

4.Convolutional neural network

Slide ref:-

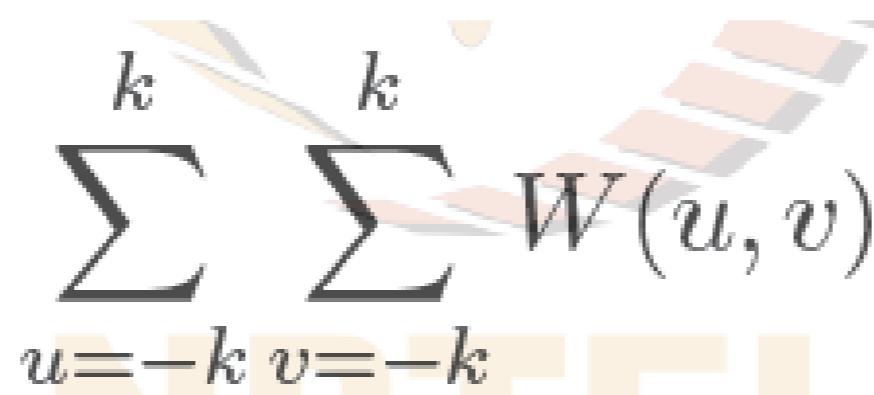
PROF. VINEETH B., IITH, NPTEL

Yunzhe Xue

<HTTP://CS231N.GITHUB.IO/CONVOLUTIONAL-NETWORKS/>

Review: Convolution Operation

- **Convolution** is a mathematical way of combining two signals to form a third signal
- One of the most important techniques in signal processing
- In case of 2D data (grayscale images), the convolution operation between a filter $W^{k \times k}$ and an image $X^{N_1 \times N_2}$ can be expressed as:

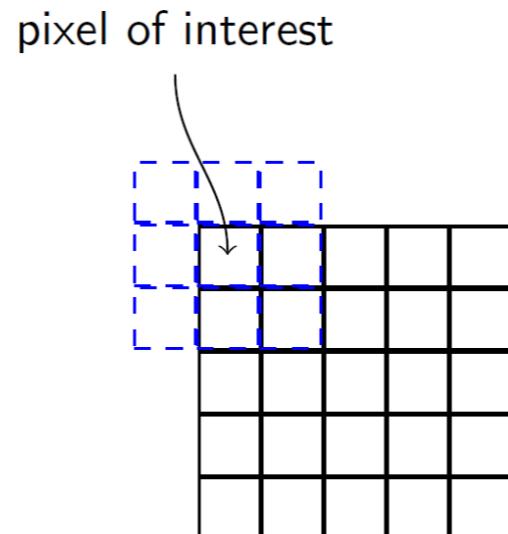
$$Y(i, j) = \sum_{u=-k}^k \sum_{v=-k}^k W(u, v) X(i - u, j - v)$$


Convolution Operation

- More generally, given a $K_1 \times K_2$ filter W , we can write it as:

$$Y(i, j) = \sum_{a=\lfloor -\frac{K_1}{2} \rfloor}^{\lfloor \frac{K_1}{2} \rfloor} \sum_{b=\lfloor -\frac{K_2}{2} \rfloor}^{\lfloor -\frac{K_2}{2} \rfloor} X(i-a, j-b) W\left(\frac{K_1}{2} + a, \frac{K_2}{2} + b\right)$$

- This allows kernel to be **centered** on pixel of interest

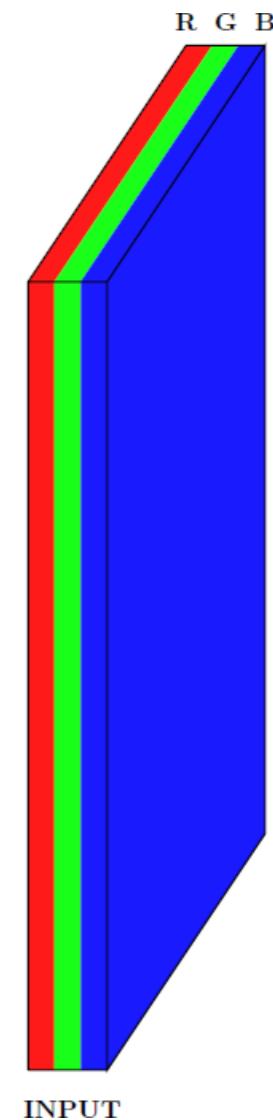


Pause and Ponder

- In the 1D case, we slide a one-dimensional filter over a one-dimensional input
- In the 2D case, we slide a two-dimensional filter over a two-dimensional input
- What would happen in the 3D case where your images are in color (RGB)?

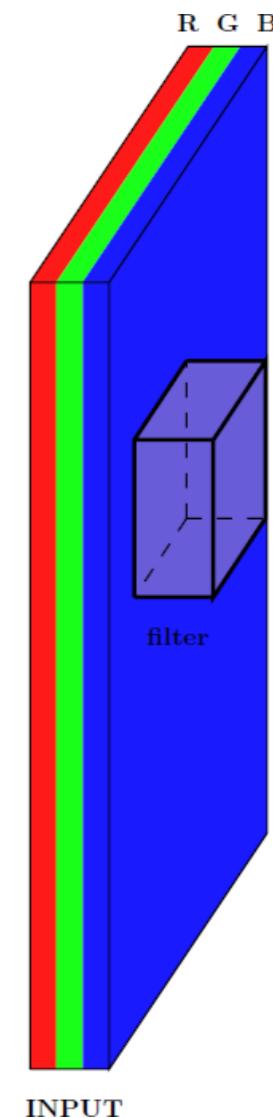
Convolution Operation

- What would a 3D filter look like?



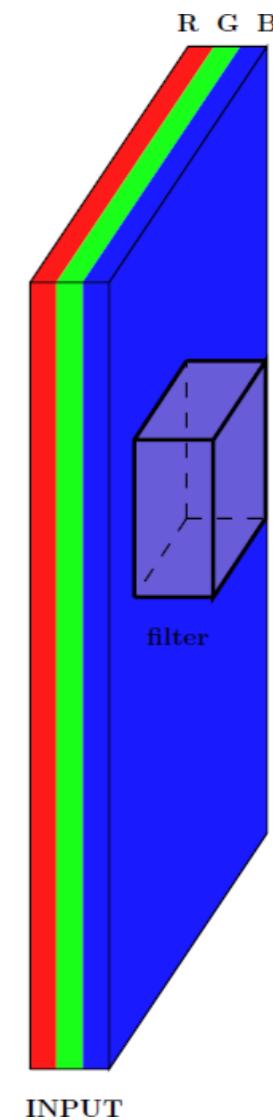
Convolution Operation

- What would a 3D filter look like?
- It will be in 3D too and we will refer to it as a volume



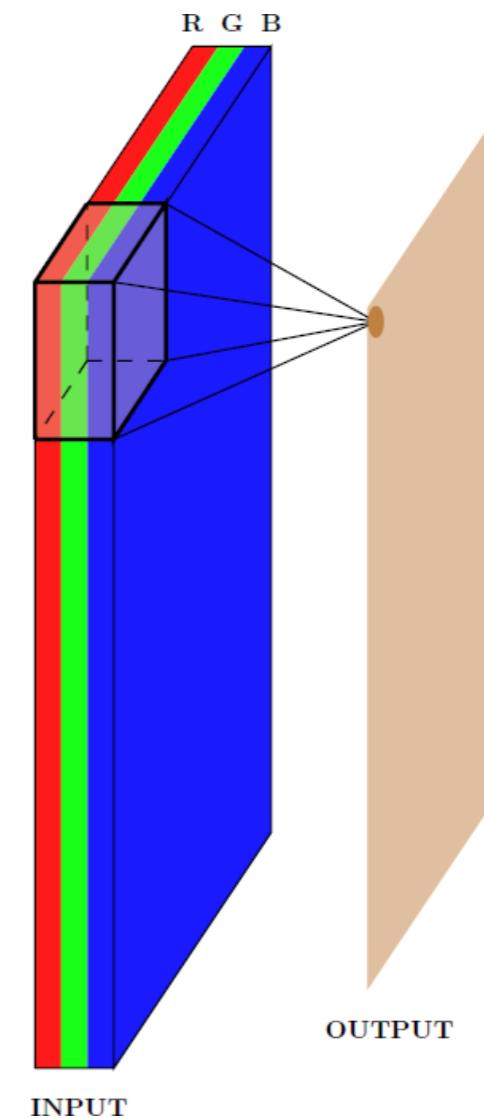
Convolution Operation

- What would a 3D filter look like?
- It will be in 3D too and we will refer to it as a volume
- Once again we will slide the volume over 3D input and perform the convolution operation



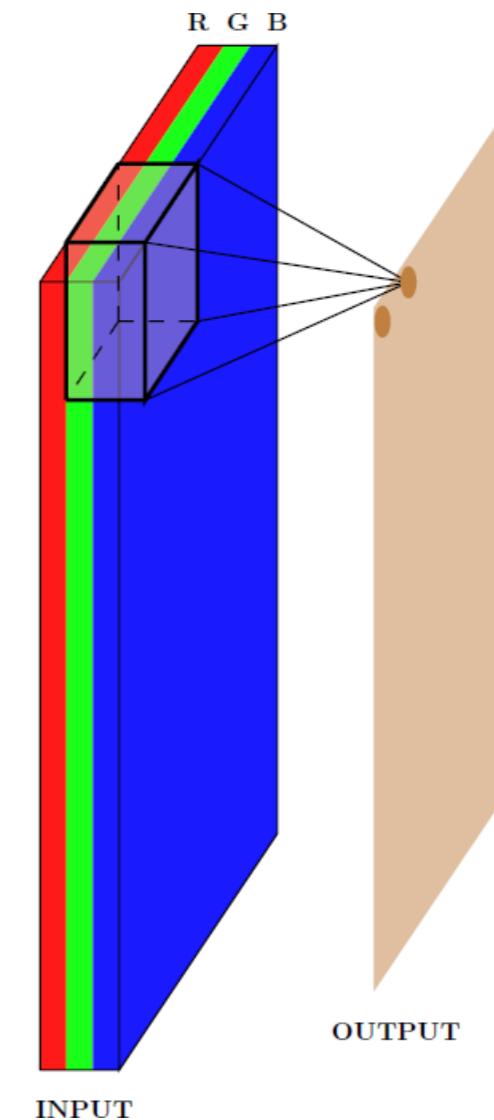
Convolution Operation

- What would a 3D filter look like?
- It will be in 3D too and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and perform the convolution operation



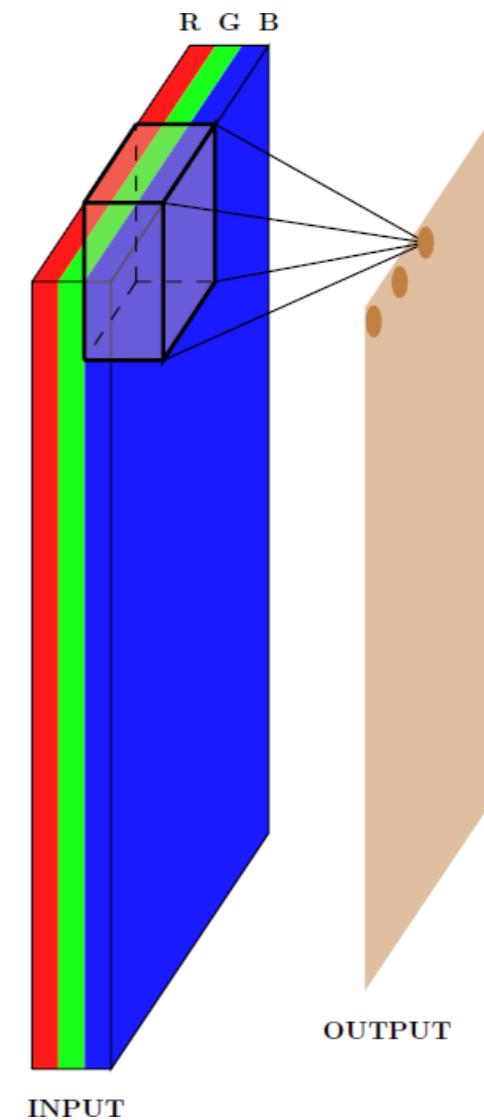
Convolution Operation

- What would a 3D filter look like?
- It will be in 3D too and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and perform the convolution operation



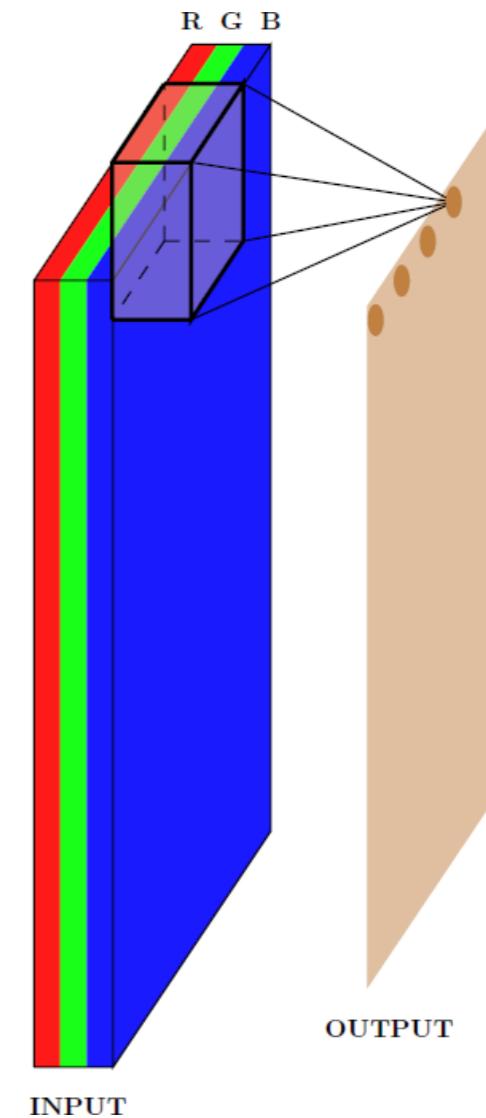
Convolution Operation

- What would a 3D filter look like?
- It will be in 3D too and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and perform the convolution operation



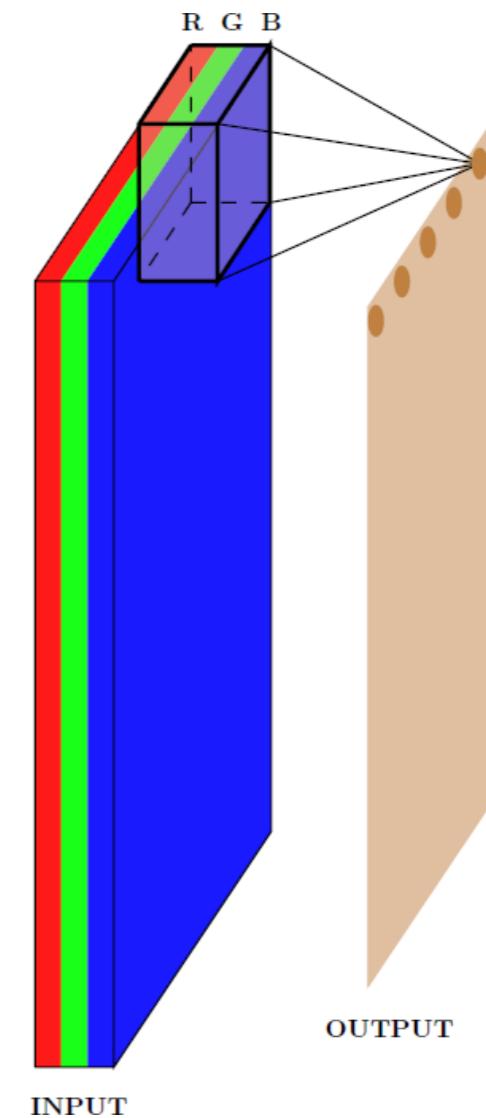
Convolution Operation

- What would a 3D filter look like?
- It will be in 3D too and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and perform the convolution operation



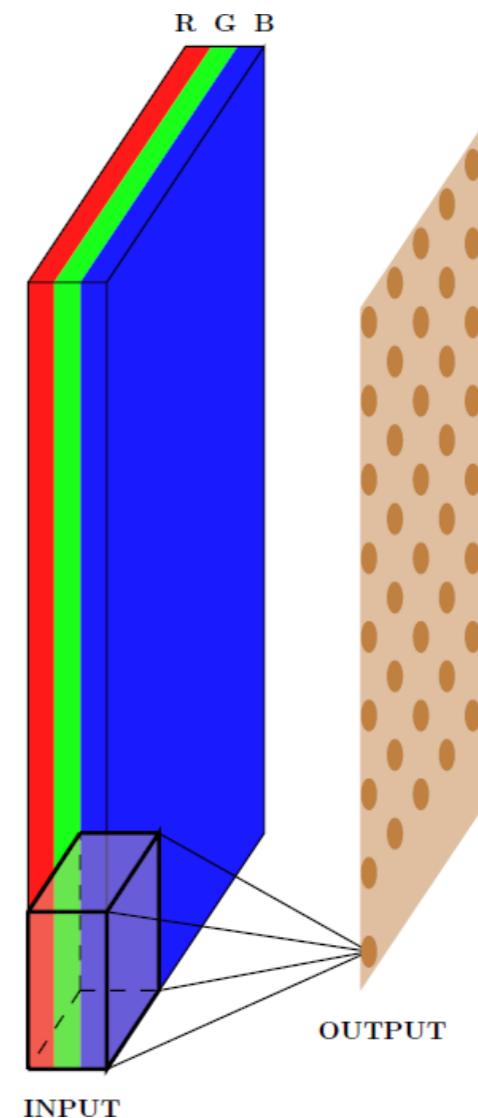
Convolution Operation

- What would a 3D filter look like?
- It will be in 3D too and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and perform the convolution operation



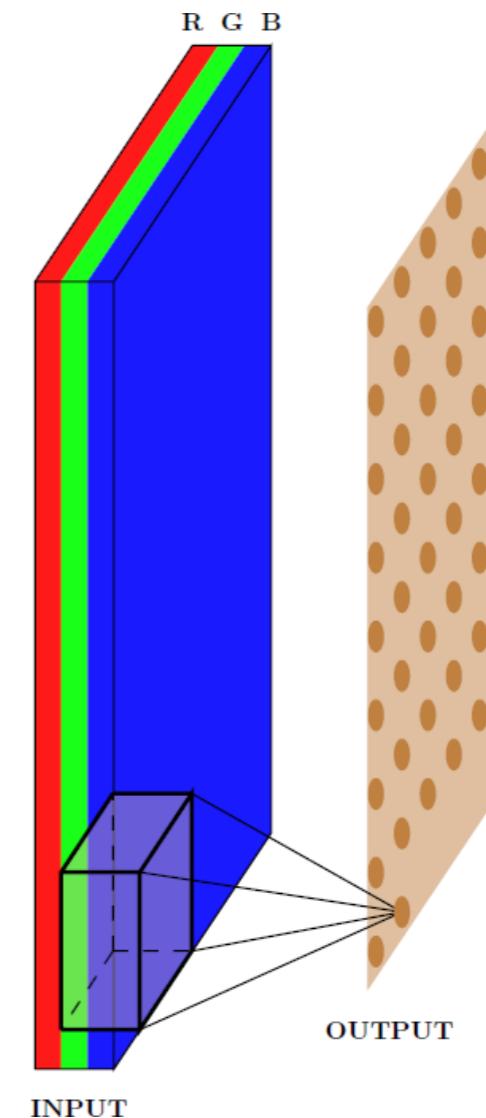
Convolution Operation

- What would a 3D filter look like?
- It will be in 3D too and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and perform the convolution operation



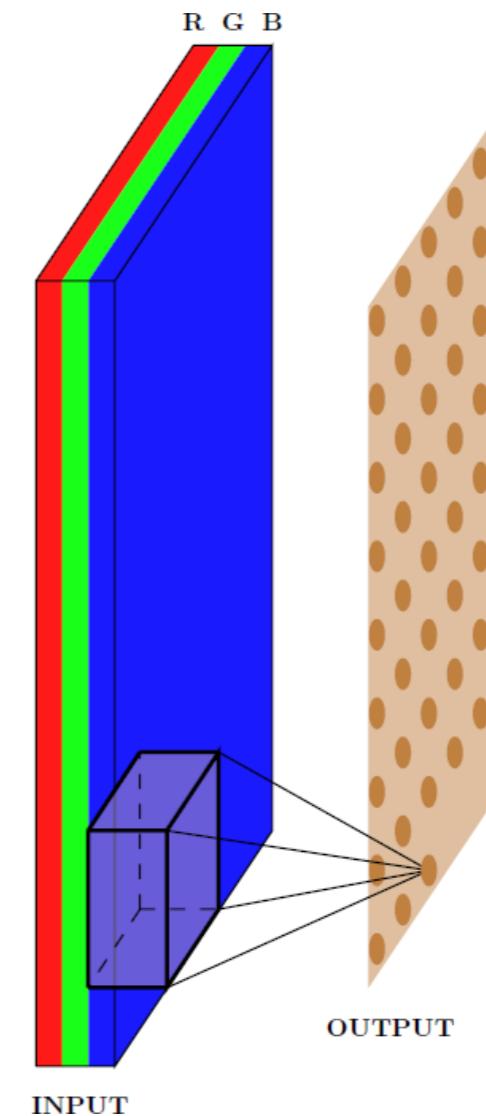
Convolution Operation

- What would a 3D filter look like?
- It will be in 3D too and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and perform the convolution operation



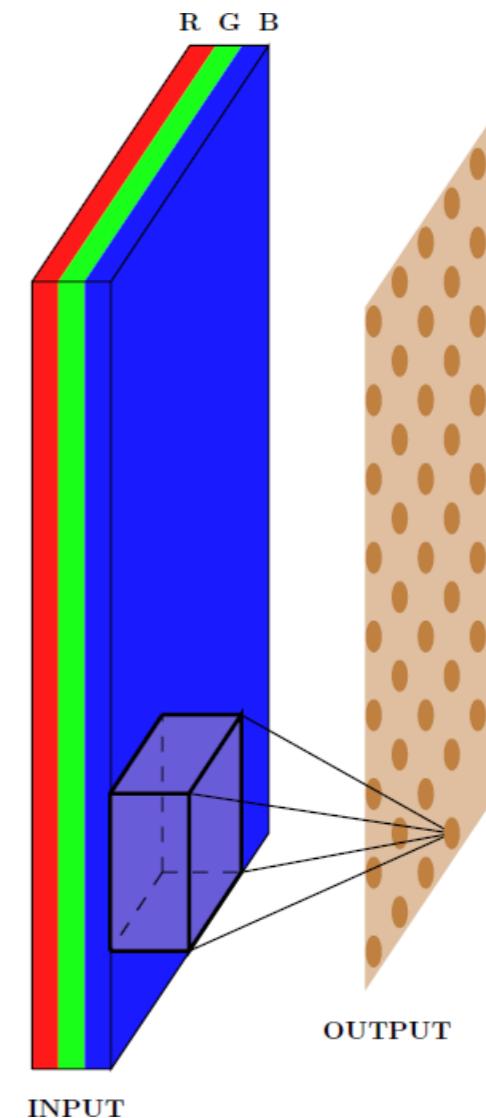
Convolution Operation

- What would a 3D filter look like?
- It will be in 3D too and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and perform the convolution operation



