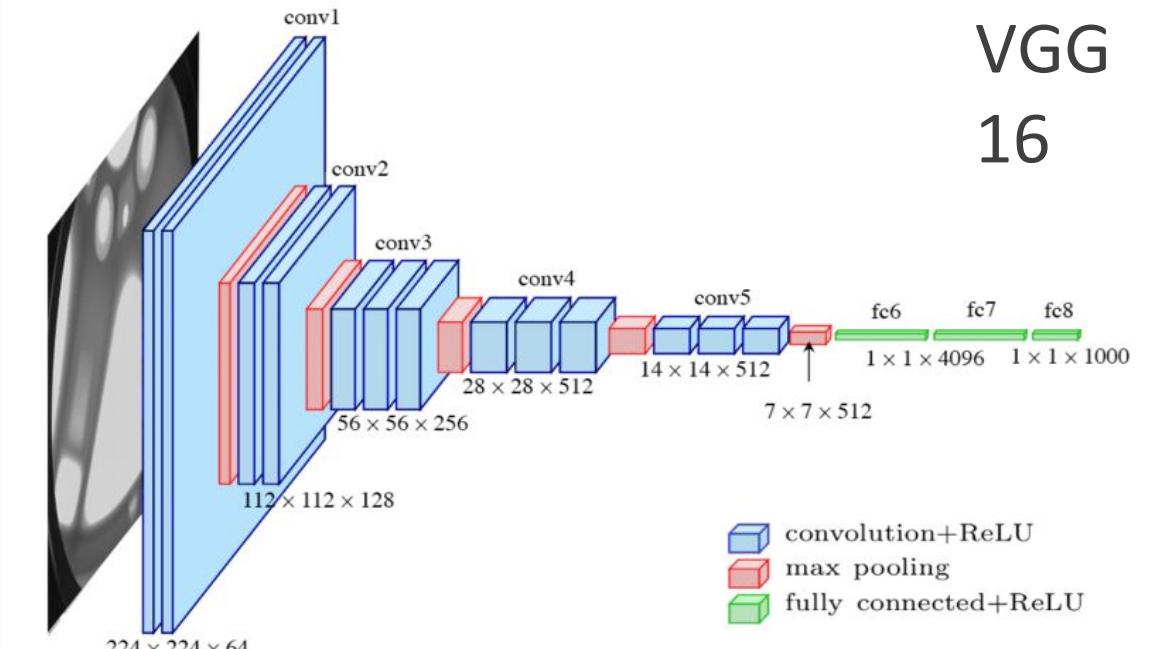
# ZFNet

- Introduced by Matthew Zeiler and Rob Fergus from NYU, won ILSVRC 2013 with 11.2% error rate. Decreased sizes of filters.
- Trained for 12 days.
- Paper presented a visualization technique named Deconvolutional Network, which helps to examine different feature activations and their relation to the input space.



