Topics:

- Convolutional Neural Networks

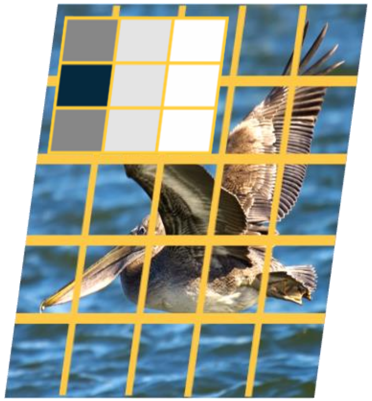
CS 4644-DL / 7643-A
ZSOLT KIRA

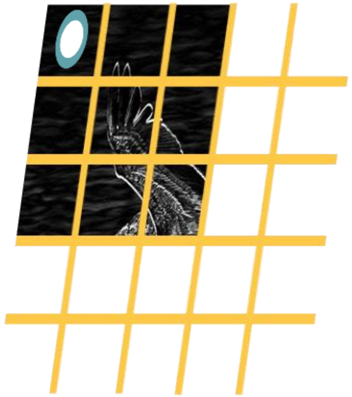
- **Assignment 2**
  - **Due 02/22**
  - Implement convolutional neural networks
  - Resources (in addition to lectures):
    - DL book: Convolutional Networks
    - CNN notes https://www.cc.gatech.edu/classes/AY2022/cs7643_spring/assets/L10_cnns_notes.pdf
    - Backprop notes https://www.cc.gatech.edu/classes/AY2023/cs7643_spring/assets/L10_cnns_backprop_notes.pdf
    - Slower OMSCS lectures on dropbox: Module 2 Lessons 5-6 (M2L5/M2L6) (https://www.dropbox.com/sh/iviro188gq0b4vs/AADdHxX_Uy1TkpF_yvIzX0nPa?dl=0)

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

$$K' = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \Rightarrow X(0:2,0:2) \cdot K' = 65 + \text{bias}$$

Dot product
(element-wise multiply and sum)



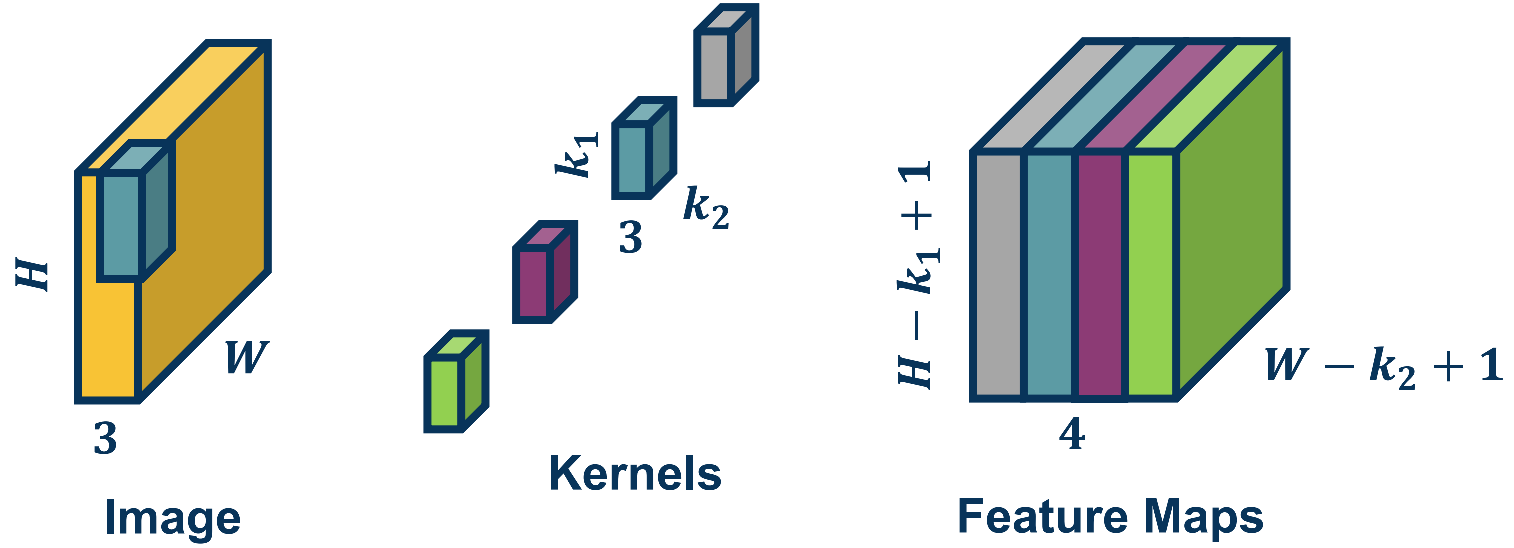


$$\boldsymbol{y(r,c) = (x * k)(r,c) = \sum_{a=0}^{k_1-1} \sum_{b=0}^{k_2-1} x(r+a, c+b)\, k(a,b)}$$

Cross-Correlation

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

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



Number of Parameters

Gradient for Convolution Layer

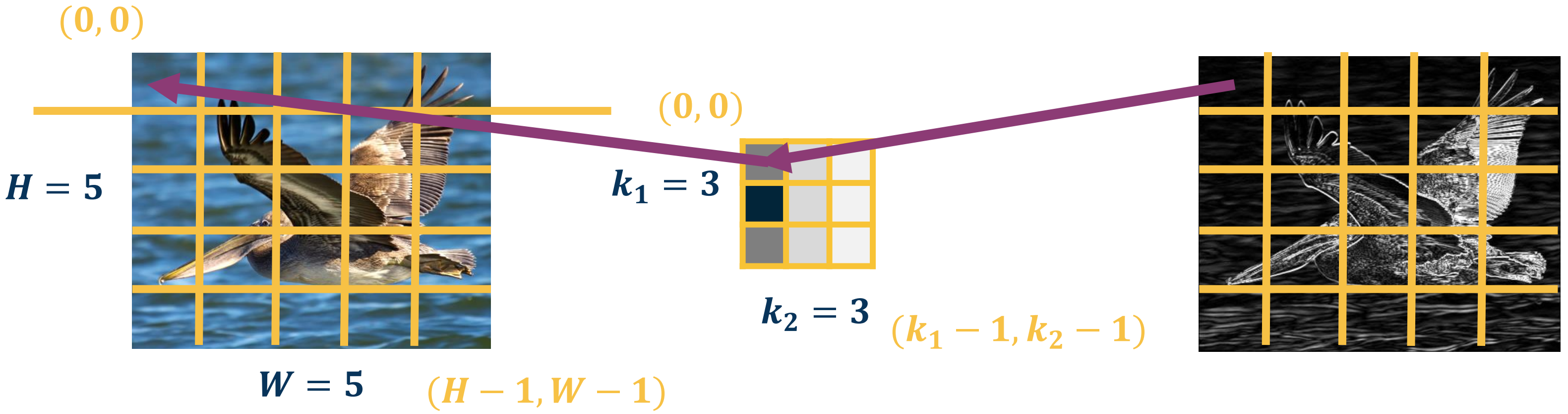
$$\frac{\partial L}{\partial k} = \frac{\partial L}{\partial h^{\ell}} \frac{\partial h^{\ell}}{\partial k}$$

Gradient for weight update

Calculate one pixel at a time $\frac{\partial L}{\partial k(a,b)} = \frac{\partial L}{\partial h^{\ell}} \frac{\partial h^{\ell}}{\partial k(a,b)}$

What does this weight affect at the output?

Everything!



What a Kernel Pixel Affects at Output

Need to incorporate all upstream gradients:

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

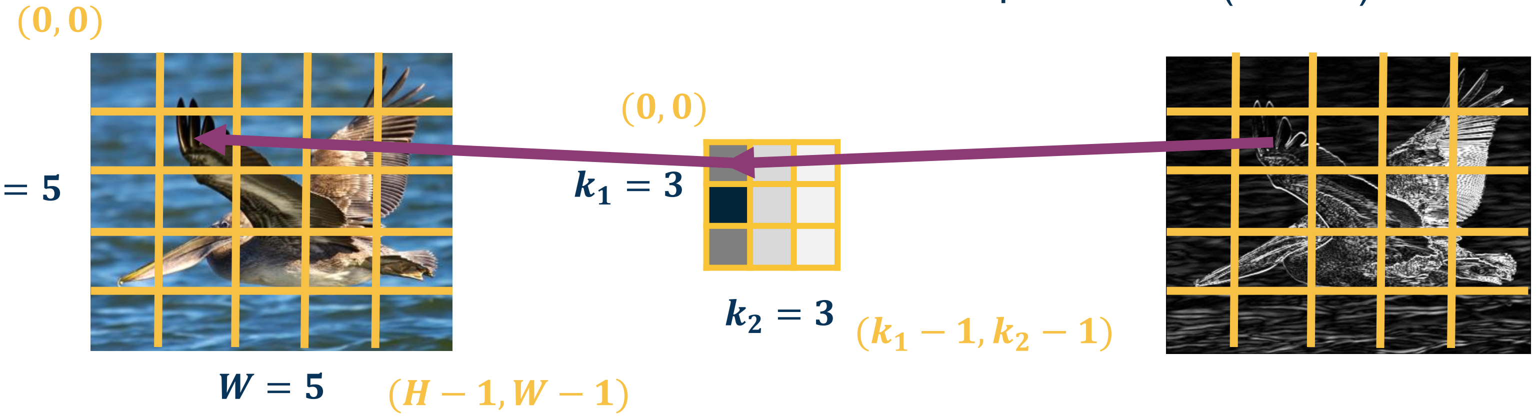
Chain Rule:

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

Sum over all output pixels

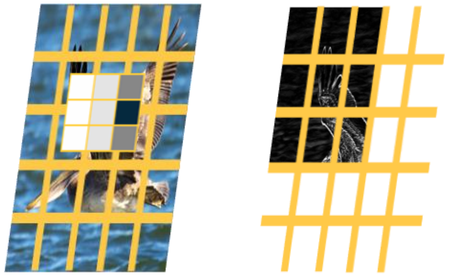
Upstream gradient (known)

We will compute



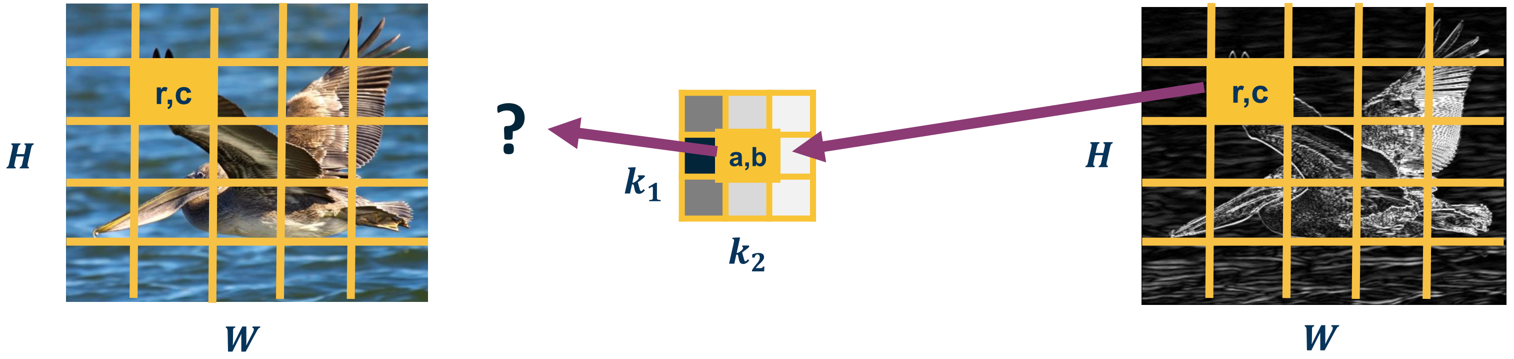
Chain Rule over all Output Pixels

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



Reasoning:

- Cross-correlation is just "dot product" of kernel and input patch (weighted sum)
- When at pixel $y(r,c)$, kernel is on input *x* such that $k(0,0)$ is multiplied by x(r,c)
- But we want derivative w.r.t. $k(a,b)$
    - $k(0,0) * x(r,c)$, $k(1,1) * x(r+1,c+1)$, $k(2,2) * x(r+2,c+2)$
    - => in general $k(a,b) * x(r+a,c+b)$
    - Just like before in fully connected layer, partial derivative w.r.t. $k(a,b)$ *only* has this term (other *x* terms go away because not multiplied by $k(a,b)$).