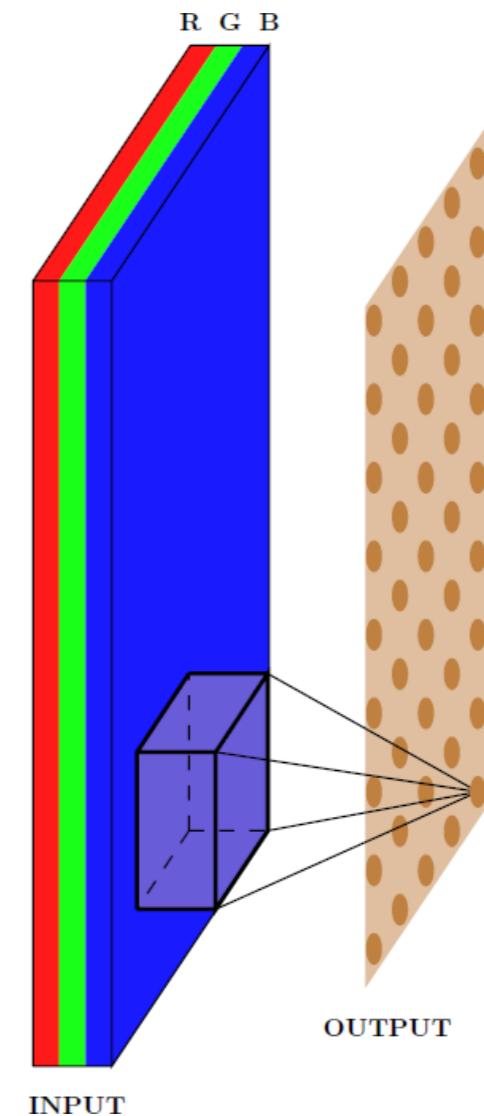
Convolution Operation

- What would a 3D filter look like?
- It will be in 3D too and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and perform the convolution operation



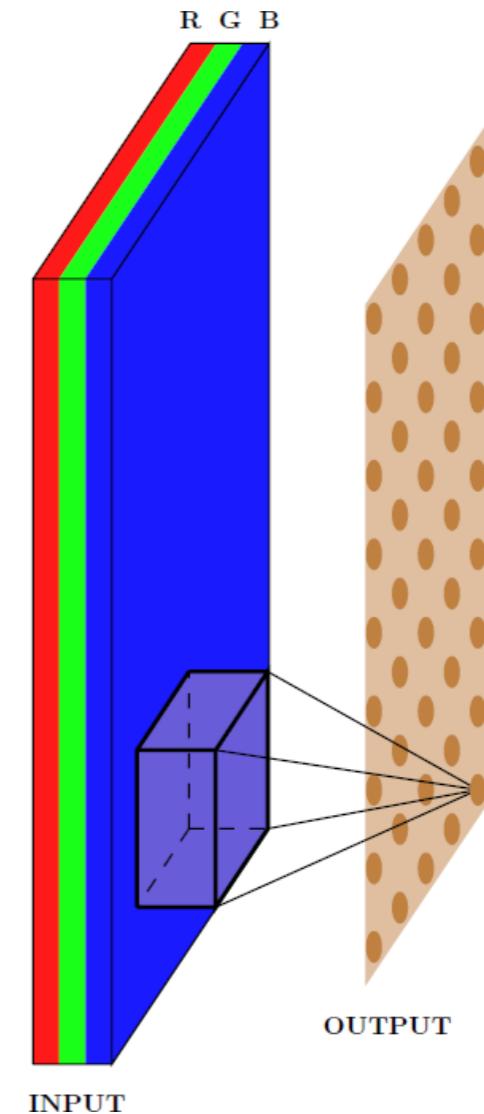
Convolution Operation

- What would a 3D filter look like?
- It will be in 3D too and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and perform the convolution operation
- We assume that the filter always extends to the depth of the image
- In effect, we are doing a 2D convolution operation on a 3D input (because the filter moves along the height and the width but not along the depth)



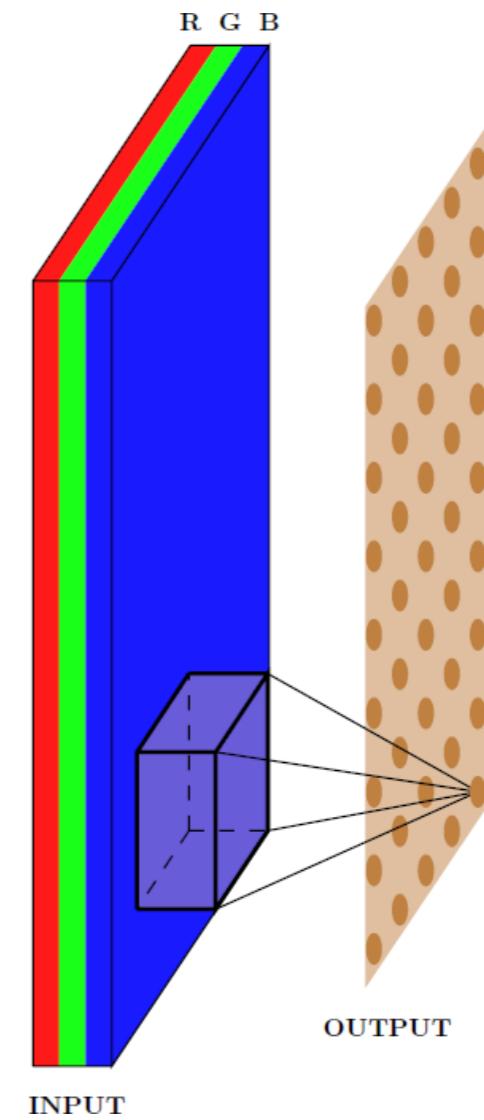
Convolution Operation

- What would a 3D filter look like?
- It will be in 3D too and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and perform the convolution operation
- We assume that the filter always extends to the depth of the image
- In effect, we are doing a 2D convolution operation on a 3D input (because the filter moves along the height and the width but not along the depth)
- As a result the output will be 2D (only width and height, no depth)



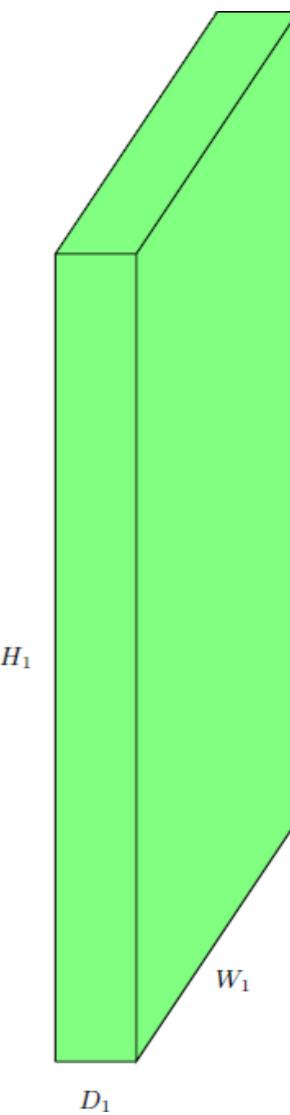
Convolution Operation

- What would a 3D filter look like?
- It will be in 3D too and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and perform the convolution operation
- We assume that the filter always extends to the depth of the image
- In effect, we are doing a 2D convolution operation on a 3D input (because the filter moves along the height and the width but not along the depth)
- As a result the output will be 2D (only width and height, no depth)
- We can apply multiple filters to get multiple feature maps



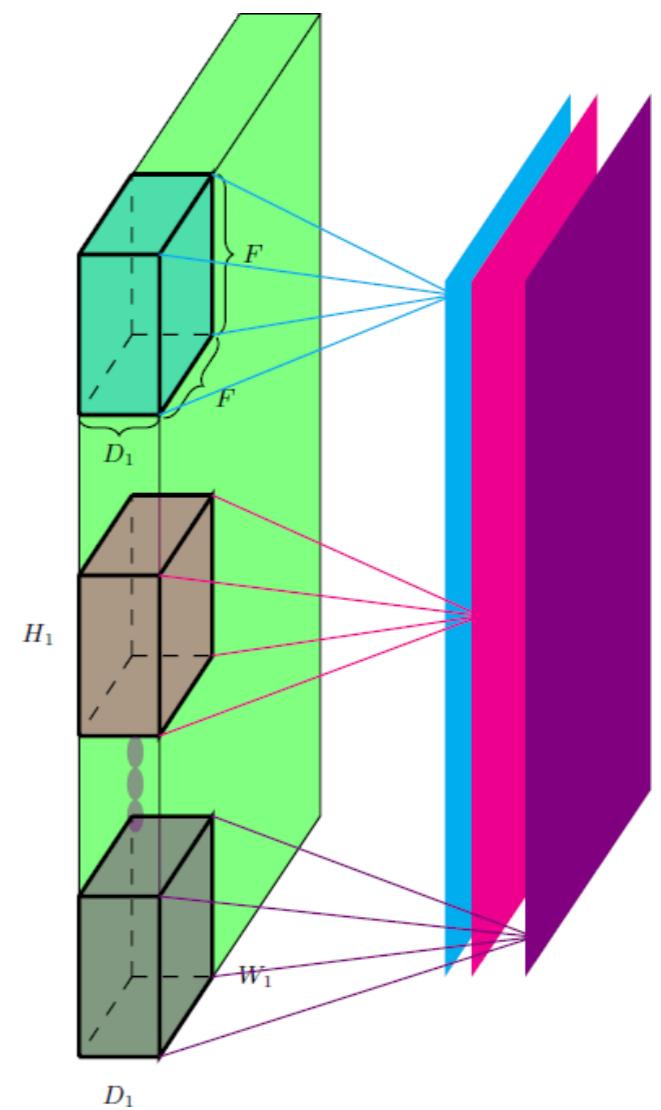
Convolution: Understanding the (Hyper)Parameters

- Input dimensions: Width (W_1) \times Height (H_1) \times Depth (D_1)



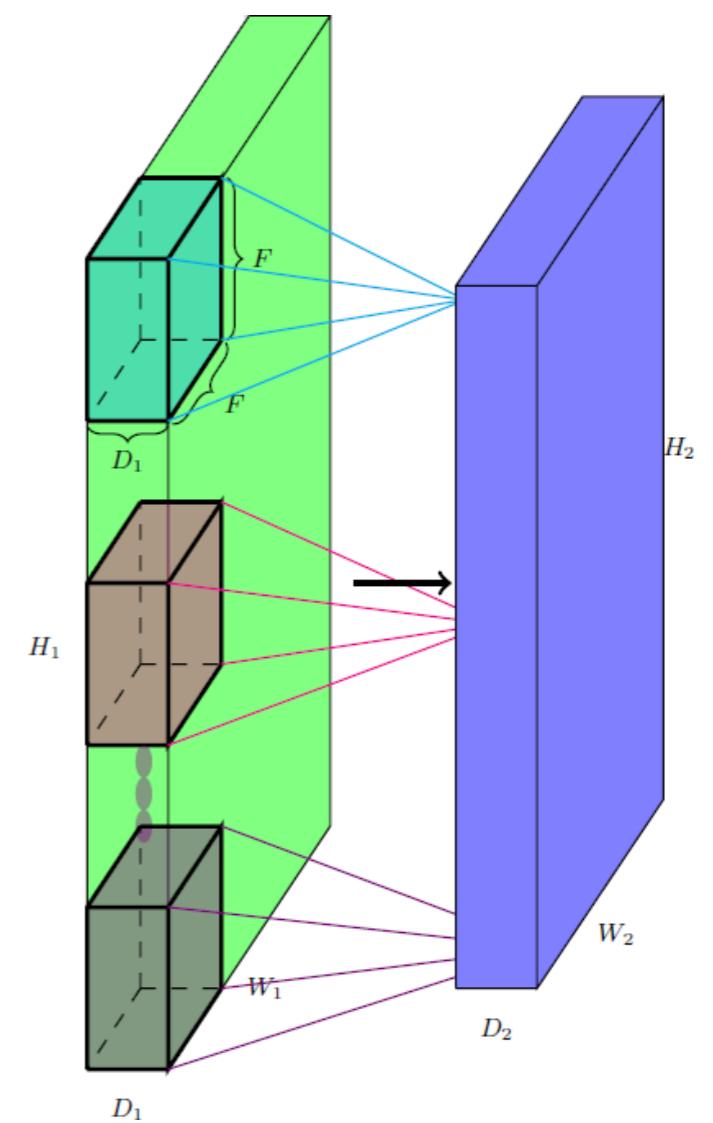
Convolution: Understanding the (Hyper)Parameters

- Input dimensions: Width (W_1) \times Height (H_1) \times Depth (D_1)
- Spatial extent (F) of each filter (the depth of each filter is same as the depth of input)
- Output dimensions is $W_2 \times H_2 \times D_2$ (we will soon see a formula for computing W_2, H_2 and D_2)



Convolution: Understanding the (Hyper)Parameters

- Input dimensions: Width (W_1) \times Height (H_1) \times Depth (D_1)
- Spatial extent (F) of each filter (the depth of each filter is same as the depth of input)
- Output dimensions is $W_2 \times H_2 \times D_2$ (we will soon see a formula for computing W_2, H_2 and D_2)
- Stride (S) (explained in following slides)
- Number of filters K

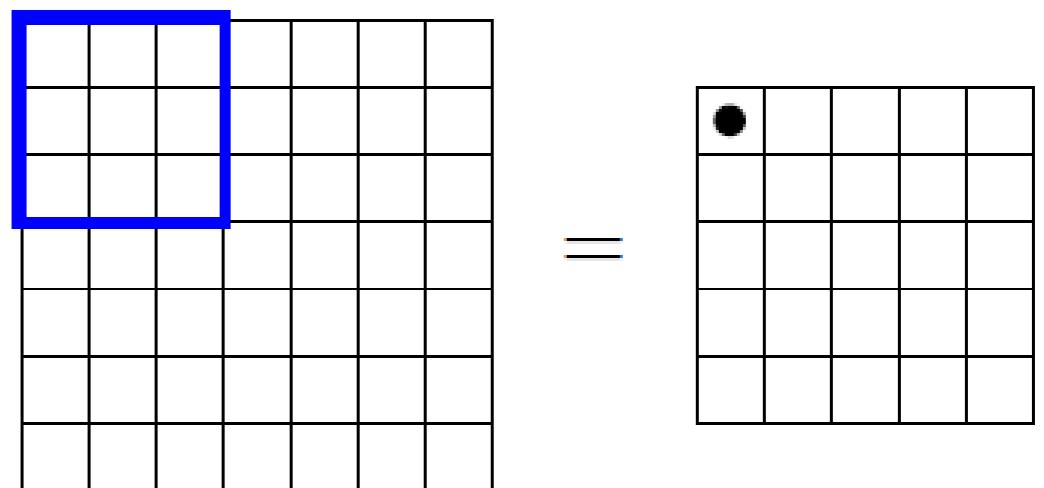


Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output

Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output

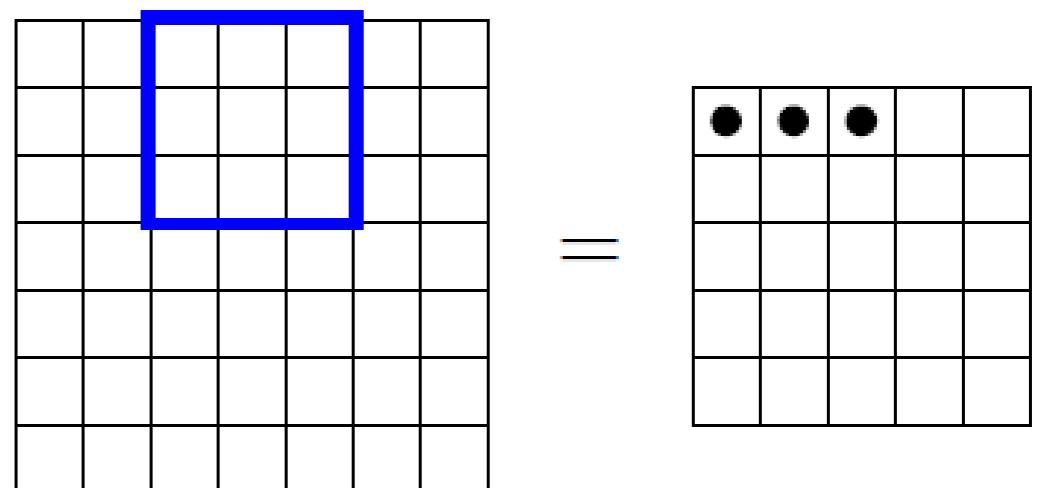


Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output

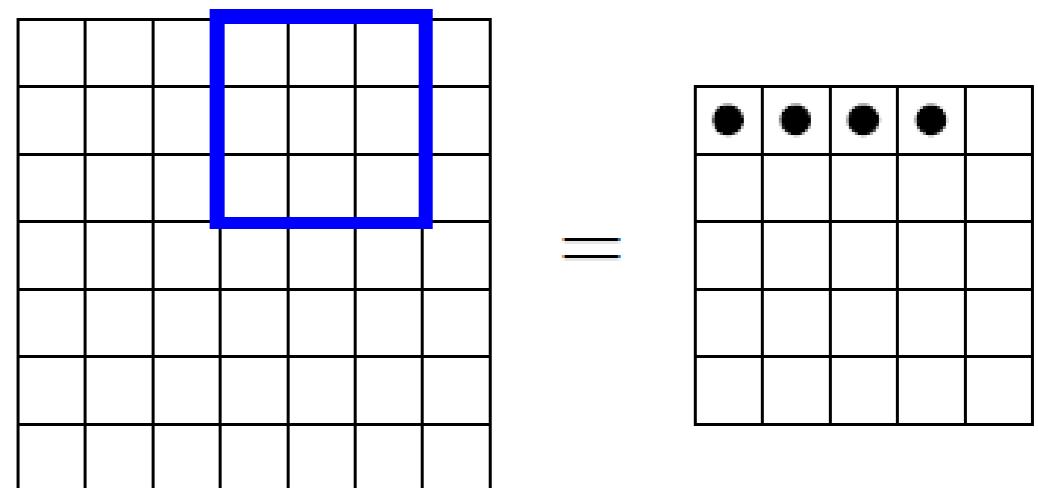
Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output



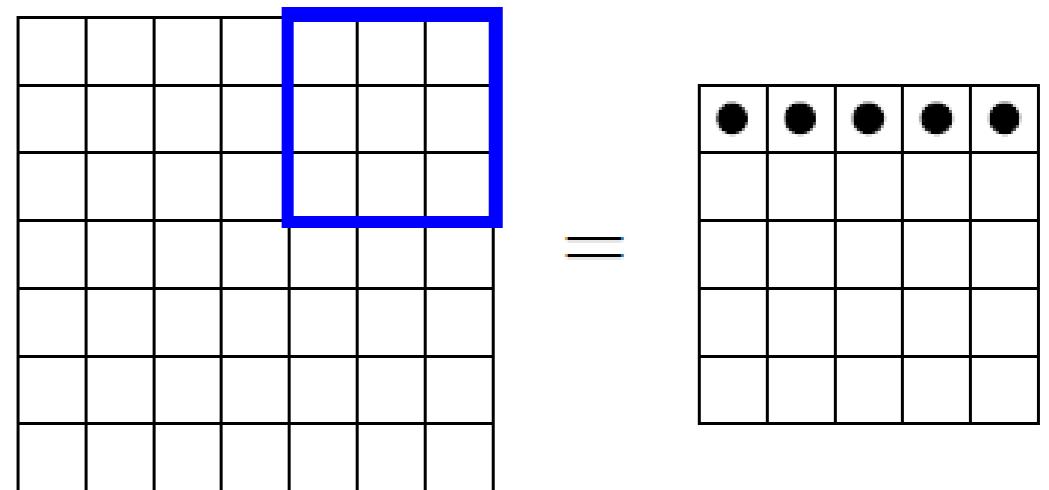
Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output



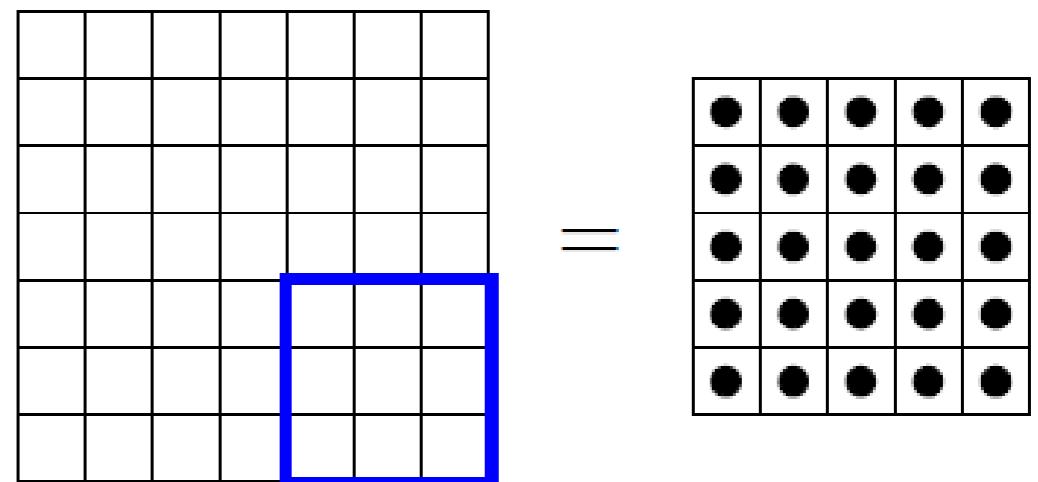
Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output



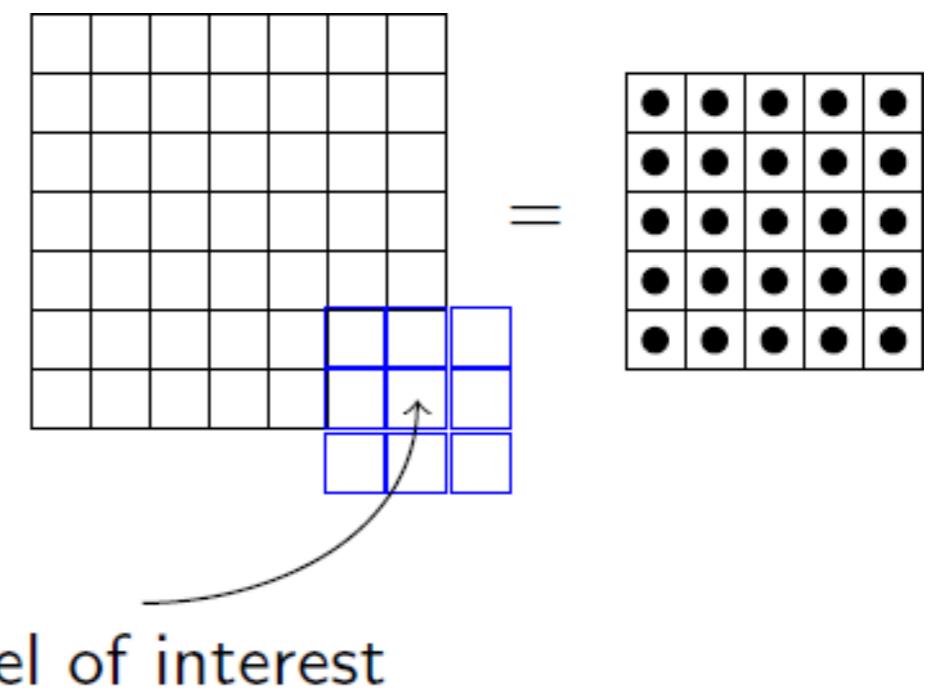
Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output



