Topics:

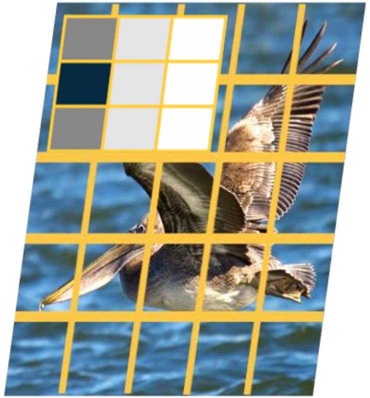
- Convolutional Neural Networks

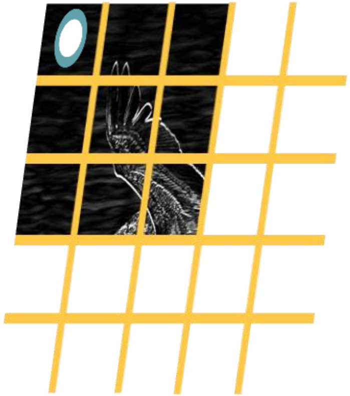
CS 4644-DL / 7643-A
ZSOLT KIRA

- **Assignment 2**
  - Implement convolutional neural networks
  - Resources (in addition to lectures):
    - DL book: Convolutional Networks
    - CNN notes https://www.cc.gatech.edu/classes/AY2022/cs7643_spring/assets/L10_cnns_notes.pdf
    - Backprop notes https://www.cc.gatech.edu/classes/AY2023/cs7643_spring/assets/L10_cnns_backprop_notes.pdf
    - **HW2 Tutorial, Conv backward @176**
    - Slower OMSCS lectures on dropbox: Module 2 Lessons 5-6 (M2L5/M2L6) (https://www.dropbox.com/sh/iviro188gq0b4vs/AADdHxX_Uy1TkpF_yvIzX0nPa?dl=0)
- **Project**
  - Info on @178

$$X(0:2,0:2) = \begin{bmatrix} 200 & 150 & 150 \\ 100 & 50 & 100 \\ 25 & 25 & 10 \end{bmatrix} \qquad K' = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \Rightarrow X(0:2,0:2) \cdot K' = 65 + \text{bias}$$

Dot product
(element-wise multiply and sum)

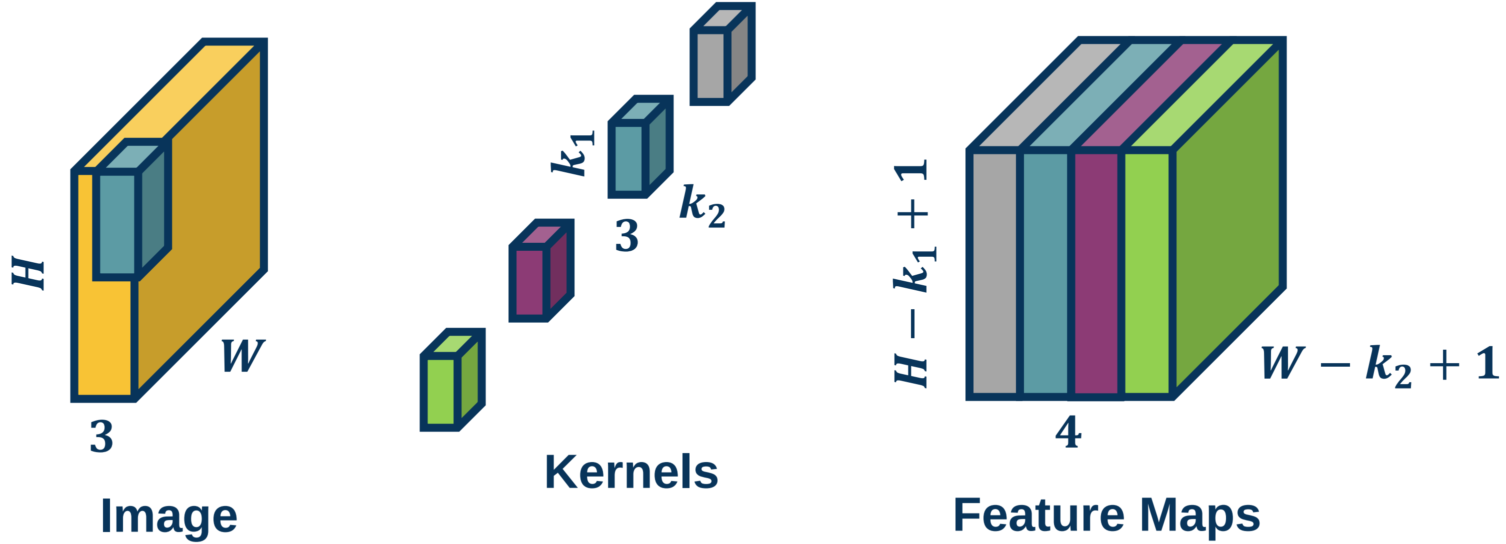




Cross-Correlation

Number of parameters with N filters is: $\boldsymbol{N * (k_1 * k_2 * 3 + 1)}$

- Example:
  $\boldsymbol{k_1 = 3, k_2 = 3,\ N = 4\ input\ channels = 3}$, then $\boldsymbol{(3 * 3 * 3 + 1) * 4 =}$112



Chain Rule over all Output Pixels

Need to incorporate all upstream gradients:

$$\left\{\frac{\partial L}{\partial y(0,0)}, \frac{\partial L}{\partial y(0,1)}, \dots, \frac{\partial L}{\partial y(H,W)}\right\}$$

Chain Rule:

$$\frac{\partial L}{\partial k(a,b)} = \sum_{r=0}^{H-1} \sum_{c=0}^{W-1} \frac{\partial L}{\partial y(r,c)} \frac{\partial y(r,c)}{\partial k(a,b)}$$

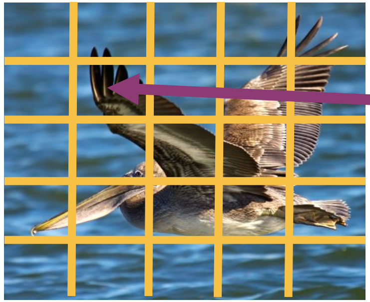
Sum over all output pixels

Upstream gradient (known)

We will compute

$(0,0)$

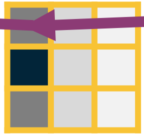
$H = 5$



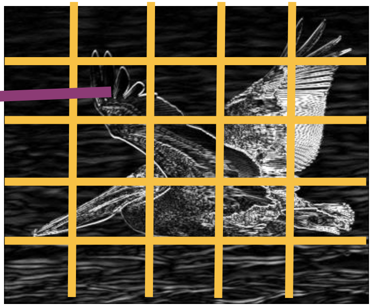
$W = 5$ $(H-1, W-1)$

$(0,0)$

$k_1 = 3$

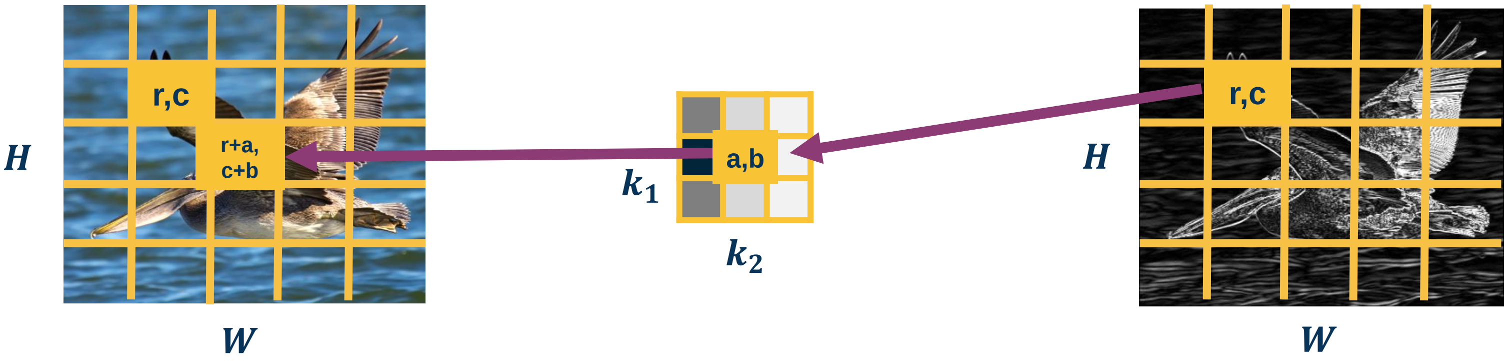


$k_2 = 3$ $(k_1 - 1, k_2 - 1)$



$$\frac{\partial y(r,c)}{\partial k(a,b)} = x(r+a, c+b)$$

$$\frac{\partial L}{\partial k(a,b)} = \sum_{r=0}^{H-1} \sum_{c=0}^{W-1} \frac{\partial L}{\partial y(r,c)} x(r+a, c+b)$$



Gradients and Cross-Correlation

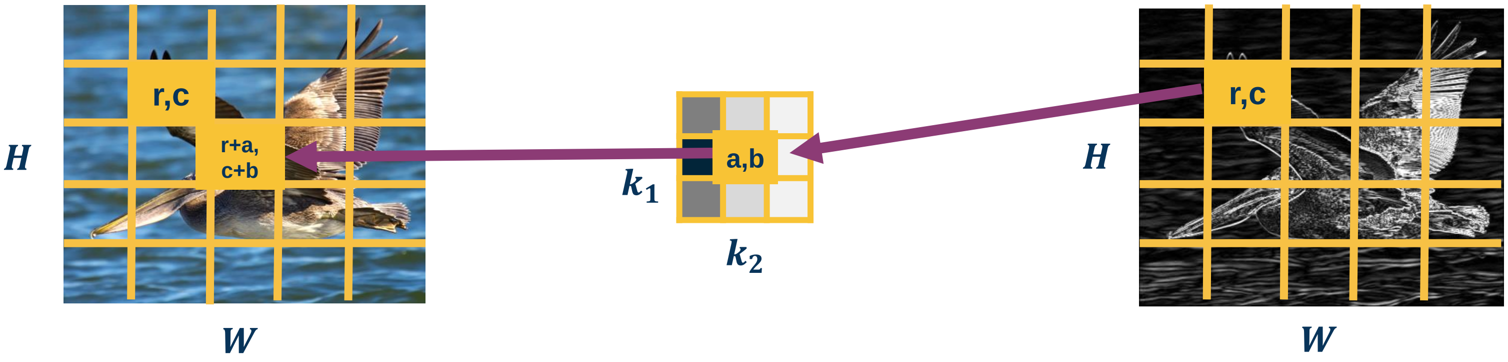
$$\frac{\partial y(r,c)}{\partial k(a,b)} = x(r+a, c+b)$$

$$\frac{\partial L}{\partial k(a,b)} = \sum_{r=0}^{H-1} \sum_{c=0}^{W-1} \frac{\partial L}{\partial y(r,c)} x(r+a, c+b)$$

Does this look familiar?

Cross-correlation between upstream gradient and input!

(until $k_1 \times k_2$ output)



$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial x}$$

Gradient for input (to pass to prior layer)

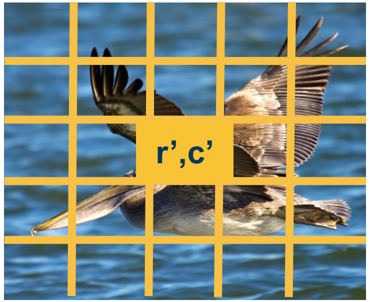
Calculate one pixel at a time $\frac{\partial L}{\partial x(r', c')}$

What does this input pixel affect at the output?