Chain Rule over all Output Pixels

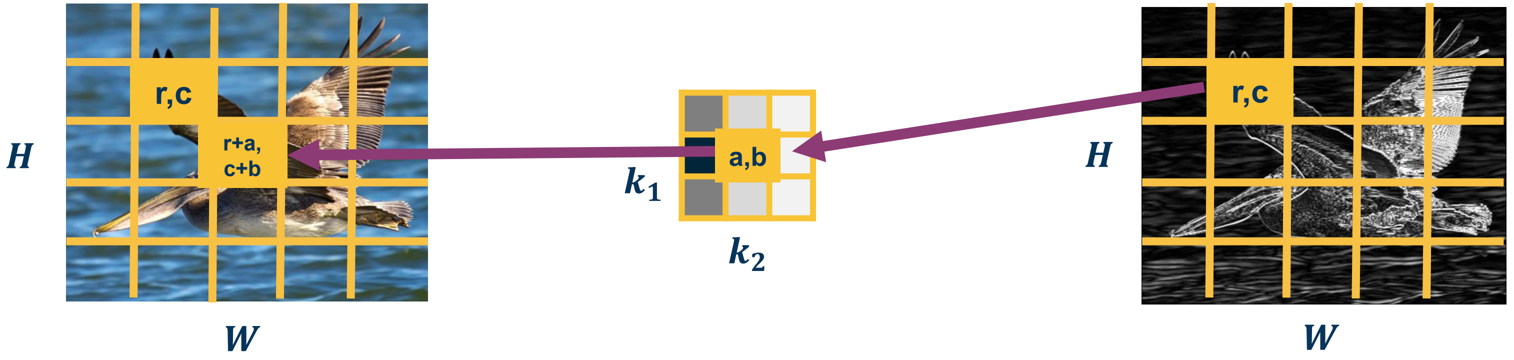
$$\frac{\partial y(r,c)}{\partial k(a,b)} = x(r+a, c+b)$$

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

Does this look familiar?

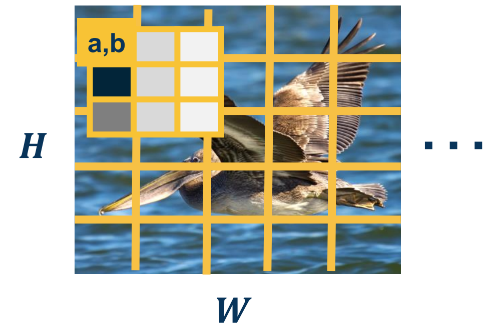
Cross-correlation between upstream gradient and input!

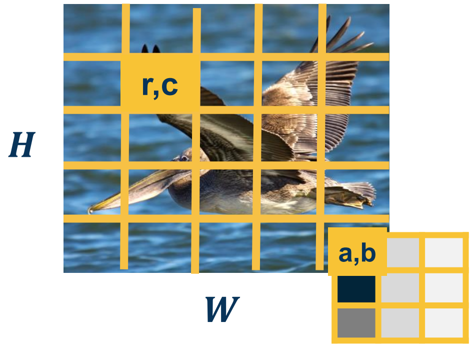
(until $k_1 \times k_2$ output)



Gradients and Cross-Correlation

Forward Pass



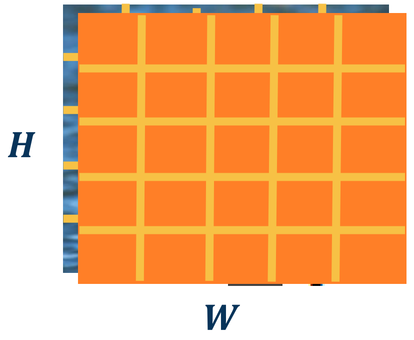


Does this look familiar?

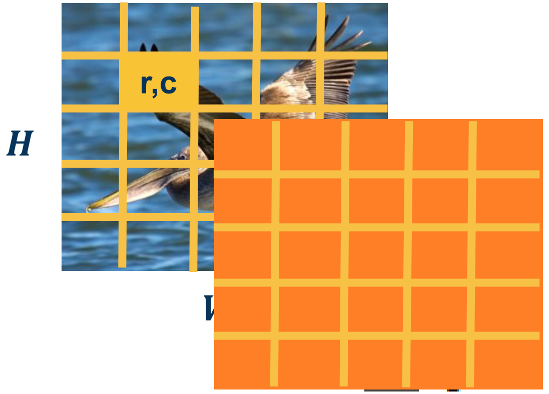
Cross-correlation between upstream gradient and input!

(until $k_1 \times k_2$ output)

Backward Pass k$(0, 0)$



Backward Pass $k(2, 2)$



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

Forward and Backward Duality

Georgia Tech

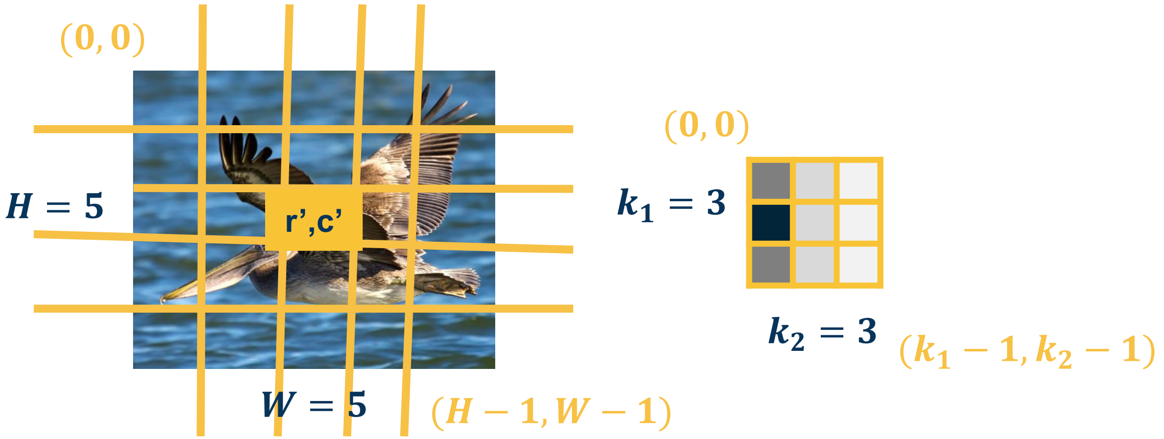
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial x}$$

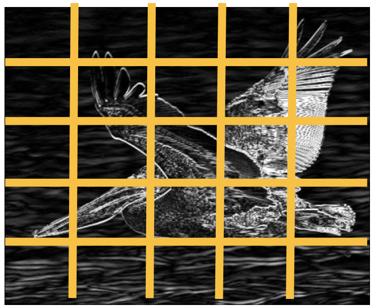
Gradient for input (to pass to prior layer)

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

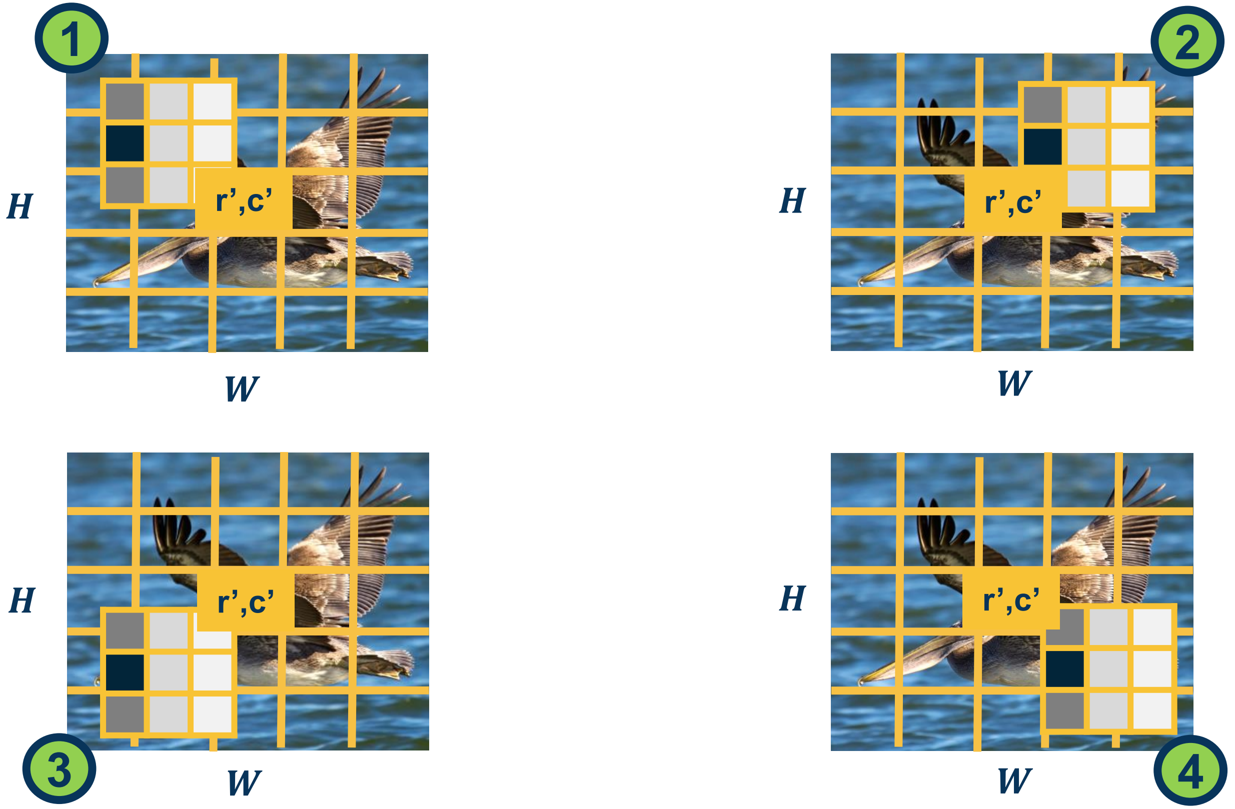
What does this input pixel affect at the output?

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

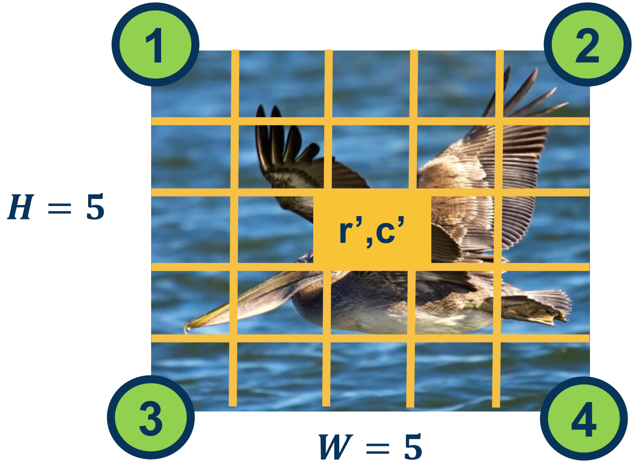


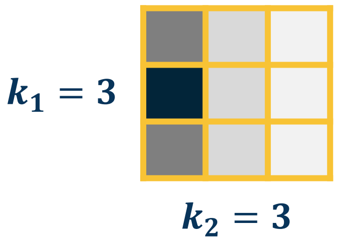


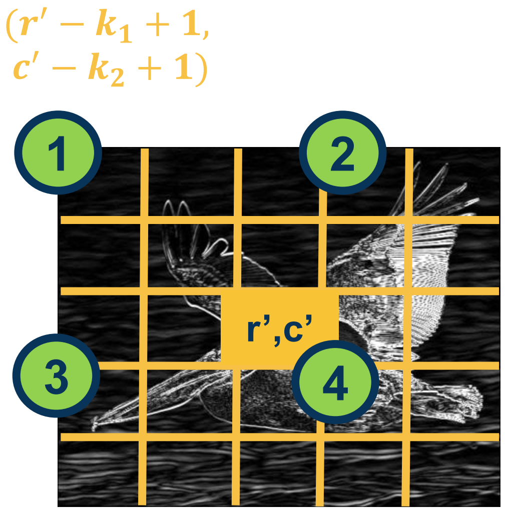
What an Input Pixel Affects at Output



Extents of Kernel Touching the Pixel







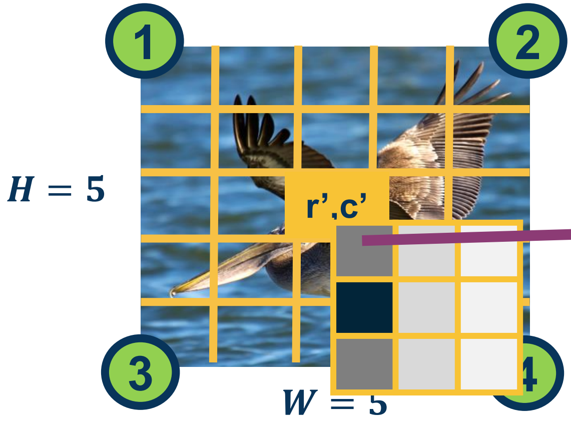
This is where the corresponding locations are for the **output**