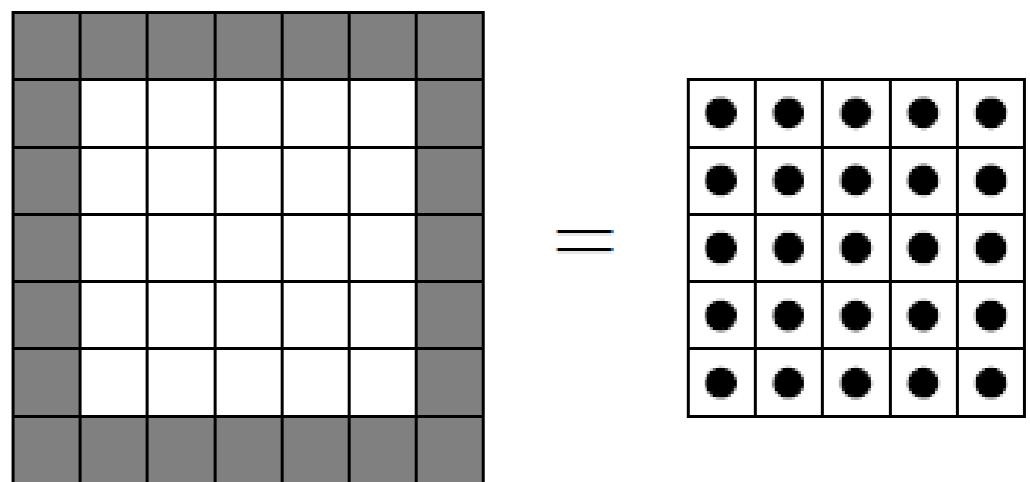
Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output
- we can't place the kernel at corners as it will cross the input boundary



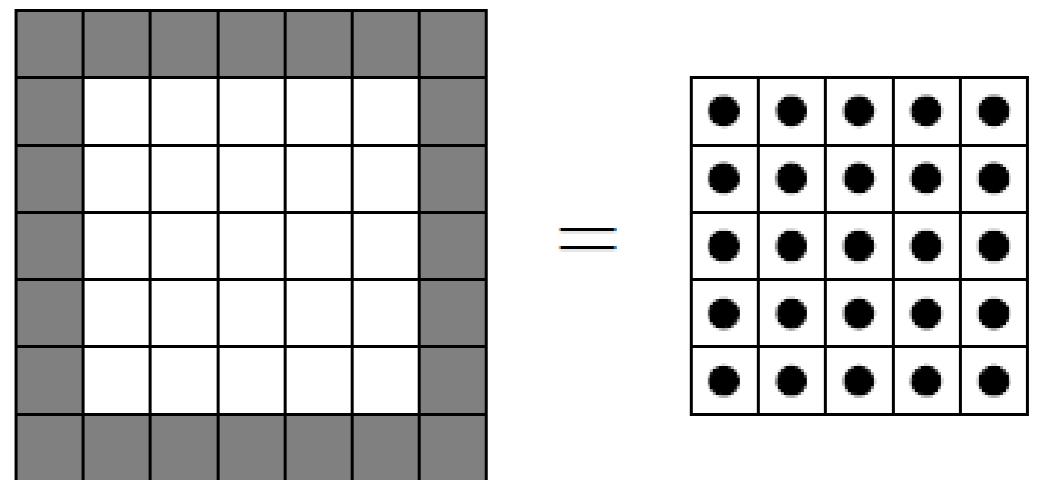
Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output
- we can't place the kernel at corners as it will cross the input boundary
- This is true for all shaded points (the kernel crosses the input boundary)



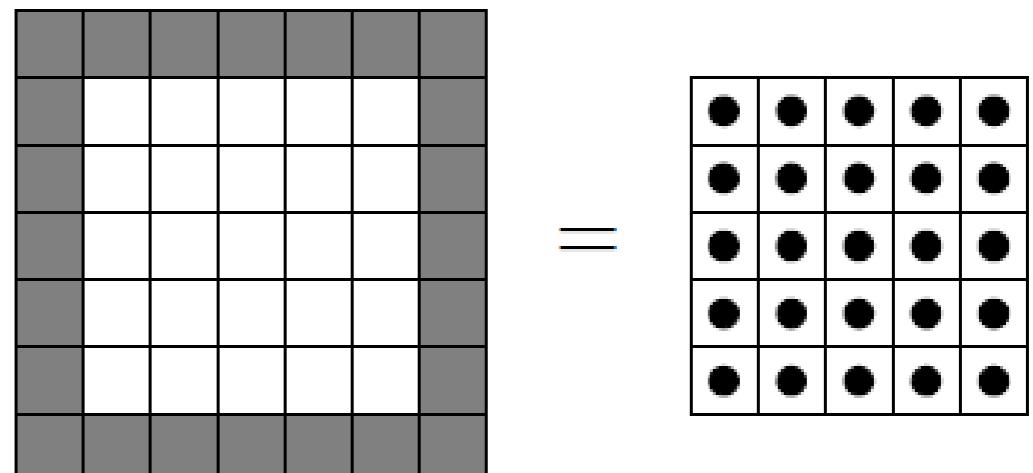
Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output
- we can't place the kernel at corners as it will cross the input boundary
- This is true for all shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than input



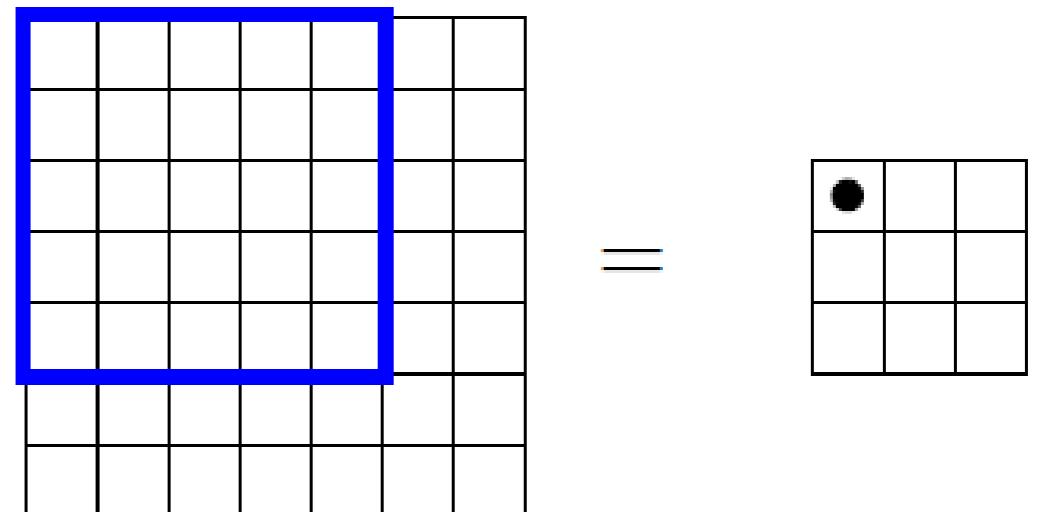
Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output
- Recall that we can't place the kernel at corners as it will cross the input boundary
- This is true for all shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than input
- As size of kernel increases, this becomes true for even more pixels



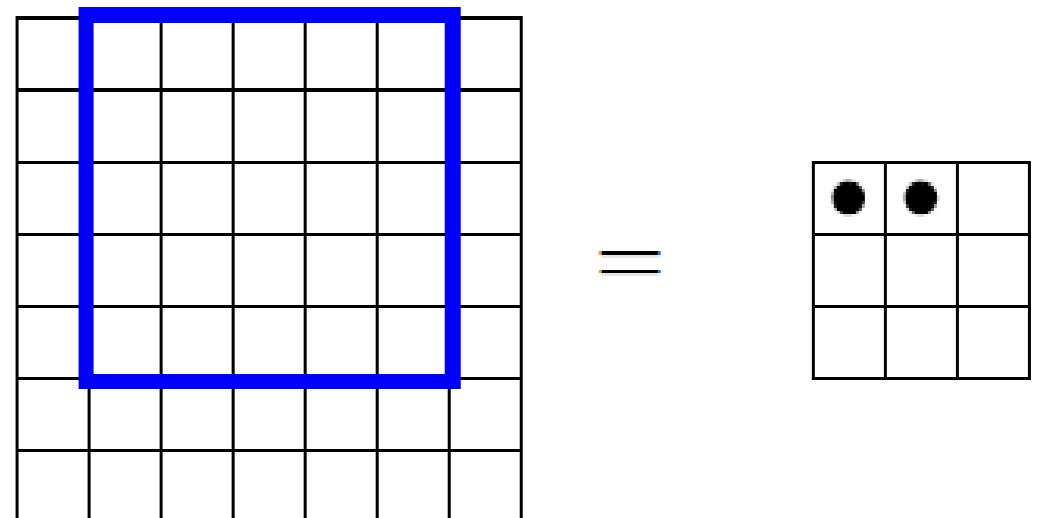
Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output
- Recall that we can't place the kernel at corners as it will cross the input boundary
- This is true for all shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than input
- As size of kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel



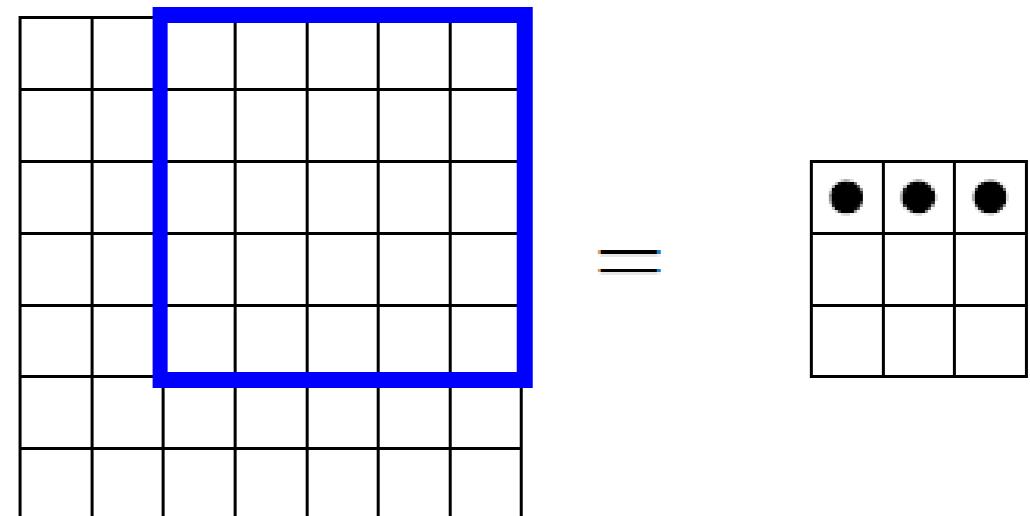
Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output
- Recall that we can't place the kernel at corners as it will cross the input boundary
- This is true for all shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than input
- As size of kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller output now



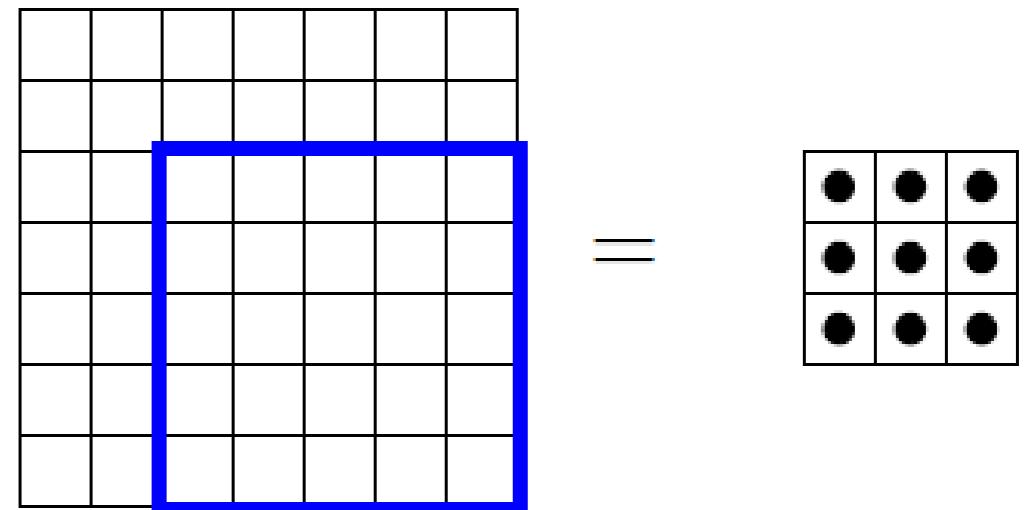
Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output
- Recall that we can't place the kernel at corners as it will cross the input boundary
- This is true for all shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than input
- As size of kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller output now



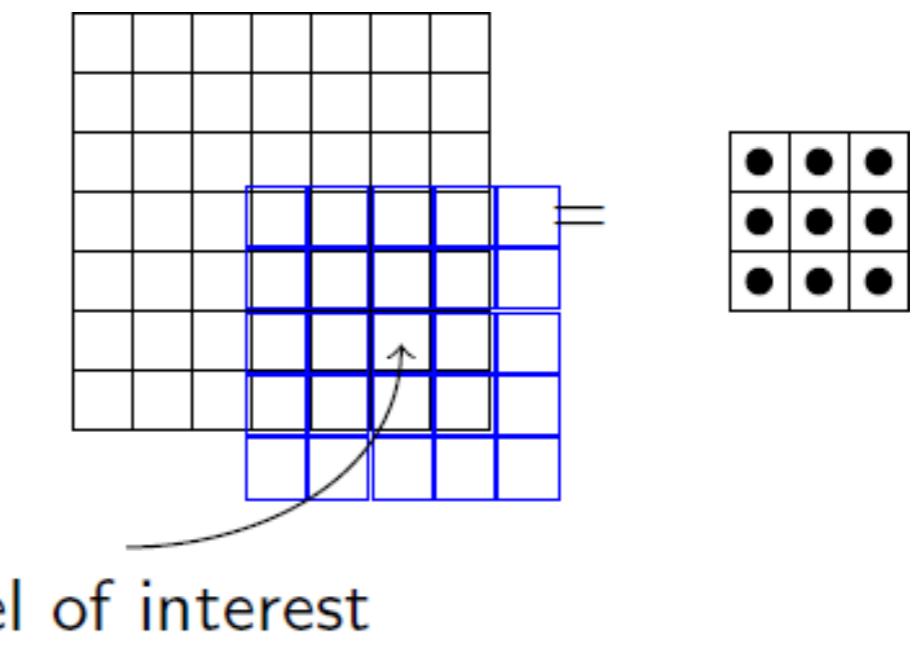
Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output
- Recall that we can't place the kernel at corners as it will cross the input boundary
- This is true for all shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than input
- As size of kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller output now



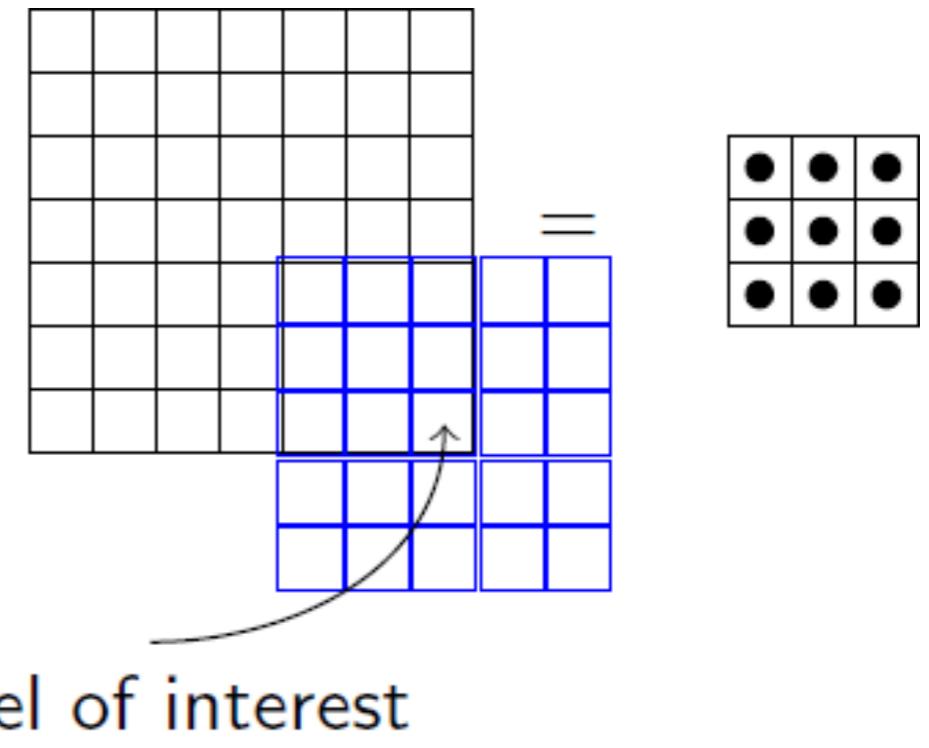
Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output
- Recall that we can't place the kernel at corners as it will cross the input boundary
- This is true for all shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than input
- As size of kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller output now



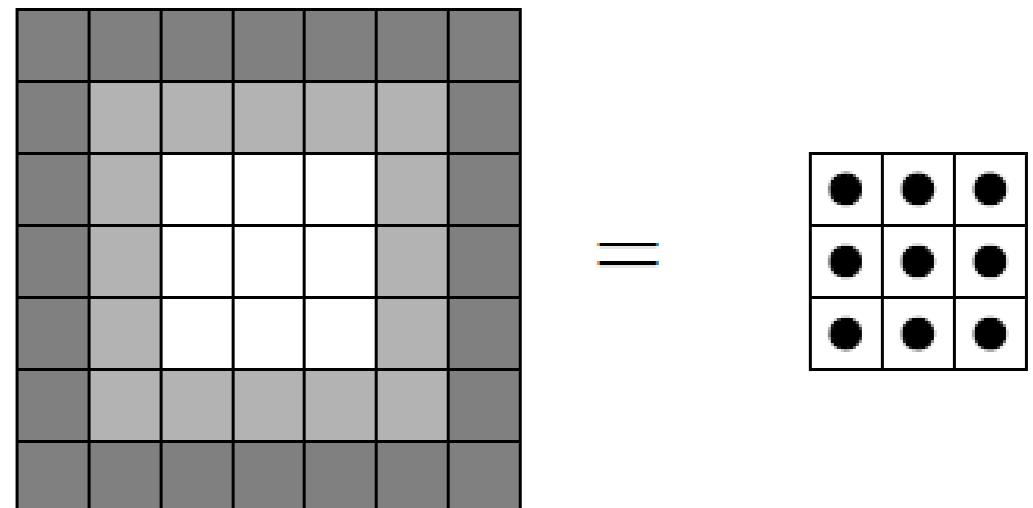
Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output
- Recall that we can't place the kernel at corners as it will cross the input boundary
- This is true for all shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than input
- As size of kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller output now



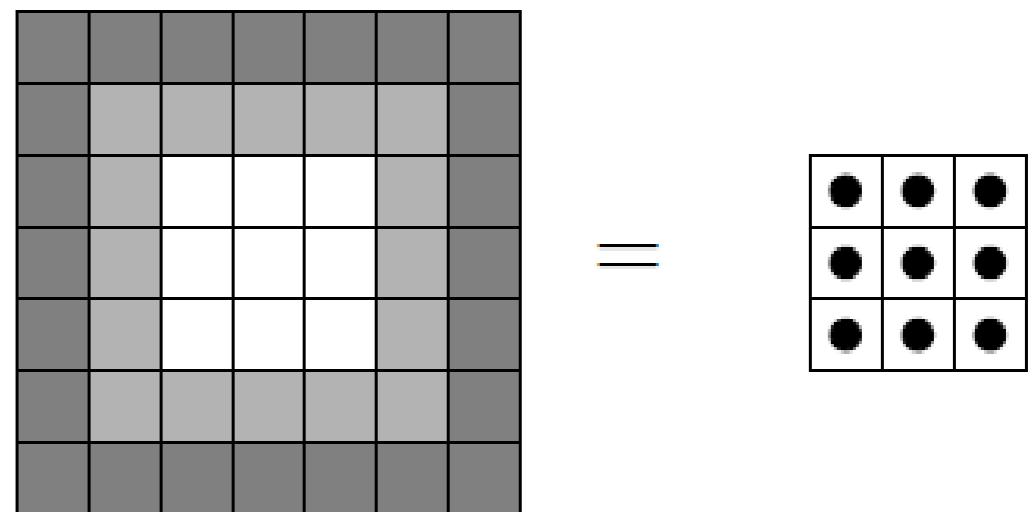
Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output
- Recall that we can't place the kernel at corners as it will cross the input boundary
- This is true for all shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than input
- As size of kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller output now



Convolution: Understanding the (Hyper)Parameters

- Let us compute dimensions (W_2, H_2) of output
- Recall that we can't place the kernel at corners as it will cross the input boundary
- This is true for all shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than input
- As size of kernel increases, this becomes true for even more pixels
- For example, let's consider a 5×5 kernel
- We have an even smaller output now



In general,

$$W_2 = W_1 - F + 1$$

$$H_2 = H_1 - F + 1$$

We will refine this formula further

Convolution: Understanding the (Hyper)Parameters

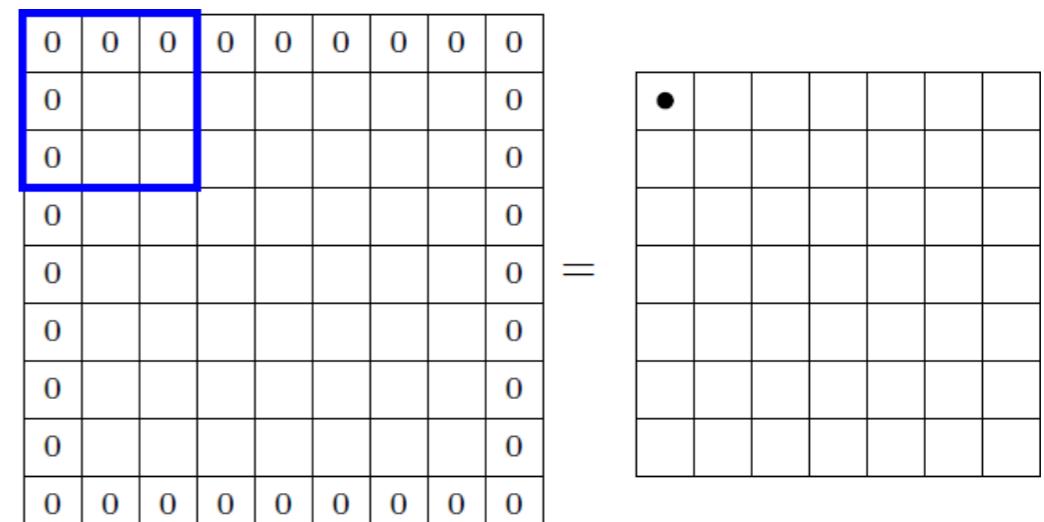
- What if we want output to be of same size as input?

Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners

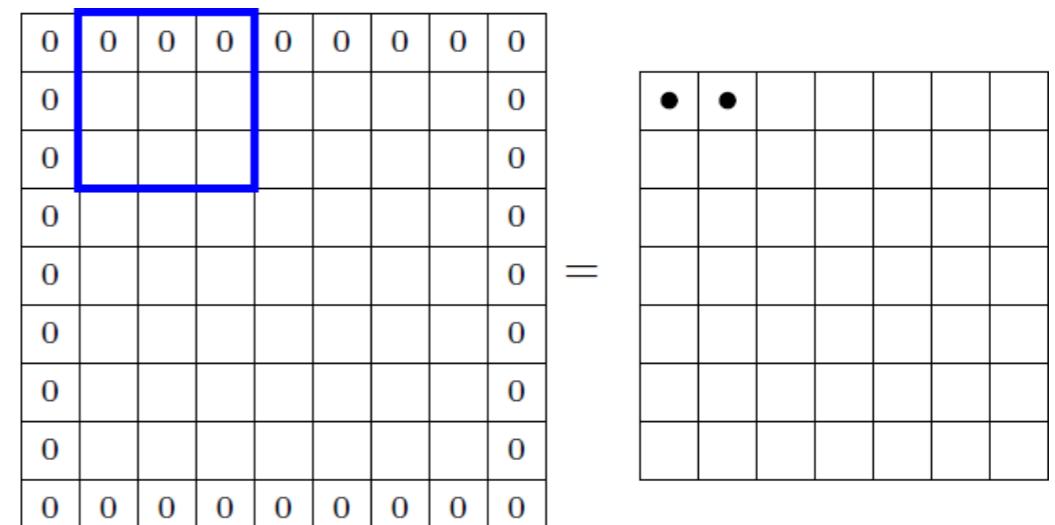
Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right;



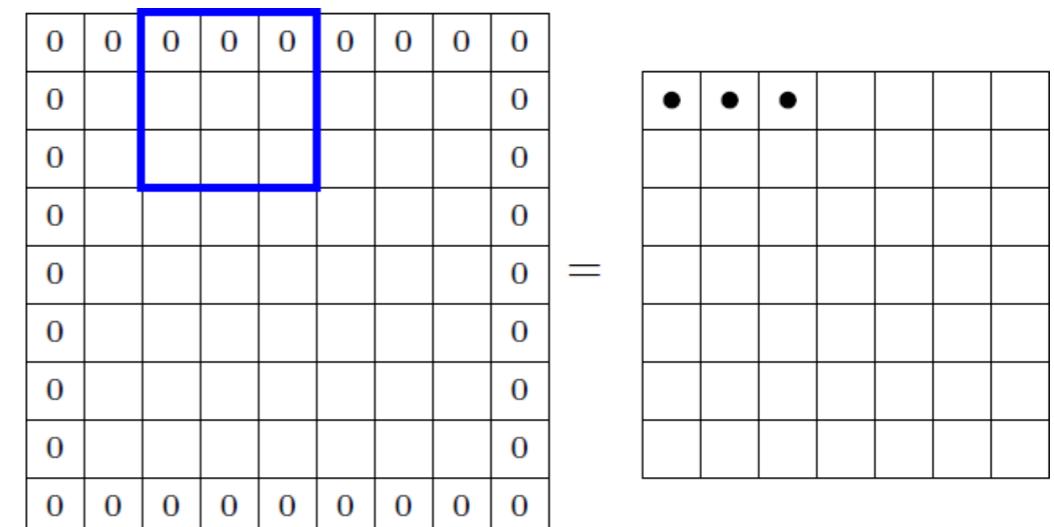
Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right;



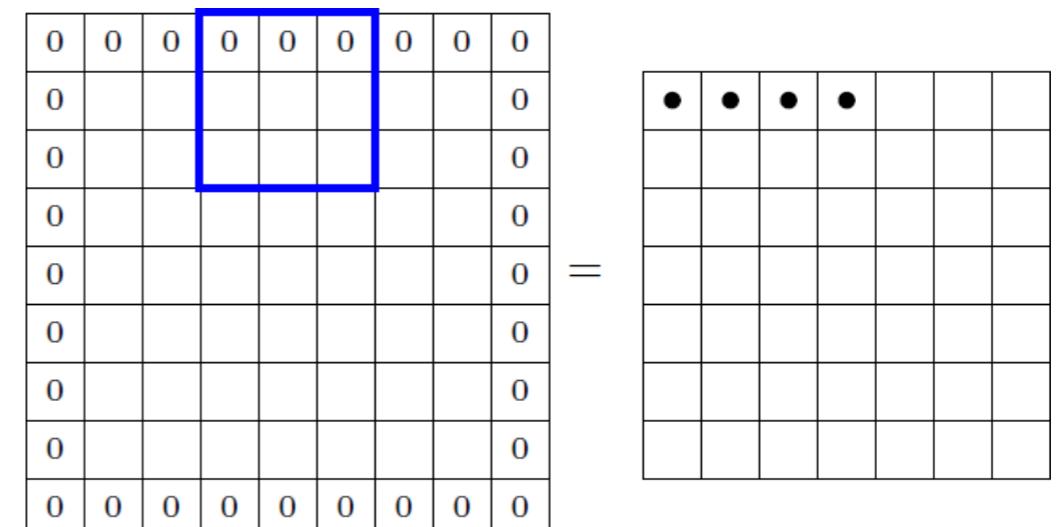
Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right;



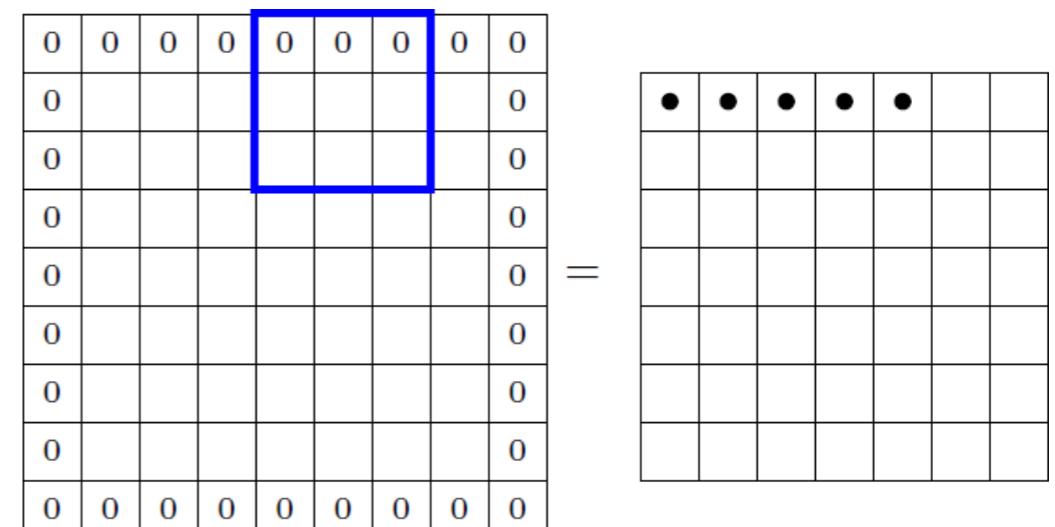
Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right;



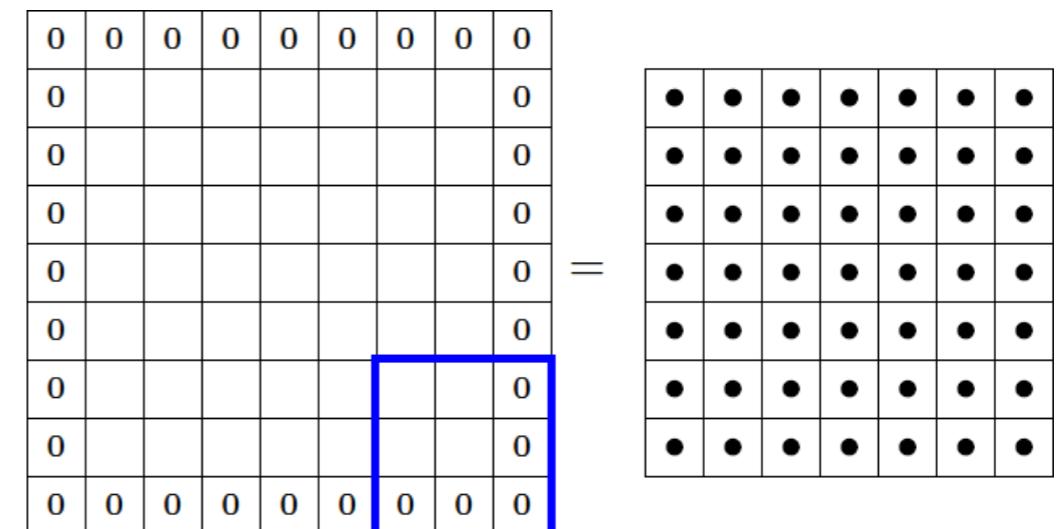
Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right;



Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right;



Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right;

Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right;

We now have:

$$W_2 = W_1 - F + 2P + 1$$

$$H_2 = H_1 - F + 2P + 1$$

We will refine this formula further

Convolution: Understanding the (Hyper)Parameters

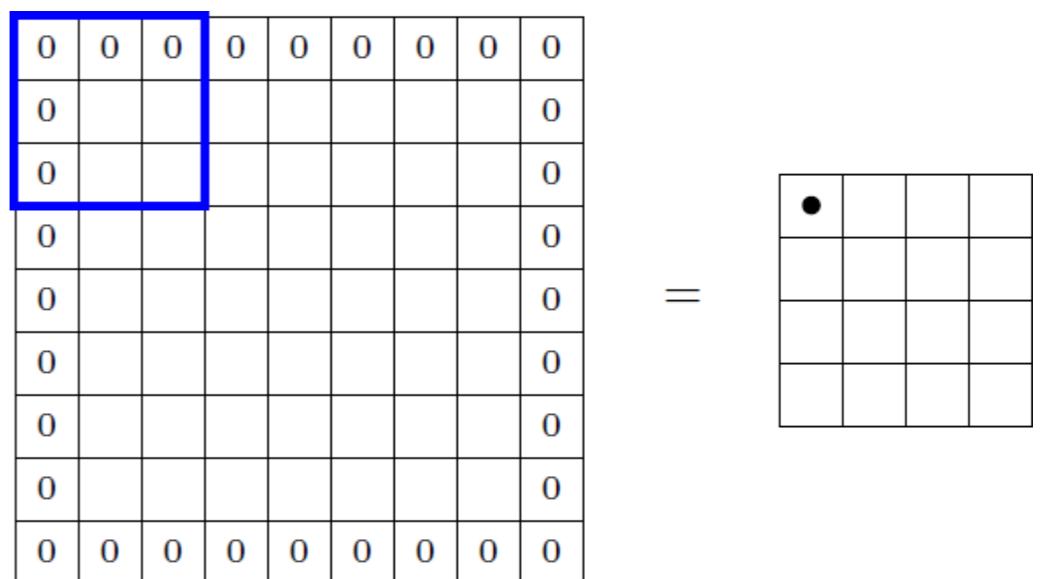
- What does **stride S** do?

Convolution: Understanding the (Hyper)Parameters

- What does **stride S** do?
- It defines the intervals at which the filter is applied (here $S = 2$)

Convolution: Understanding the (Hyper)Parameters

- What does **stride** S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions



Convolution: Understanding the (Hyper)Parameters

- What does **stride S** do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions

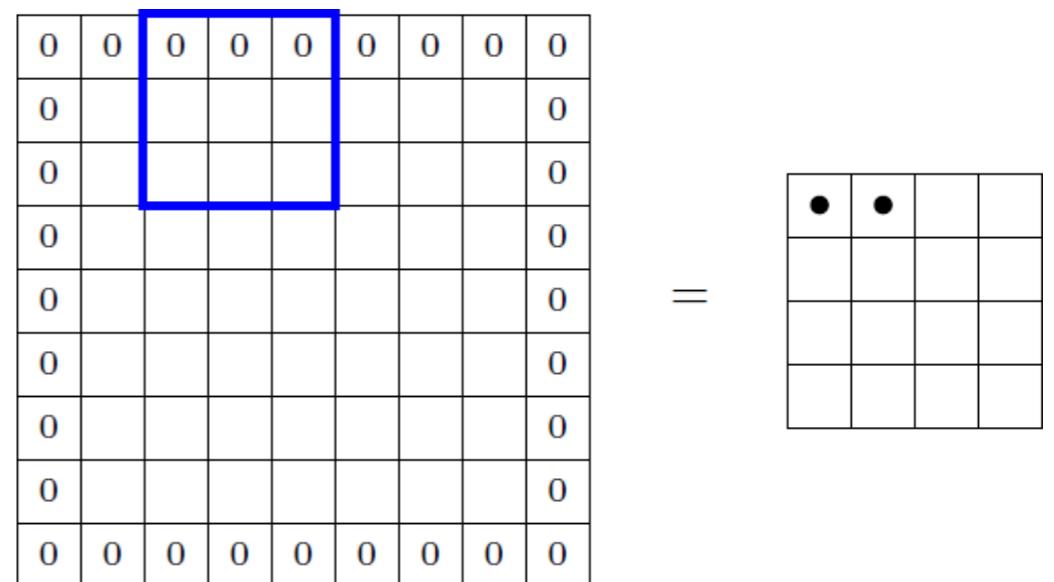
0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

=

•			

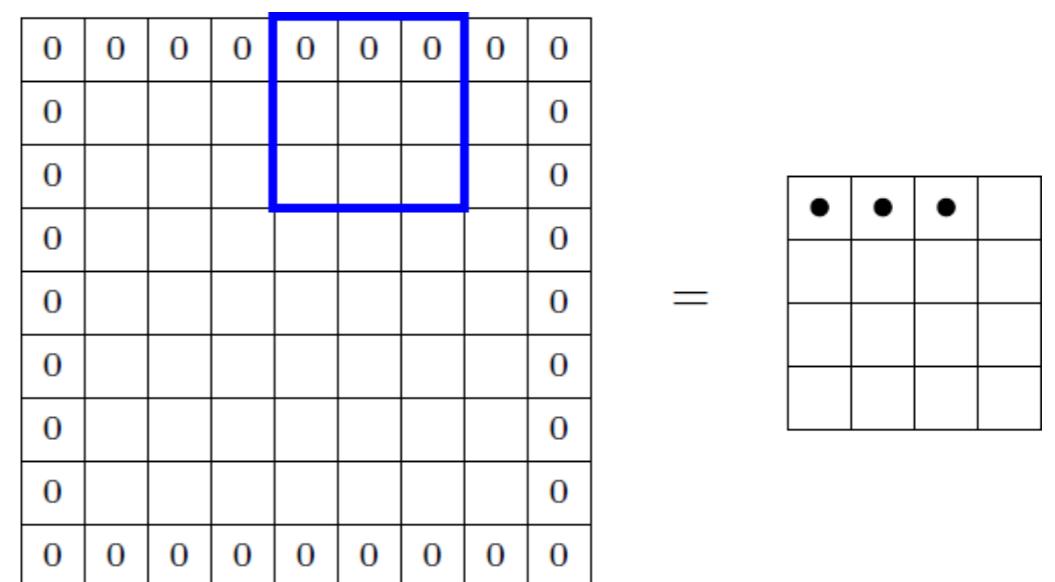
Convolution: Understanding the (Hyper)Parameters

- What does **stride S** do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions



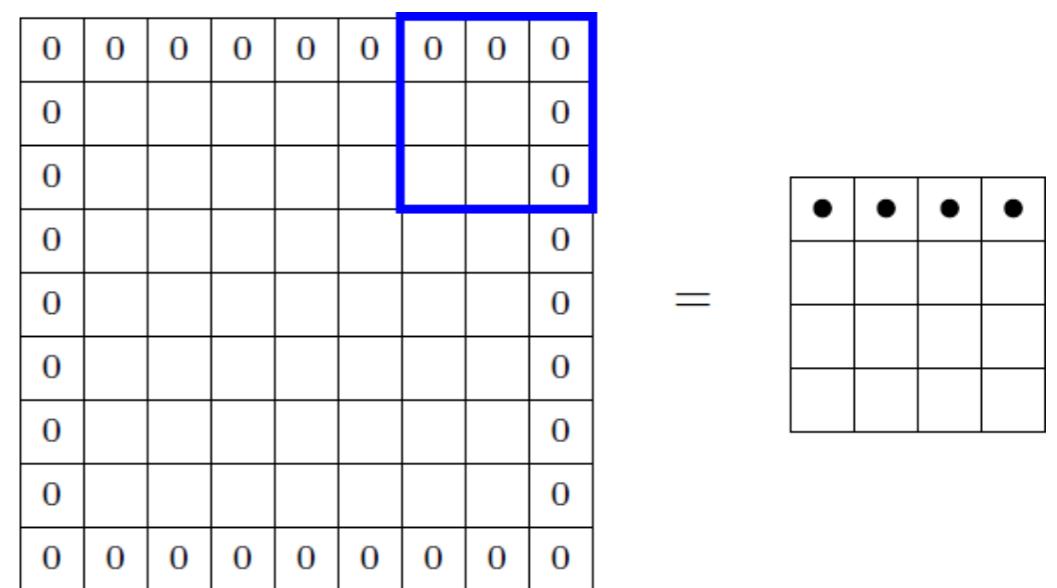
Convolution: Understanding the (Hyper)Parameters

- What does **stride S** do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions



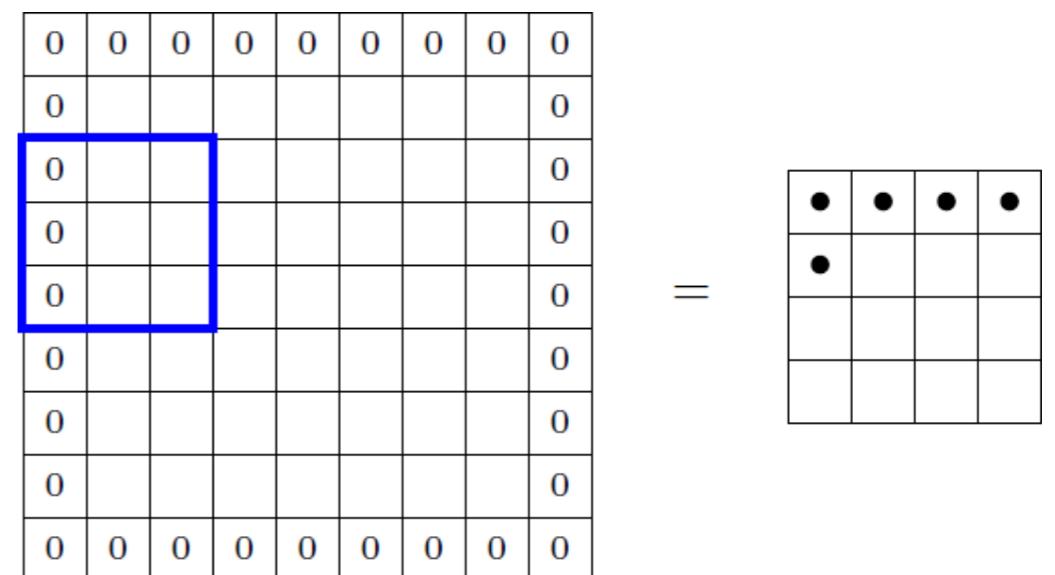
Convolution: Understanding the (Hyper)Parameters

- What does **stride S** do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions



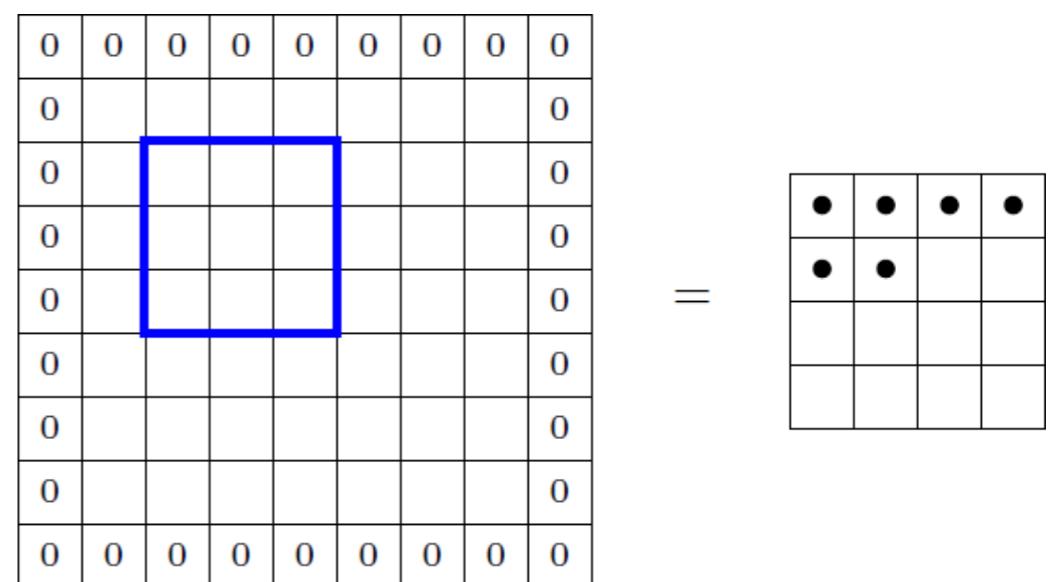
Convolution: Understanding the (Hyper)Parameters

- What does **stride S** do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions



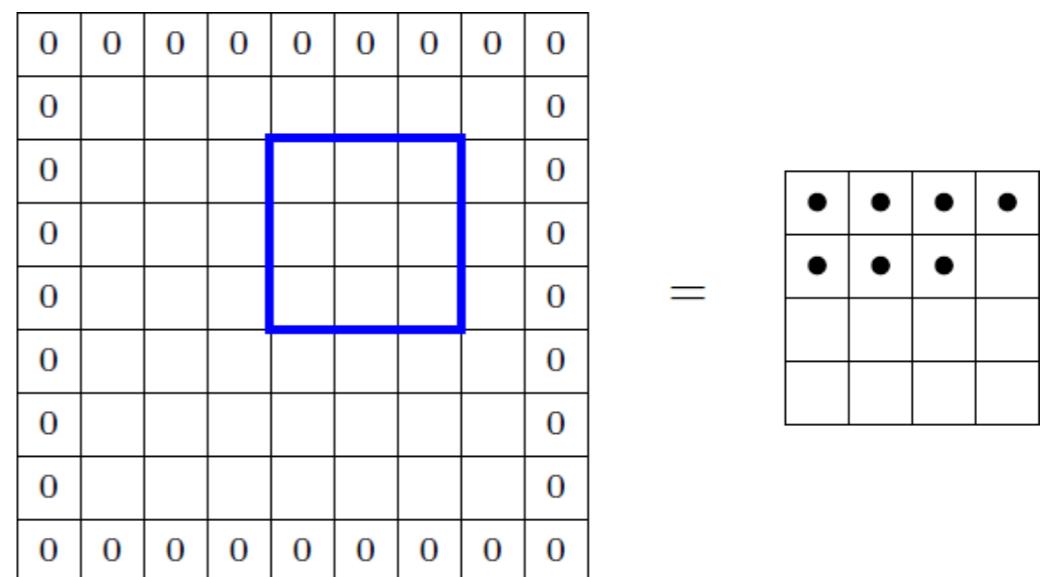
Convolution: Understanding the (Hyper)Parameters

- What does **stride** S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions



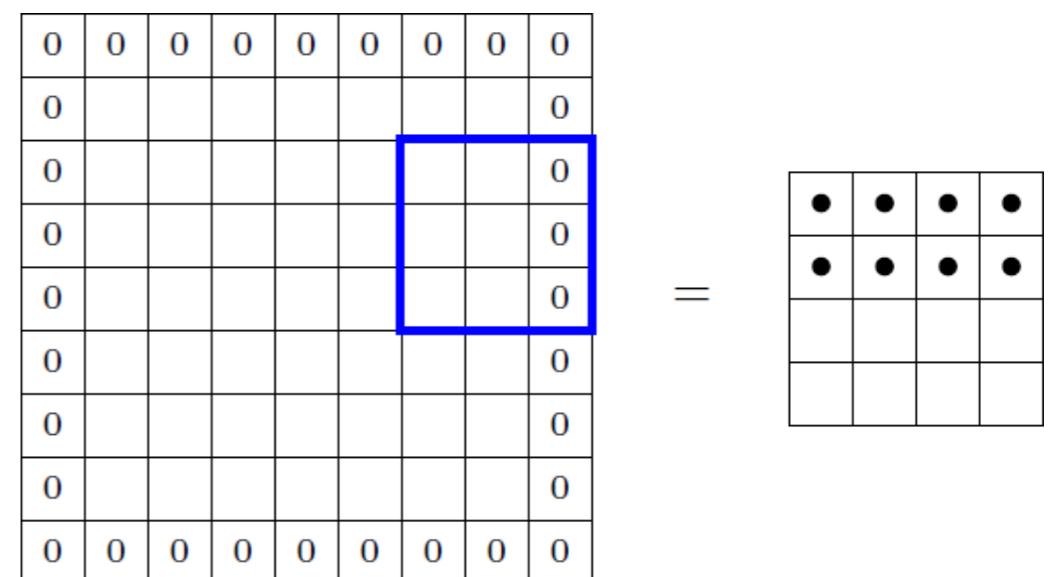
Convolution: Understanding the (Hyper)Parameters

- What does **stride S** do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions



Convolution: Understanding the (Hyper)Parameters

- What does **stride** S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions



Convolution: Understanding the (Hyper)Parameters

What does **stride S** do?