Neighborhood around it (where part of the kernel touches it)

$(0, 0)$

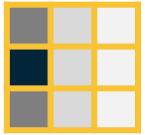
$H = 5$



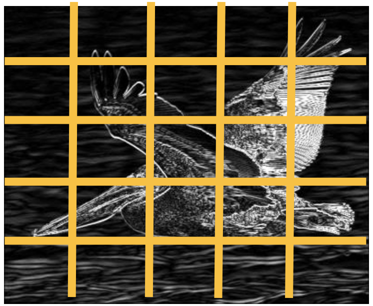
$W = 5$ $(H-1, W-1)$

$(0, 0)$

$k_1 = 3$



$k_2 = 3$ $(k_1 - 1, k_2 - 1)$



What an Input Pixel Affects at Output

Summing Gradient Contributions

Chain rule for affected pixels (sum gradients):

$$\frac{\partial L}{\partial x(r',c')} = \sum_{Pixels\ p} \frac{\partial L}{\partial y(p)} \frac{\partial y(p)}{\partial x(r',c')}$$

$$x(r',c') * k(0,0) \Rightarrow y(r',c')$$
$$x(r',c') * k(1,1) \Rightarrow y(r'-1,c'-1)$$
$$\ldots$$
$$x(r',c') * k(a,b) \Rightarrow y(r'-a,c'-b)$$

$$\frac{\partial L}{\partial x(r',c')} = \sum_{a=0}^{k_1-1} \sum_{b=0}^{k_2-1} \frac{\partial L}{\partial y(?,?)} \frac{\partial y(?,?)}{\partial x(r',c')}$$

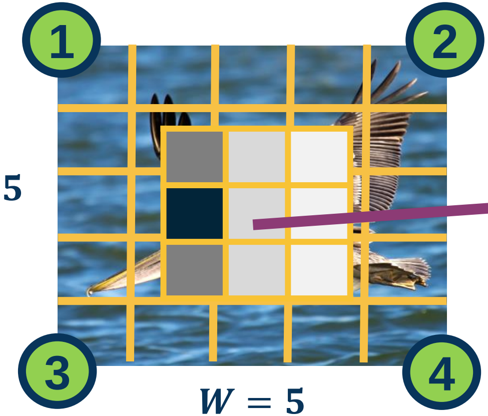
Chain rule for affected pixels (sum gradients):

$$\frac{\partial L}{\partial x(r',c')} = \sum_{Pixels\ p} \frac{\partial L}{\partial y(p)} \frac{\partial y(p)}{\partial x(r',c')}$$

Let's derive it analytically this time (as opposed to visually)

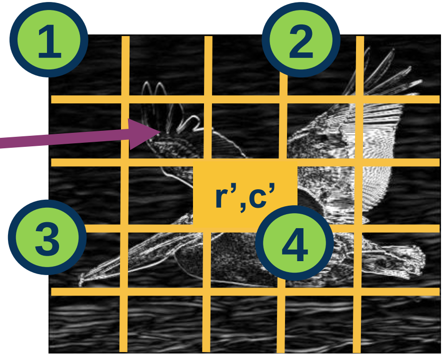
$$\frac{\partial L}{\partial x(r',c')} = \sum_{a=0}^{k_1-1} \sum_{b=0}^{k_2-1} \frac{\partial L}{\partial y(r'-a,c'-b)} \frac{\partial y(r'-a,c'-b)}{\partial x(r',c')}$$

$H = 5$



$W = 5$

$(r'-k_1+1, c'-k_2+1)$



Summing Gradient Contributions

Plugging in to earlier equation:

$$\frac{\partial L}{\partial x(r',c')} = \sum_{a=0}^{k_1-1}\sum_{b=0}^{k_2-1} \frac{\partial L}{\partial y(r'-a,c'-b)} \frac{\partial y(r'-a,c'-b)}{\partial x(r',c')}$$

$$= \sum_{a=0}^{k_1-1}\sum_{b=0}^{k_2-1} \frac{\partial L}{\partial y(r'-a,c'-b)} k(a,b)$$

Does this look familiar?

Convolution between upstream gradient and kernel!

(can implement by flipping kernel and cross- correlation)

Again, all operations can be implemented via matrix multiplications (same as FC layer)!

Backwards is Convolution

- Convolutions are mathematical descriptions of striding linear operation

- In practice, we implement **cross-correlation neural networks!** (still called convolutional neural networks due to history)
  - Can connect to convolutions via duality (flipping kernel)
  - Convolution formulation has mathematical properties explored in ECE

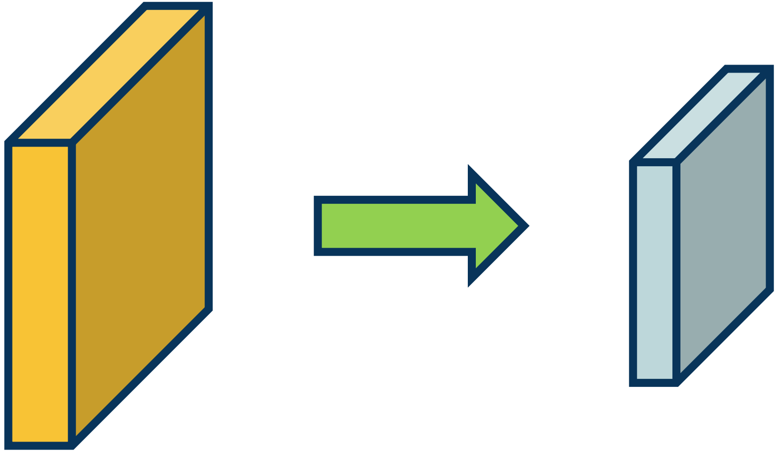
- Duality for forwards and backwards:
  - **Forward**: Cross-correlation
  - **Backwards w.r.t. K**: Cross-correlation b/w upstream gradient and input
  - **Backwards w.r.t. X**: Convolution b/w upstream gradient and kernel
    - In practice implement via cross-correlation and flipped kernel

- All operations still implemented via **efficient linear algebra** (e.g. matrix-matrix multiplication)

Summary

Pooling Layers

- **Dimensionality reduction** is an important aspect of machine learning
- Can we make a layer to **explicitly down-sample** image or feature maps?



- **Yes!** We call one class of these operations **pooling** operations

Parameters

- **kernel_size** – the size of the window to take a max over
- **stride** – the stride of the window. Default value is `kernel_size`
- **padding** – implicit zero padding to be added on both sides

From: https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html#torch.nn.MaxPool2d

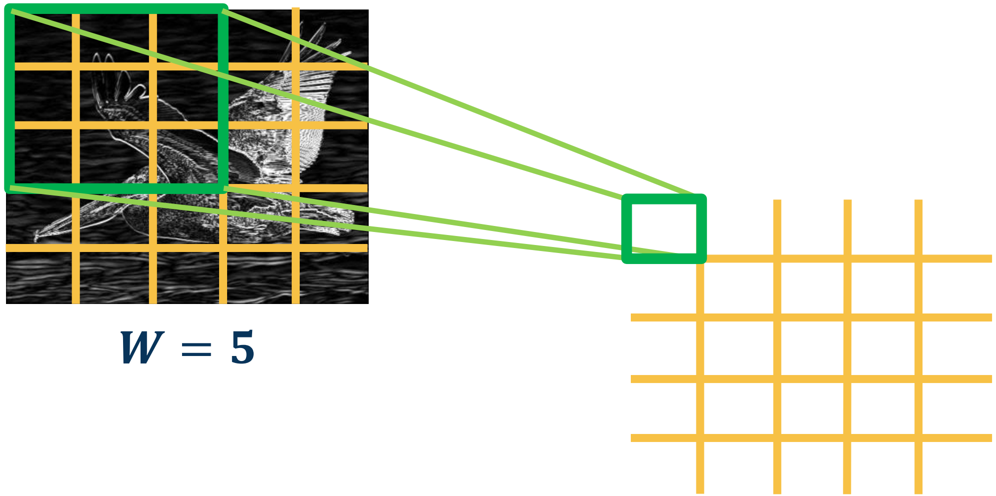
Pooling Layers

Example: Max pooling

- Stride window across image but perform per-patch **max operation**

$$X(0{:}2, 0{:}2) = \begin{bmatrix} 200 & 150 & 150 \\ 100 & 50 & 100 \\ 25 & 25 & 10 \end{bmatrix} \Rightarrow \max(0{:}2,0{:}2) = 200$$

$H = 5$

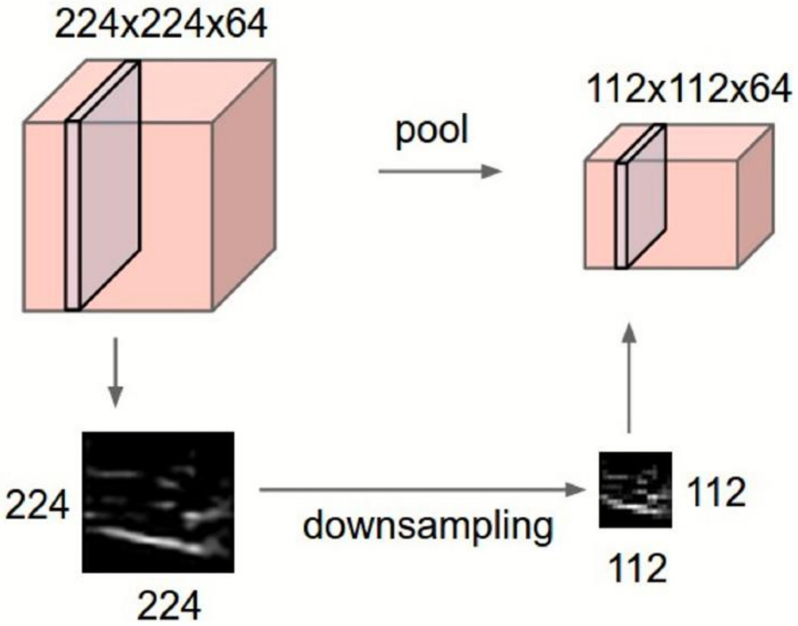


$W = 5$

How many learned parameters does this layer have?

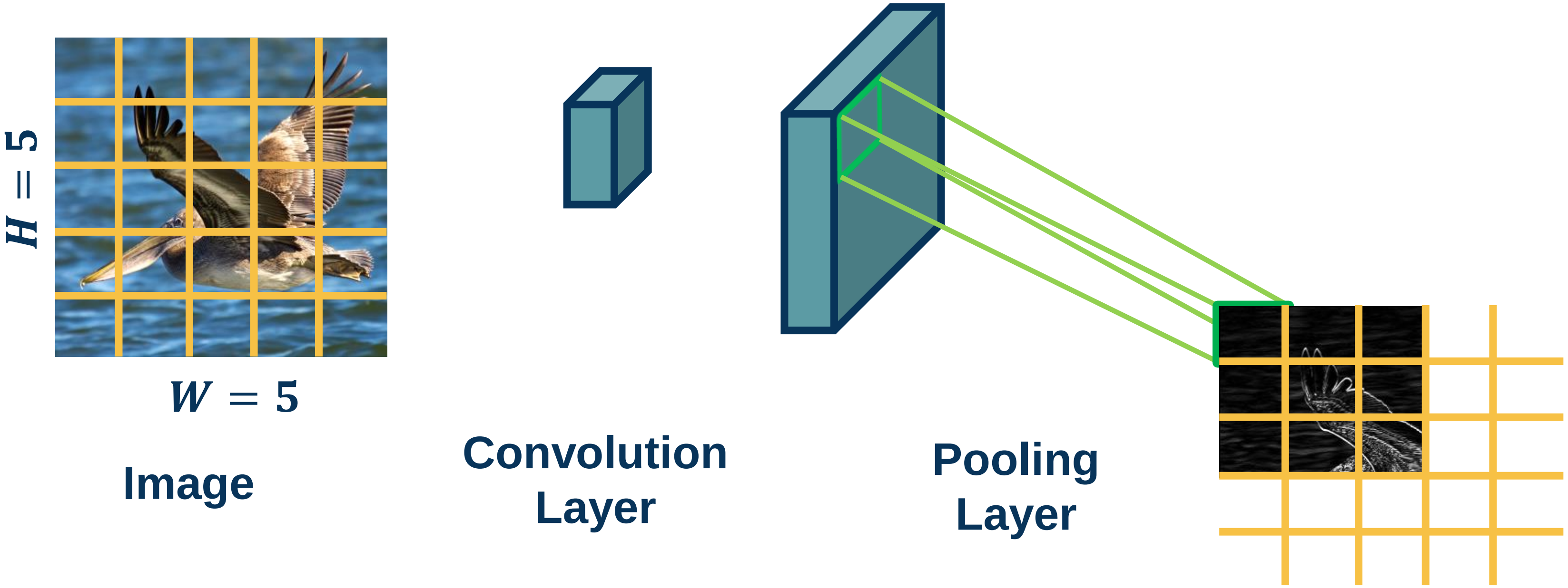
None!