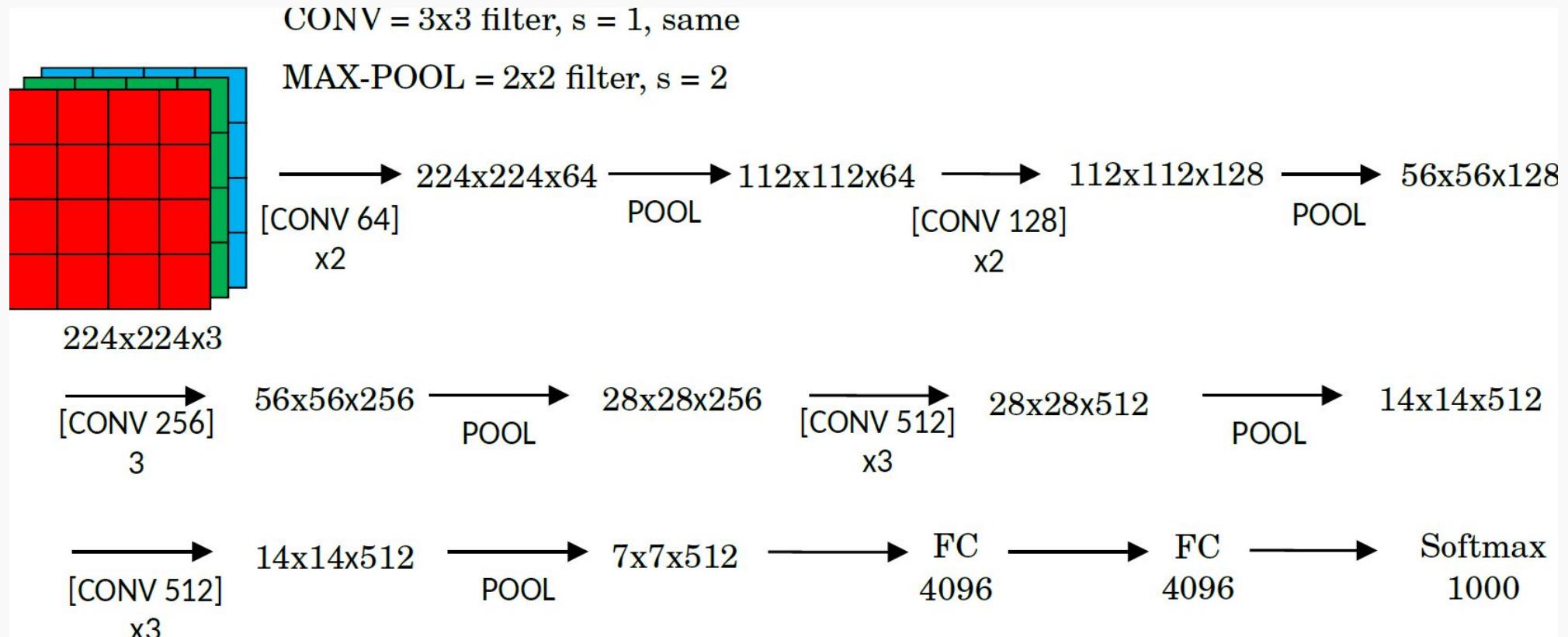
# VGG

- Introduced by **Simonyan and Zisserman** (Oxford) in 2014
- **Simplicity and depth as main points.** Used **3x3 filters exclusively** and 2x2 MaxPool layers with stride 2.
- Showed that two 3x3 filters have an effective receptive field of 5x5.
- As spatial size decreases, depth increases.
- Trained for two to three weeks.
- Still used as of today.



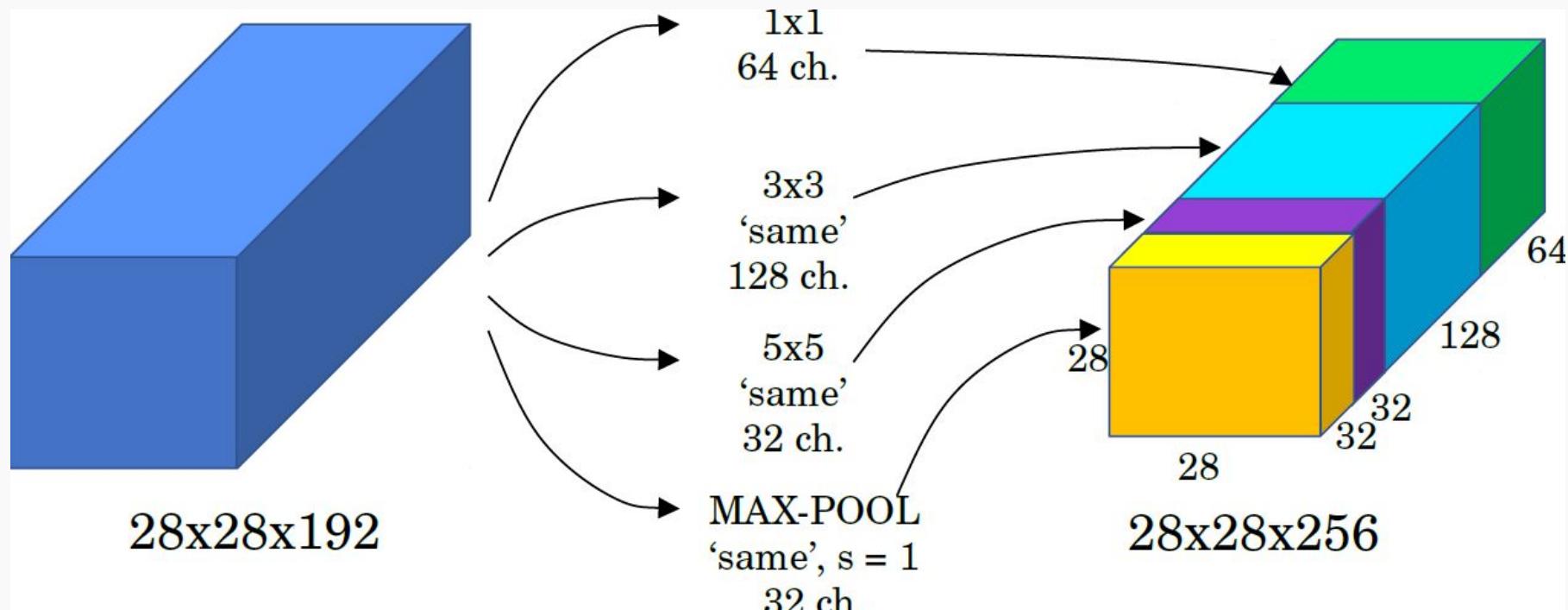
# VGG

- ImageNet Challenge 2014; 16 or 19 layers; 138 million parameters (~ 522 MB).
- Convolutional layers use ‘same’ padding and stride s=1.
- Max-pooling layers use a filter size f=2 and stride s=2.