It defines the intervals at which the filter is applied (here $S = 2$)

Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

=

•	•	•	•
•	•	•	•
•			

Convolution: Understanding the (Hyper)Parameters

What does **stride** S do?

It defines the intervals at which the filter is applied (here $S = 2$)

Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

=

•	•	•	•
•	•	•	•
•	•		

Convolution: Understanding the (Hyper)Parameters

What does **stride** S do?

It defines the intervals at which the filter is applied (here $S = 2$)

Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

=

•	•	•	•
•	•	•	•
•	•	•	

Convolution: Understanding the (Hyper)Parameters

What does **stride S** do?

It defines the intervals at which the filter is applied
(here $S = 2$)

Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

=

•	•	•	•
•	•	•	•
•	•	•	•

Convolution: Understanding the (Hyper)Parameters

What does **stride** S do?

It defines the intervals at which the filter is applied (here $S = 2$)

Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

=

•	•	•	•
•	•	•	•
•	•	•	•
•	•	•	•

Convolution: Understanding the (Hyper)Parameters

- What does **stride** S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions

So our final formula should mostly look like,

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

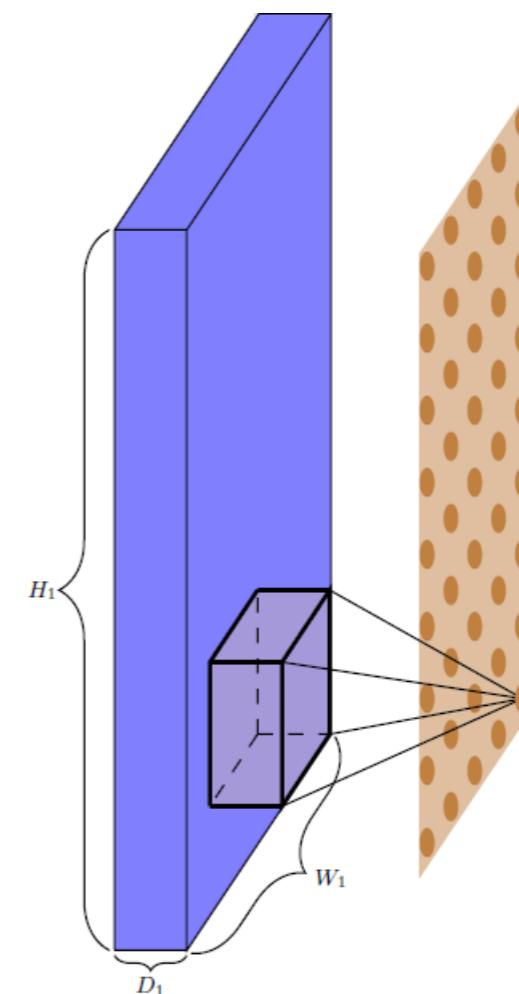
Not done yet, we will refine this formula further!

Convolution: Understanding the (Hyper)Parameters

- Finally, coming to depth of output

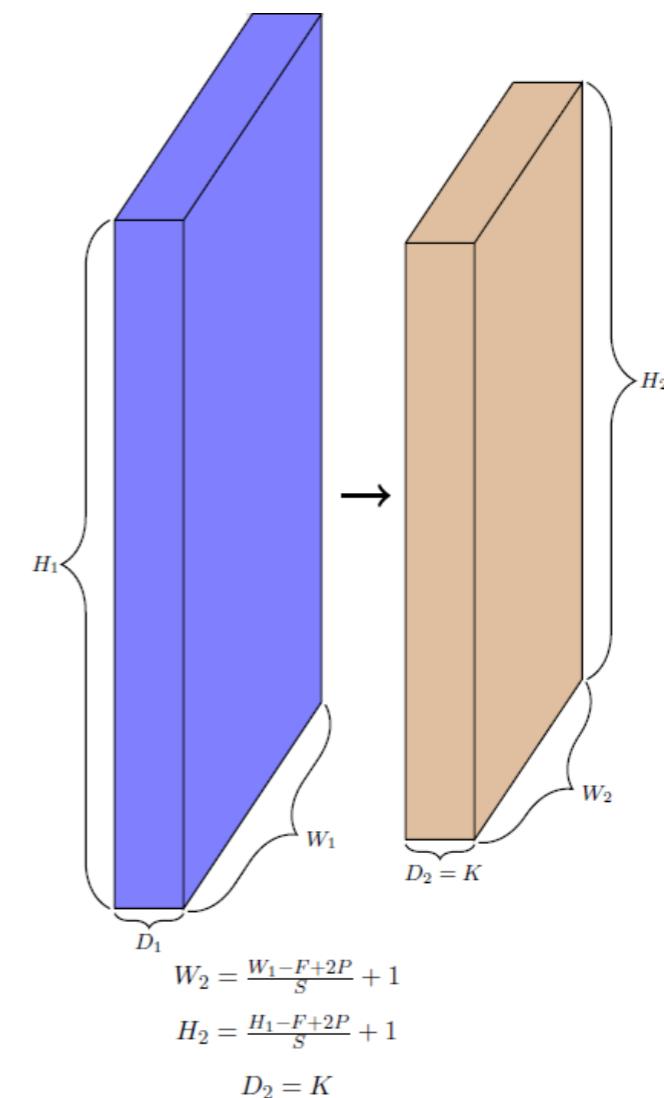
Convolution: Understanding the (Hyper)Parameters

- Finally, coming to depth of output
- Each filter gives us one 2D output



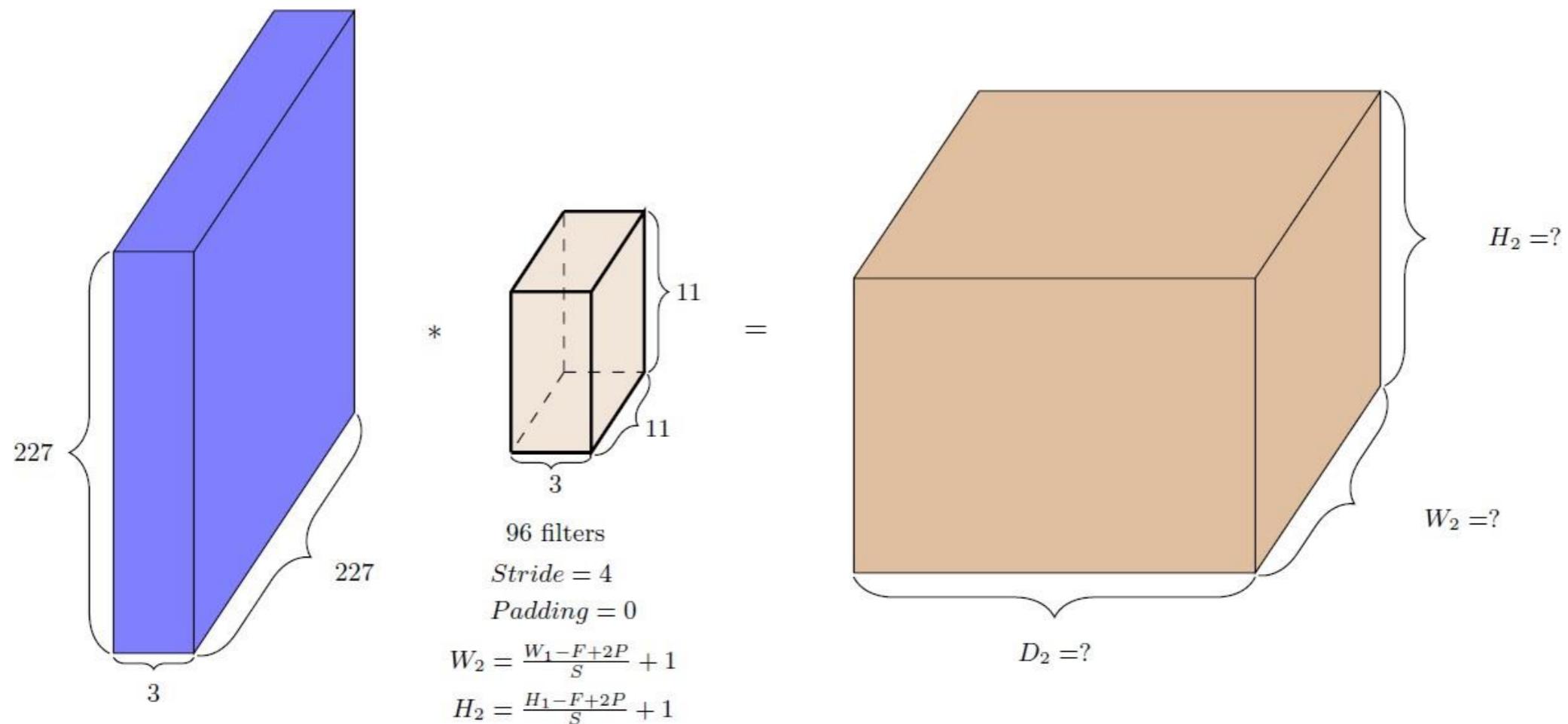
Convolution: Understanding the (Hyper)Parameters

- Finally, coming to depth of output
- Each filter gives us one 2D output
- K filters will give us K such 2D outputs
- We can think of resulting output as
- $K \times W_2 \times H_2$ volume
- Thus $D_2 = K$



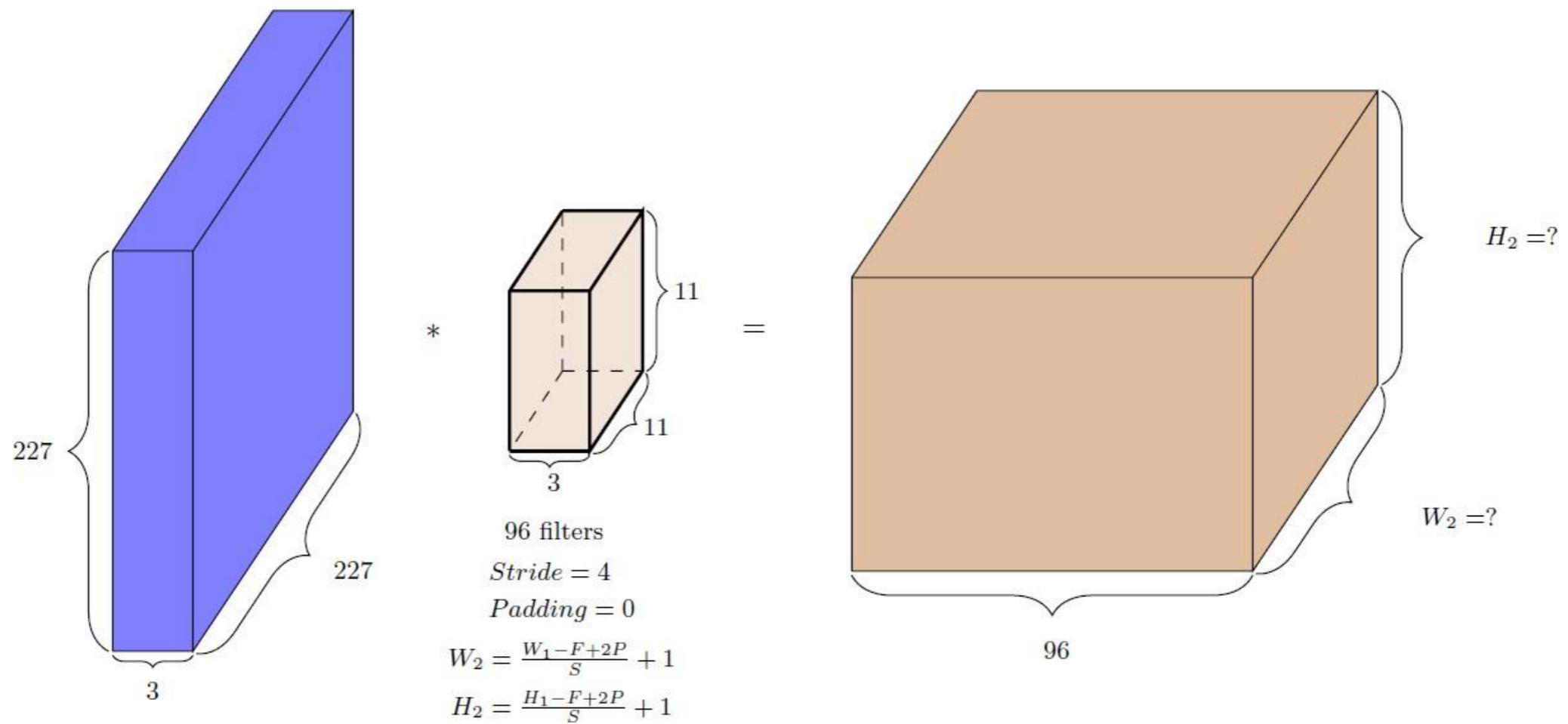
Quick Exercise

Work out output dimensions for the setting below!



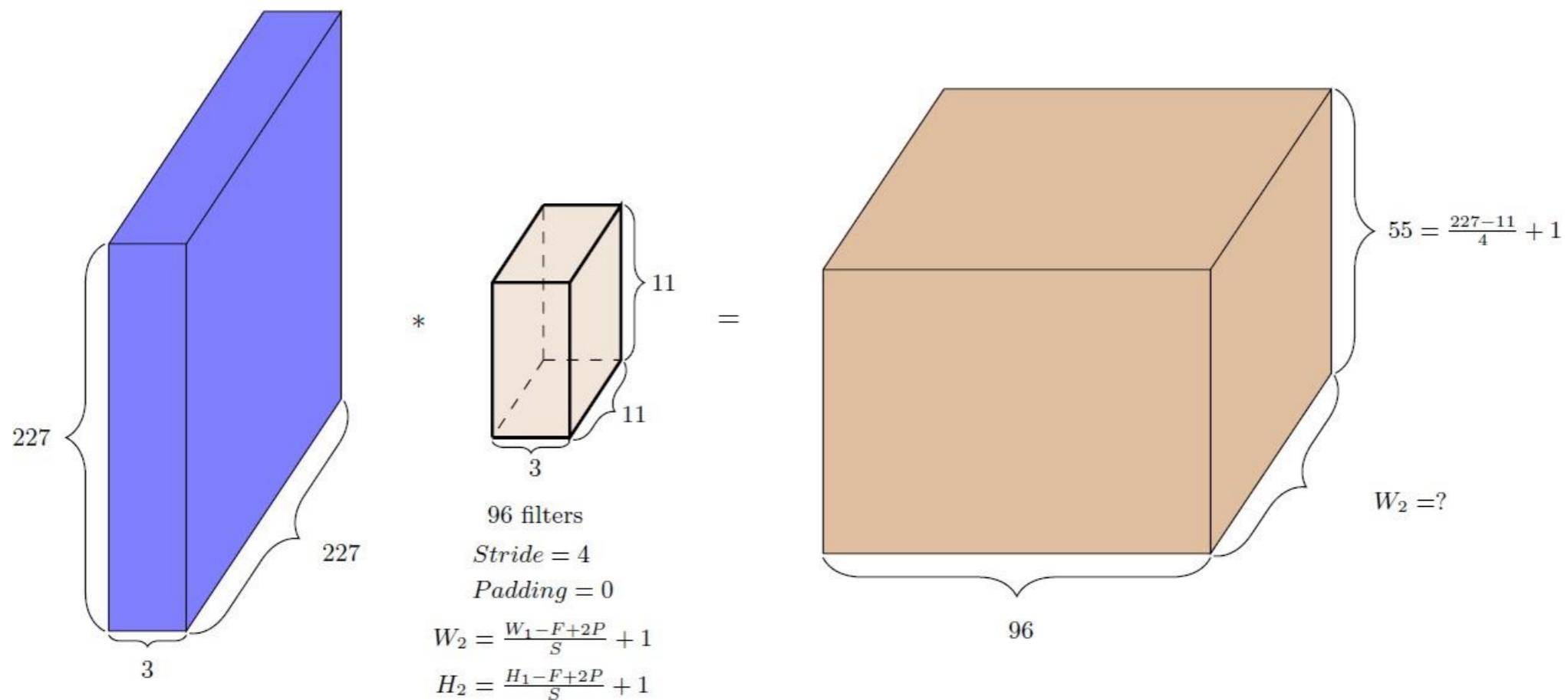
Quick Exercise

Work out output dimensions for the setting below!



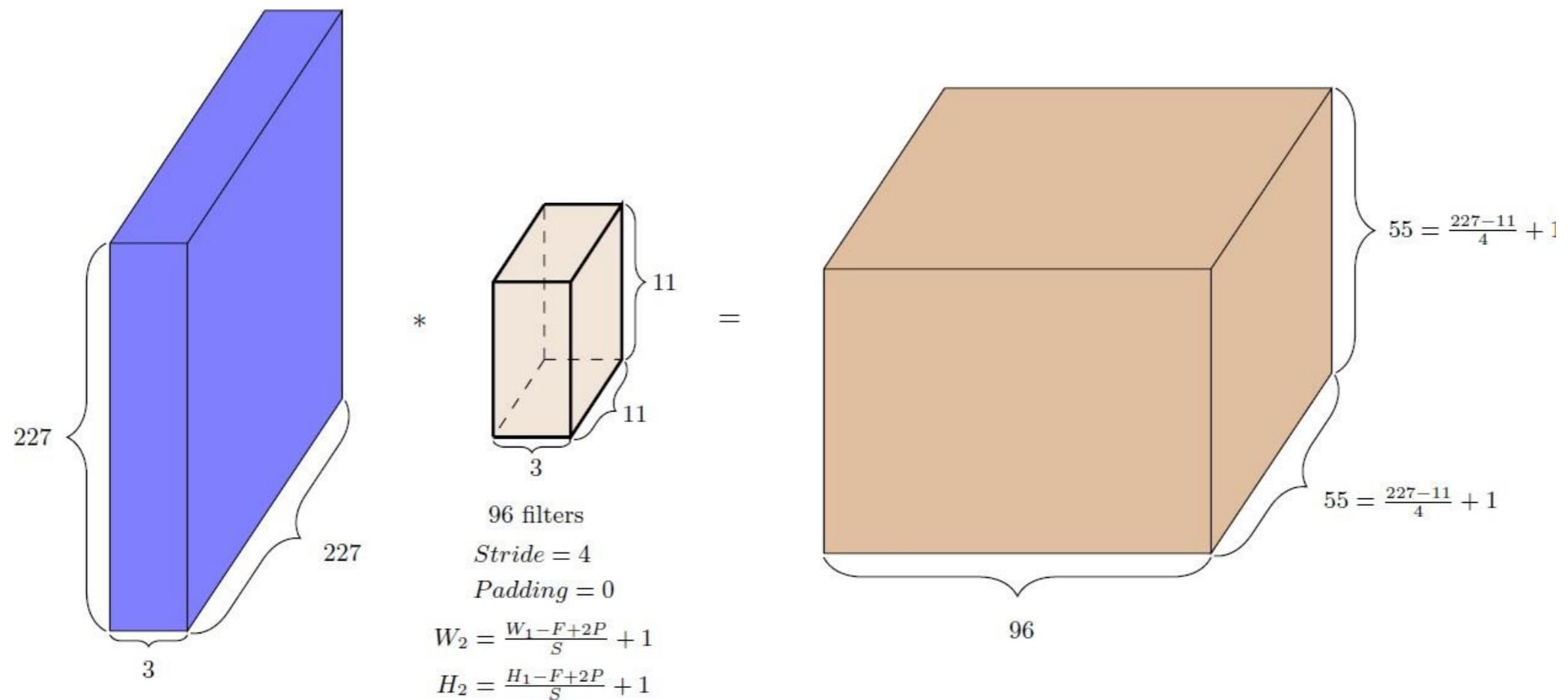
Quick Exercise

Work out output dimensions for the setting below!



Quick Exercise

Work out output dimensions for the setting below!



Pause and Ponder

- What is the connection between convolution and neural networks? Won't feedforward neural networks do?
- We will try to understand this by considering the task of "image classification"

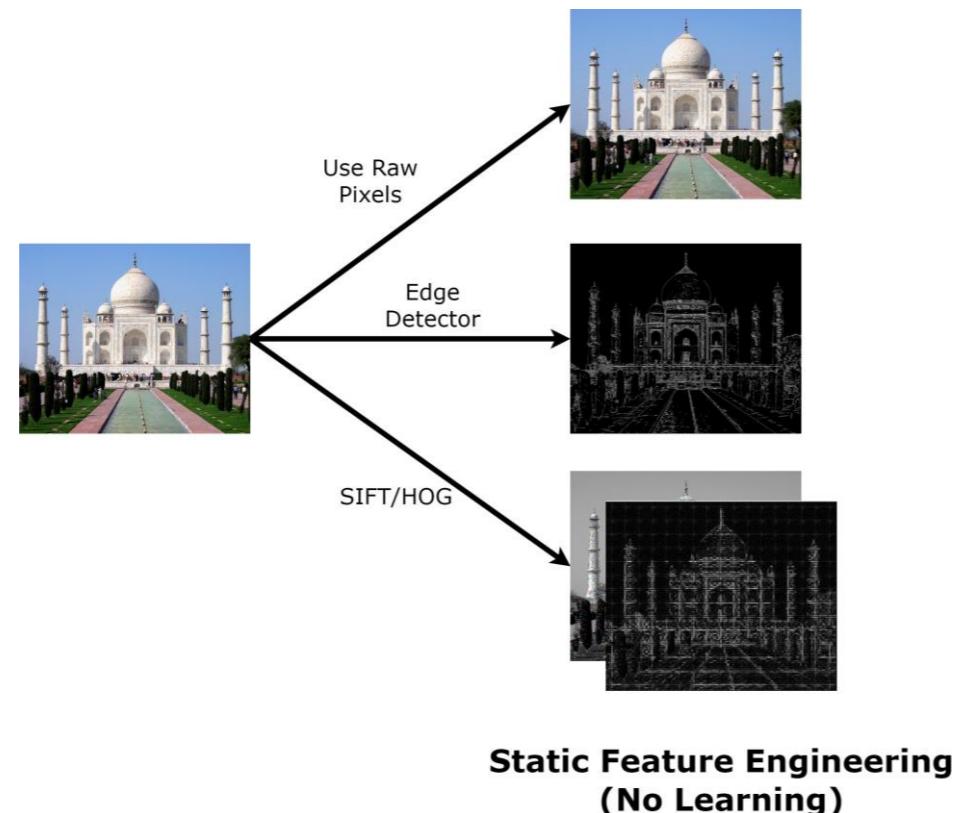
Convolution plays a fundamental role in image classification tasks and is particularly essential in Convolutional Neural Networks (CNNs). Unlike feedforward neural networks, which process each pixel independently, CNNs leverage convolution layers to detect patterns within small regions of an image, preserving the spatial structure.

CNNs apply convolution to extract low-level features like edges in early layers, gradually recognizing high-level features such as textures, shapes, or even objects in deeper layers. This enables CNNs to automatically learn hierarchical representations, making them superior to feedforward networks for tasks like facial recognition, self-driving cars, and medical imaging.

By maintaining spatial relationships within an image, CNNs efficiently recognize objects regardless of orientation or position, making convolution a powerful technique in modern AI-driven vision systems.