- makes the representations spatially smaller
- saves computation (GPU mem & speed), allows go deeper
- operates over each activation map independently:



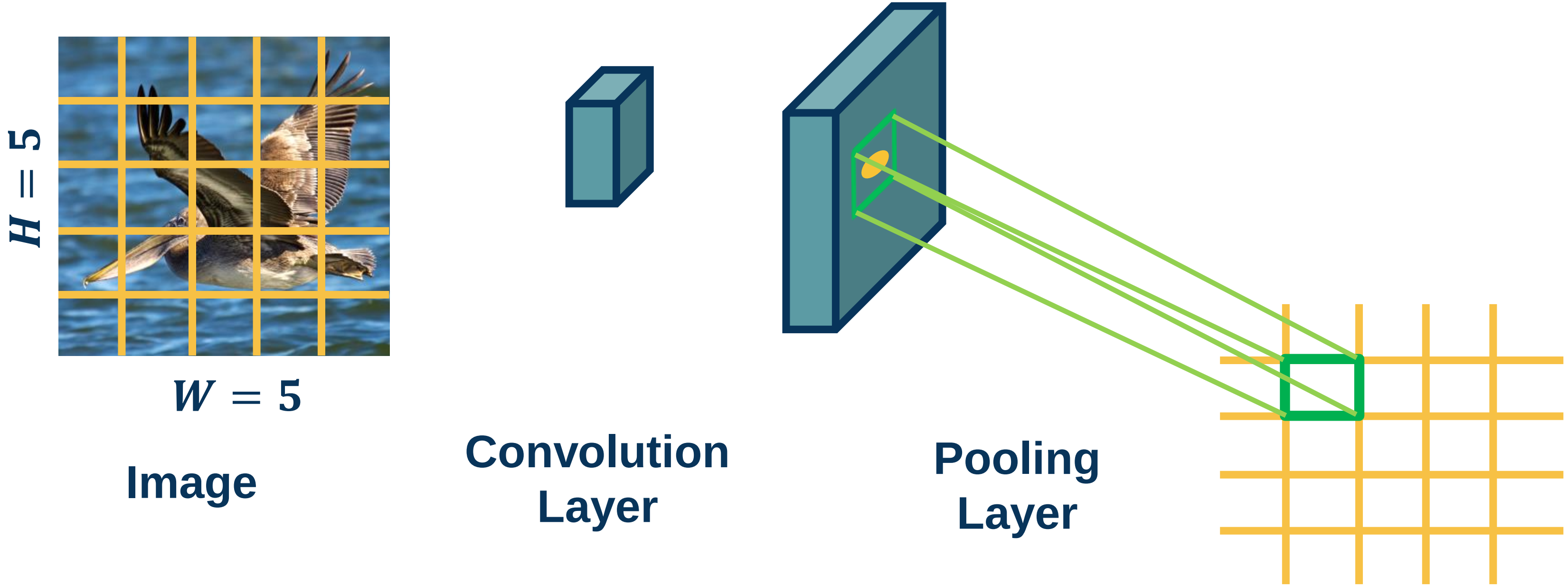
From: Slides by CS 231n, Danfei Xu

Since the **output** of convolution and pooling layers are **(multi-channel) images**, we can sequence them just as any other layer



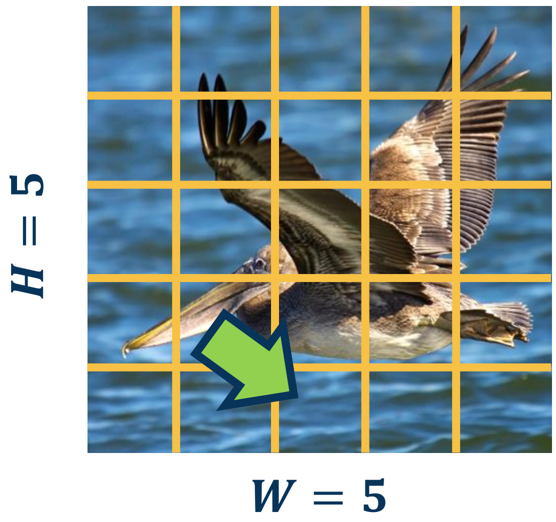
This combination adds some **invariance** to translation of the features

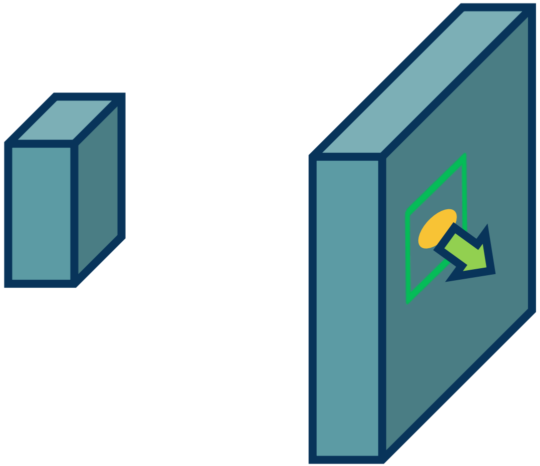
- If feature (such as beak) translated a little bit, output values still **remain the same**



Convolution by itself has the property of **equivariance**

- If feature (such as beak) translated a little bit, output values **move by the same translation**

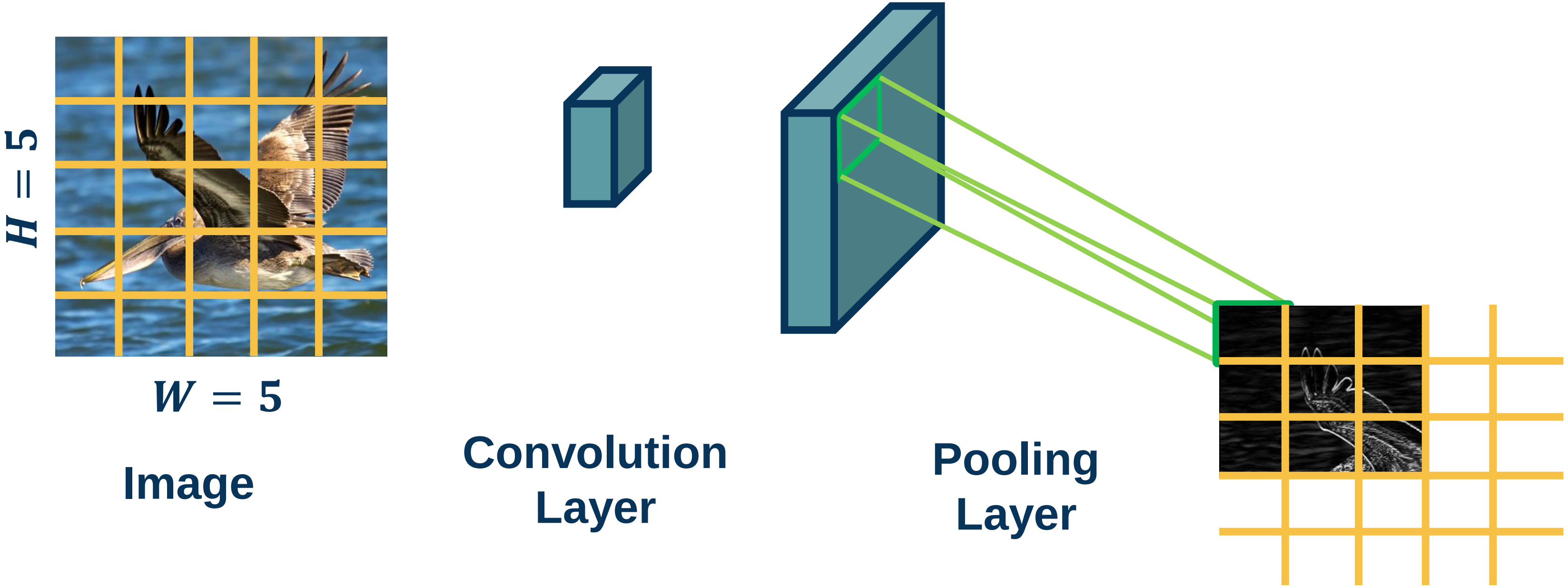




Invariance vs. Equivariance

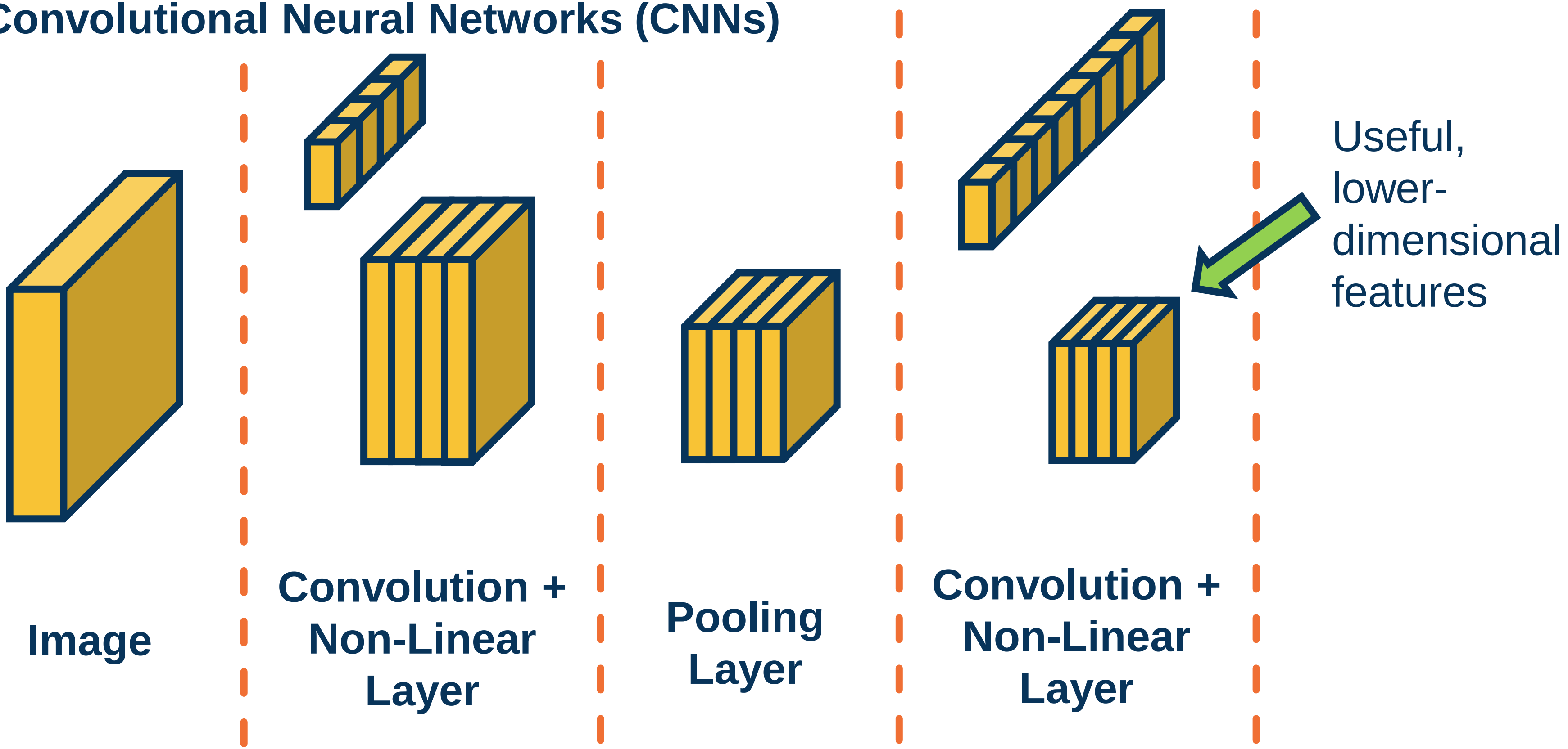
Simple Convolutional Neural Networks

Since the **output** of convolution and pooling layers are **(multi-channel) images**, we can sequence them just as any other layer