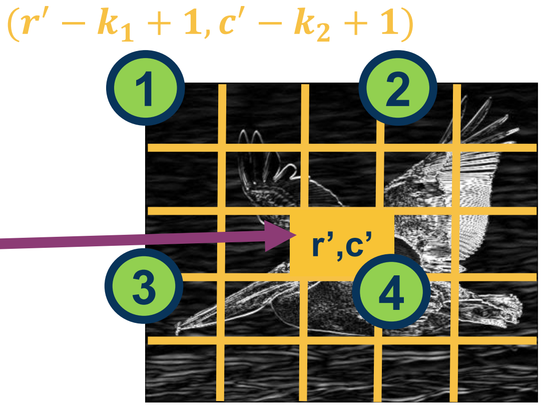
Extents at the Output

Chain rule for affected pixels (sum gradients):

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

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





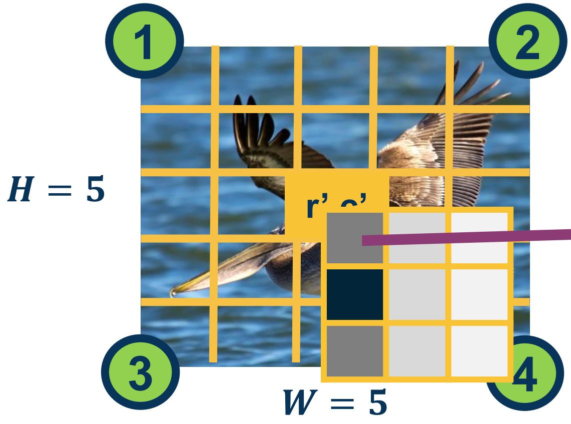
Summing Gradient Contributions

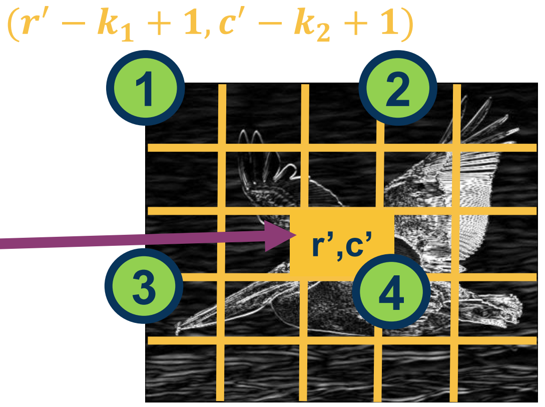
Chain rule for affected pixels (sum gradients):

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

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

$x(r',c') * k(0,0) \Rightarrow y(r',c')$
$x(r',c') * k(1,1) \Rightarrow ?$





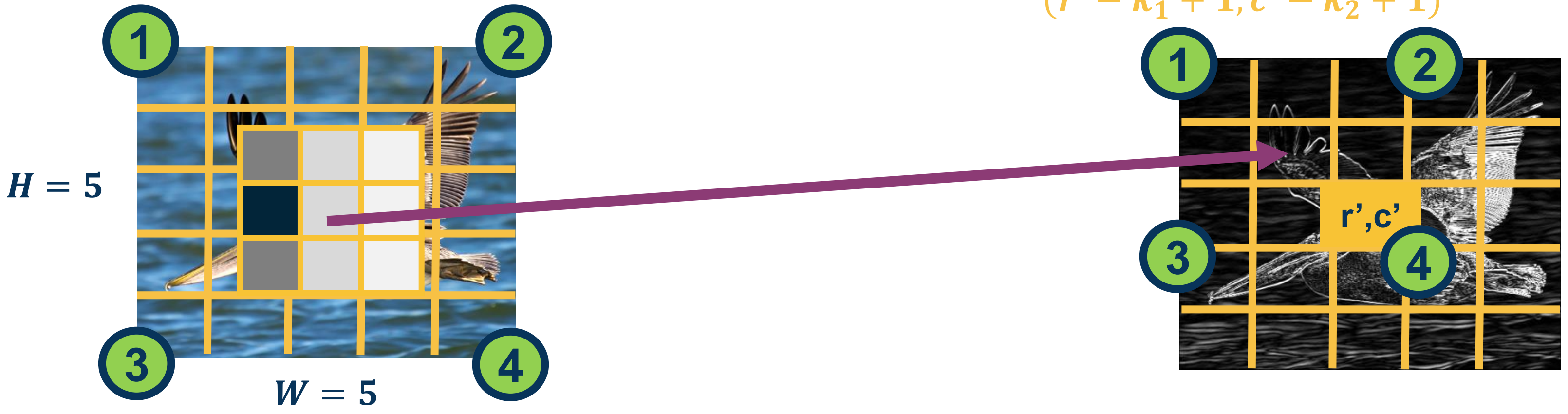
Summing Gradient Contributions

Chain rule for affected pixels (sum gradients):

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

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

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



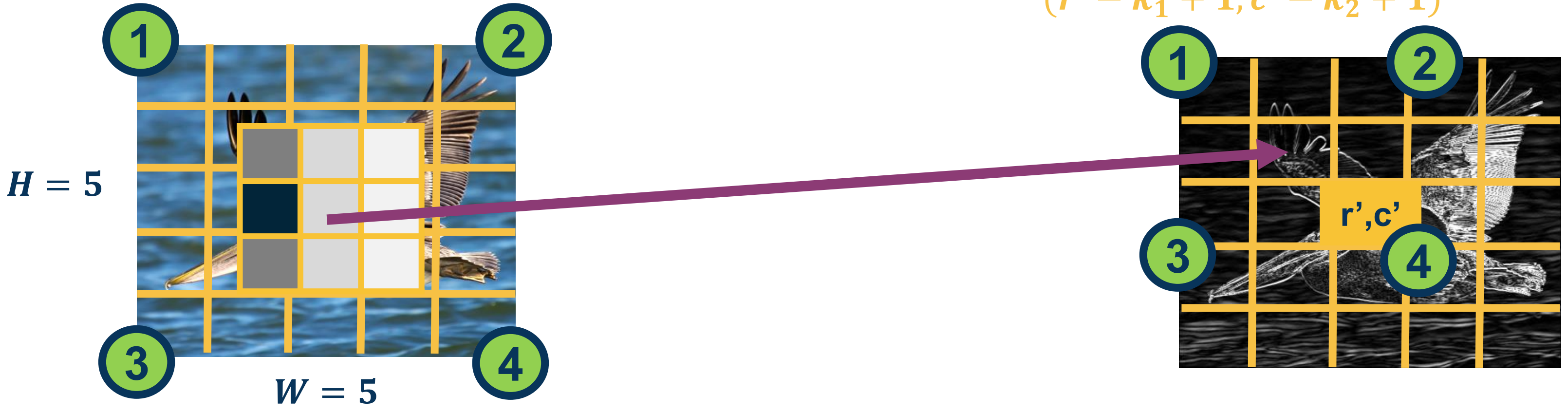
Summing Gradient Contributions

Chain rule for affected pixels (sum gradients):

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

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

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



Summing Gradient Contributions

Definition of cross-correlation (use a', b' to distinguish from prior variables):

$$\boldsymbol{y}(\boldsymbol{r}',\boldsymbol{c}') = (\boldsymbol{x} * \boldsymbol{k})(\boldsymbol{r}',\boldsymbol{c}') = \sum_{\boldsymbol{a}'=0}^{\boldsymbol{k}_1-1} \sum_{\boldsymbol{b}'=0}^{\boldsymbol{k}_2-1} \boldsymbol{x}(\boldsymbol{r}'+\boldsymbol{a}',\boldsymbol{c}'+\boldsymbol{b}')\,\boldsymbol{k}(\boldsymbol{a}',\boldsymbol{b}')$$

Plug in what we actually wanted :

$$\boldsymbol{y}(\boldsymbol{r}'-\boldsymbol{a},\boldsymbol{c}'-\boldsymbol{b}) = (\boldsymbol{x} * \boldsymbol{k})(\boldsymbol{r}',\boldsymbol{c}') = \sum_{\boldsymbol{a}'=0}^{\boldsymbol{k}_1-1} \sum_{\boldsymbol{b}'=0}^{\boldsymbol{k}_2-1} \boldsymbol{x}(\boldsymbol{r}'-\boldsymbol{a}+\boldsymbol{a}',\boldsymbol{c}'-\boldsymbol{b}+\boldsymbol{b}')\,\boldsymbol{k}(\boldsymbol{a}',\boldsymbol{b}')$$

What is $\dfrac{\partial \boldsymbol{y}(\boldsymbol{r}'-\boldsymbol{a},\boldsymbol{c}'-\boldsymbol{b})}{\partial \boldsymbol{x}(\boldsymbol{r}',\boldsymbol{c}')} = \mathbf{k}(\boldsymbol{a},\boldsymbol{b})$ (we want term with $\boldsymbol{x}(\boldsymbol{r}',\boldsymbol{c}')$ in it; this happens when $\mathbf{a}' = \mathbf{a}$ and $\mathbf{b}' = \mathbf{b}$)

Calculating the Gradient

Plugging in to earlier equation:

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

Does this look familiar?

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

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

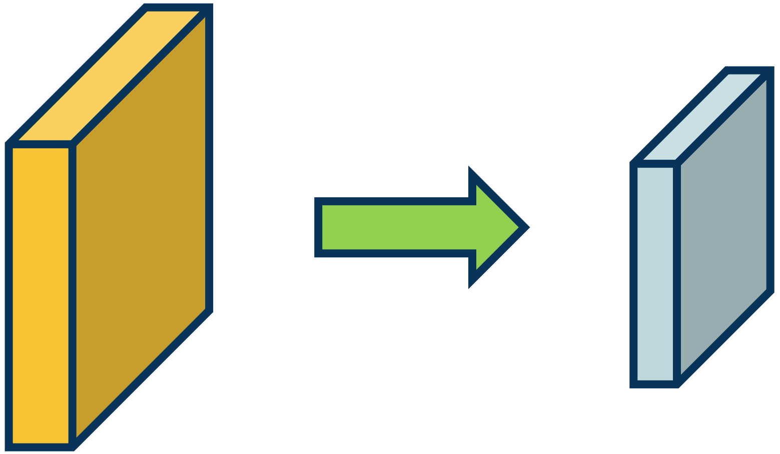
- All operations still implemented via **efficient linear algebra** (e.g. matrix-matrix multiplication)

Summary

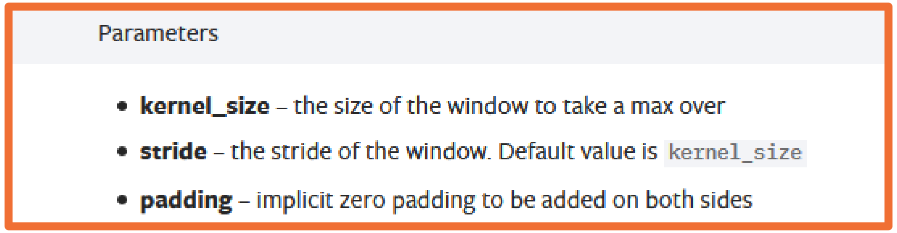
Georgia Tech

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

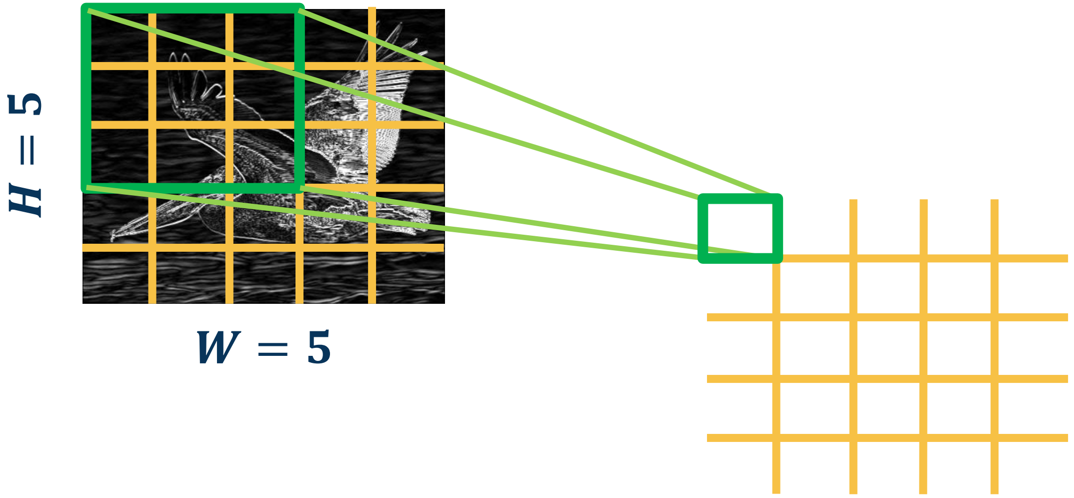
From: https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html#torch.nn.MaxPool2d

Pooling Layers

Example: Max pooling

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

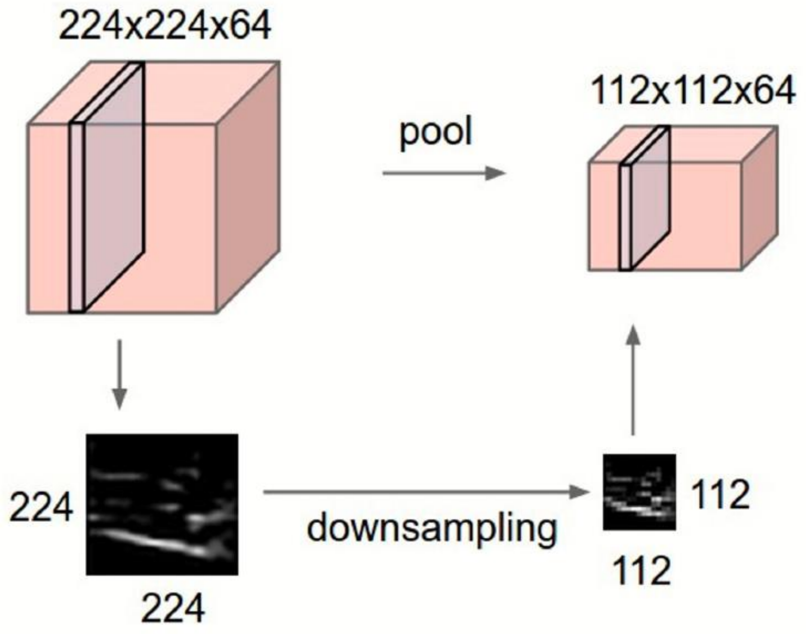
$$X(0{:}2, 0{:}2) = \begin{bmatrix} 200 & 150 & 150 \\ 100 & 50 & 100 \\ 25 & 25 & 10 \end{bmatrix} \Rightarrow \max(0{:}2,0{:}2) = 200$$



How many learned parameters does this layer have?