Traditional Machine Learning for Vision

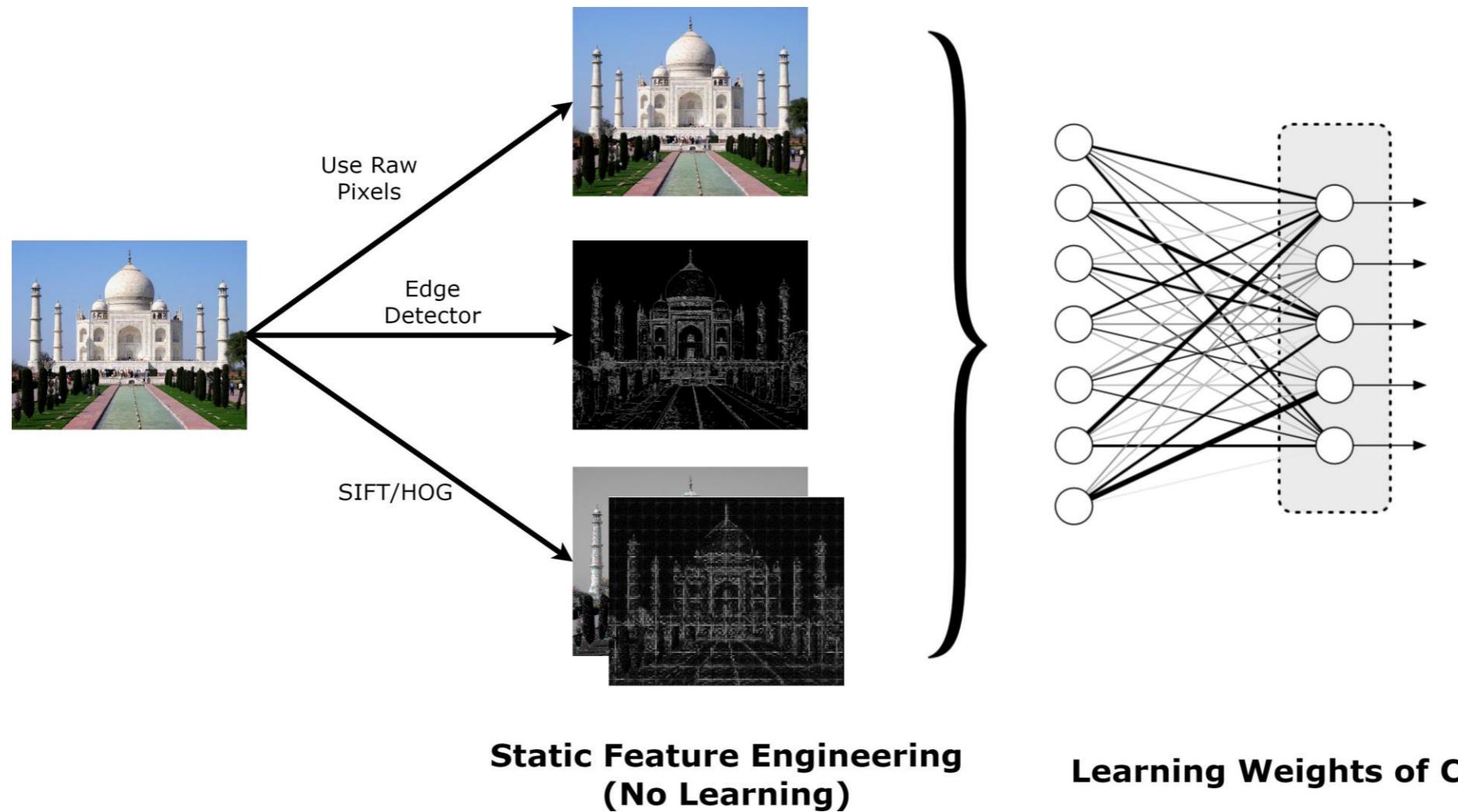
- Traditional ML-based computer vision solutions involve static feature engineering from images (e.g. recall SIFT, LBP, HoG, etc)
- Though effective, static feature engineering was a bottleneck of pre-DL vision solutions



SIFT - Scale Invariant Feature Transform
LBP - Local Binary Patterns
HoG - Histogram of Oriented Gradients

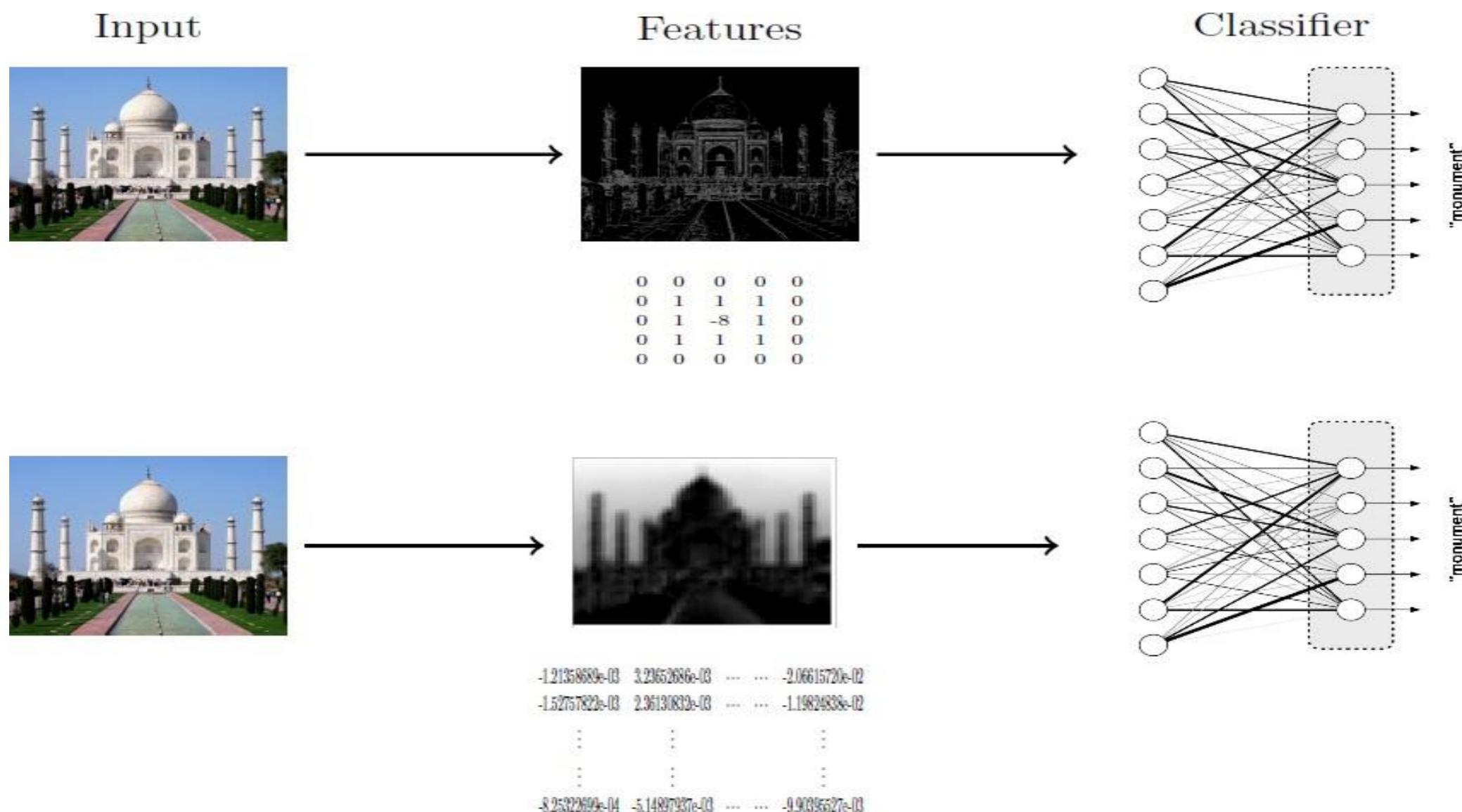
Traditional Machine Learning for Vision

- Traditional ML-based computer vision solutions involve static feature engineering from images (e.g. recall SIFT, LBP, HoG, etc)
- Though effective, static feature engineering was a bottleneck of pre-DL vision solutions



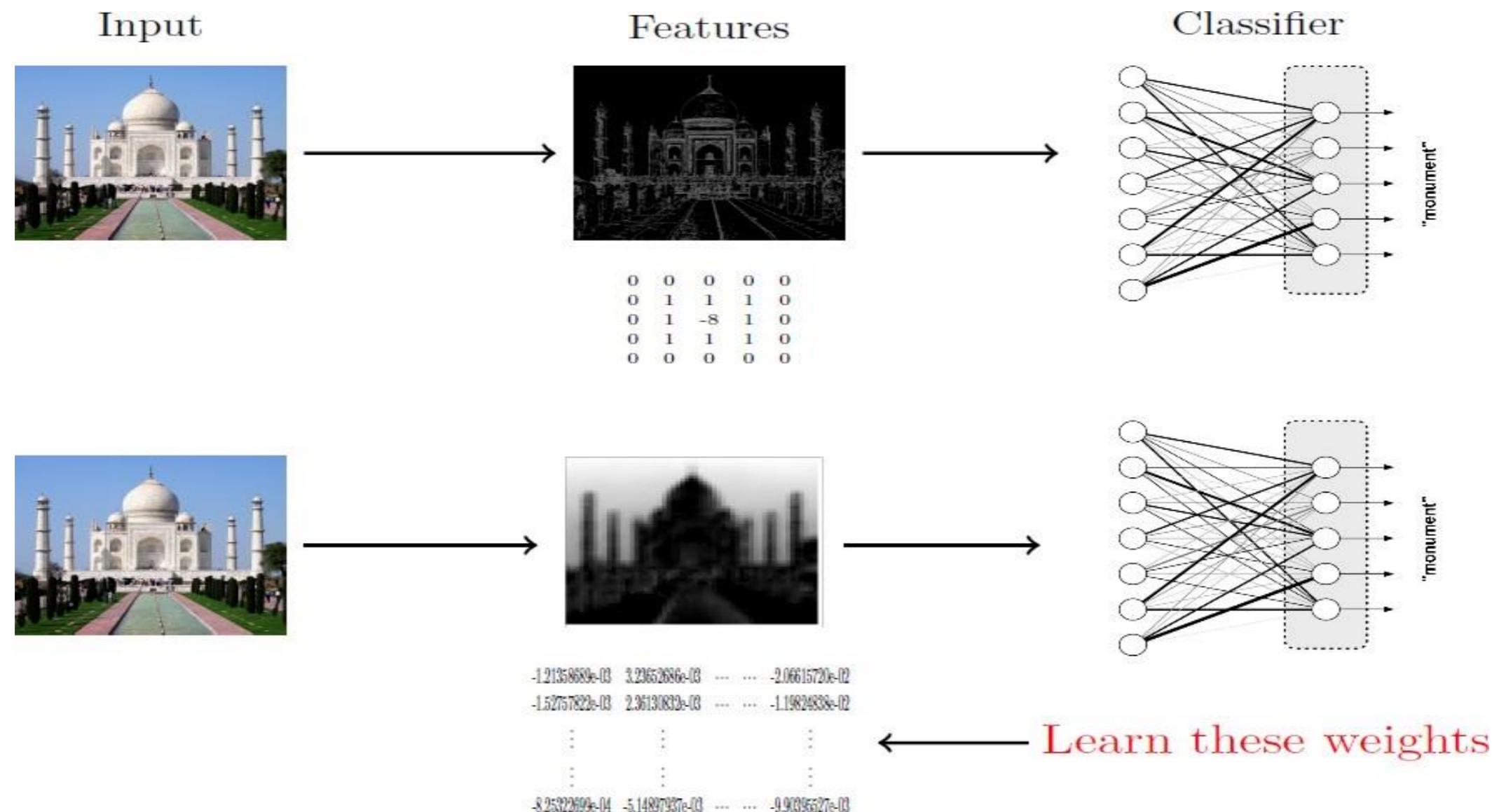
Beyond Static Feature Engineering

- Instead of using handcrafted kernels such as edge detectors can we **learn meaningful kernels/filters** in addition to learning the weights of the classifier?



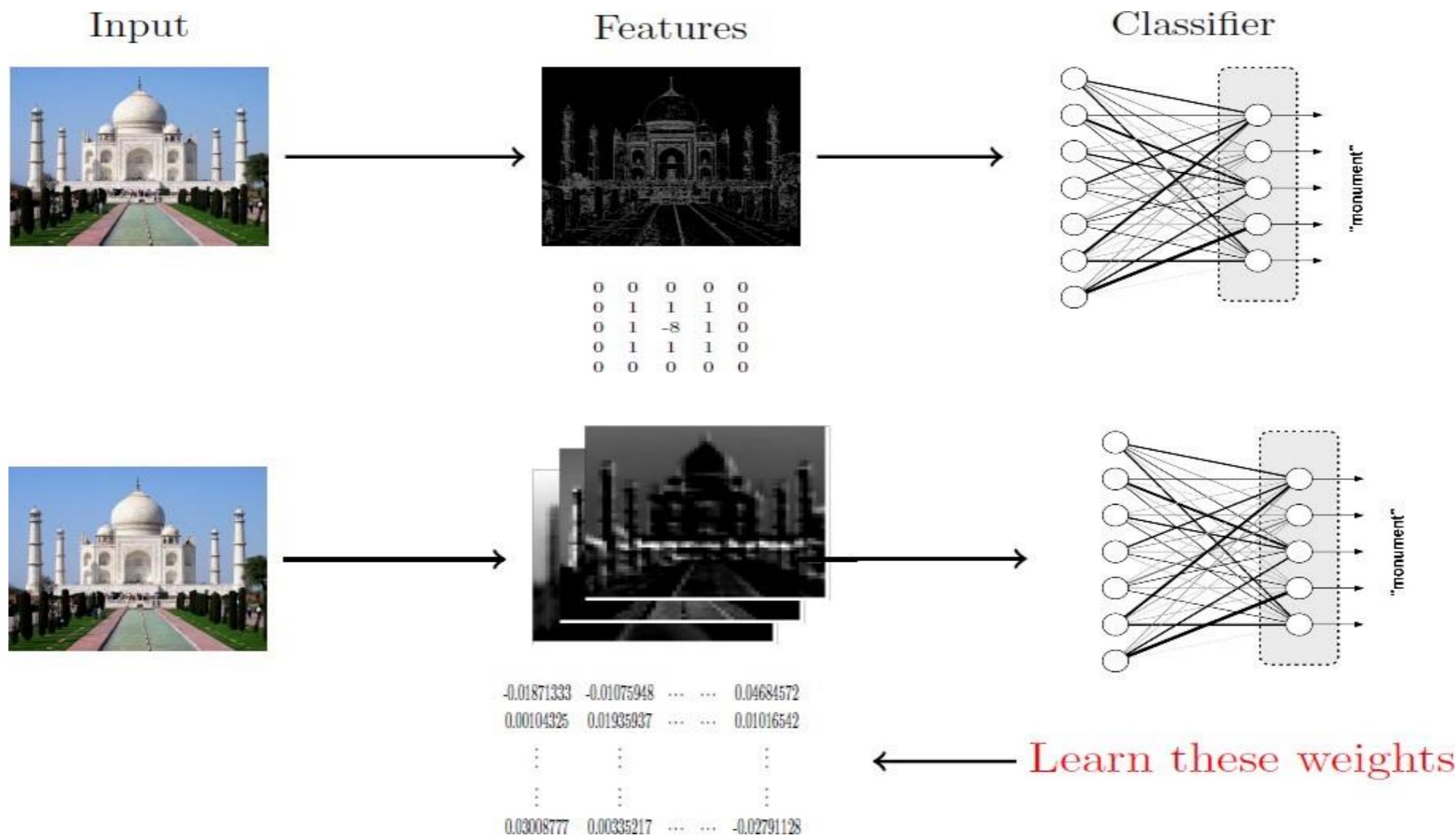
Beyond Static Feature Engineering

- Instead of using handcrafted kernels such as edge detectors can we **learn meaningful kernels/filters** in addition to learning the weights of the classifier?



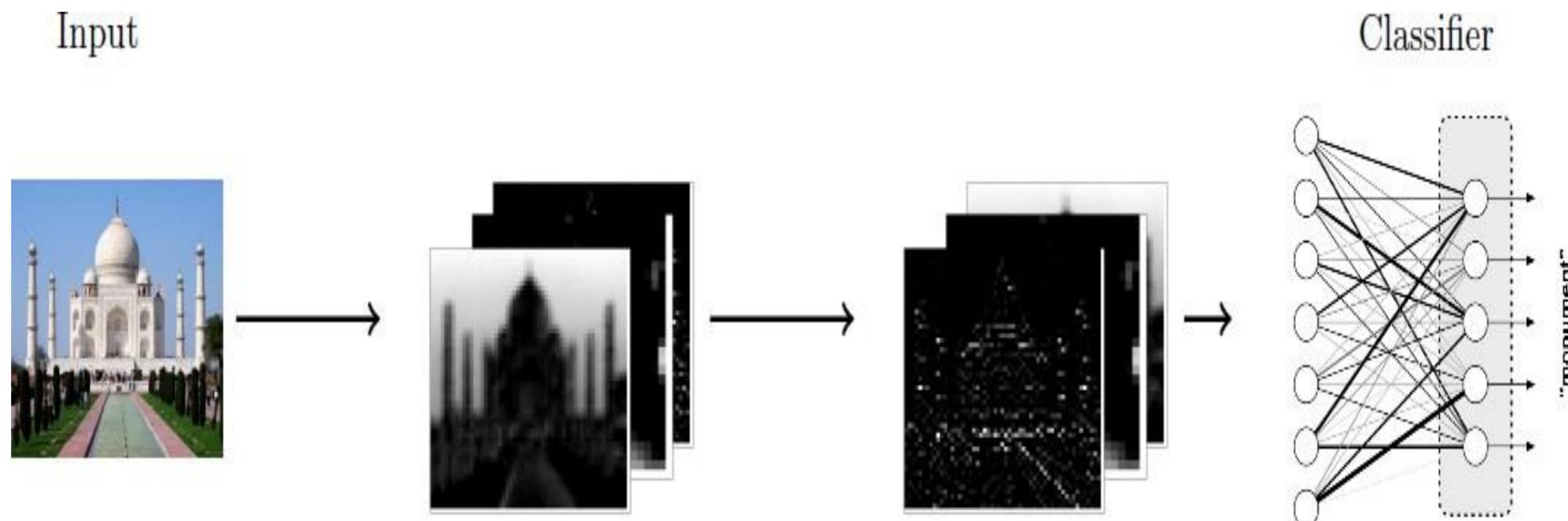
Beyond Static Feature Engineering

- Even better: Instead of using handcrafted kernels such as edge detectors can we learn **multiple meaningful kernels/filters** in addition to learning the weights of the classifier?



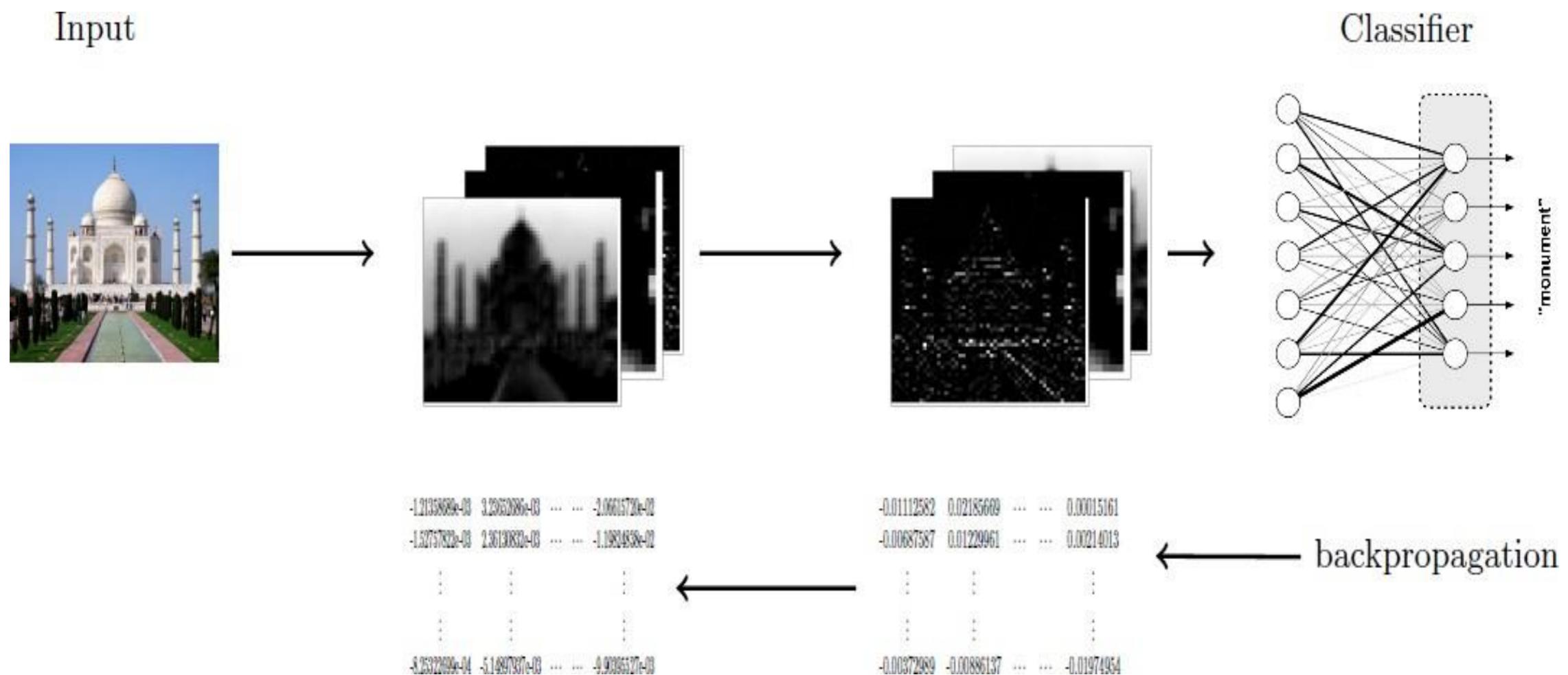
Beyond Static Feature Engineering

- Can we learn multiple **layers** of meaningful kernels/filters in addition to learning the weights of the classifier?



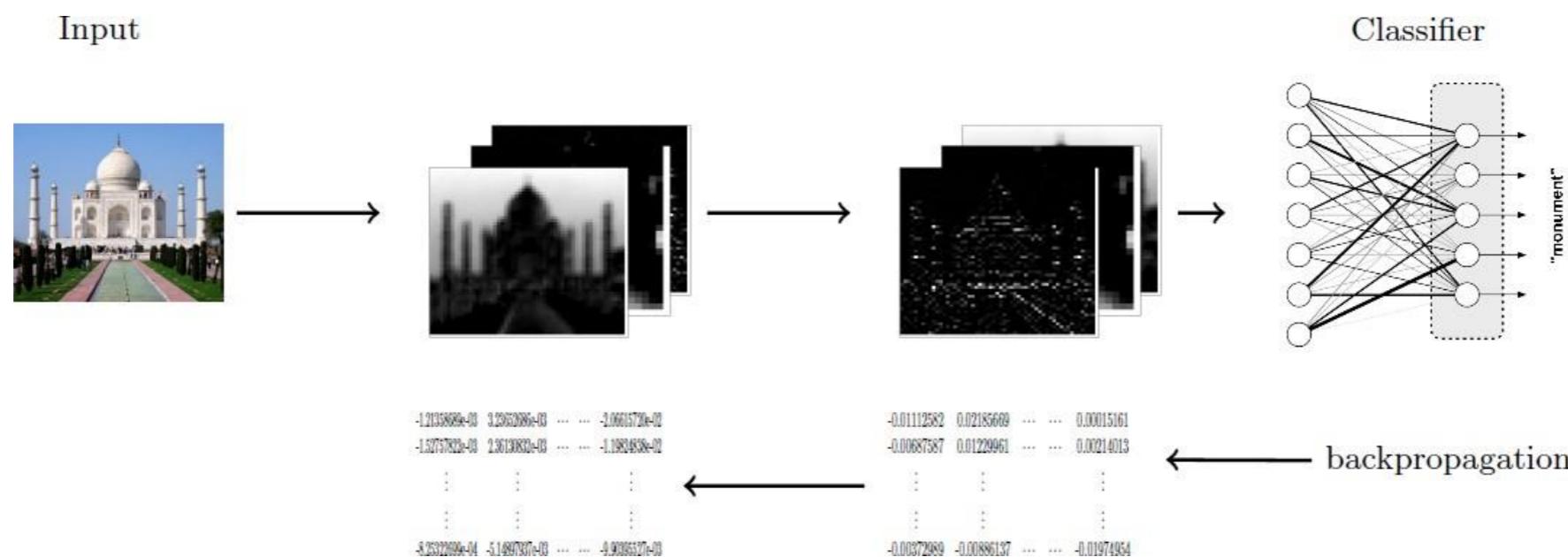
Beyond Static Feature Engineering

- Can we learn multiple **layers** of meaningful kernels/filters in addition to learning the weights of the classifier? **Yes, we can!**
- Simply by treating these kernels as parameters and learning them in addition to the weights of the classifier (using backpropagation)



Beyond Static Feature Engineering

- Can we learn multiple **layers** of meaningful kernels/filters in addition to learning the weights of the classifier? **Yes, we can!**
- Simply by treating these kernels as parameters and learning them in addition to the weights of the classifier (using backpropagation, discussed in the next lecture)



- Such a network is called a **Convolutional Neural Network!**

Parameter Explosion:
Images are large (e.g., 224x224 pixels with 3 color channels = 150,528 inputs).
Flattening images leads to huge input layers and massive numbers of parameters in fully connected layers.
This causes a lot of computation, memory usage, and risk of overfitting.