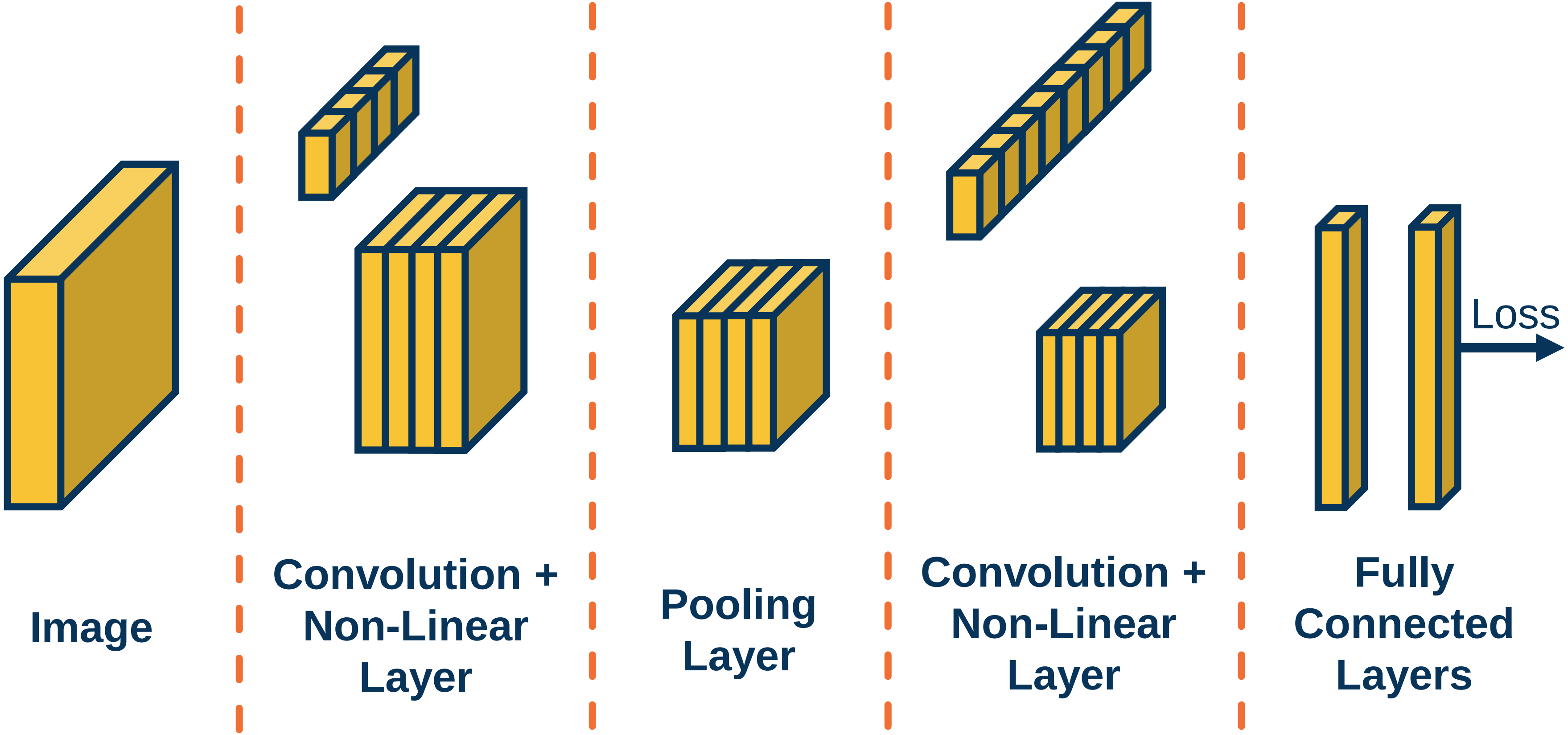


Combining Convolution & Pooling Layers

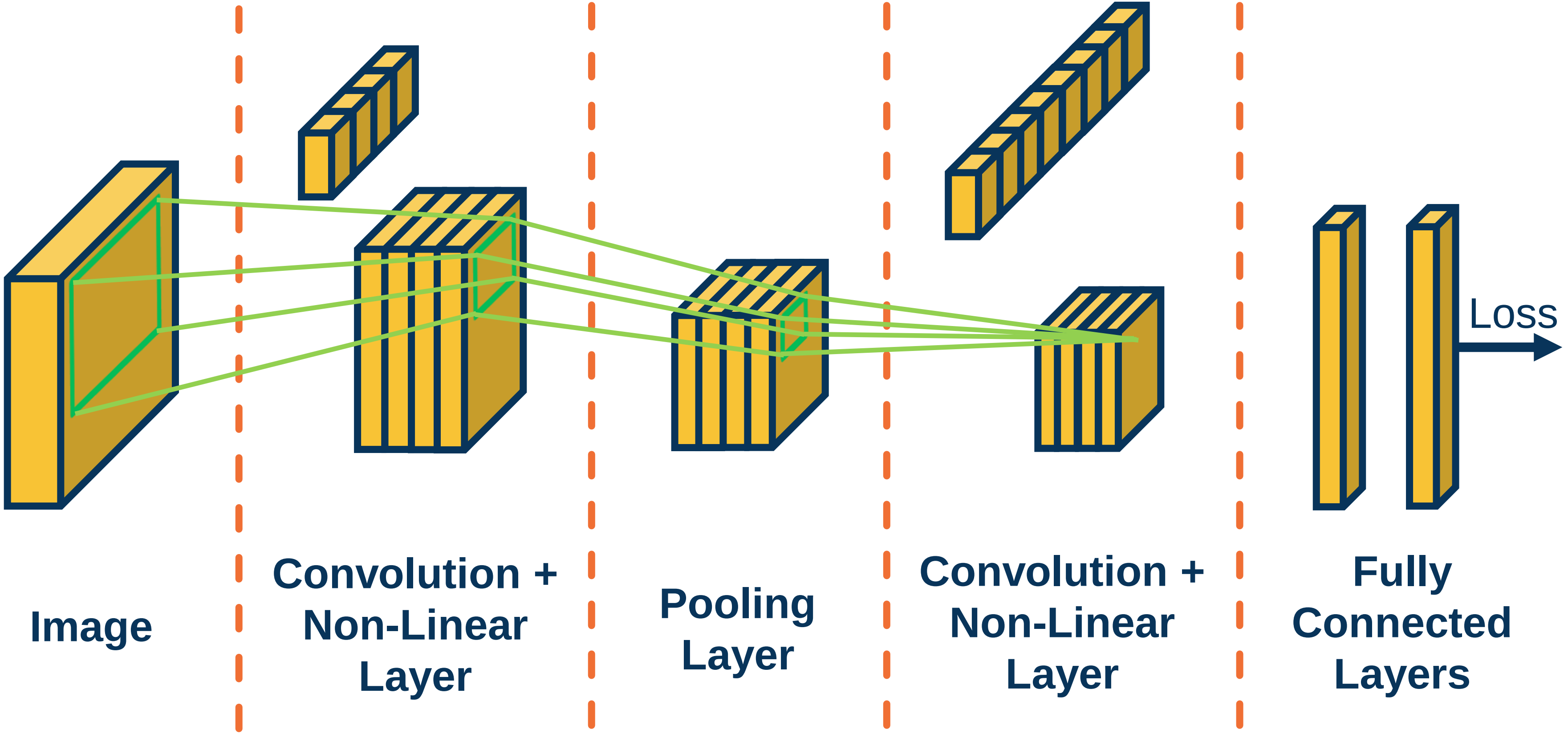
Convolutional Neural Networks (CNNs)



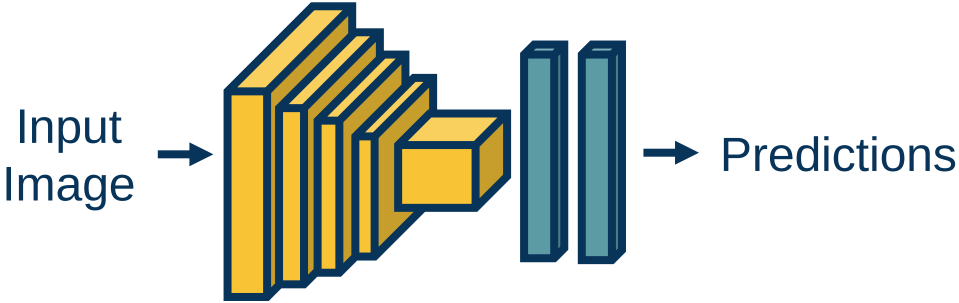
Alternating Convolution and Pooling



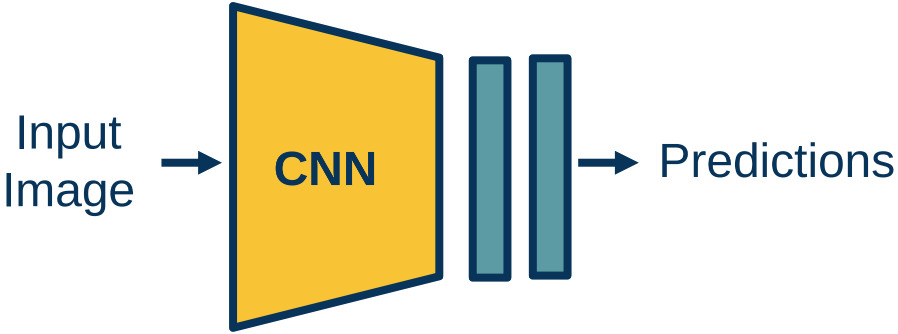
Adding a Fully Connected Layer



Receptive Fields



Convolutional Neural Networks



Typical Depiction of CNNs

These architectures have existed **since 1980s**

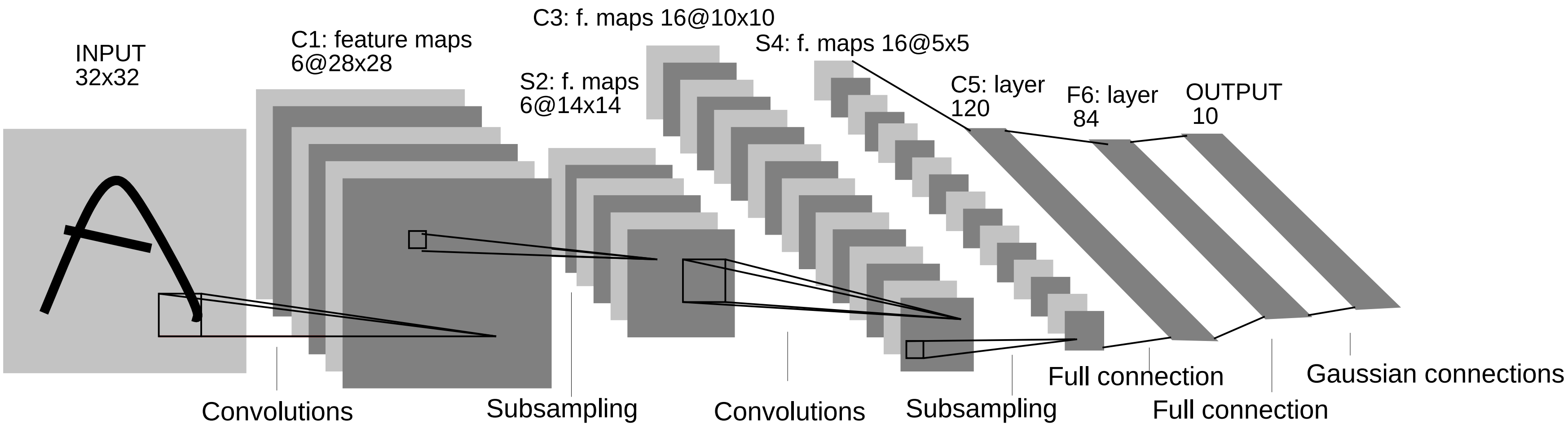


Image Credit: Yann LeCun, Kevin Murphy

Handwriting Recognition

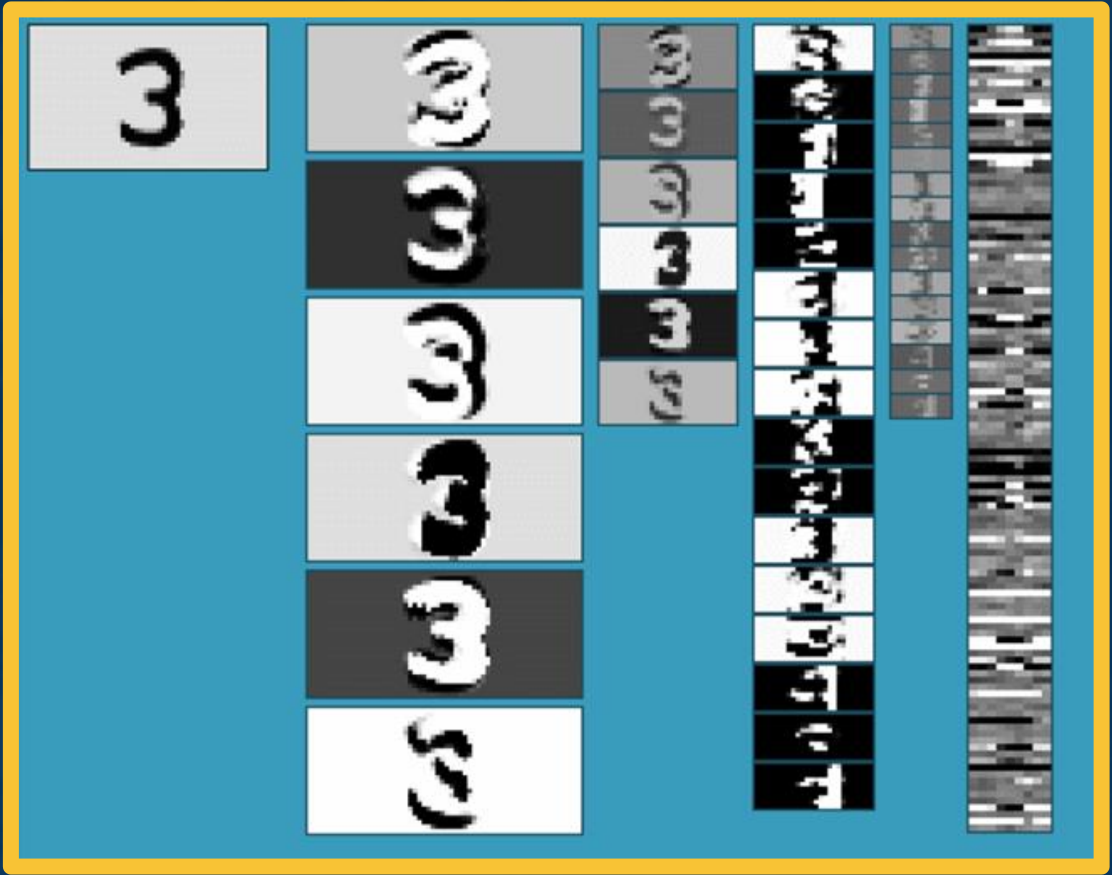


Image Credit: Yann LeCun

Translation Equivariance (Conv Layers) & Invariance (Output)