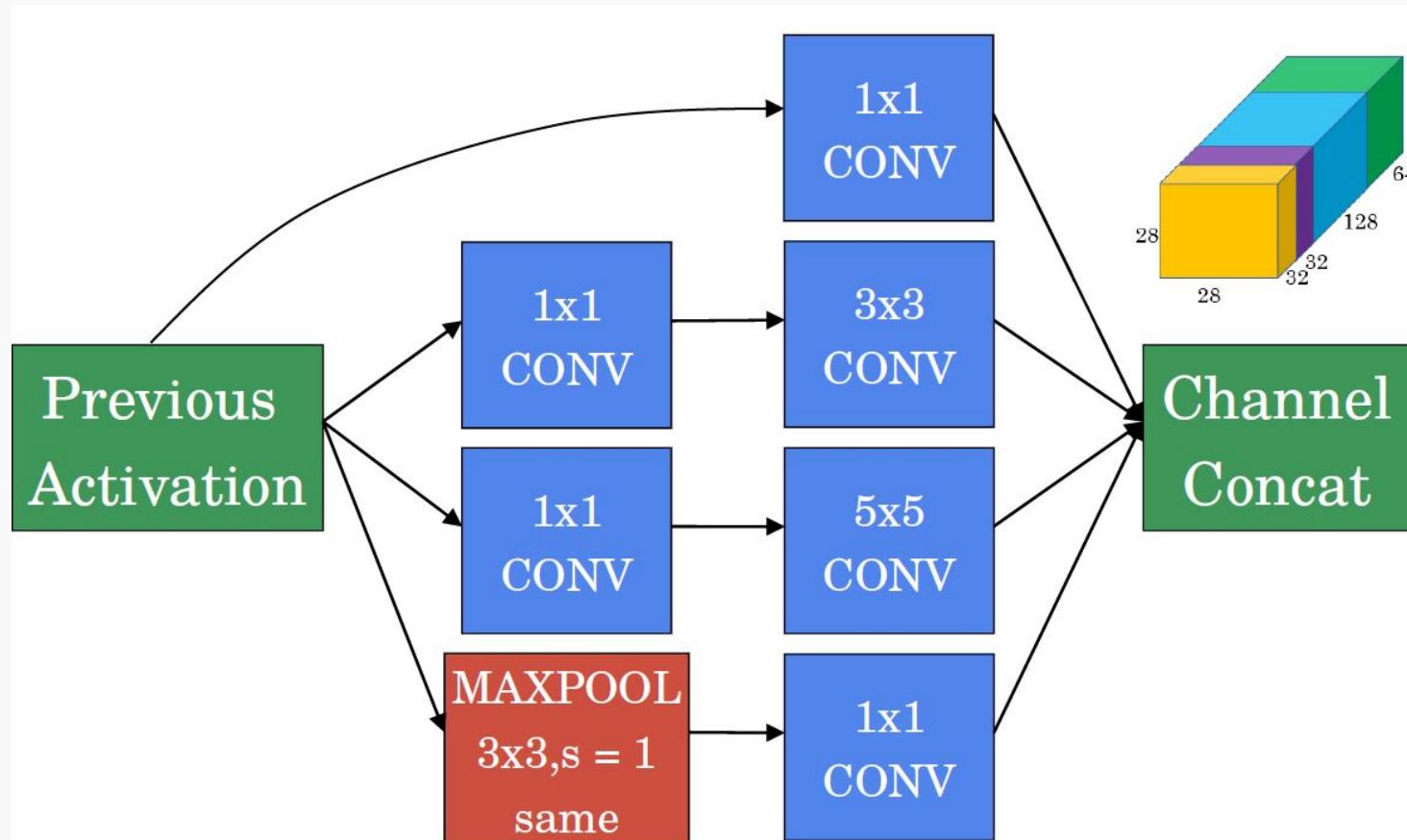
# SOTA Deep Models: Inception (GoogLeNet)

- The motivation behind inception networks is to use more than a single type of convolution layer at each layer.
- Use  $1 \times 1$ ,  $3 \times 3$ ,  $5 \times 5$  convolutional layers, and max-pooling layers in parallel.
- All modules use same convolution.
- Basic implementation:



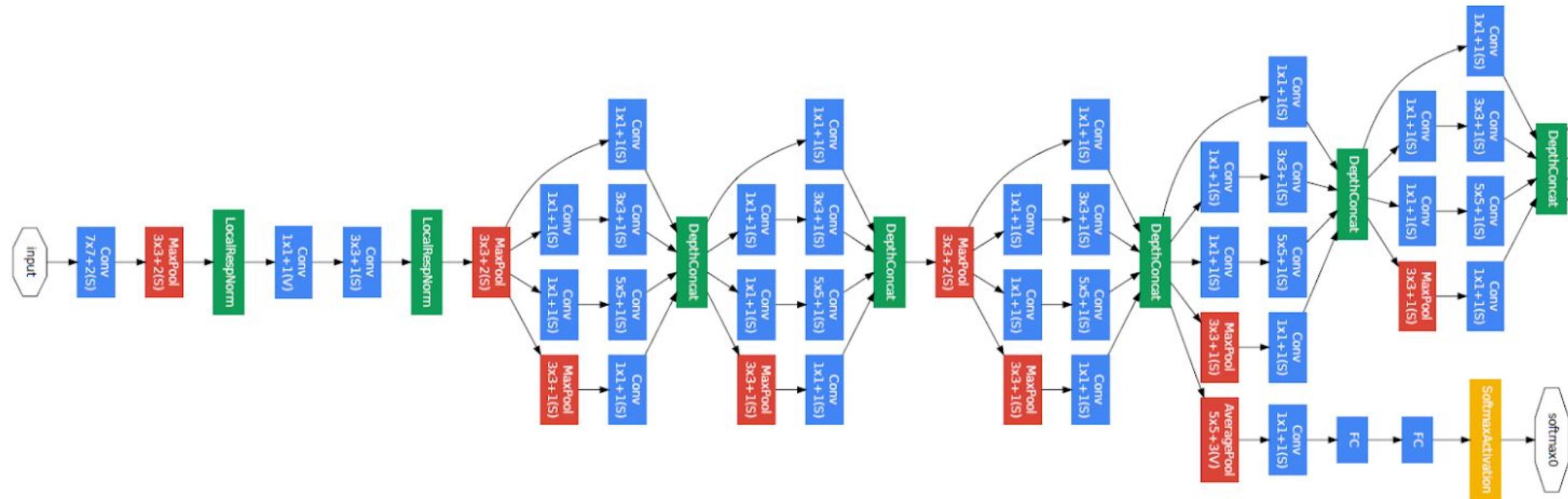
# SOTA Deep Models: Inception (GoogLeNet)

- Use  $1 \times 1$  convolutions that reduce the size of the channel dimension.
  - The number of channels can vary from the input to the output..