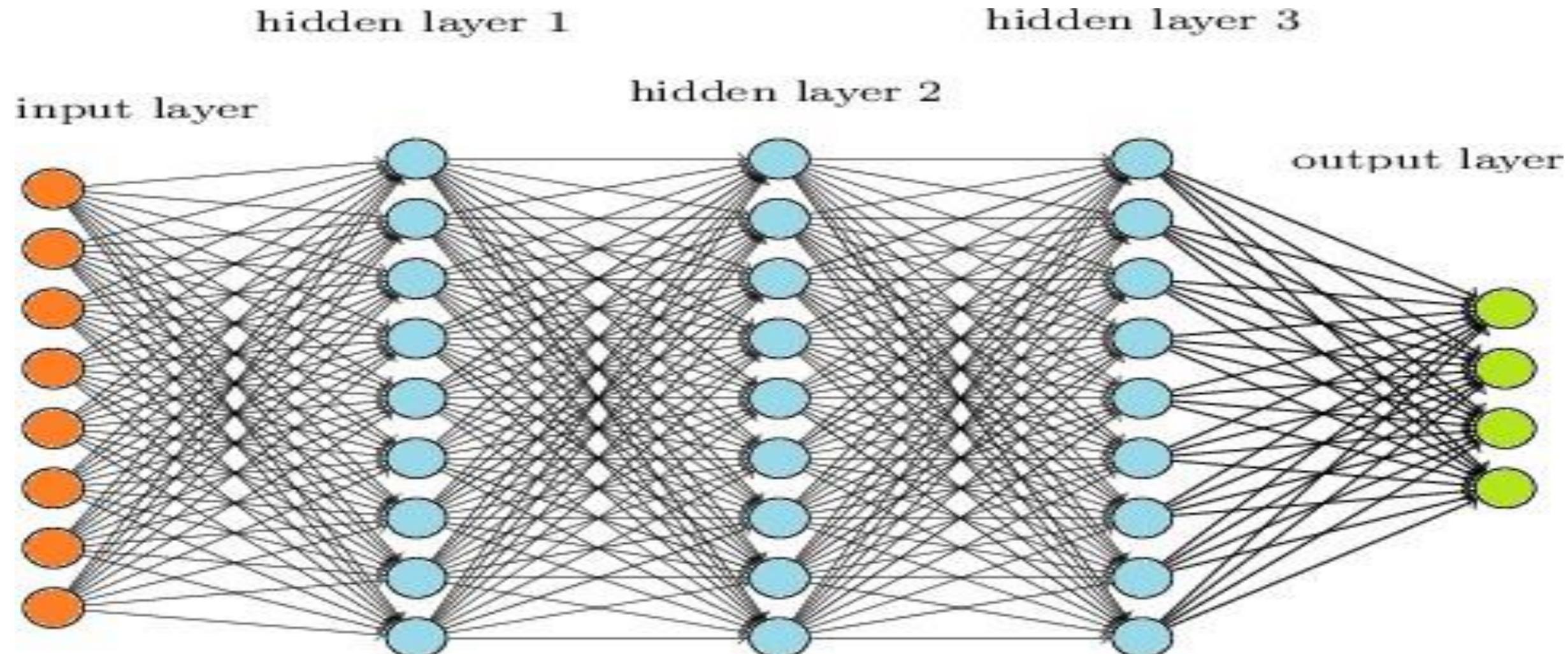
Spatial Structure Loss:
Flattening destroys the 2D spatial structure of images.
Pixels near each other spatially lose their local relationships.
CNNs keep spatial structure by using small, local filters that slide across the image.

Parameter Sharing:
CNNs use shared filters across the entire image, drastically reducing parameters.
FNNs have separate weights for every input connection, which doesn't scale well.

Better Feature Learning:
CNNs learn hierarchical features, starting from edges to textures to shapes.
FNNs struggle to learn such structured features from raw pixels due to lack of spatial awareness.

Pause and Ponder

- Learning kernels/filters by treating them as parameters definitely is interesting
- But why not directly use flattened images with fully connected neural networks (or feedforward neural networks, FNNs) instead?



Challenges of Applying FNNs to Images

On a reasonably *simple* dataset like MNIST, we can get about 2% error (or even better) using FNNs, but



- Ignores spatial (2-D) structure of input images – unroll each 28×28 image into a 784-D vector
 - Pixels that are spatially separate are treated the same way as pixels that are adjacent
- No obvious way for networks to learn same features (e.g. edges) at different places in the input image
- Can get computationally expensive for large images
 - For a 1MP color image with 20 neurons in the first hidden layer, how many weights in the first layer?

MNIST Dataset

Challenges of Applying FNNs to Images

On a reasonably *simple* dataset like MNIST, we can get about 2% error (or even better) using FNNs, but



- Ignores spatial (2-D) structure of input images – unroll each 28×28 image into a 784-D vector
 - Pixels that are spatially separate are treated the same way as pixels that are adjacent
- No obvious way for networks to learn same features (e.g. edges) at different places in the input image
- Can get computationally expensive for large images
 - For a 1MP color image with 20 neurons in the first hidden layer, how many weights in the first layer?
60 million!

MNIST Dataset

Credit: Steve Renals

How do Convolutional Neural Networks Solve these Challenges?

- **Local receptive fields**, in which hidden units are connected to local patches of the layer below, serve two purposes:
 - Capture local spatial relationships in pixels (which would not be captured by FNNs)
 - Greatly reduces number of parameters in the model
 - For a 1MP color image ,a filter size of $K_1 \times K_2$ in the first hidden layer, how many weights in a convolutional layer?

How do Convolutional Neural Networks Solve these Challenges?

- **Local receptive fields**, in which hidden units are connected to local patches of the layer below, serve two purposes:
 - Capture local spatial relationships in pixels (which would not be captured by FNNs)
 - Greatly reduces number of parameters in the model
 - For a 1MP color image a filter size of $K_1 \times K_2$ in the first hidden layer, how many weights in a convolutional layer? $K_1 \times K_2$, compare with 60 million for FNNs on the previous slide!

How do Convolutional Neural Networks Solve these Challenges?

- **Local receptive fields**, in which hidden units are connected to local patches of the layer below, serve two purposes:
 - Capture local spatial relationships in pixels (which would not be captured by FNNs)
 - Greatly reduces number of parameters in the model
 - For a 1MP color image a filter size of $K_1 \times K_2$ in the first hidden layer, how many weights in a convolutional layer? $K_1 \times K_2$, compare with 60 million for FNNs on the previous slide!
- **Weight sharing**, which also serves two purposes:
 - Enables translation-invariance of neural network to objects in images
 - Reduces number of parameters in the model

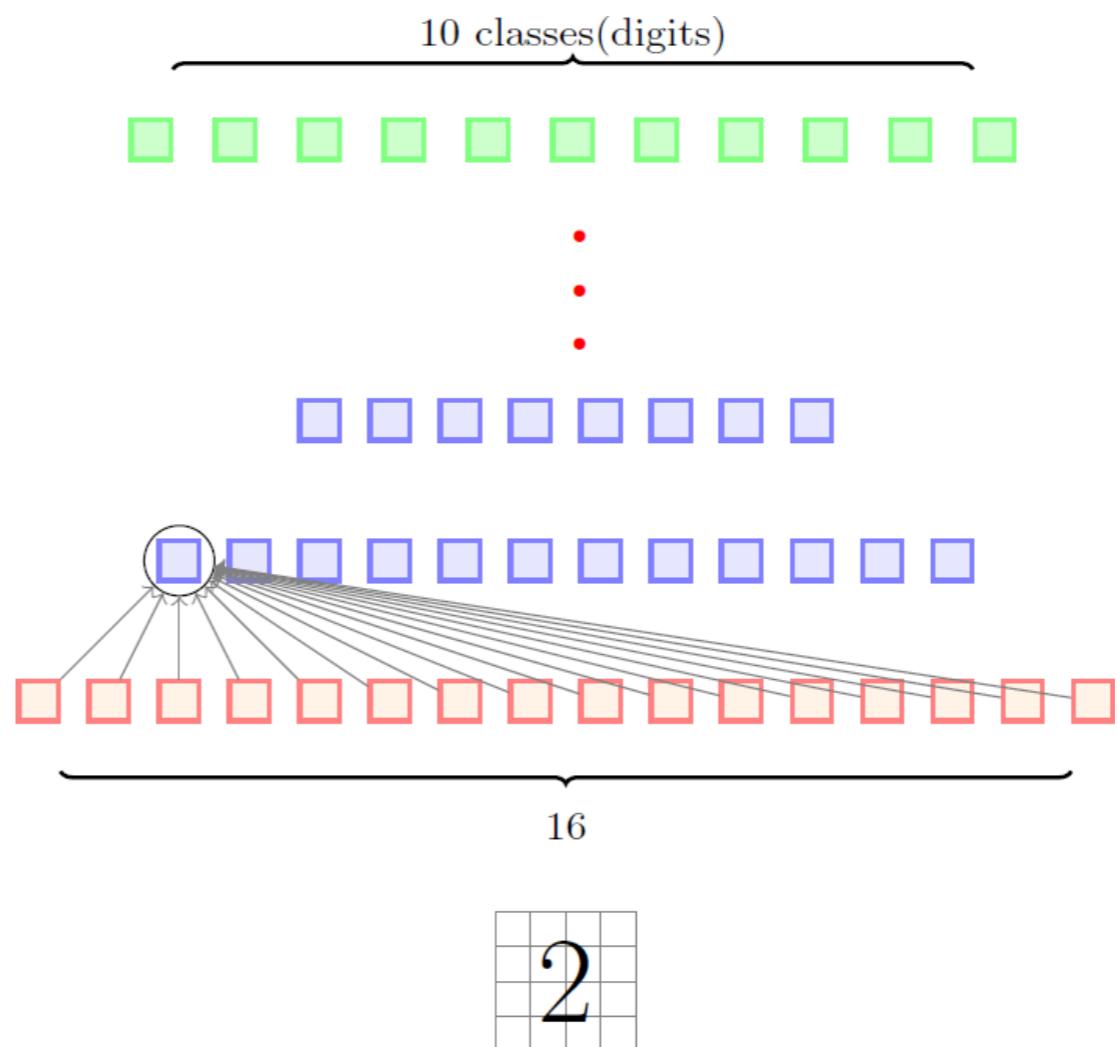
How do Convolutional Neural Networks Solve these Challenges?

- **Local receptive fields**, in which hidden units are connected to local patches of the layer below, serve two purposes:
 - Capture local spatial relationships in pixels (which would not be captured by FNNs)
 - Greatly reduces number of parameters in the model
 - For a 1MP color image a filter size of $K_1 \times K_2$ in the first hidden layer, how many weights in a convolutional layer? $K_1 \times K_2$, compare with 60 million for FNNs on the previous slide!
- **Weight sharing**, which also serves two purposes:
 - Enables translation-invariance of neural network to objects in images
 - Reduces number of parameters in the model
- **Pooling** which condenses information from previous layer, serves two purposes:
 - Aggregates information, especially minor variations
 - Reduces size of output of a previous layer, which reduces number of computations in later layers

Credit: Steve Renals

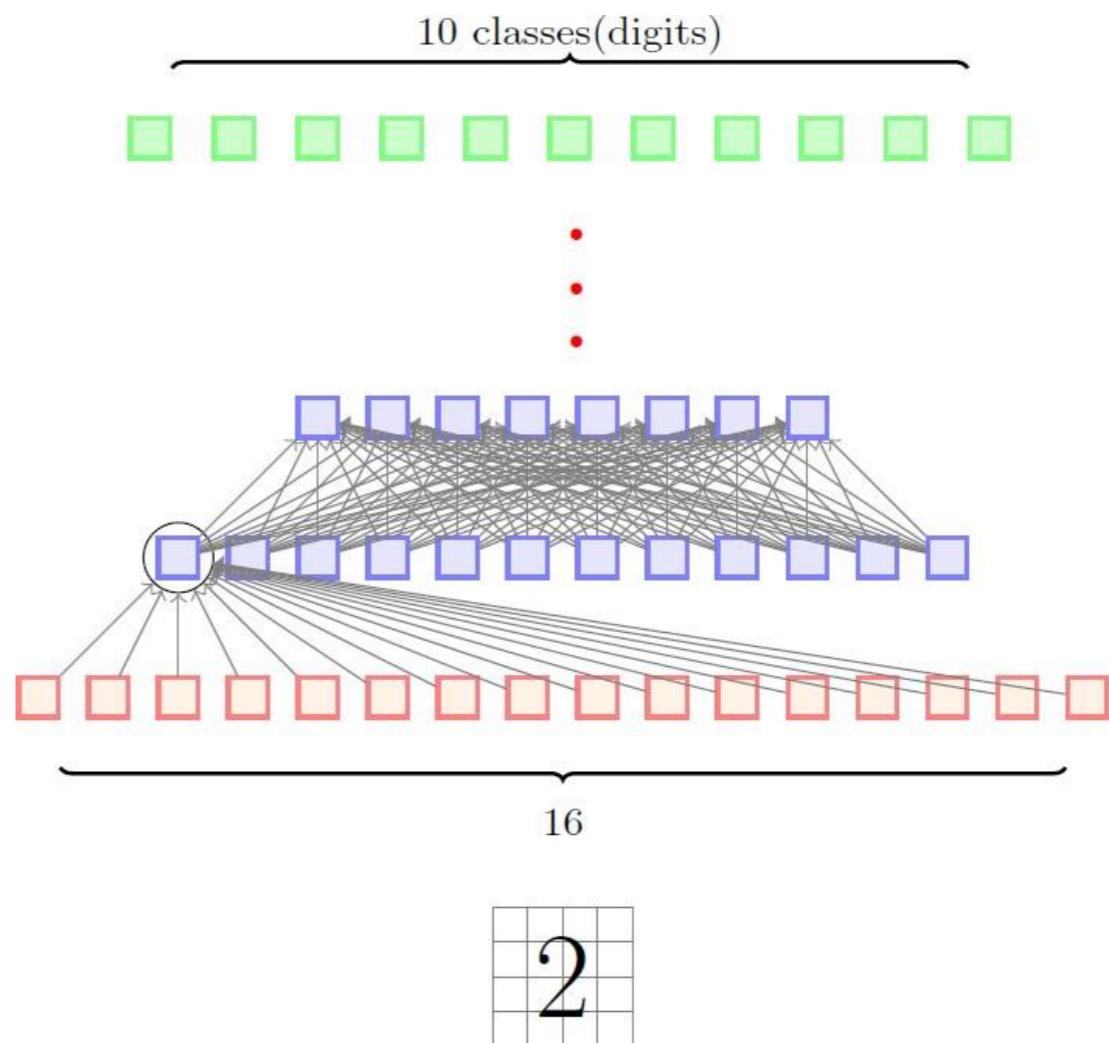
Local Receptive Fields

- This is what a regular feedforward neural network will look like
- There are many dense connections here



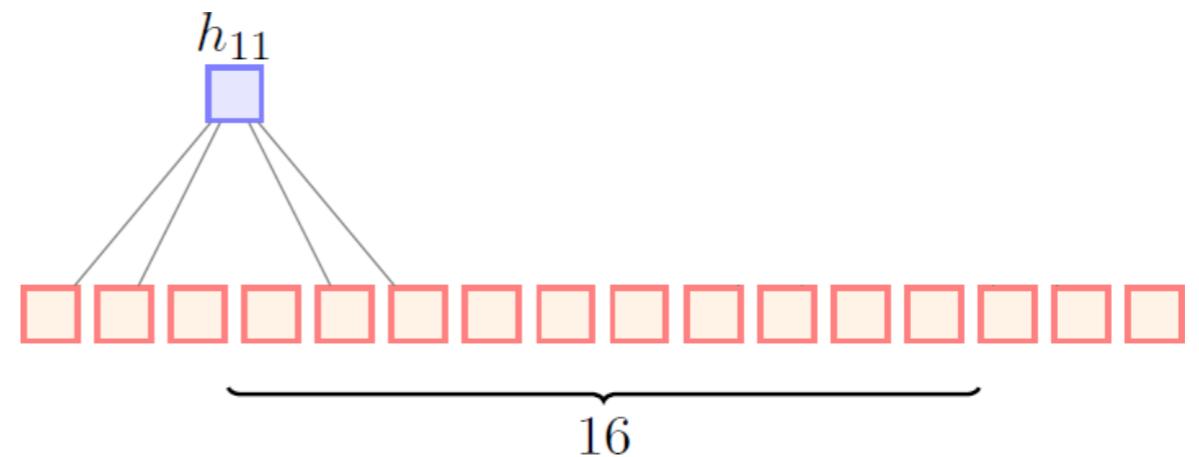
Local Receptive Fields

- This is what a regular feedforward neural network will look like
- There are many dense connections here
- All 16 input neurons are contributing to computation of h_{11}
- Let us contrast this to what happens in case of convolution



Local Receptive Fields

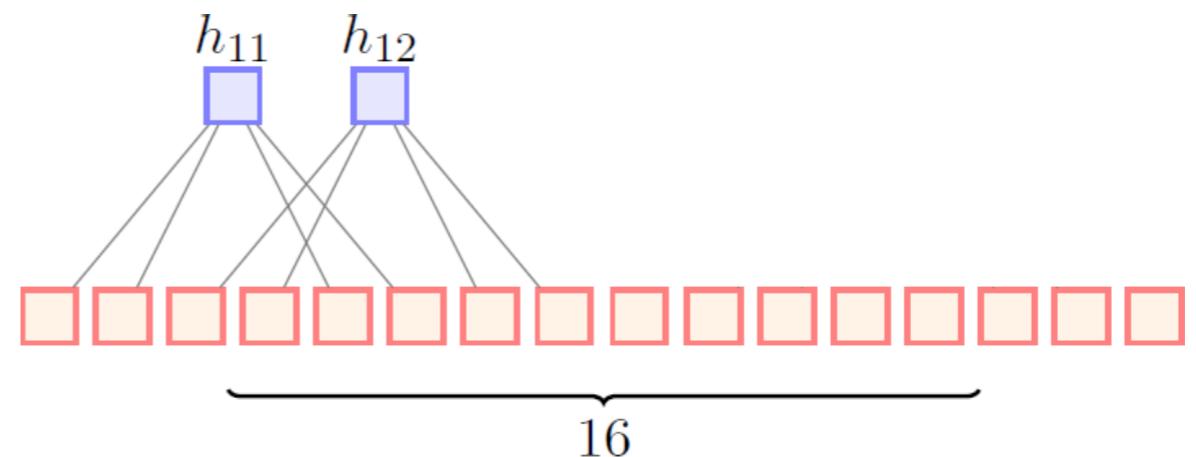
- Only a few local neurons participate in computation of h_{11}
- E.g. only pixels 1, 2, 5, 6 contribute to h_{11}



$$\begin{matrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \end{matrix} * \begin{matrix} \bullet & \bullet \\ \bullet & \bullet \end{matrix} = h_{11}$$

Local Receptive Fields

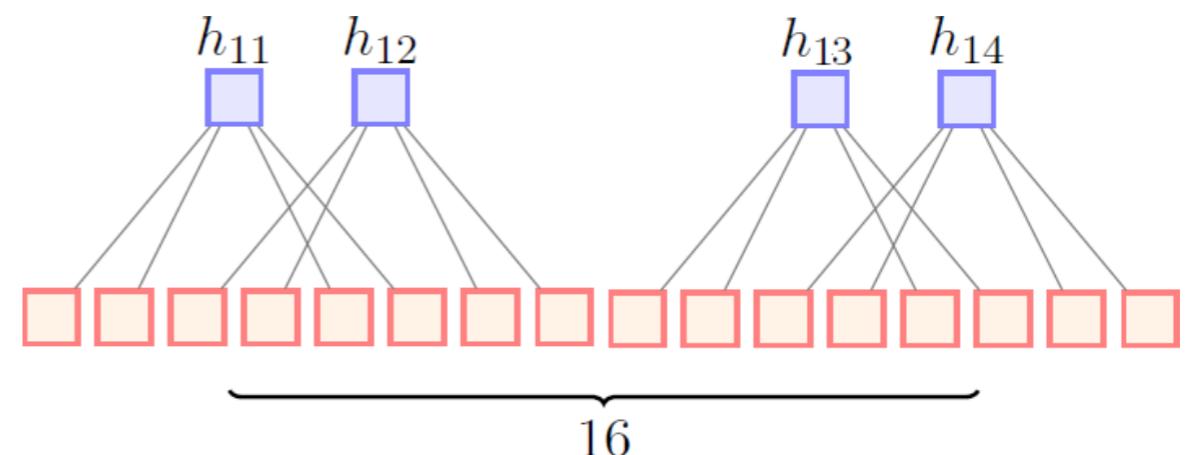
- Only a few local neurons participate in computation of h_{11}
- E.g. only pixels 1, 2, 5, 6 contribute to h_{11}



The diagram shows the computation of h_{12} as a convolution of a 4x4 input matrix and a 2x2 kernel matrix. The input matrix is a 4x4 grid of black dots, with a 2x2 submatrix highlighted in gray. The kernel matrix is a 2x2 grid of blue dots. The result of the convolution is shown as a blue square labeled h_{12} .

Local Receptive Fields

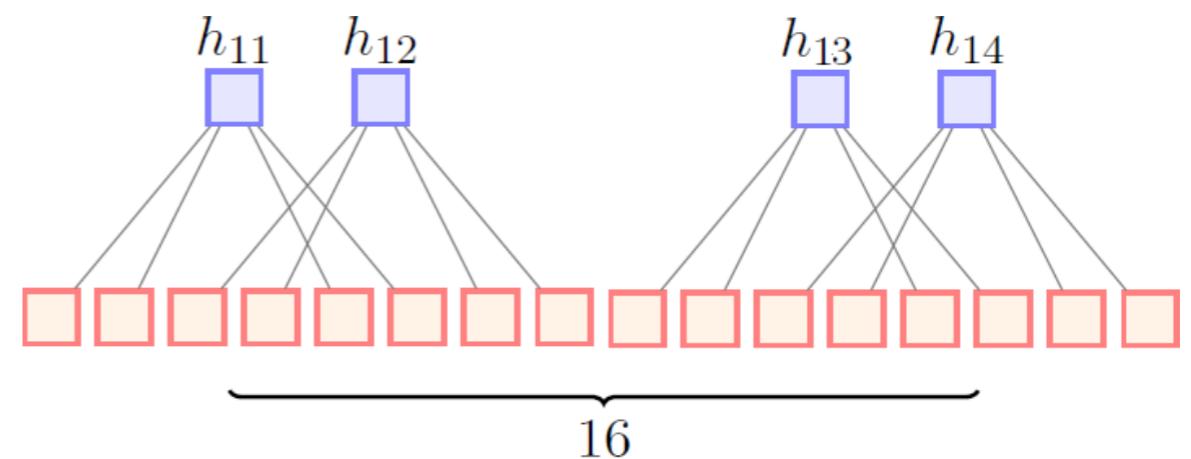
- Only a few local neurons participate in computation of h_{11}
- E.g. only pixels 1, 2, 5, 6 contribute to h_{11}
- Similar for other pixels



The diagram shows the computation of h_{14} . It consists of two parts: a 4x4 input grid and a 2x2 kernel. The input grid contains black dots at positions (1,1), (1,2), (2,1), (2,2), (3,2), (4,1), (4,2), and (4,3). A gray circle highlights the 3x3 receptive field of the center pixel (3,2). The kernel is a 2x2 grid with blue dots at all four positions. An asterisk (*) between the input grid and the kernel indicates convolution, resulting in a single blue square labeled h_{14} .