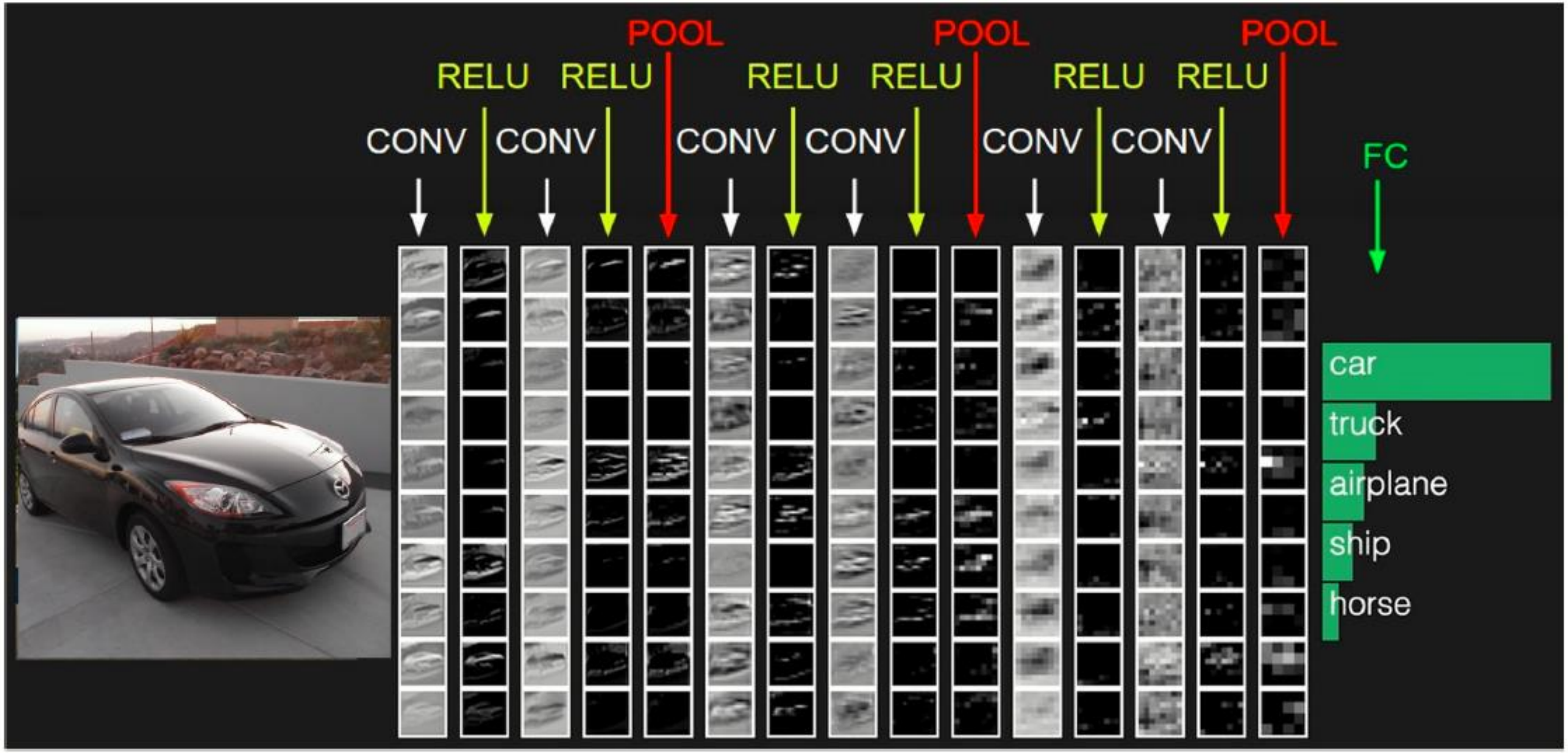
Image Credit: Yann LeCun

(Some) Rotation Invariance

Image Credit: Yann LeCun

(Some) Scale Invariance

Image Credit: Yann LeCun

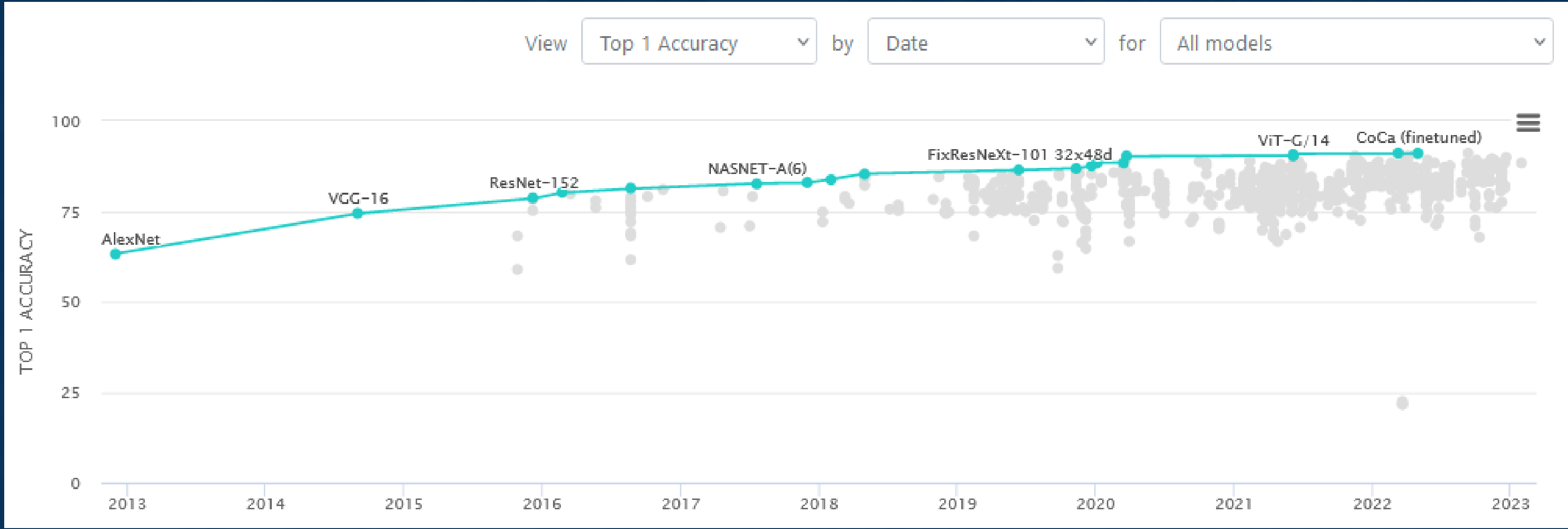


A More Modern Canonical CNN

Advanced Convolutional Networks

The **ImageNet** dataset contains 14,197,122 annotated images according to the WordNet hierarchy. ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a benchmark for image classification and object detection based on the dataset.

The Importance of Benchmarks

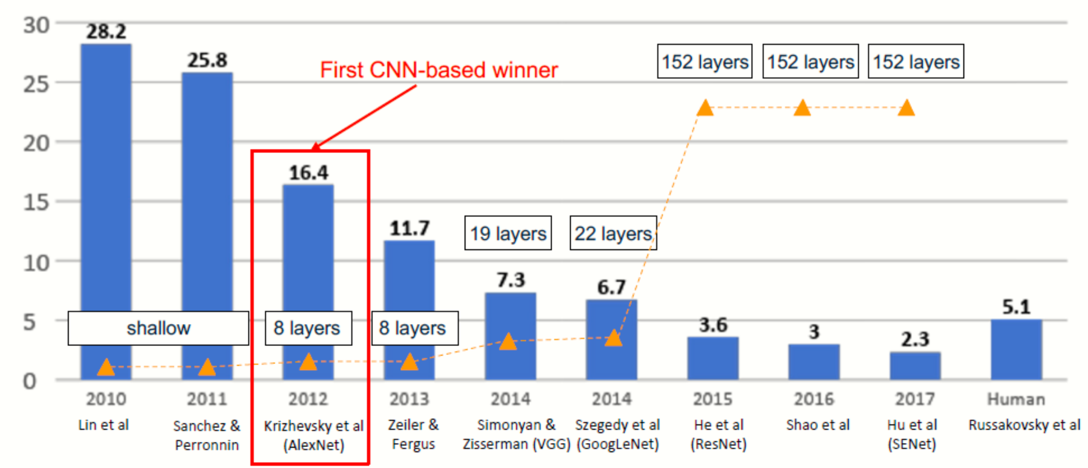


From: https://paperswithcode.com

Georgia Tech

Case Studies

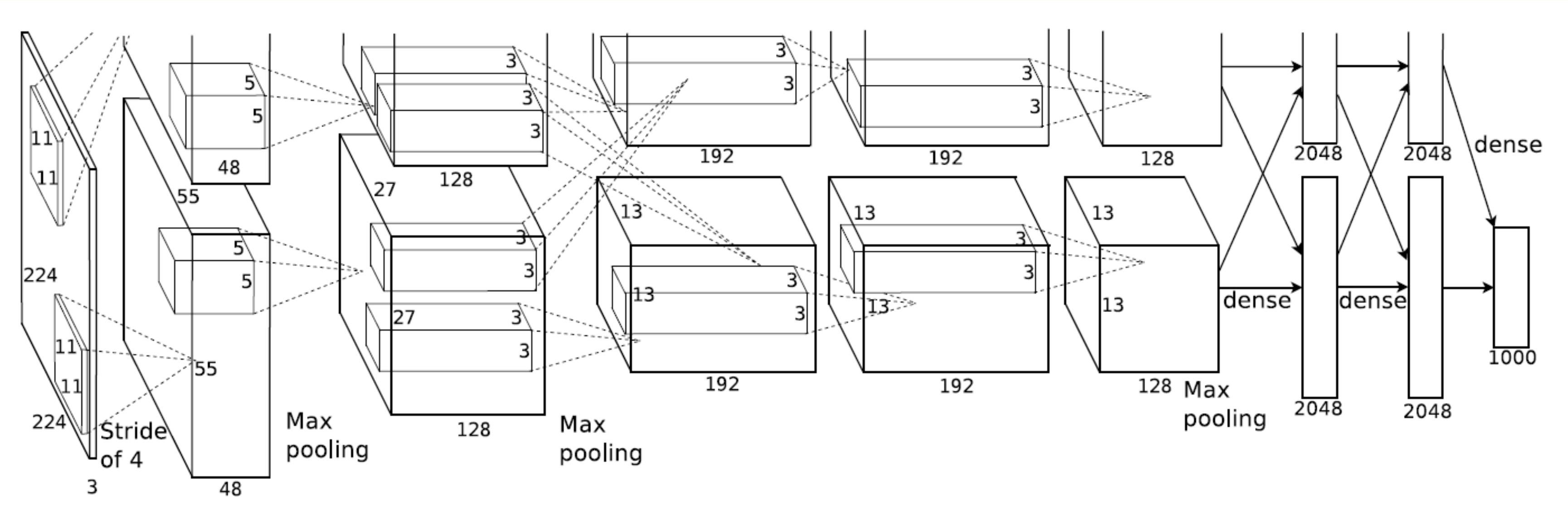
- AlexNet
- VGG
- GoogLeNet
- ResNet



Also....

- SENet
- Wide ResNet
- ResNeXT
- DenseNet
- MobileNets
- NASNet
- EfficientNet
- ConvNeXt v1/v2

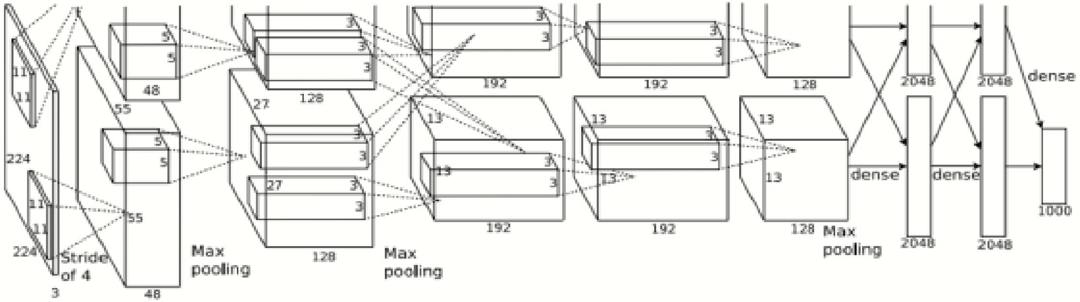
AlexNet - Architecture



From: Krizhevsky et al., ImageNet Classification with Deep ConvolutionalNeural Networks, 2012.