# SOTA Deep Models: Inception (GoogLeNet)

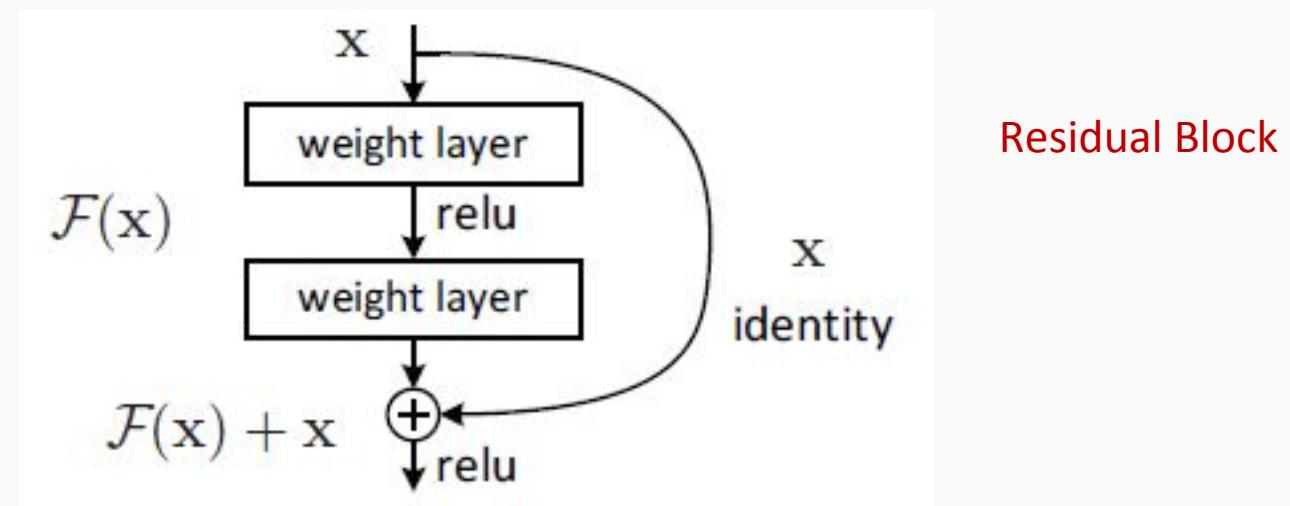
- The inception network is formed by concatenating other inception modules.
- It includes several softmax output units to enforce regularization.





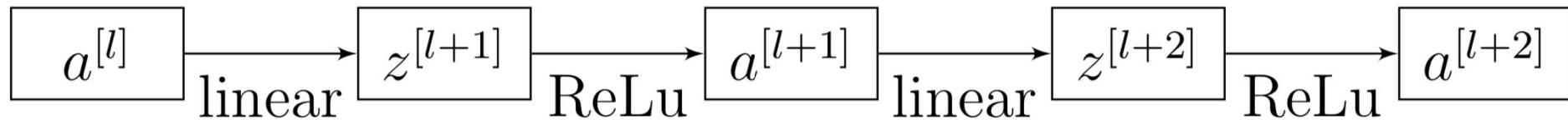
# ResNet

- Presented by He et al. (Microsoft), 2015. Won ILSVRC 2015 in multiple categories.
- Main idea: Residual block. Allows for extremely deep networks.
- Authors believe that it is easier to optimize the residual mapping than the original one. Furthermore, residual block can decide to “shut itself down” if needed.

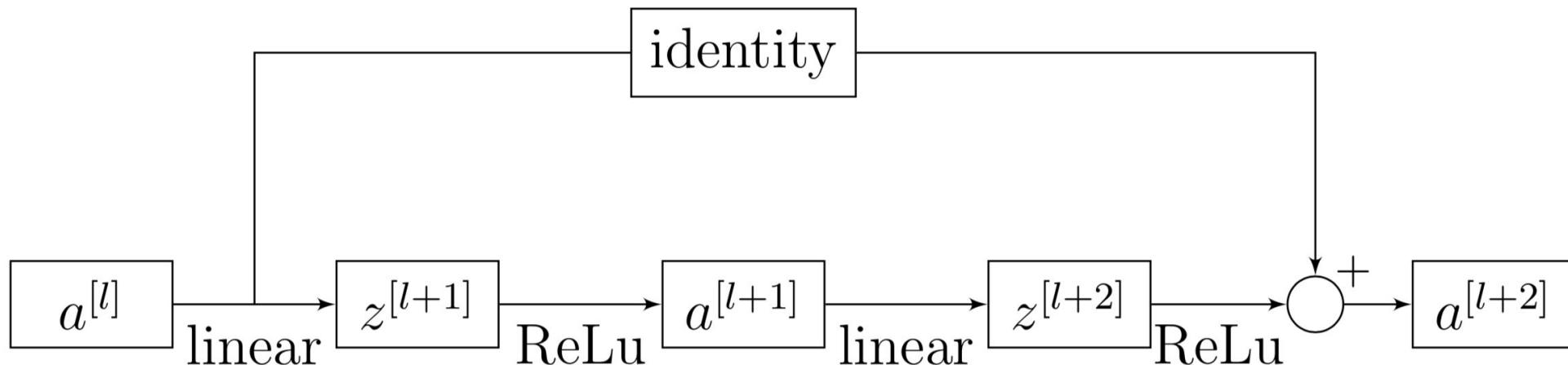


# ResNet

- Residual nets appeared in 2016 to train very deep NN (100 or more layers).
- Their architecture uses ‘residual blocks’.
- Plain network structure:

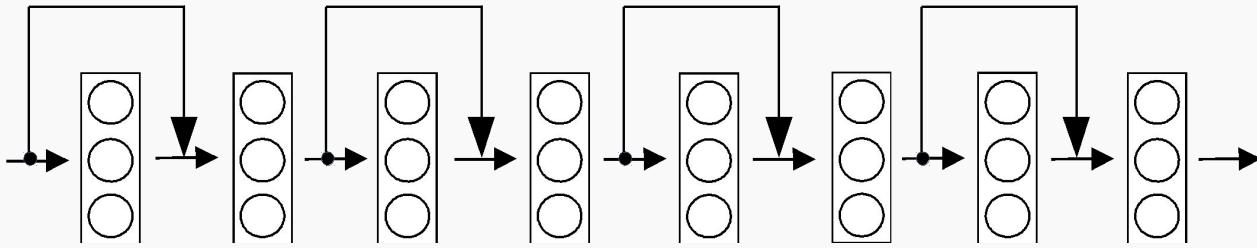


- **Residual network block**

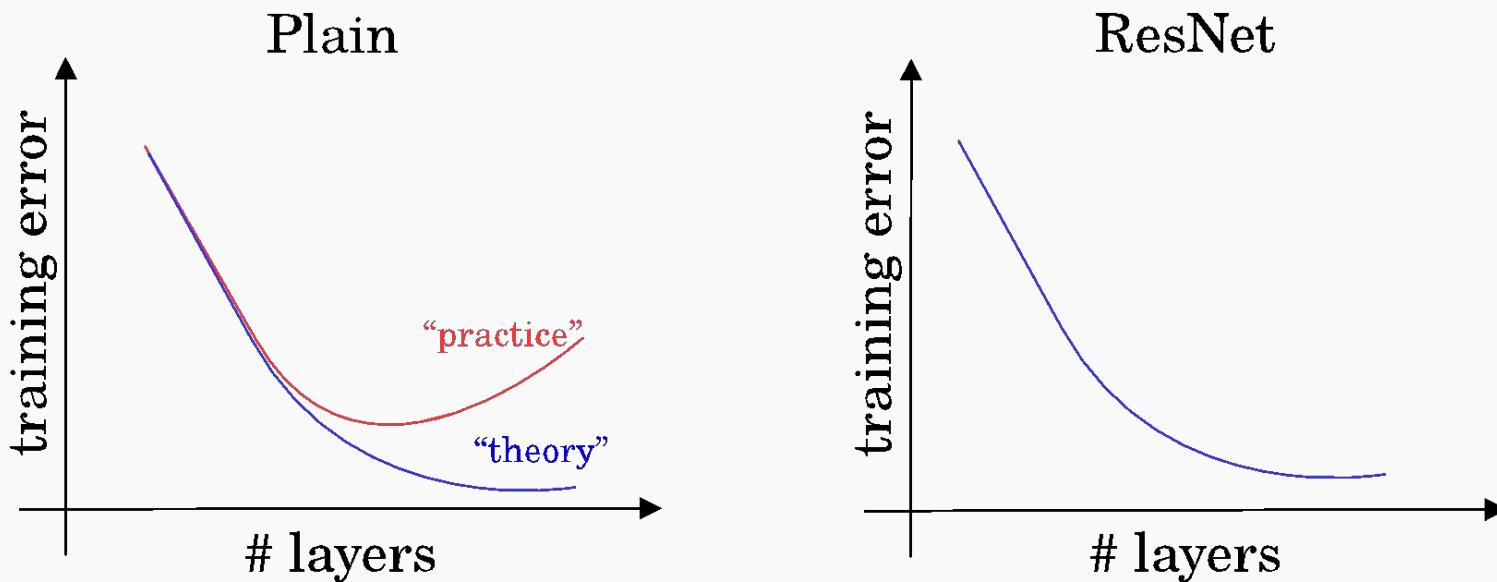


# ResNet

The residual network stacks blocks sequentially



The idea is to allow the network to become deeper without increasing the training time