None!

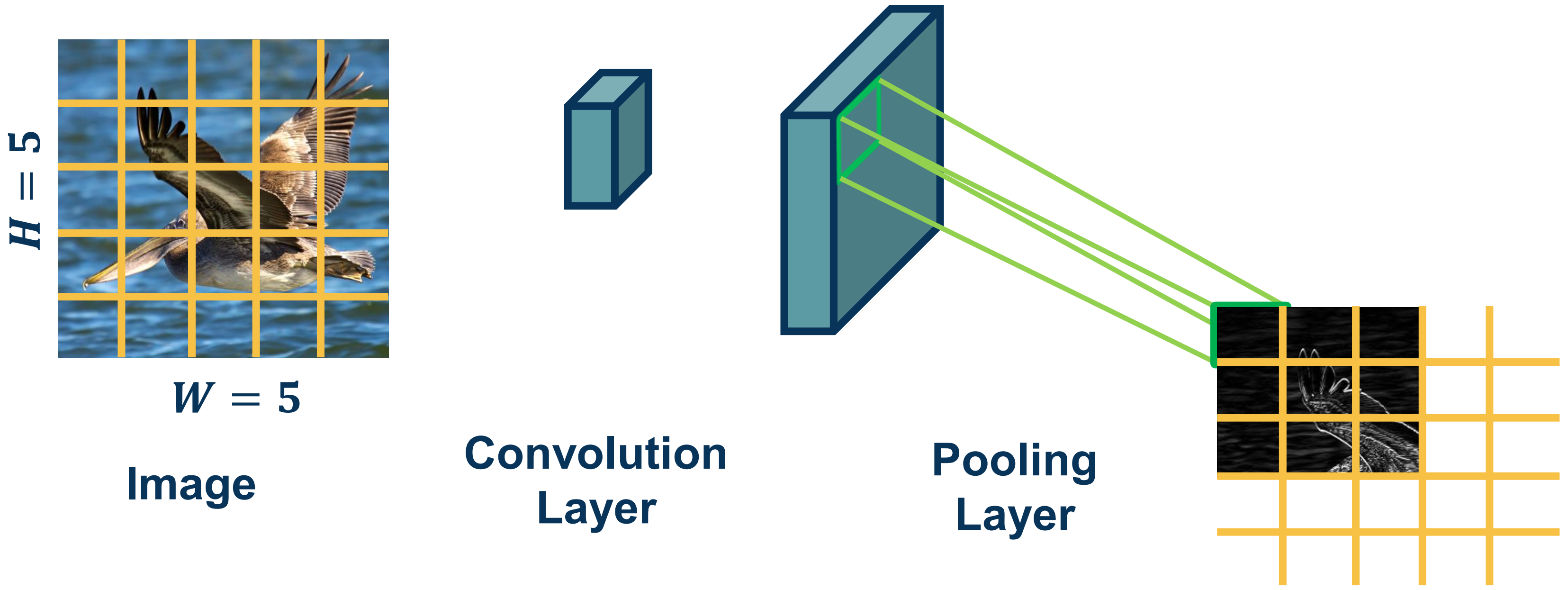
- makes the representations spatially smaller
- saves computation (GPU mem & speed), allows go deeper
- operates over each activation map independently:



From: Slides by CS 231n, Danfei Xu

Pooling with Tensors

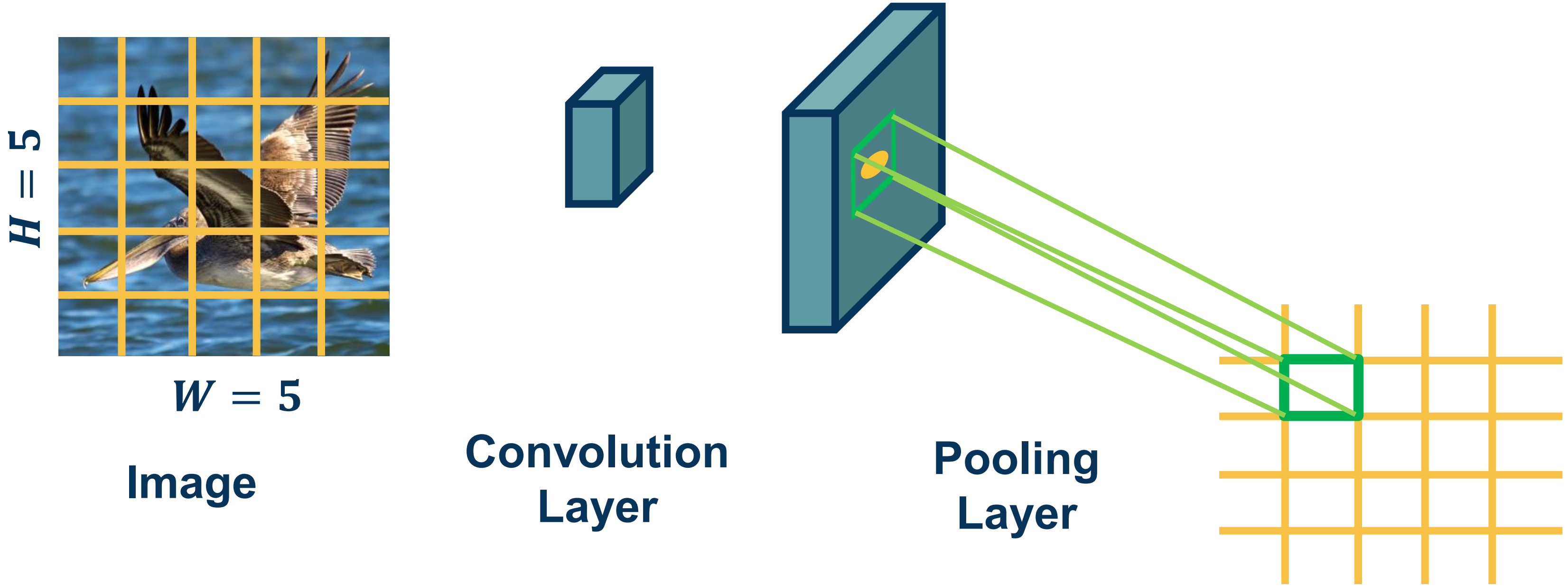
Since the **output** of convolution and pooling layers are **(multi-channel) images**, we can sequence them just as any other layer



Combining Convolution & Pooling Layers

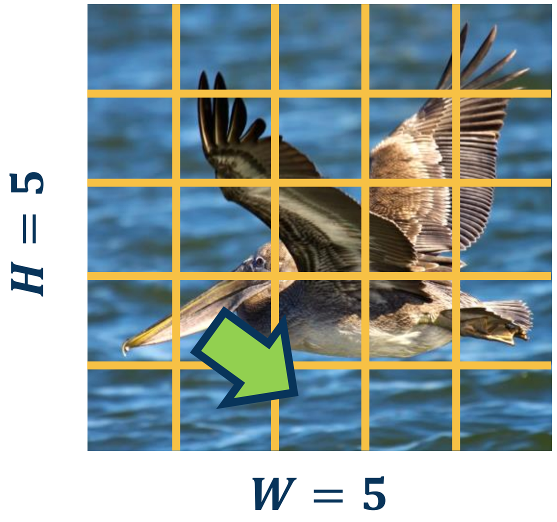
This combination adds some **invariance** to translation of the features

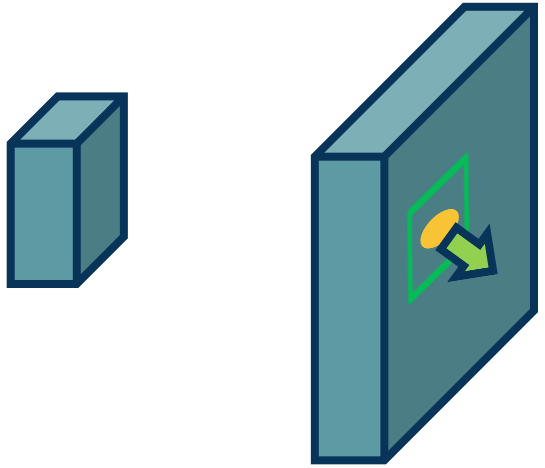
- If feature (such as beak) translated a little bit, output values still **remain the same**



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

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

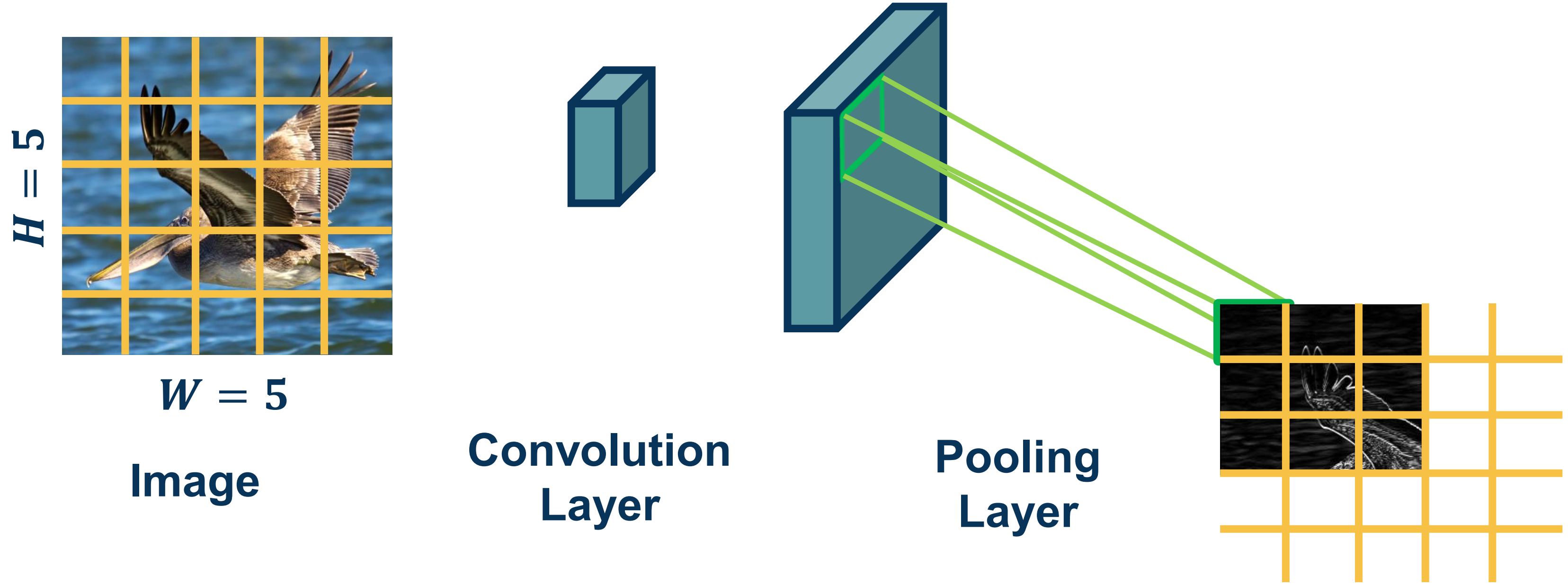




Invariance vs. Equivariance

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)