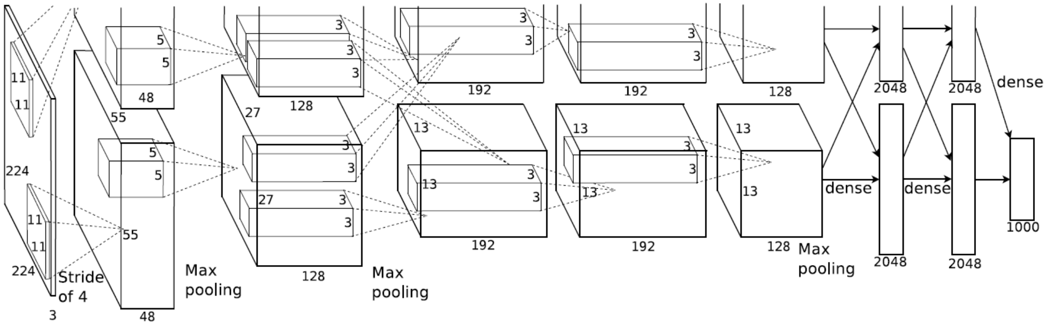
Case Study: AlexNet

[Krizhevsky et al. 2012]



Architecture:

CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

=>

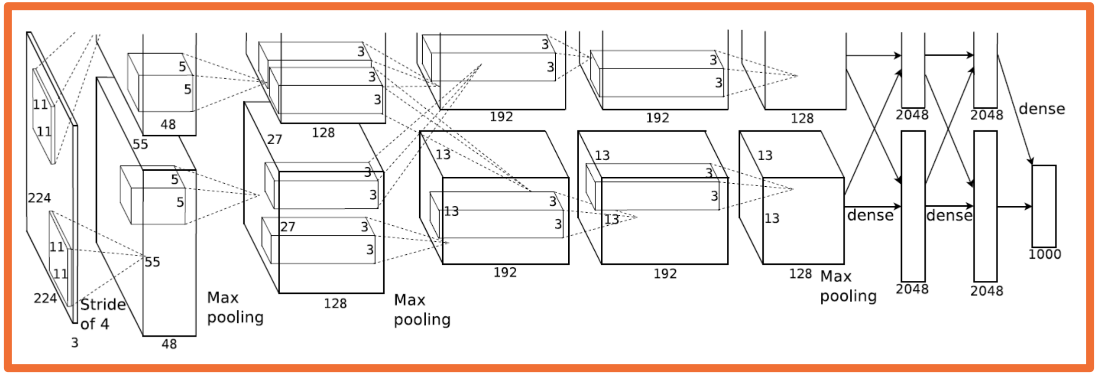
Q: what is the output volume size? Hint: (227-11)/4+1 = 55

W' = (W - F + 2P) / S + 1

From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

AlexNet – Layers and Key Aspects

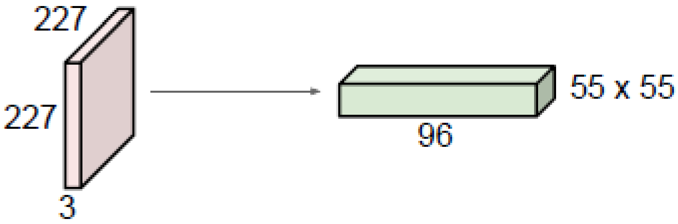
Georgia Tech



Input: 227x227x3 images

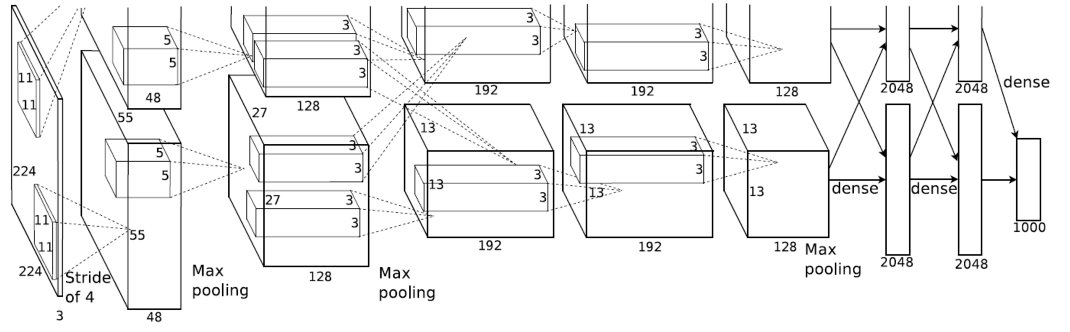
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

W' = (W - F + 2P) / S + 1



From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

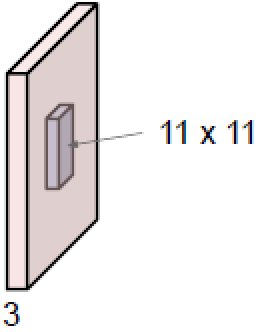
AlexNet – Layers and Key Aspects



Input: 227x227x3 images

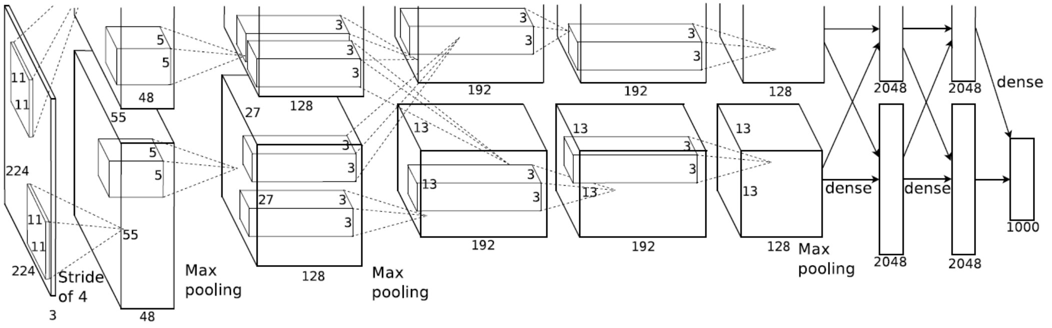
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?



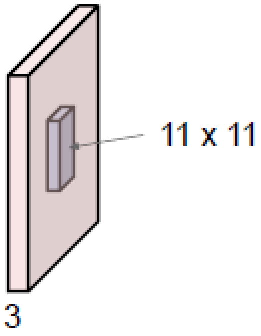
From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

AlexNet – Layers and Key Aspects



Input: 227x227x3 images

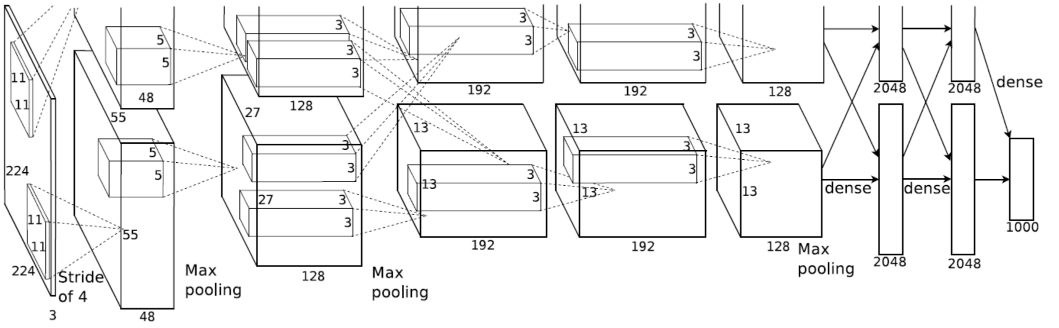
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3 + 1)*96 = **35K**



From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

AlexNet – Layers and Key Aspects

Full (simplified) AlexNet architecture:
[224x224x3] 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)



Key aspects:

- ReLU instead of sigmoid or tanh
- Specialized normalization layers
- PCA-based data augmentation
- Dropout
- Ensembling

From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

AlexNet – Layers and Key Aspects

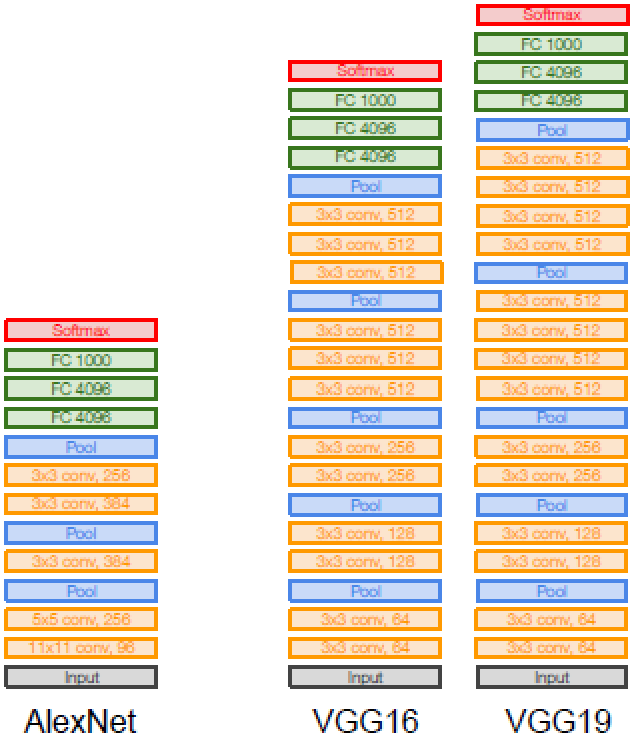
Georgia Tech

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13 (ZFNet)
-> 7.3% top 5 error in ILSVRC'14



From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

VGGNet

Q: Why use smaller filters? (3x3 conv)

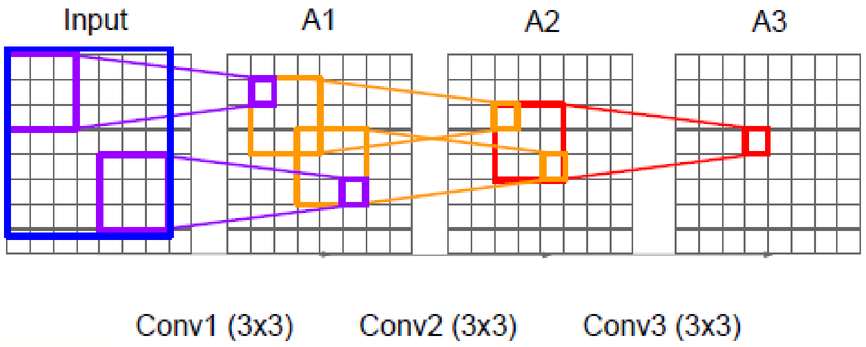
Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?

From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

VGGNet

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer

From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

VGGNet

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

From: Simonyan & Zimmerman, Very Deep Convolutional Networks for Large-Scale Image Recognition
From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

VGG

INPUT: [224x224x3]	memory: 224*224*3=150K	params: 0 (not counting biases)
CONV3-64: [224x224x64]	memory: 224*224*64=3.2M	params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]	memory: 224*224*64=3.2M	params: (3*3*64)*64 = 36,864
POOL2: [112x112x64]	memory: 112*112*64=800K	params: 0
CONV3-128: [112x112x128]	memory: 112*112*128=1.6M	params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]	memory: 112*112*128=1.6M	params: (3*3*128)*128 = 147,456
POOL2: [56x56x128]	memory: 56*56*128=400K	params: 0
CONV3-256: [56x56x256]	memory: 56*56*256=800K	params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824
POOL2: [28x28x256]	memory: 28*28*256=200K	params: 0
CONV3-512: [28x28x512]	memory: 28*28*512=400K	params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]	memory: 14*14*512=100K	params: 0
CONV3-512: [14x14x512]	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]	memory: 7*7*512=25K	params: 0
FC: [1x1x4096]	memory: 4096	params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]	memory: 4096	params: 4096*4096 = 16,777,216
FC: [1x1x1000]	memory: 1000	params: 4096*1000 = 4,096,000

Most memory usage in convolution layers

Most parameters in FC layers

From: Simonyan & Zimmerman, Very Deep Convolutional Networks for Large-Scale Image Recognition
From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Parameters and Memory

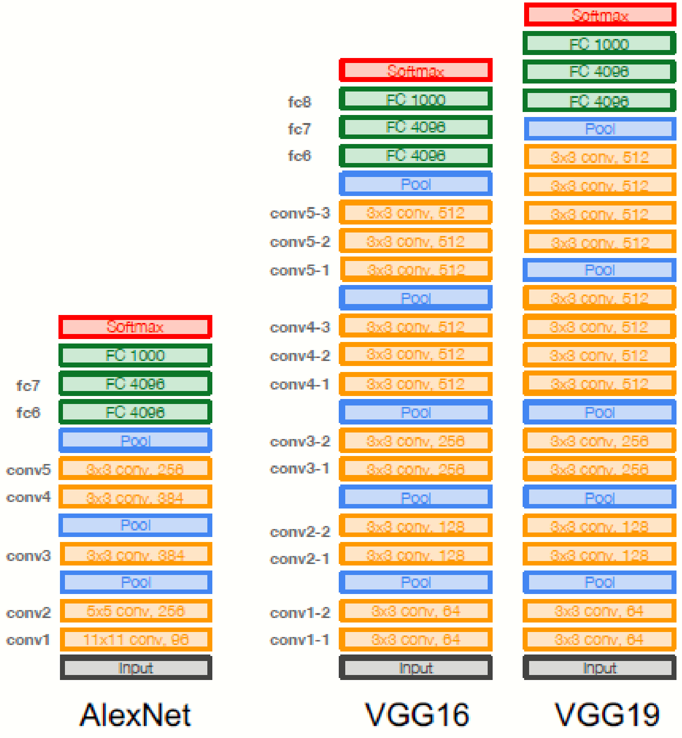
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