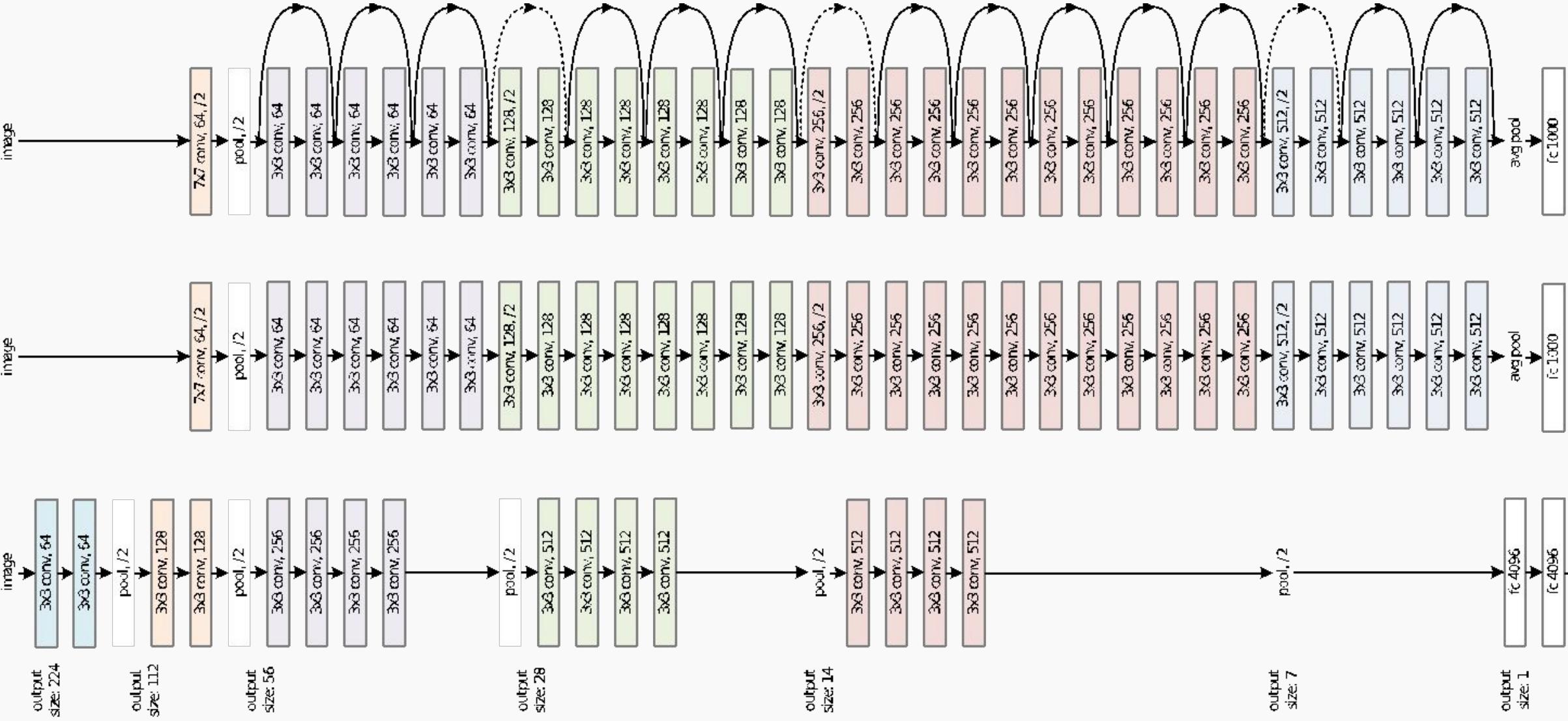
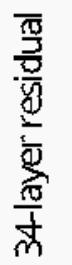
# ResNet

Residual networks implement blocks with convolutional layers that use ‘same’ padding option (even when max-pooling).

- This allows the block to learn the identity function.
- The designer may want to reduce the size of features and use ‘valid’ padding.
- In such case, the shortcut path can implement a new set of convolutional layers that reduces the size appropriately.

Number of Layers	Number of Parameters
ResNet 18	11.174M
ResNet 34	21.282M
ResNet 50	23.521M
ResNet 101	42.513M
ResNet 152	58.157M