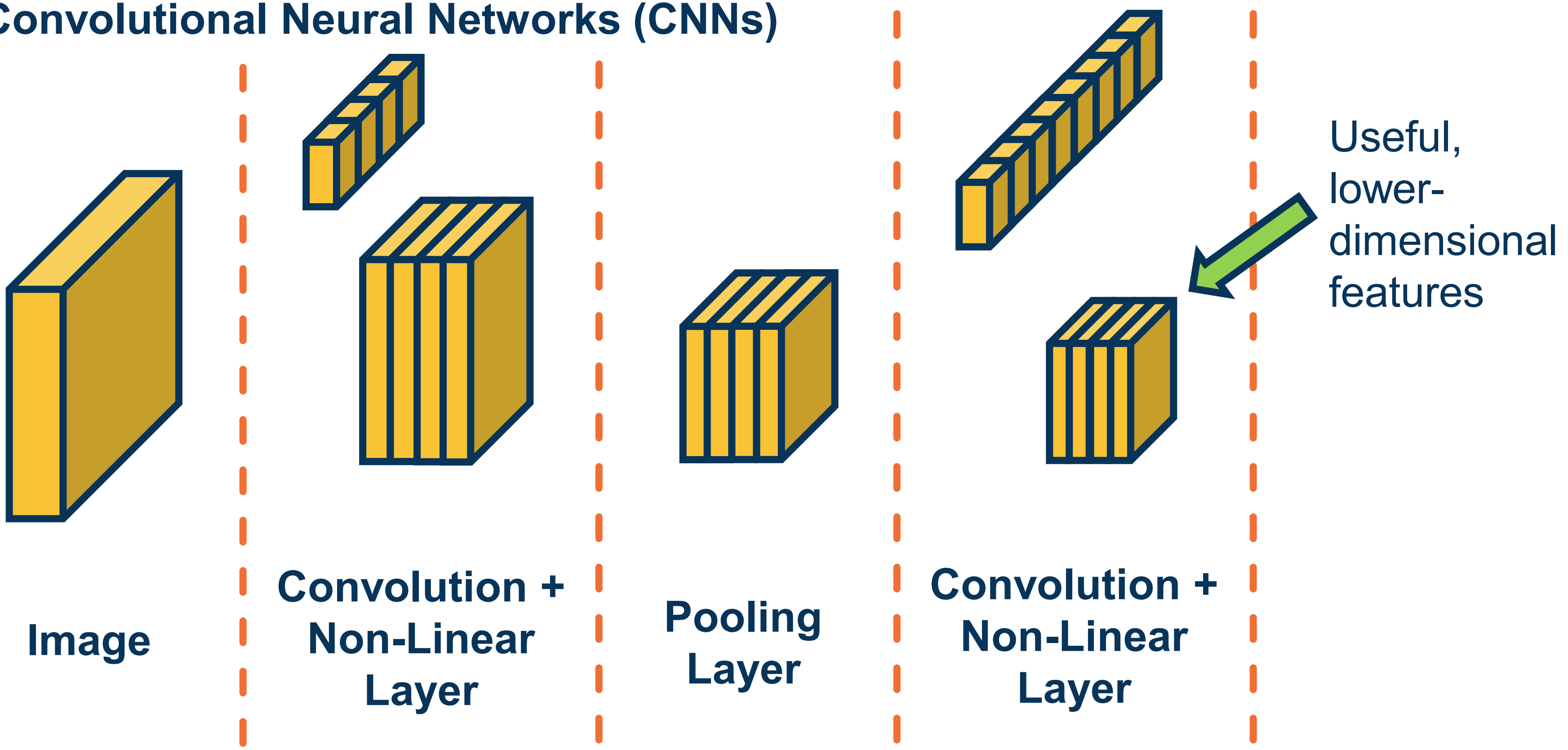


Image
Convolution + Non-Linear Layer
Pooling Layer
Convolution + Non-Linear Layer
Useful, lower-dimensional features

Alternating Convolution and Pooling

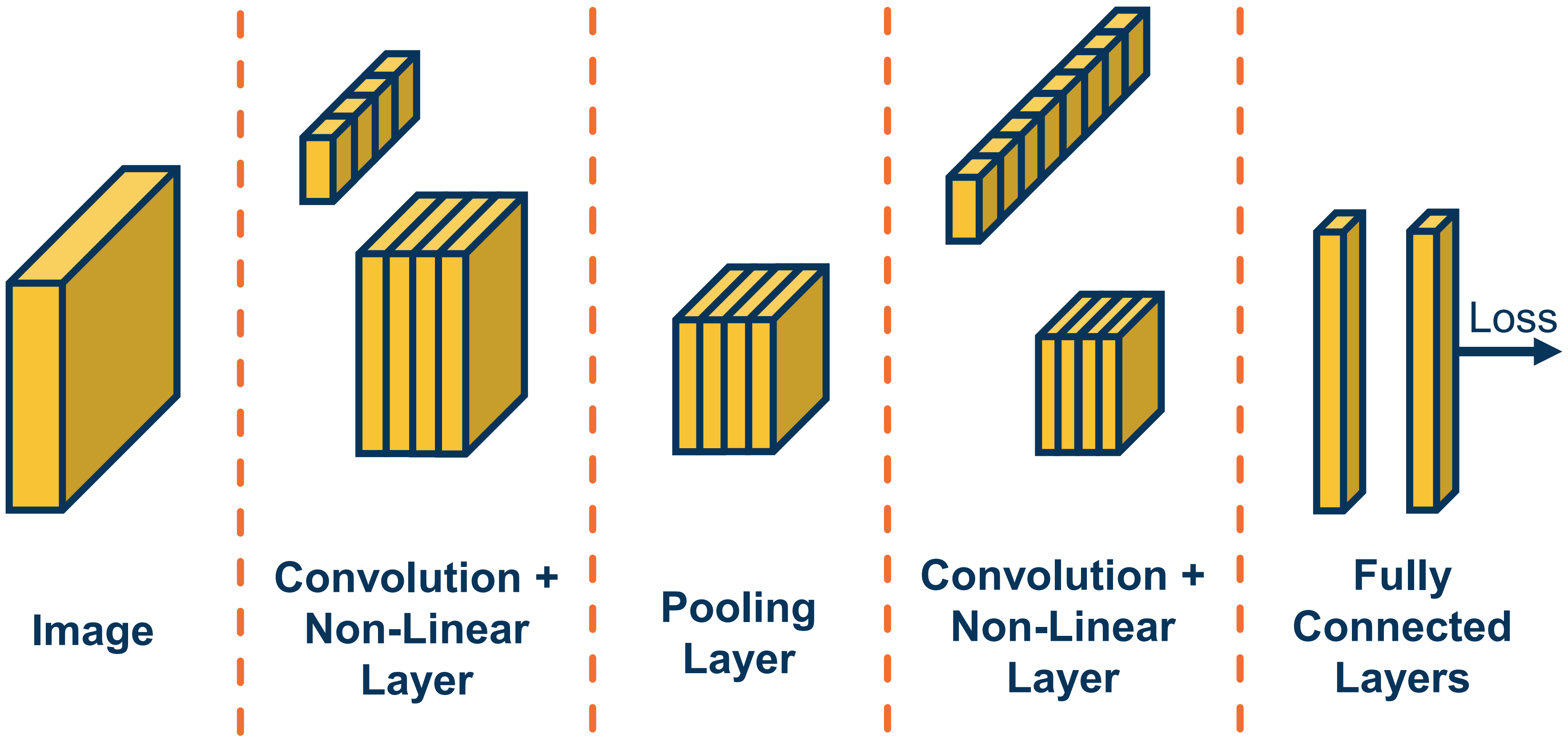
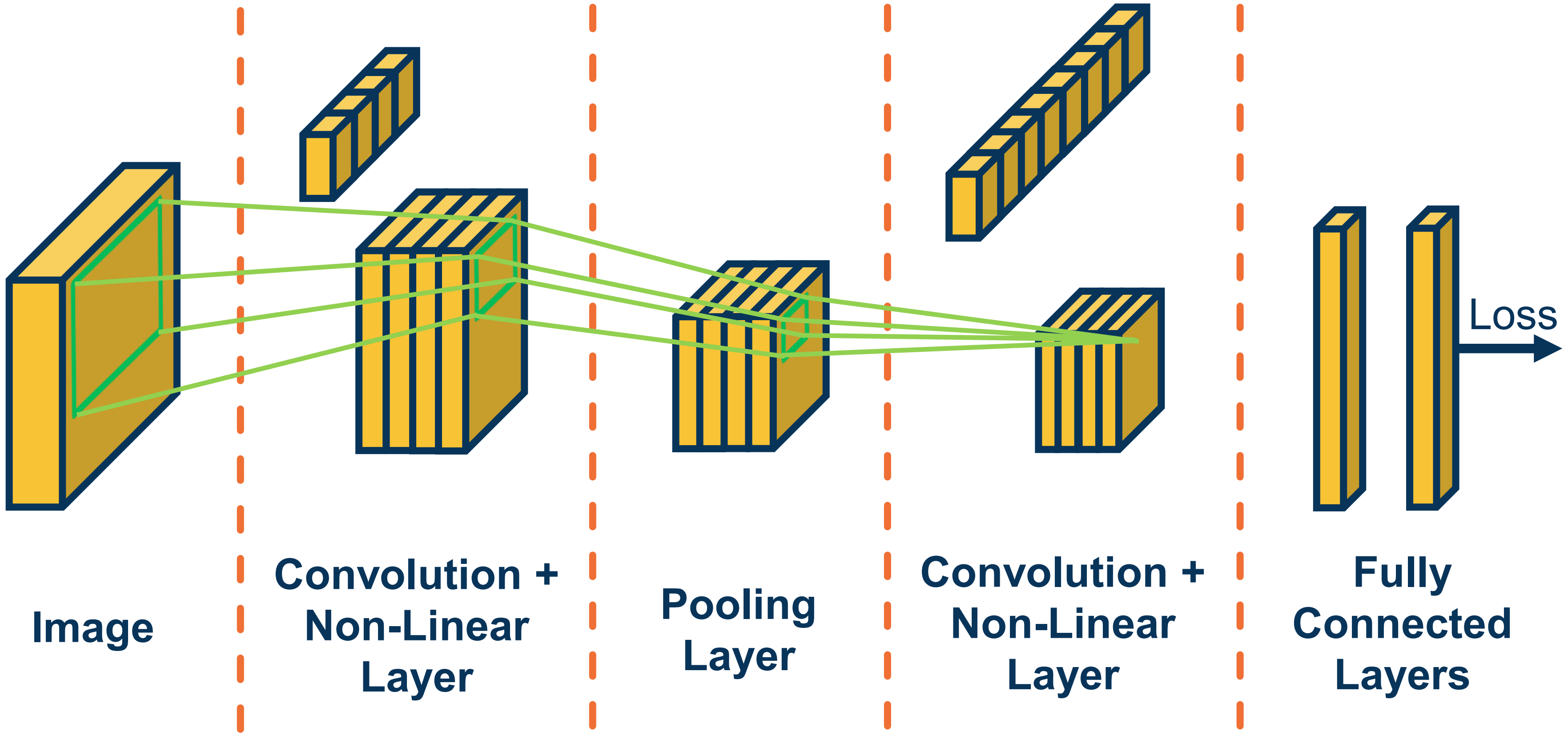
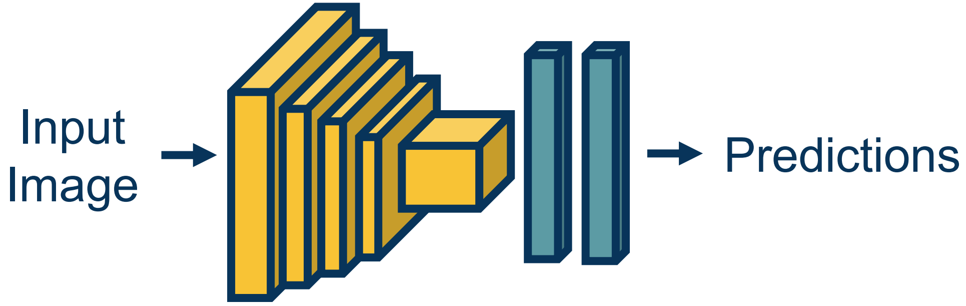


Image
Convolution + Non-Linear Layer
Pooling Layer
Convolution + Non-Linear Layer
Fully Connected Layers
Loss