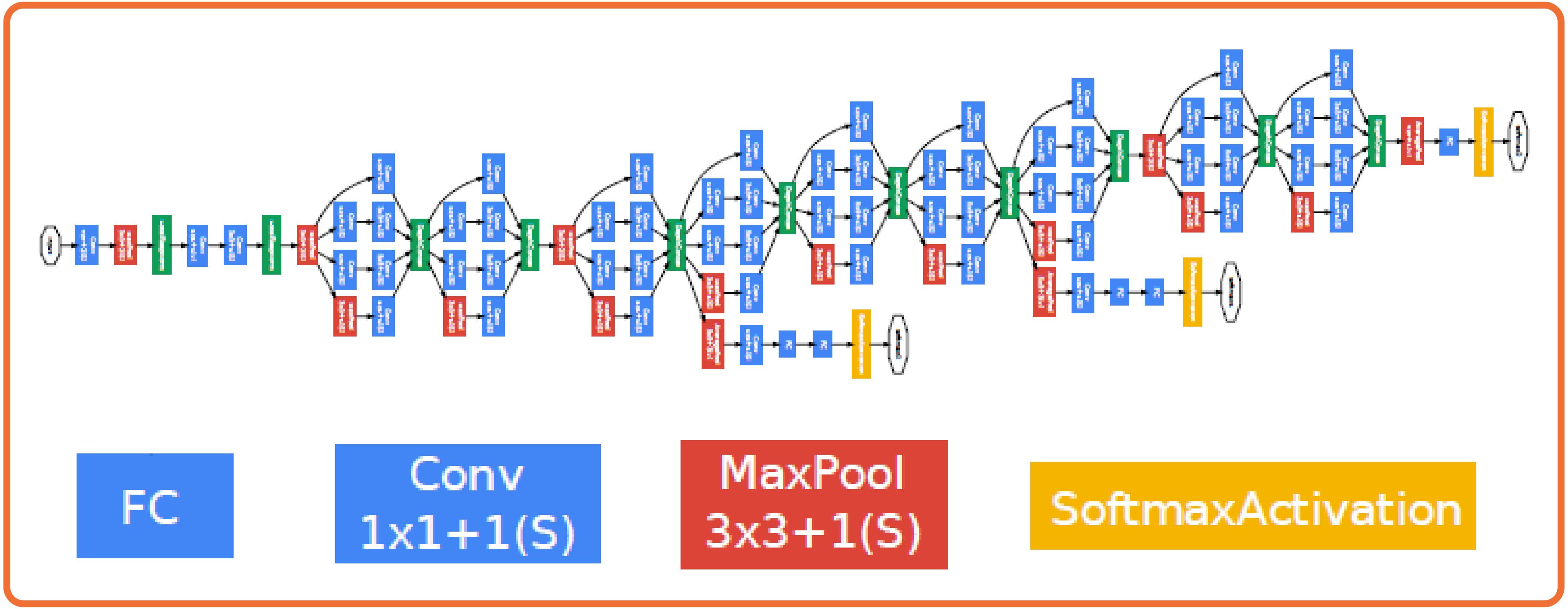
Details:

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks

Still very expensive!
TOTAL memory: 24M * 4 bytes ~= 96MB / image
(only forward! ~*2 for bwd)
TOTAL params: 138M parameters



But have become **deeper and more complex**



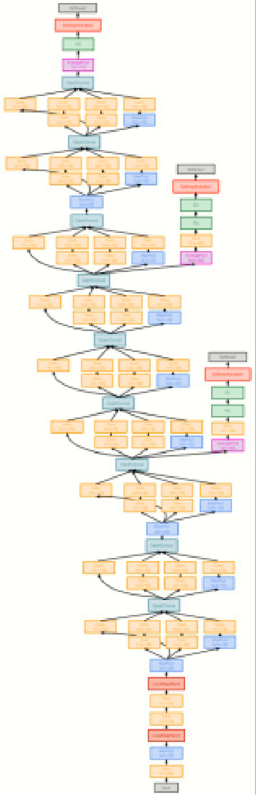
From: Szegedy et al. Going deeper with convolutions

Case Study: GoogLeNet

[Szegedy et al., 2014]

Deeper networks, focus on computational efficiency

- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!
  12x less than AlexNet
  27x less than VGG-16
- Efficient "Inception" module
- No FC layers



From: Szegedy et al. Going deeper with convolutions

Case Study: GoogLeNet

[Szegedy et al., 2014]

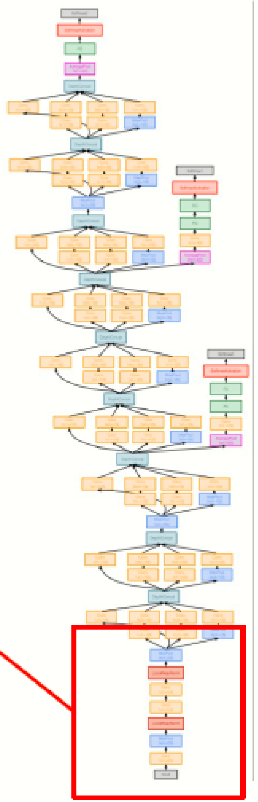
Deeper networks, focus on computational efficiency

- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!
  12x less than AlexNet
  27x less than VGG-16
- Efficient "Inception" module
- No FC layers

Stem Network: aggressively reduce the input feature volume

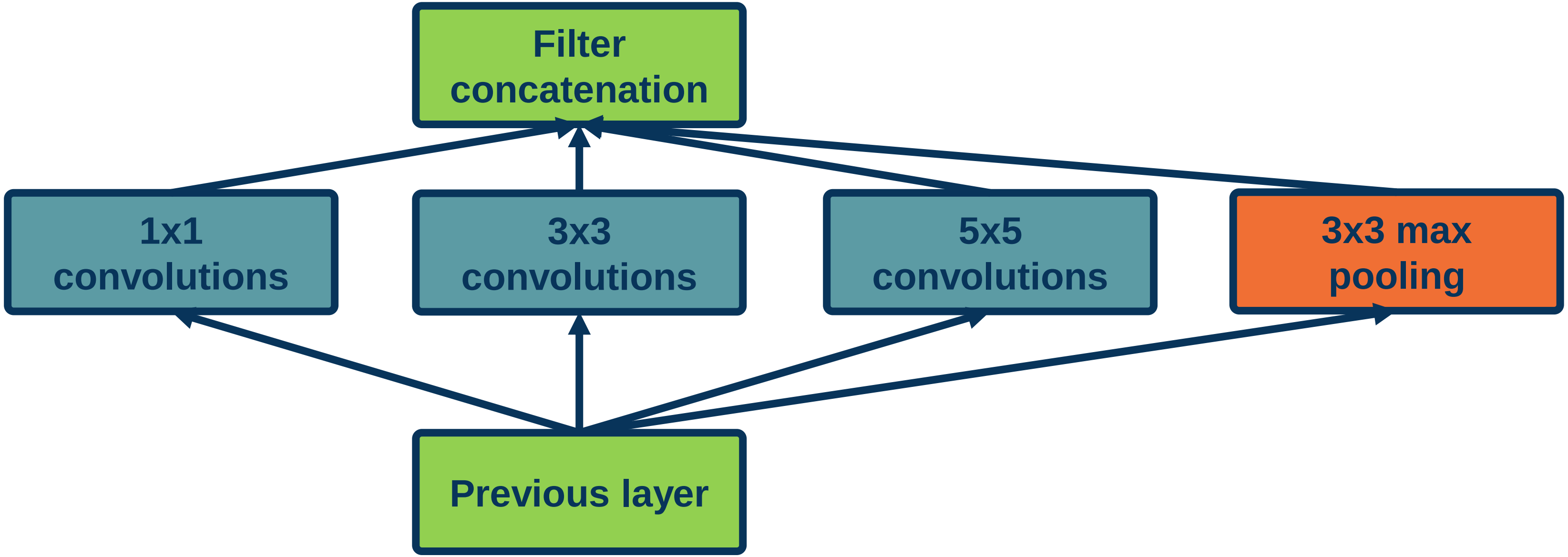
- Conv 7 x 7 x 64 with stride 2
- MaxPool
- Conv 1 x 1 x 64
- Conv 3 x 3 x 192
- MaxPool

Reduce 224 x 224 spatial solution to 28 x 28 with just 418 MFLOP! (Comparing to 7485 MFLOP of VGG)



From: Szegedy et al. Going deeper with convolutions

Key idea: Repeated blocks and multi-scale features



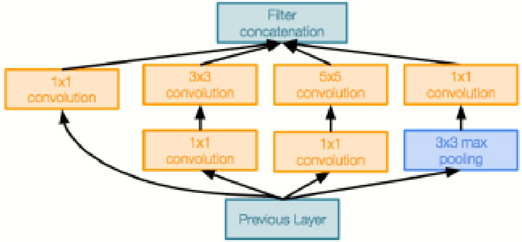
From: Szegedy et al. Going deeper with convolutions

Case Study: GoogLeNet

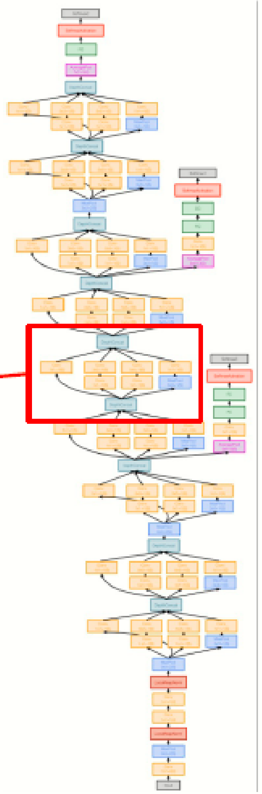
[Szegedy et al., 2014]

"Inception module": design a good local network topology (network within a network) and then stack these modules on top of each other

Multiple conv filter size diversifies learned features

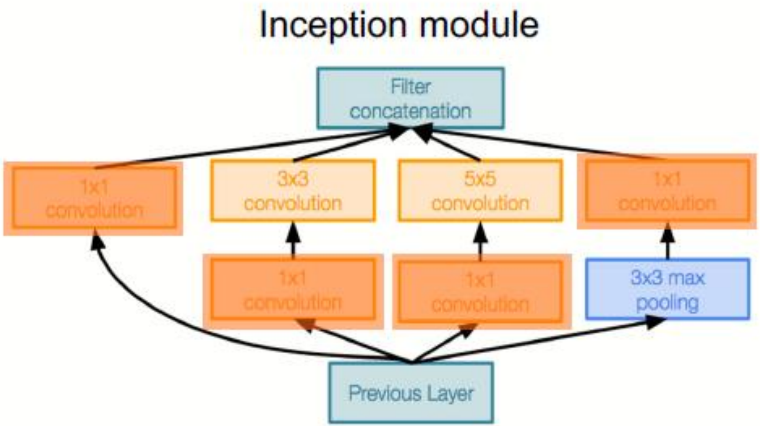


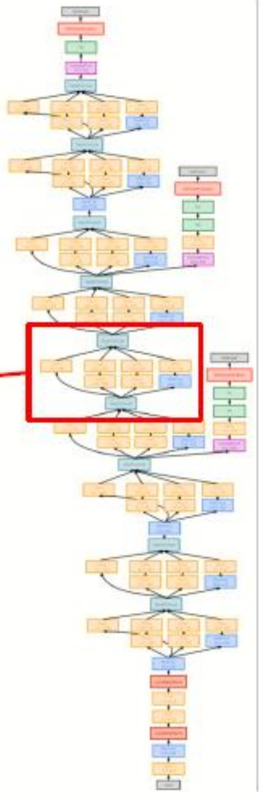
Inception module



Case Study: GoogLeNet

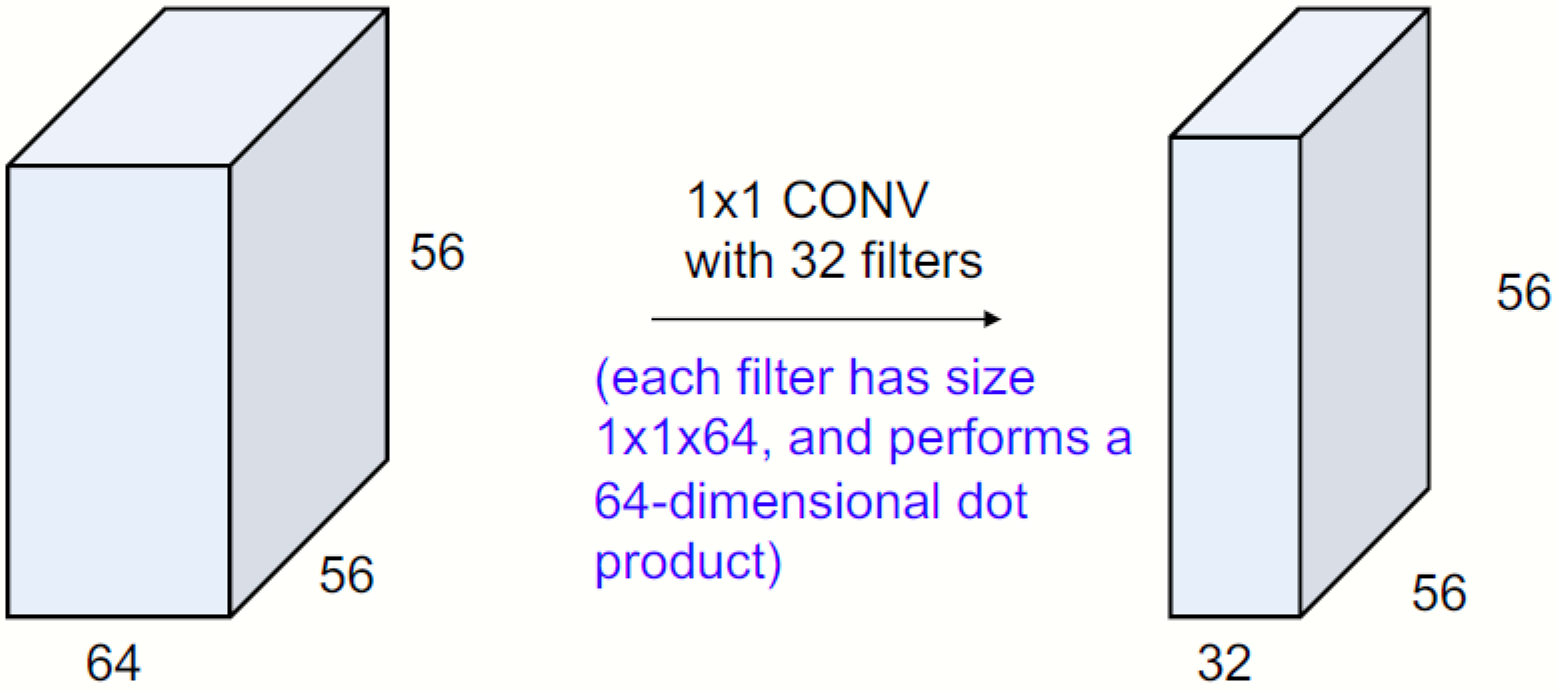
[Szegedy et al., 2014]