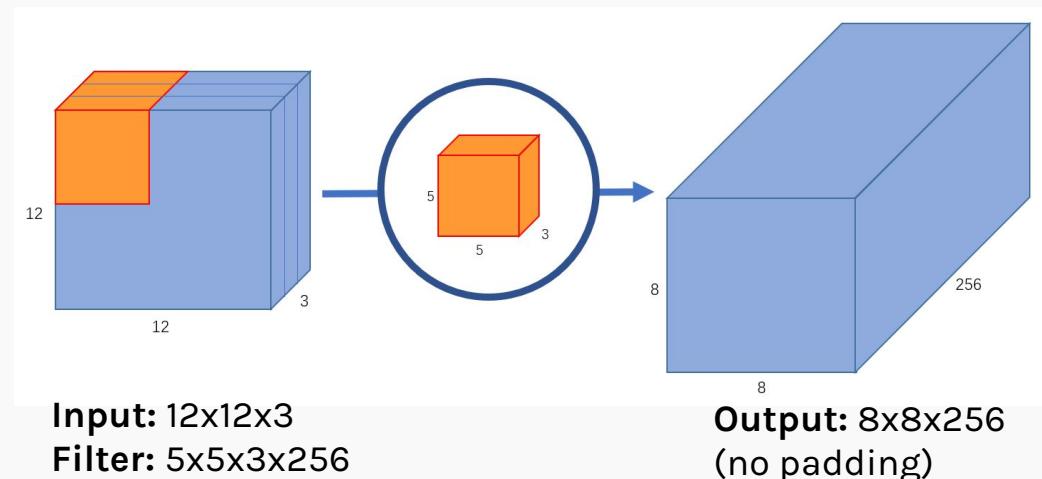
## ResNet



# SOTA Deep Models: MobileNet

## Standard Convolution

Filters and combines inputs into a new set of outputs in one step

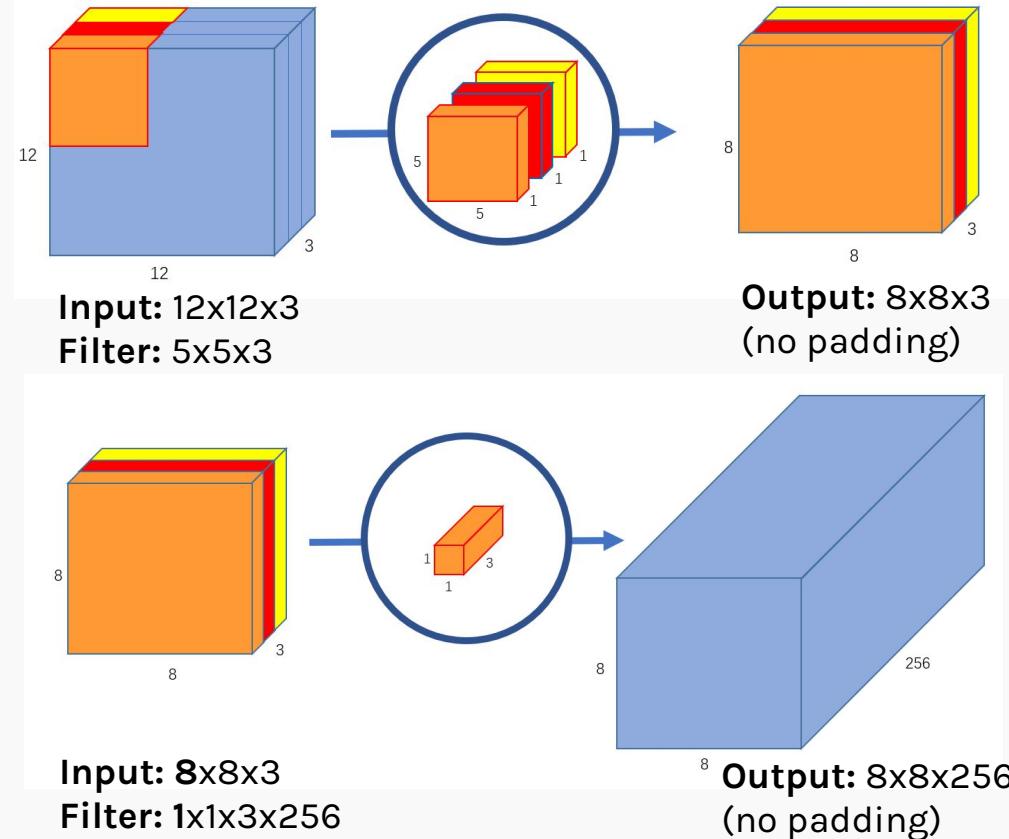


**MACs:**  $(5 \times 5) \times 3 \times 256 \times (12 \times 12) \sim 2.8M$

**Parameters:**  $(5 \times 5 \times 3) \times 256 + 256 \sim 20K$

## Depth-Wise Separable Convolution (DW)

It combines a depth wise convolution and a pointwise convolution



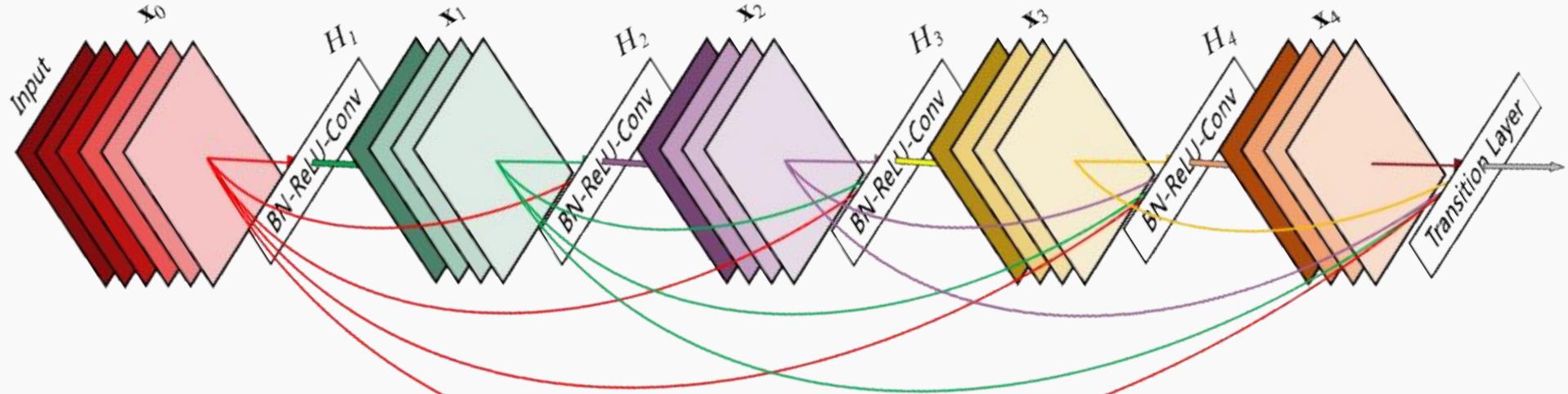
**MACs:**  $(5 \times 5) \times 3 \times (12 \times 12) + 3 \times 256 \times (8 \times 8) \sim 60K$

**Parameters:**  $(5 \times 5 \times 3 + 3) + (1 \times 1 \times 3 \times 256 + 256) \sim 1K$



# SOTA Deep Models: DenseNets

- **Goal:** allow maximum information (and gradient) flow → connect every layer directly with each other.
- DenseNets exploit the potential of the network through feature reuse → no need to learn redundant feature maps.
- DenseNets layers are very narrow (e.g. 12 filters), and they just add a small set of new feature-maps.