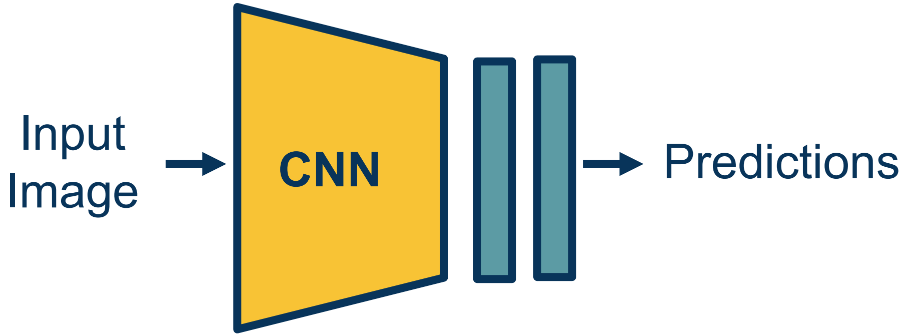
Adding a Fully Connected Layer


Loss
Image
Convolution + Non-Linear Layer
Pooling Layer
Convolution + Non-Linear Layer
Fully Connected Layers

Receptive Fields



Convolutional Neural Networks



Typical Depiction of CNNs

These architectures have existed **since 1980s**

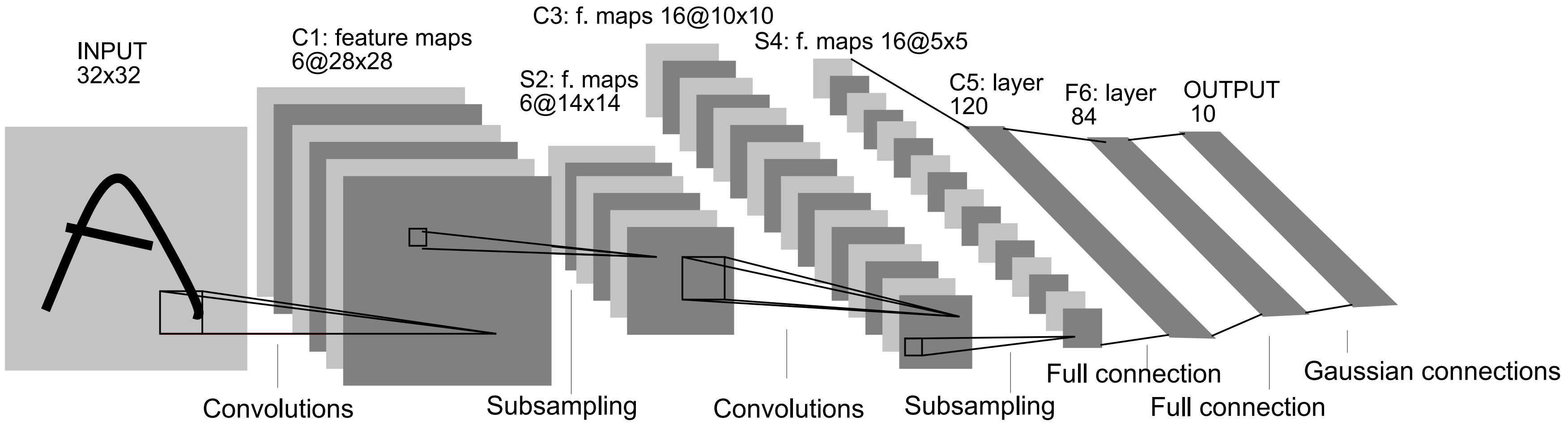


Image Credit: Yann LeCun, Kevin Murphy

Handwriting Recognition

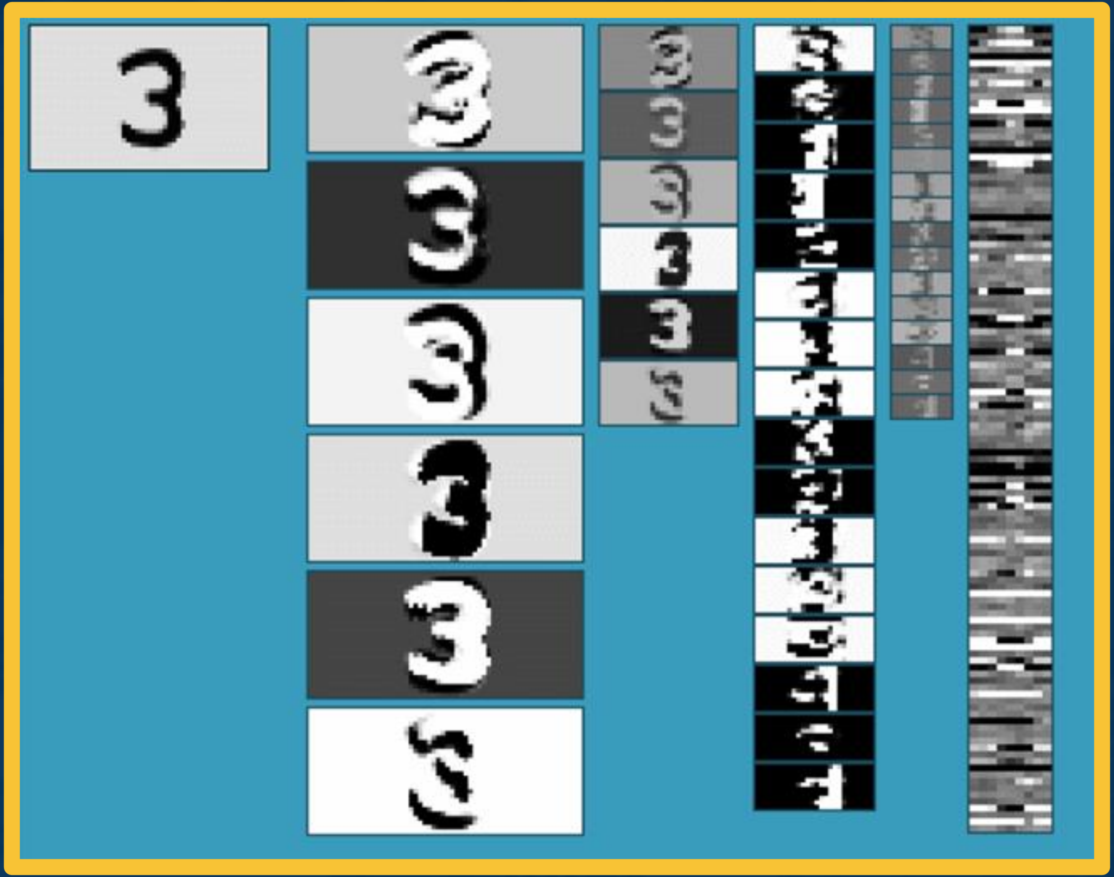


Image Credit: Yann LeCun

Translation Equivariance (Conv Layers) & Invariance (Output)