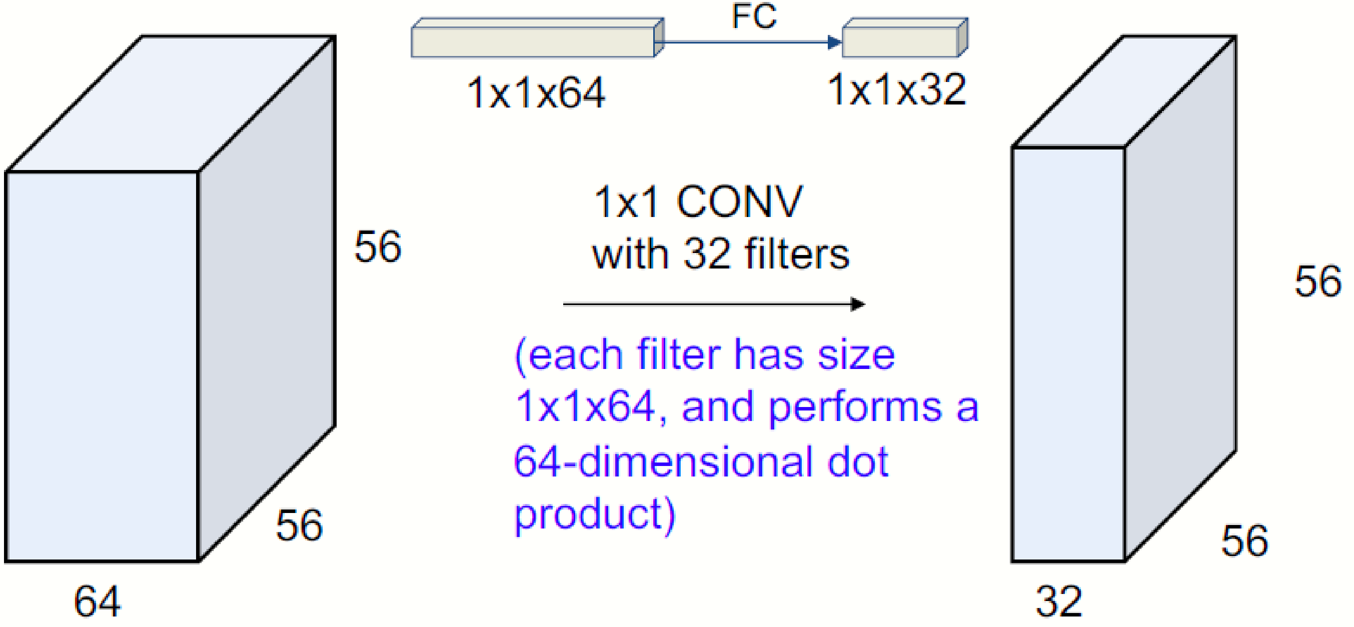
Uses 1x1 "Bottleneck" layers to reduce channel dimension before expensive conv (we will revisit this with ResNet!)

Inception Module



1x1 Convolutions

Alternatively, interpret it as applying the same FC layer on each input pixel



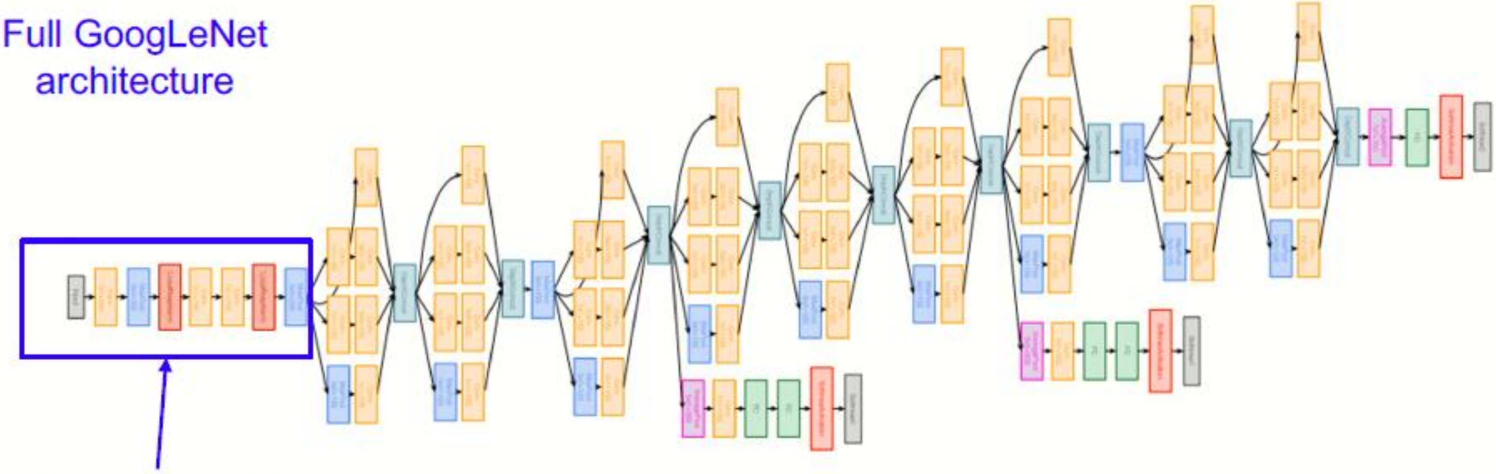
(each filter has size 1x1x64, and performs a 64-dimensional dot product)

1x1 Convolutions

Case Study: GoogLeNet

[Szegedy et al., 2014]

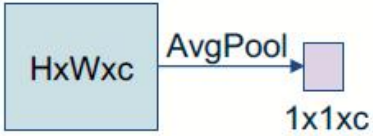
Full GoogLeNet architecture



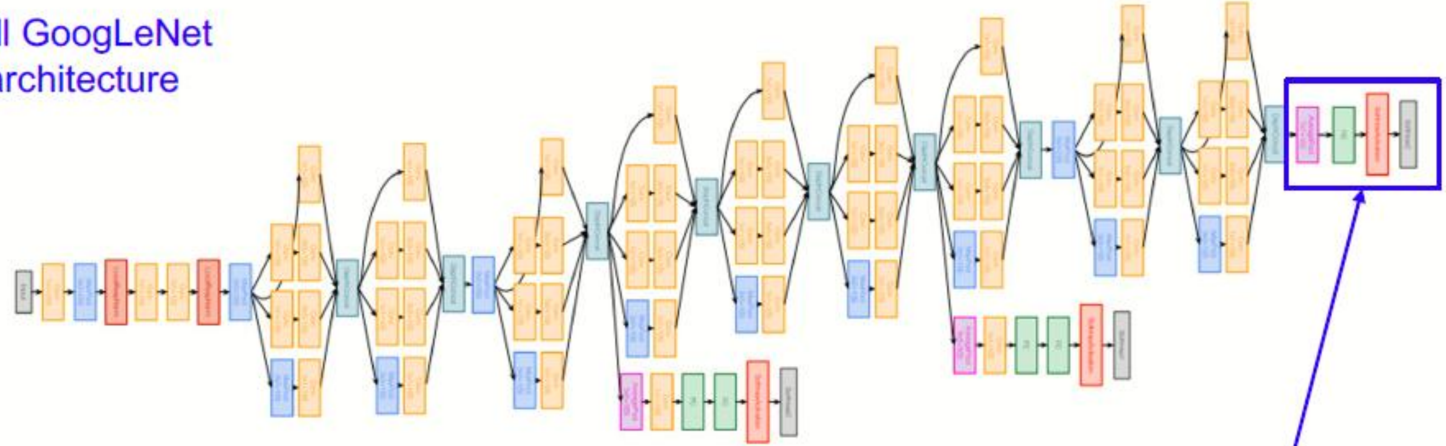
Stem Network: Conv-Pool- 2x Conv-Pool

Case Study: GoogLeNet

[Szegedy et al., 2014]



Full GoogLeNet architecture



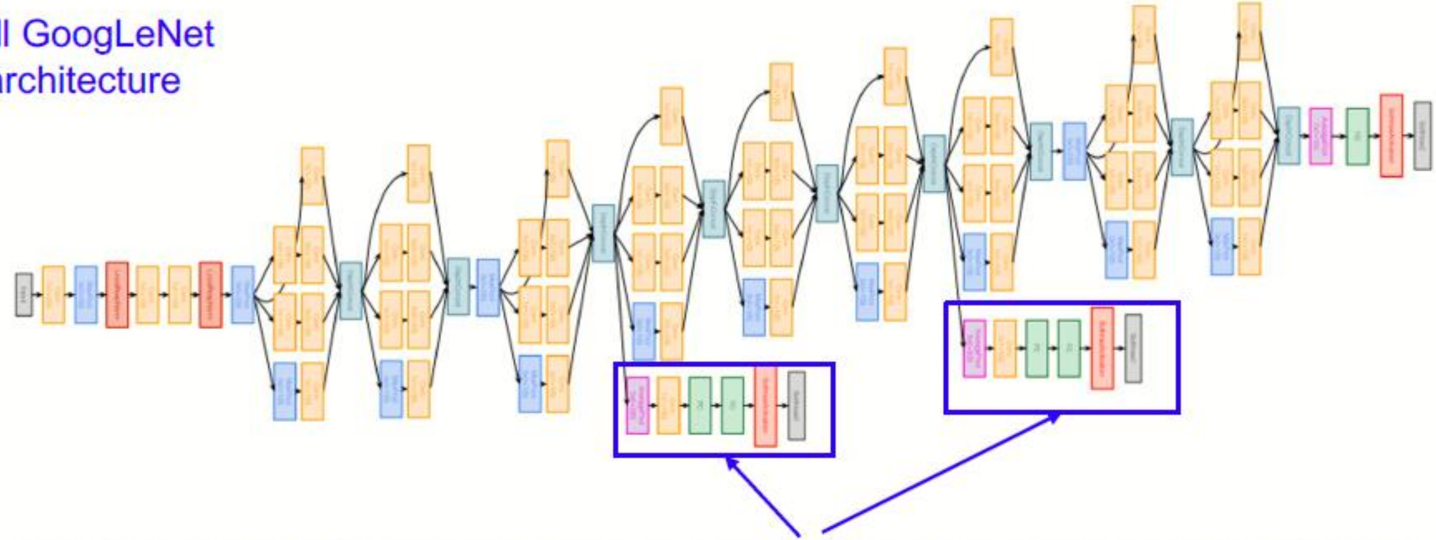
Note: after the last convolutional layer, a global average pooling layer is used that **spatially** averages across each feature map, before final FC layer. No longer multiple expensive FC layers!
(Also used in ResNet)

Classifier output

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



Auxiliary classification outputs to inject additional gradient at lower layers (AvgPool-1x1Conv-FC-FC-Softmax)

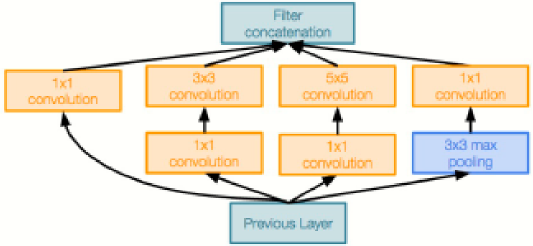
Why?

Case Study: GoogLeNet

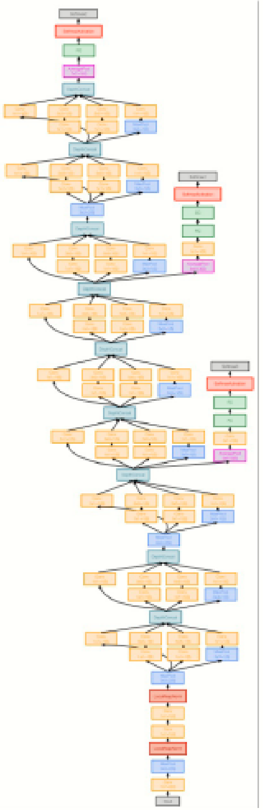
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- Avoids expensive FC layers
- 12x less params than AlexNet
- 27x less params than VGG-16
- ILSVRC’14 classification winner (6.7% top 5 error)



Inception module



1x1 Convolutions

- Convolutional neural networks (CNNs) stack pooling, convolution, non-linearities, and fully connected (FC) layers

- Feature engineering => architecture engineering!
  - Tons of small details and tips/tricks
  - Considerations: Memory, compute/FLO, dimensionality reduction, diversity of features, number of parameters/capacity, etc.

Summary

Georgia Tech