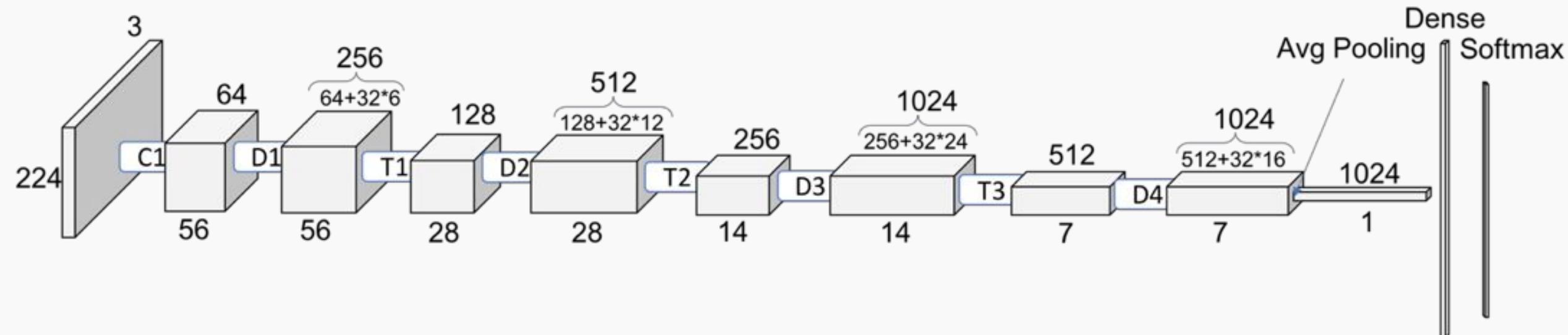
# SOTA Deep Models: DenseNets

- DenseNets do not sum the output feature maps of the layer with the incoming feature maps but concatenate them:

$$a^{[l]} = g([a^{[0]}, a^{[1]}, \dots, a^{[l-1]}])$$

- Dimensions of the feature maps remains constant within a block, but the number of filters changes between them → **growth rate**:

$$k^{[l]} = k^{[0]} + k(l - 1)$$



# Beyond

---

- MobileNetV2 (<https://arxiv.org/abs/1801.04381>)
- Inception-Resnet, v1 and v2  
(<https://arxiv.org/abs/1602.07261>)
- Wide-Resnet (<https://arxiv.org/abs/1605.07146>)
- Xception (<https://arxiv.org/abs/1610.02357>)
- ResNeXt (<https://arxiv.org/pdf/1611.05431>)
- ShuffleNet, v1 and v2 (<https://arxiv.org/abs/1707.01083>)
- Squeeze and Excitation Nets  
(<https://arxiv.org/abs/1709.01507> )

# What's next

---

Advanced topics start today on Transfer Learning :

4:30pm @ MD 115

## Next Week:

Segmentation -

Autoencoders

Start of RNNs



# Advanced Sec. 2: Object Detection and Semantic Segmentation

- **IMAGE CLASSIFICATION**

assigning 1 single label to the **entire picture** = Easy!

- **Algorithms.: VGG/Resnet/Densenet**



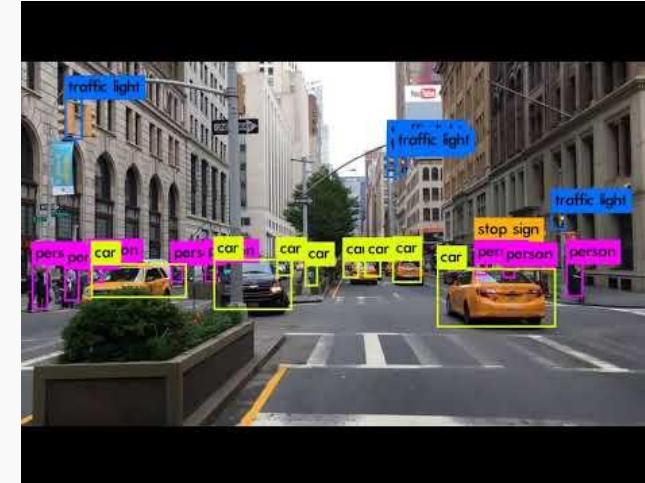
DOG

- **OBJECT DETECTION**

detect, classify and locate **every object** in the picture

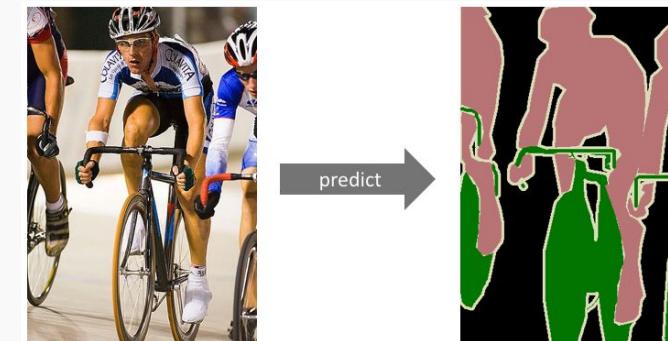
- **Algorithms:**

R-CNN/Fast-R-CNN/Faster-R-CNN &  
YOLO



- **SEMANTIC SEGMENTATION**

assigning a meaningful label to **every pixel** in the image



# Latest events on Image Recognition

## You Only Look Once (YOLO) - 2016