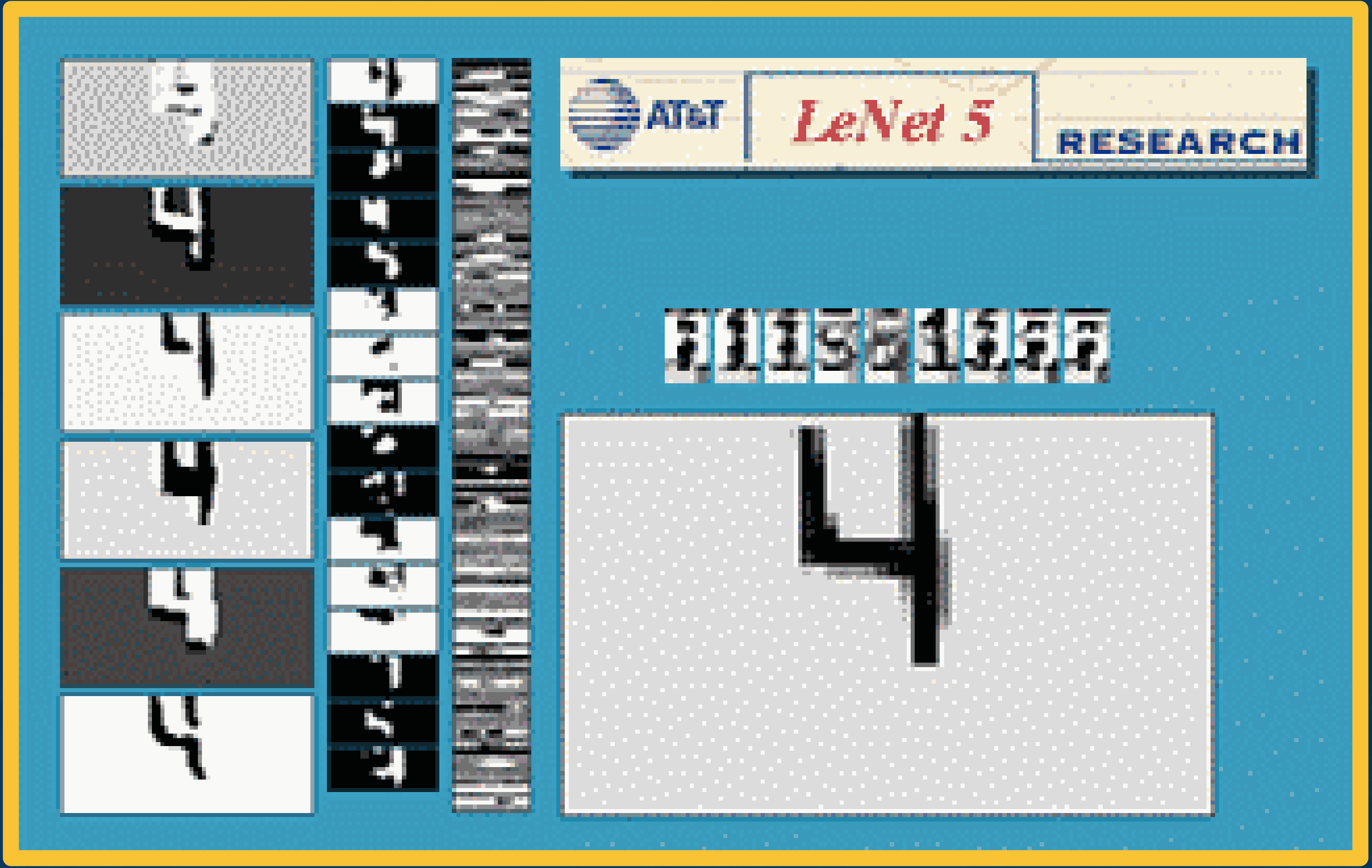


Image Credit: Yann LeCun

(Some) Rotation Invariance

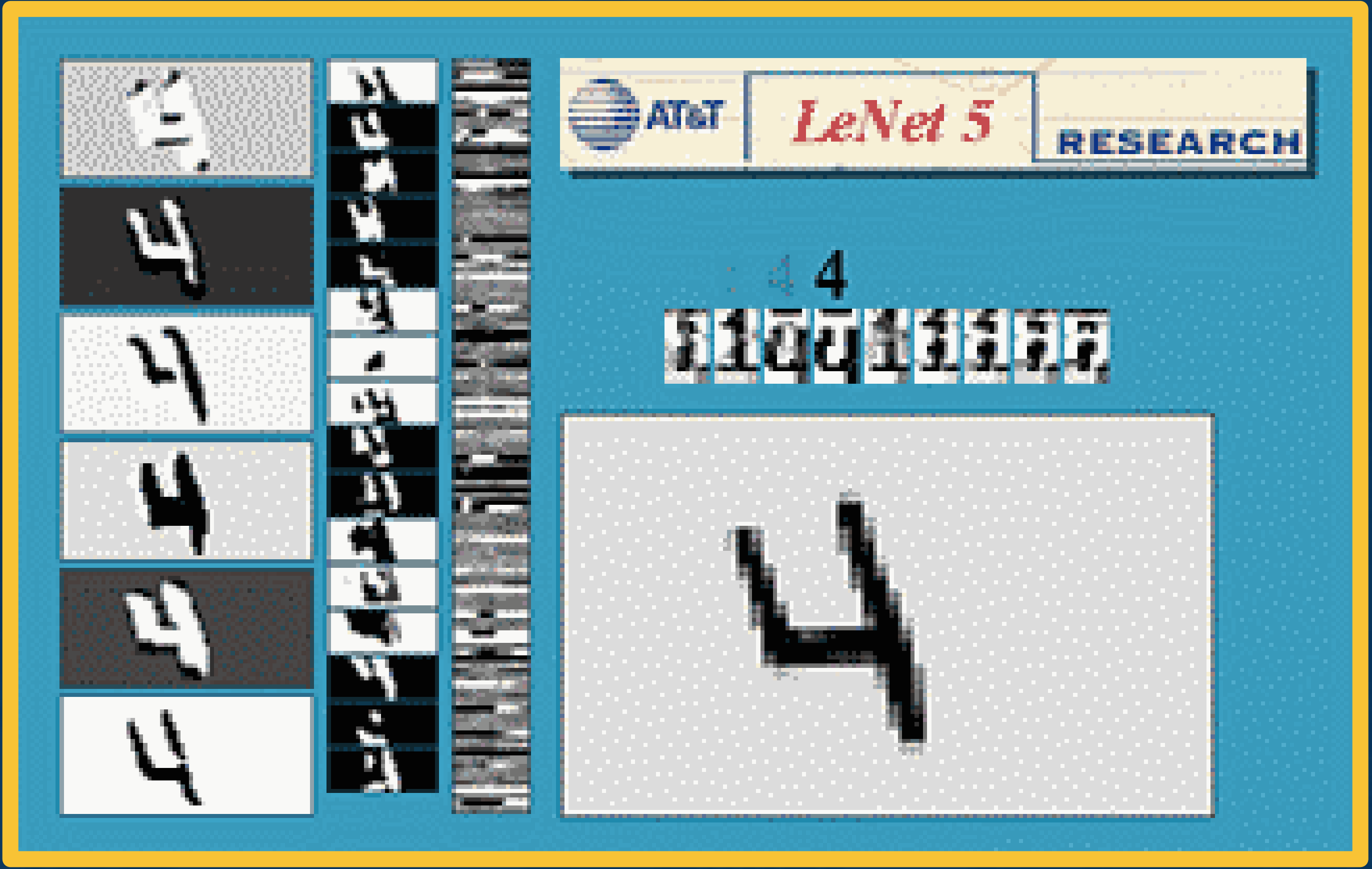


Image Credit: Yann LeCun

(Some) Scale Invariance

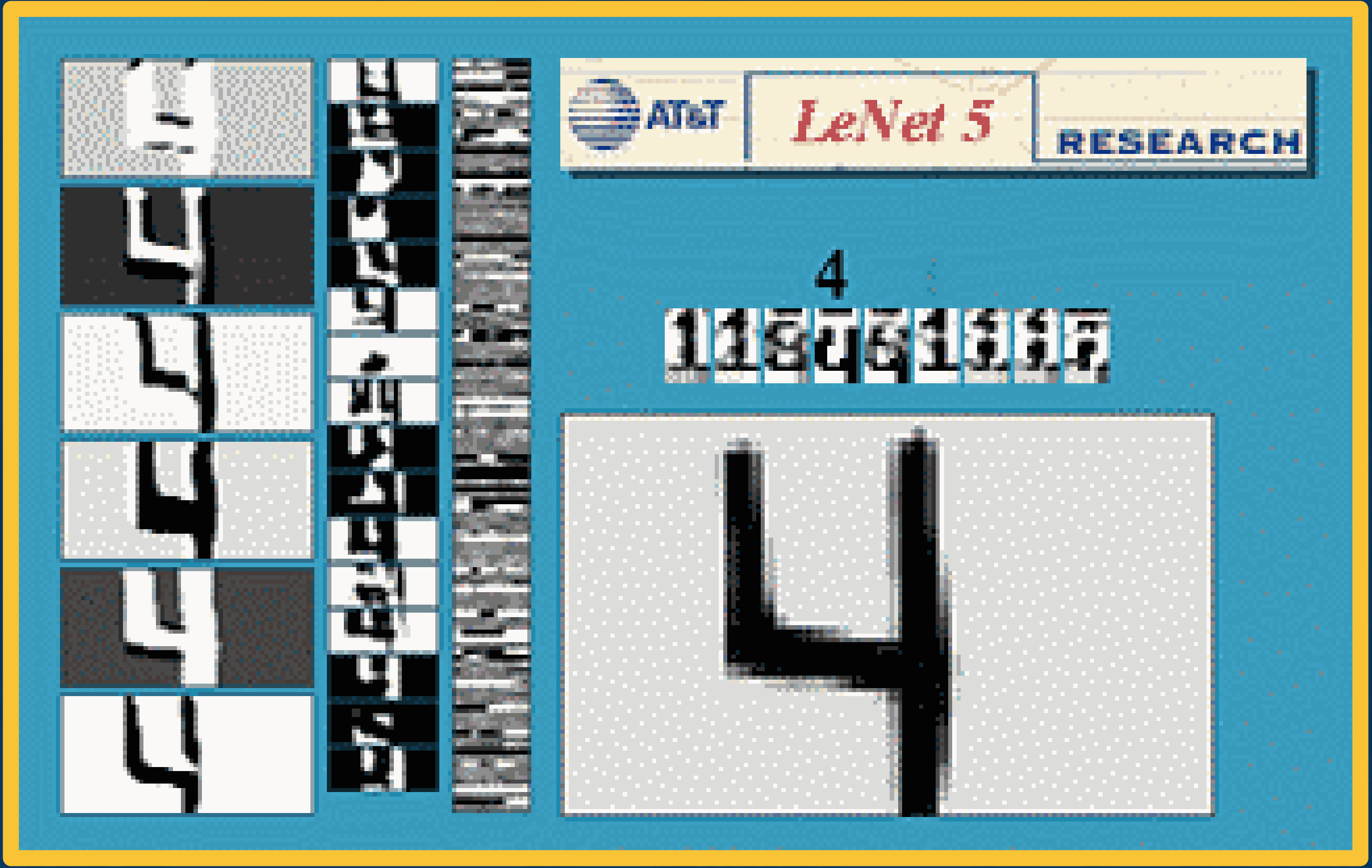
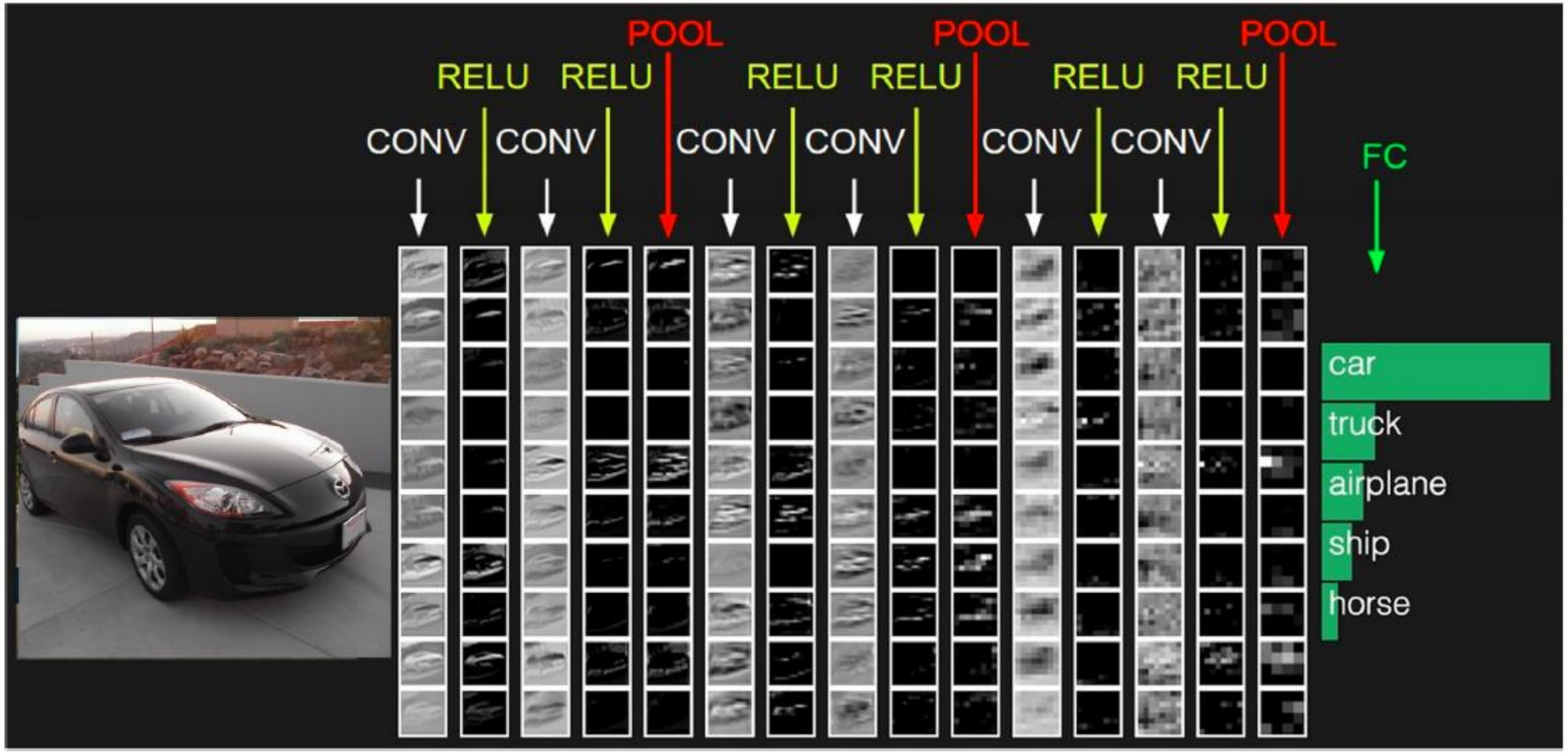


Image Credit: Yann LeCun



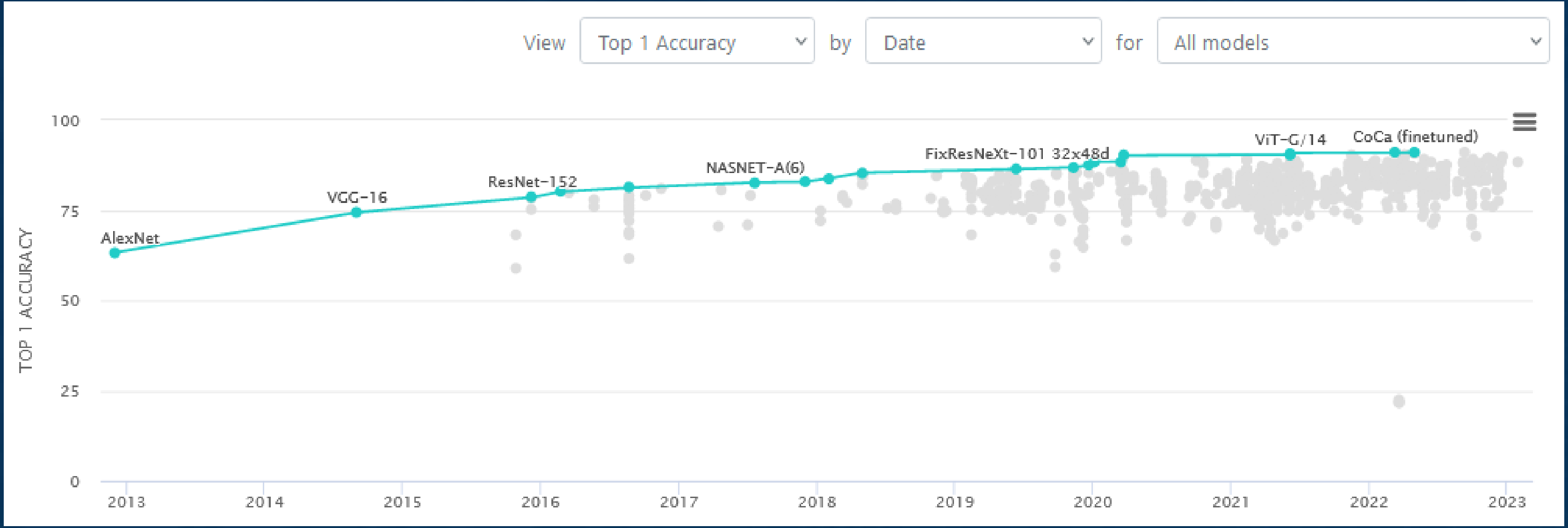
A More Modern Canonical CNN

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.

Benchmarking Models

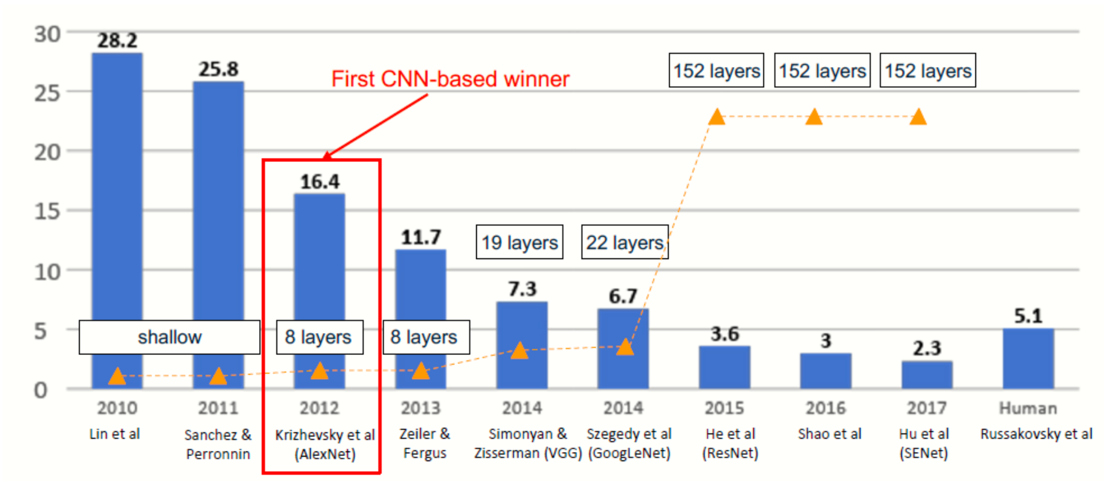
The Importance of Benchmarks



From: https://paperswithcode.com

Case Studies

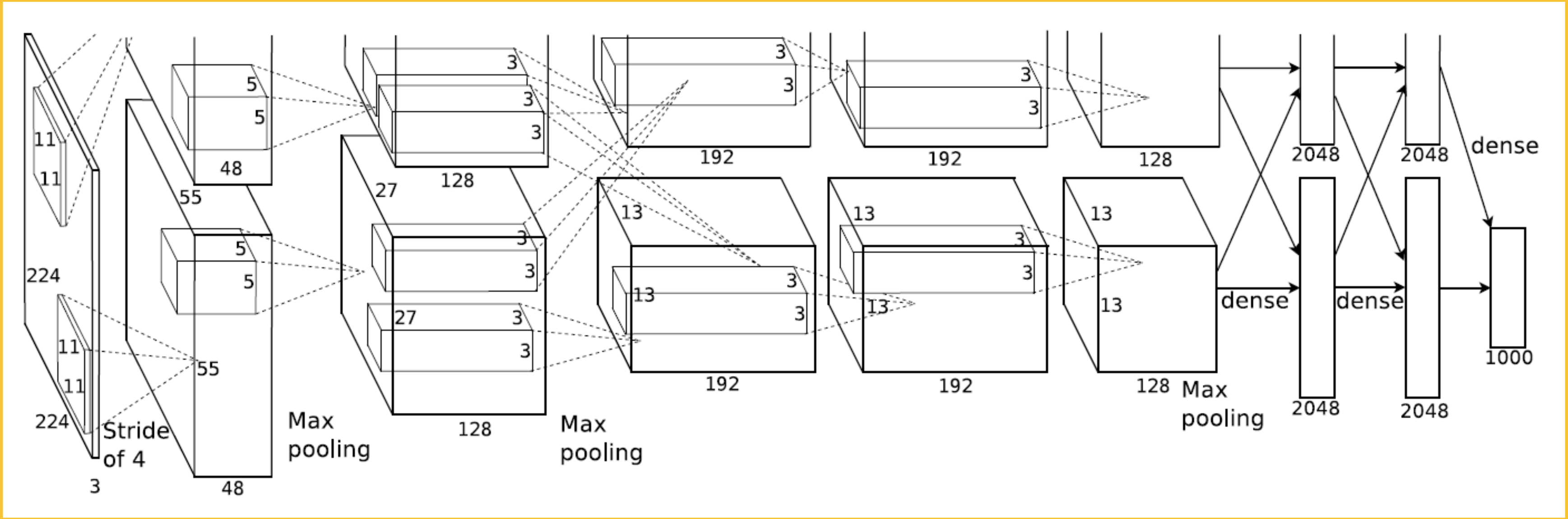
- AlexNet
- VGG
- GoogLeNet
- ResNet



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

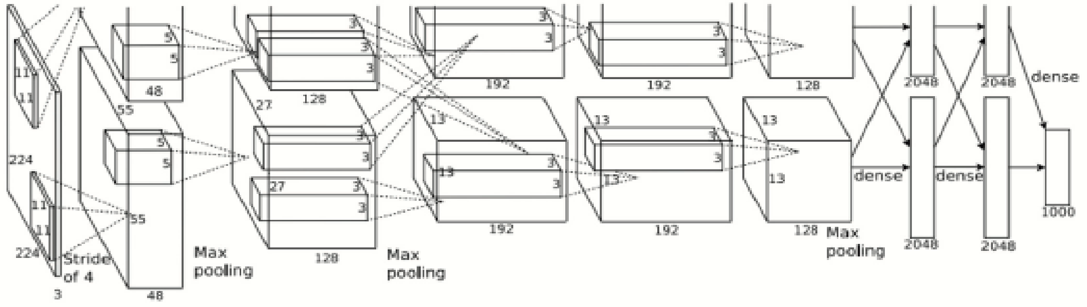
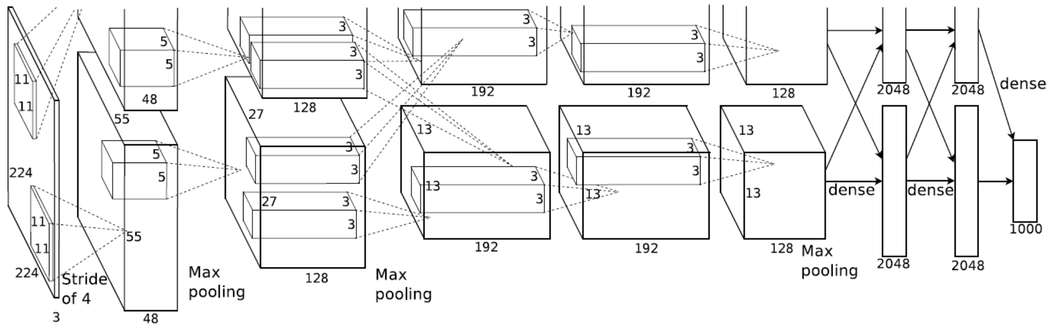


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.



Input: 227x227x3 images

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

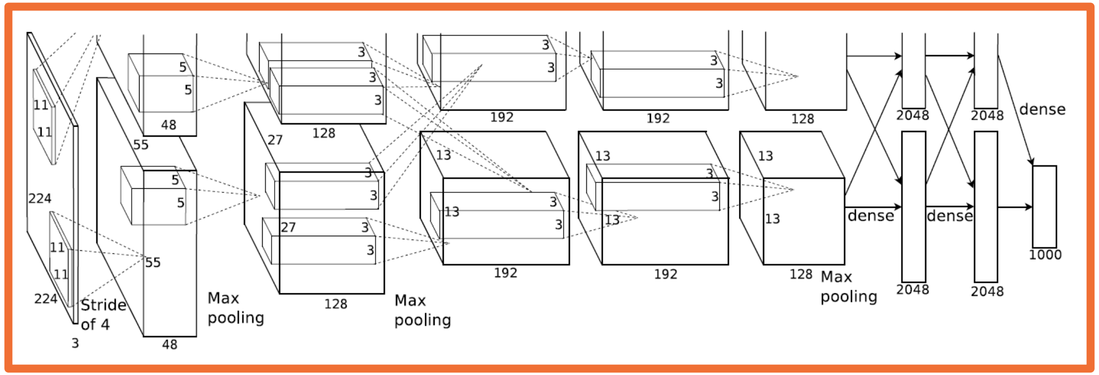
=>

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

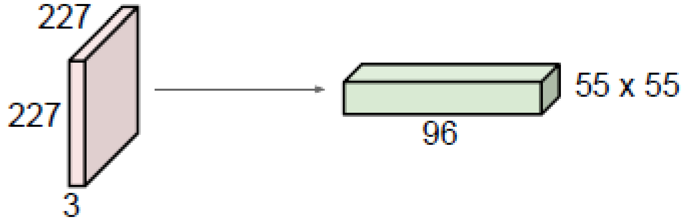
AlexNet – Layers and Key Aspects



Input: 227x227x3 images

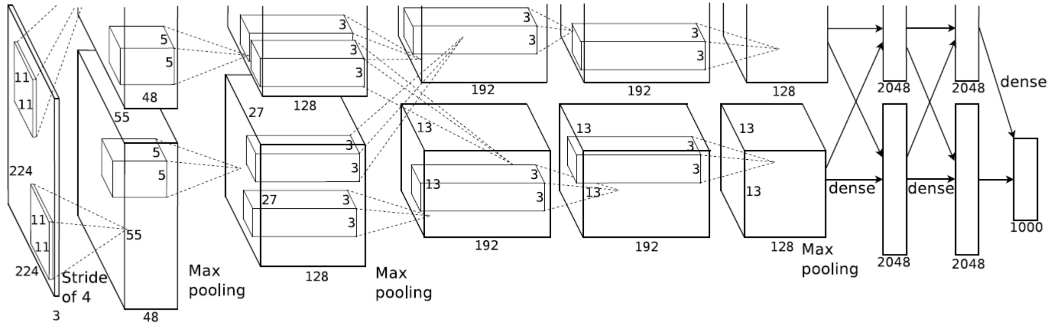
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

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



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

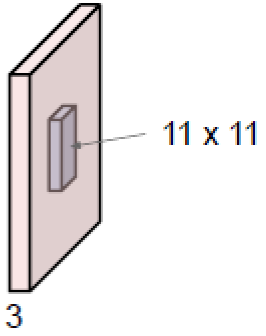
AlexNet – Layers and Key Aspects



Input: 227x227x3 images

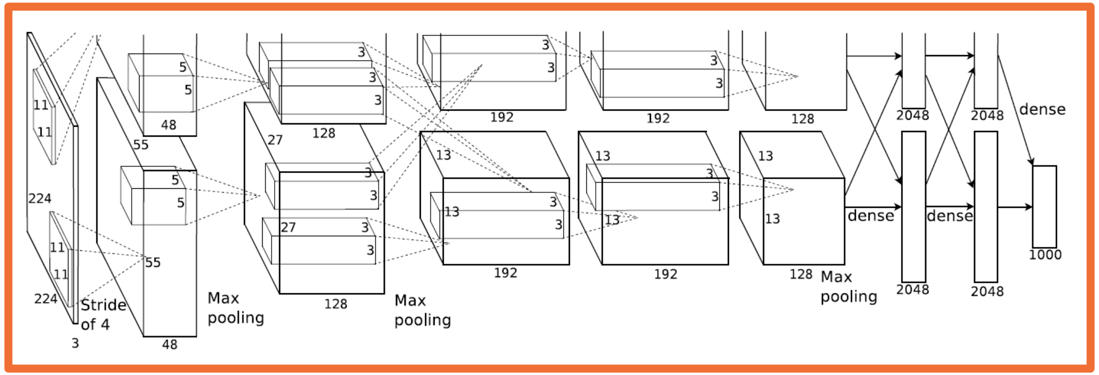
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

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



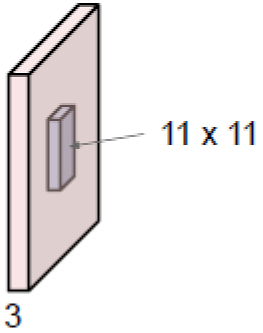
From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

AlexNet – Layers and Key Aspects



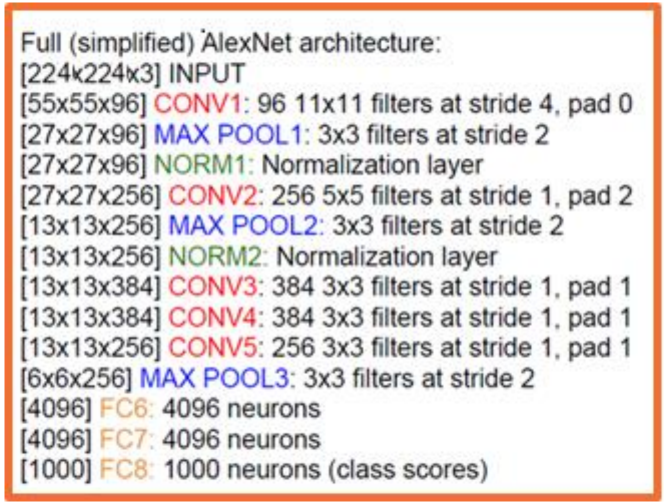
Input: 227x227x3 images

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

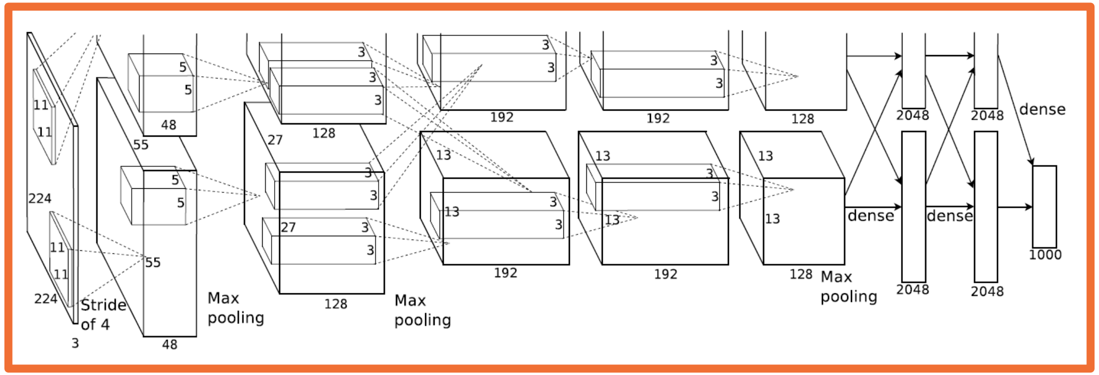


From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

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