Local Receptive Fields

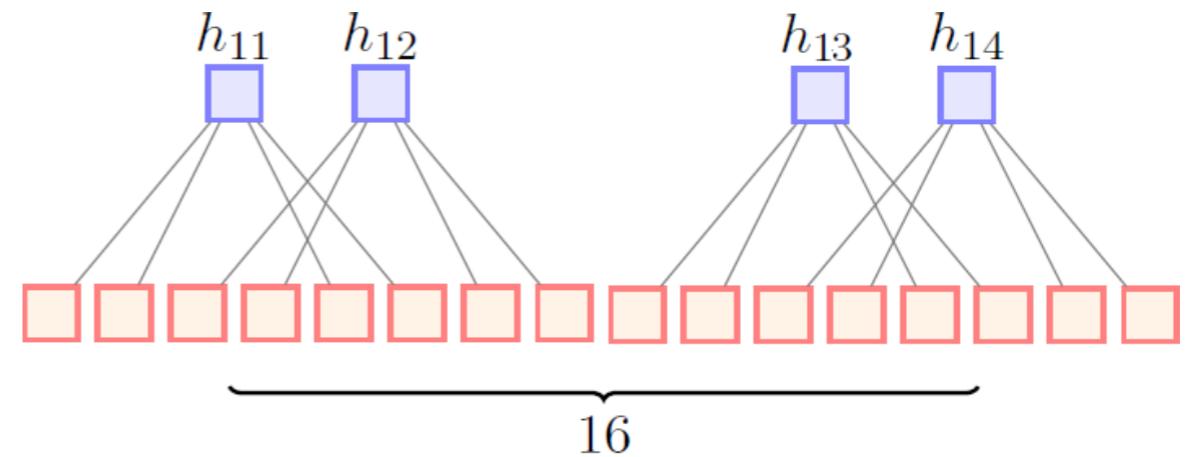
- Only a few local neurons participate in computation of h_{11}
- E.g. only pixels 1, 2, 5, 6 contribute to h_{11}
- Similar for other pixels
- The connections are much sparser
- This **sparse connectivity** reduces the number of parameters in the model



The diagram illustrates the computation of the output h_{14} . It shows a 4x4 input feature map with black dots at positions (1,1), (1,2), (1,3), (1,4), (2,2), (2,3), (2,4), (3,2), (3,3), (3,4), (4,2), (4,3), and (4,4). A 2x2 receptive field mask with blue dots at (1,1) and (2,2) is applied to the feature map. A gray circle highlights the central element (2,3) in the receptive field. The result of the multiplication is shown as a blue square labeled h_{14} .

Local Receptive Fields

- Only a few local neurons participate in computation of h_{11}
- E.g. only pixels 1, 2, 5, 6 contribute to h_{11}
- Similar for other pixels
- The connections are much sparser
- This **sparse connectivity** reduces the number of parameters in the model
- We are taking advantage of the structure of the image (interactions between neighboring pixels are interesting in images)



$$\begin{matrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \end{matrix} * \begin{matrix} \bullet & \bullet \\ \bullet & \bullet \end{matrix} = h_{14}$$

Local Receptive Fields

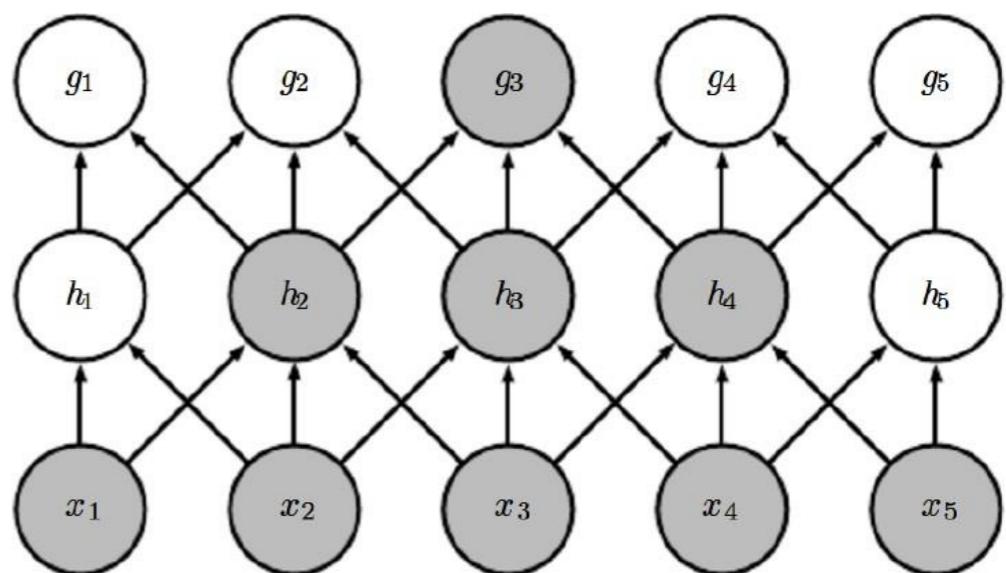
- But is sparse connectivity really a good thing?

Local Receptive Fields

- But is sparse connectivity really a good thing?
- Aren't we losing information (by losing interactions between some input pixels)

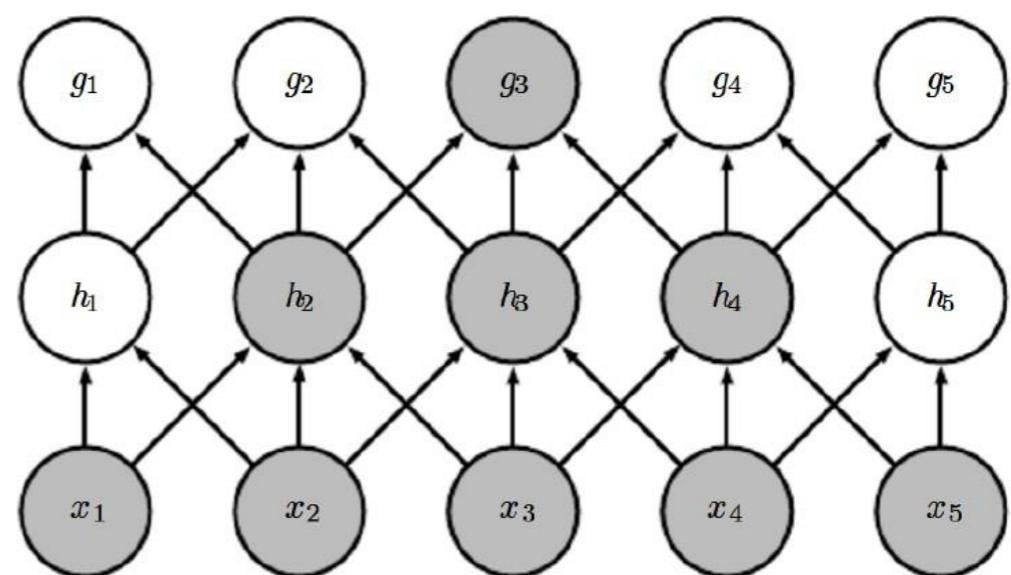
Local Receptive Fields

- But is sparse connectivity really a good thing?
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really



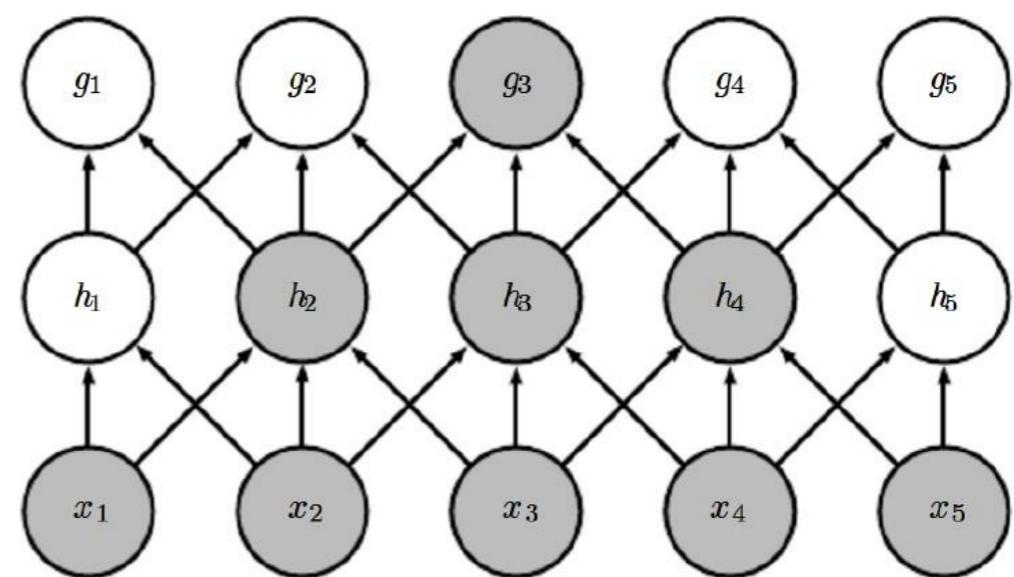
Local Receptive Fields

- But is sparse connectivity really a good thing?
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really
- The two highlighted neurons (x_1x_5) do not interact in layer 1



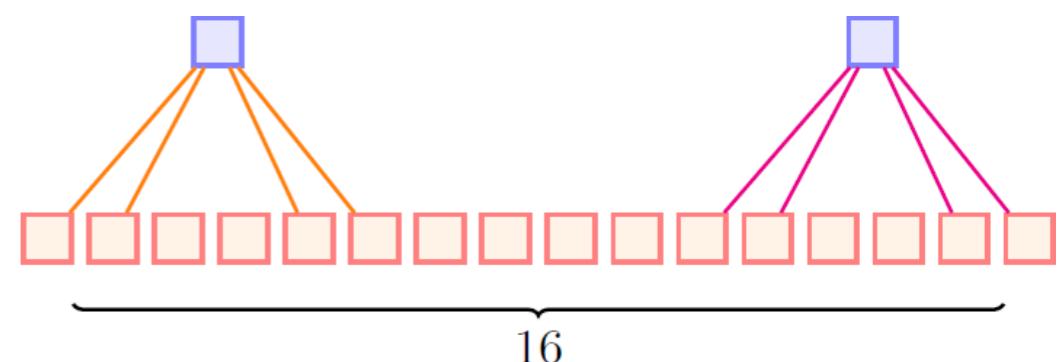
Local Receptive Fields

- But is sparse connectivity really a good thing?
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really
- The two highlighted neurons ($x_1 x_5$) do not interact in layer 1
- But they indirectly contribute to the computation of g_3 and hence interact indirectly

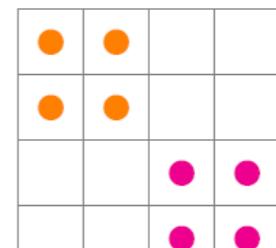


Weight Sharing

- Consider the following network; do we want the kernel weights to be different for different parts of the image?



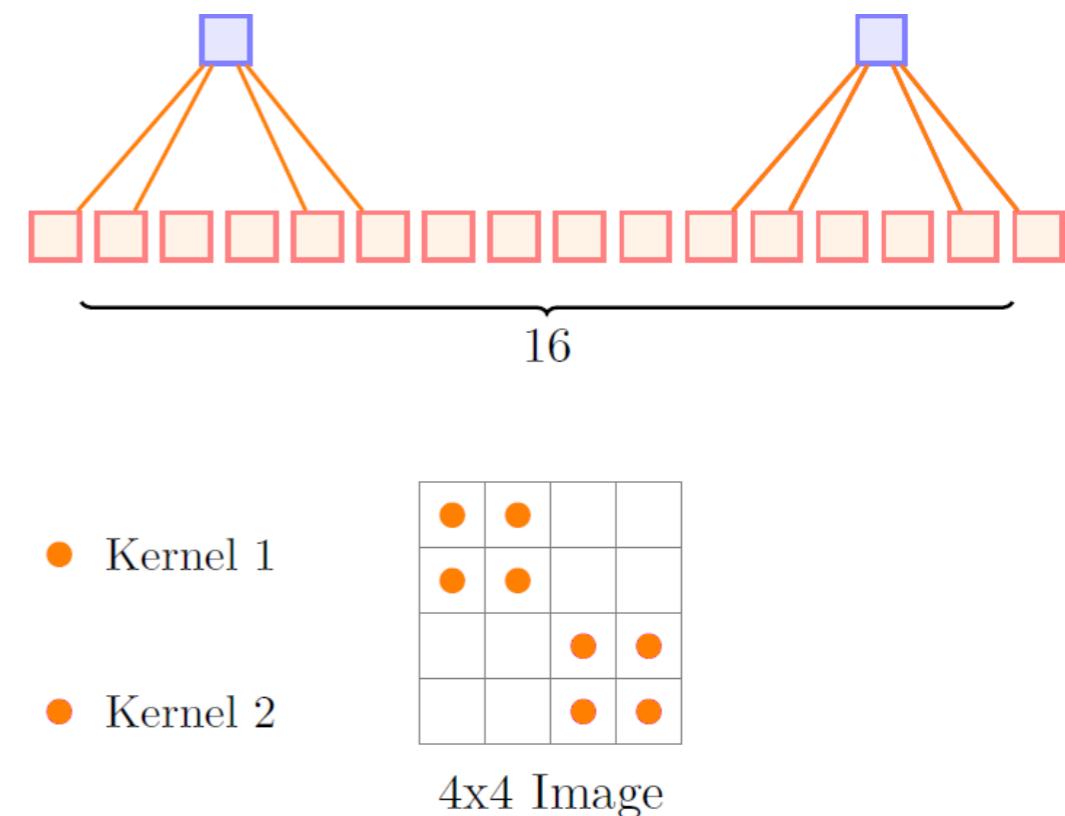
- Kernel 1
- Kernel 2



4x4 Image

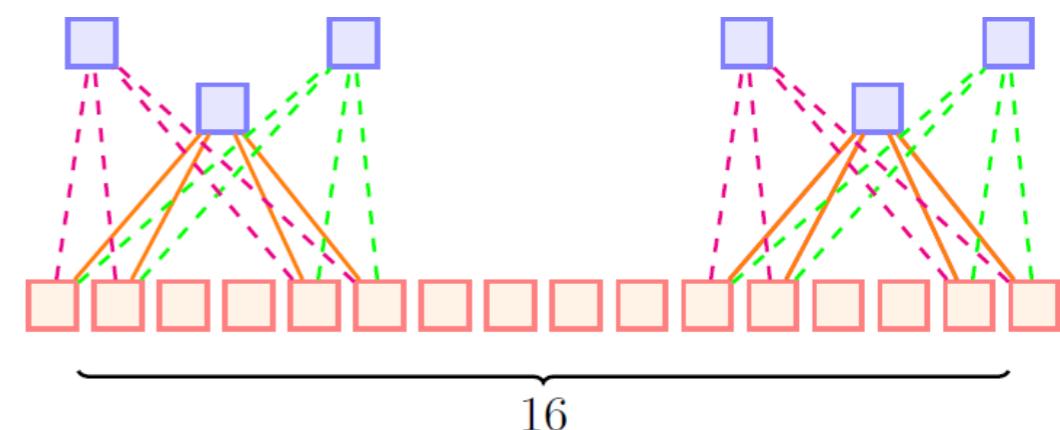
Weight Sharing

- Consider the following network; do we want the kernel weights to be different for different parts of the image?
- Not really. We would want the filter to respond to an object or an artefact in an image in the same way irrespective of where it is located in the image
==> **translation-invariance**

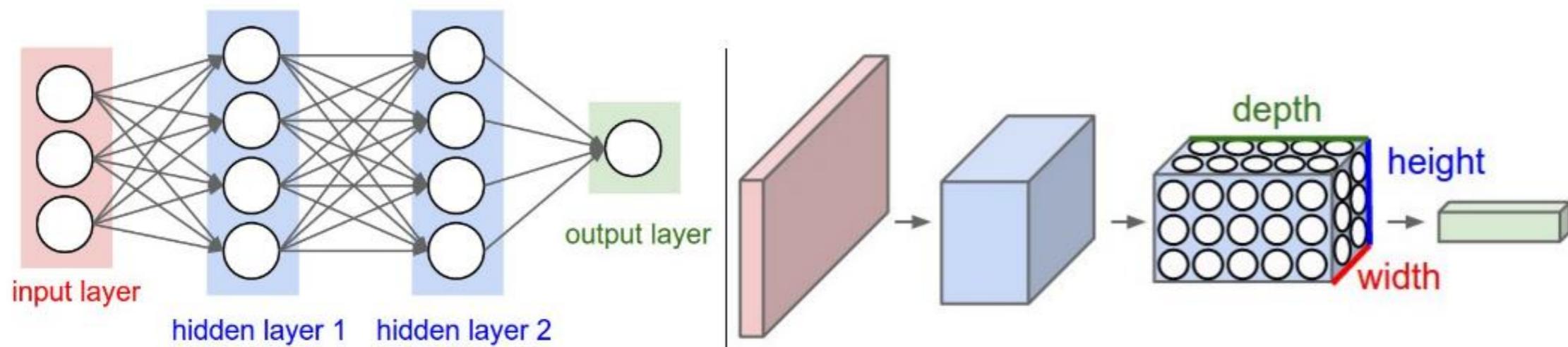


Weight Sharing

- Consider the following network; do we want the kernel weights to be different for different parts of the image?
- Not really. We would want the filter to respond to an object or an artefact in an image in the same way irrespective of where it is located in the image
==> **translation-invariance**
- We can have as many different kernels to capture different kinds of artifacts, but each one is intended to give the same response on all parts of the image
- This is called **weight sharing**

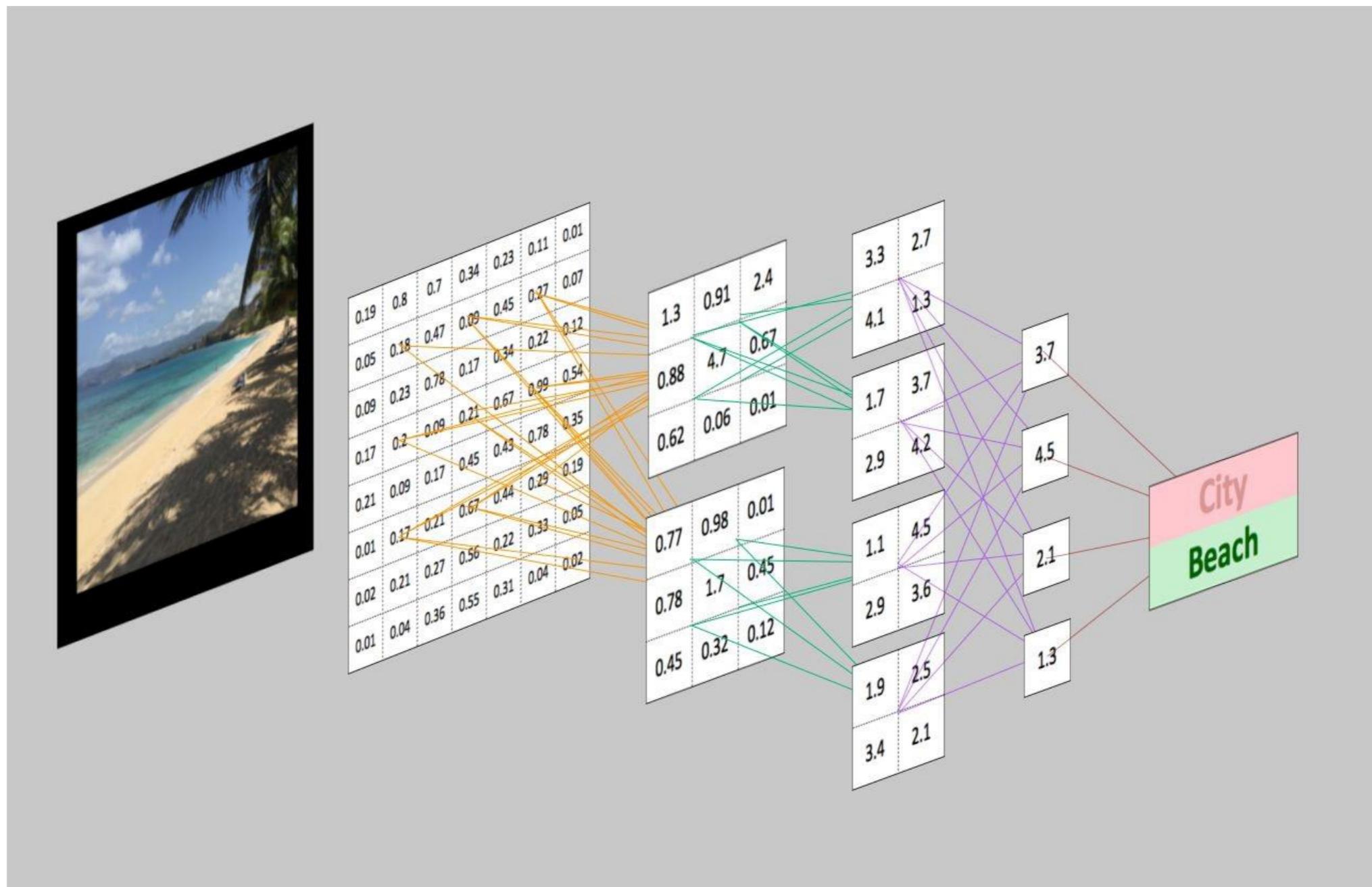


Dense neural network and Convolutional neural network



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Introduction

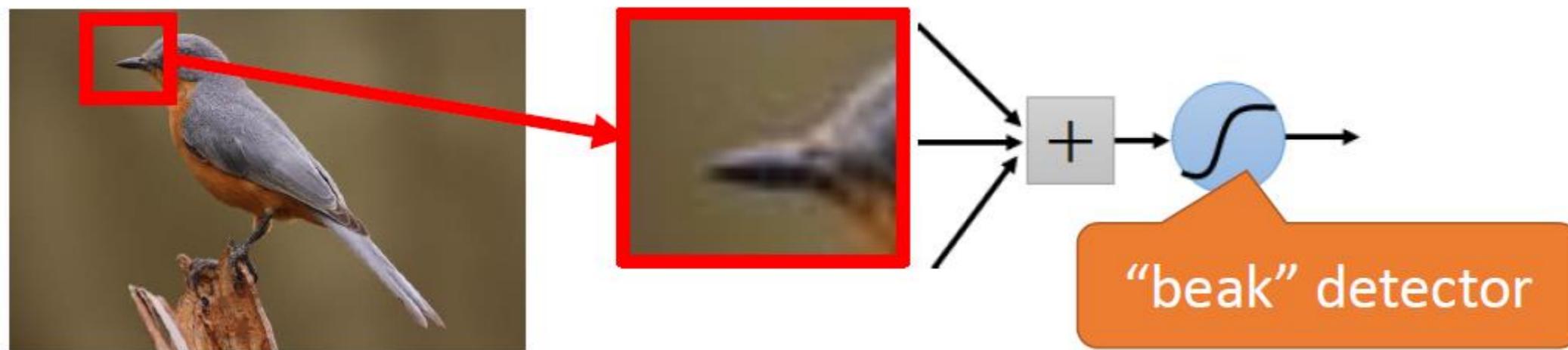


Why CNN for Image

- Some patterns are much smaller than the whole image

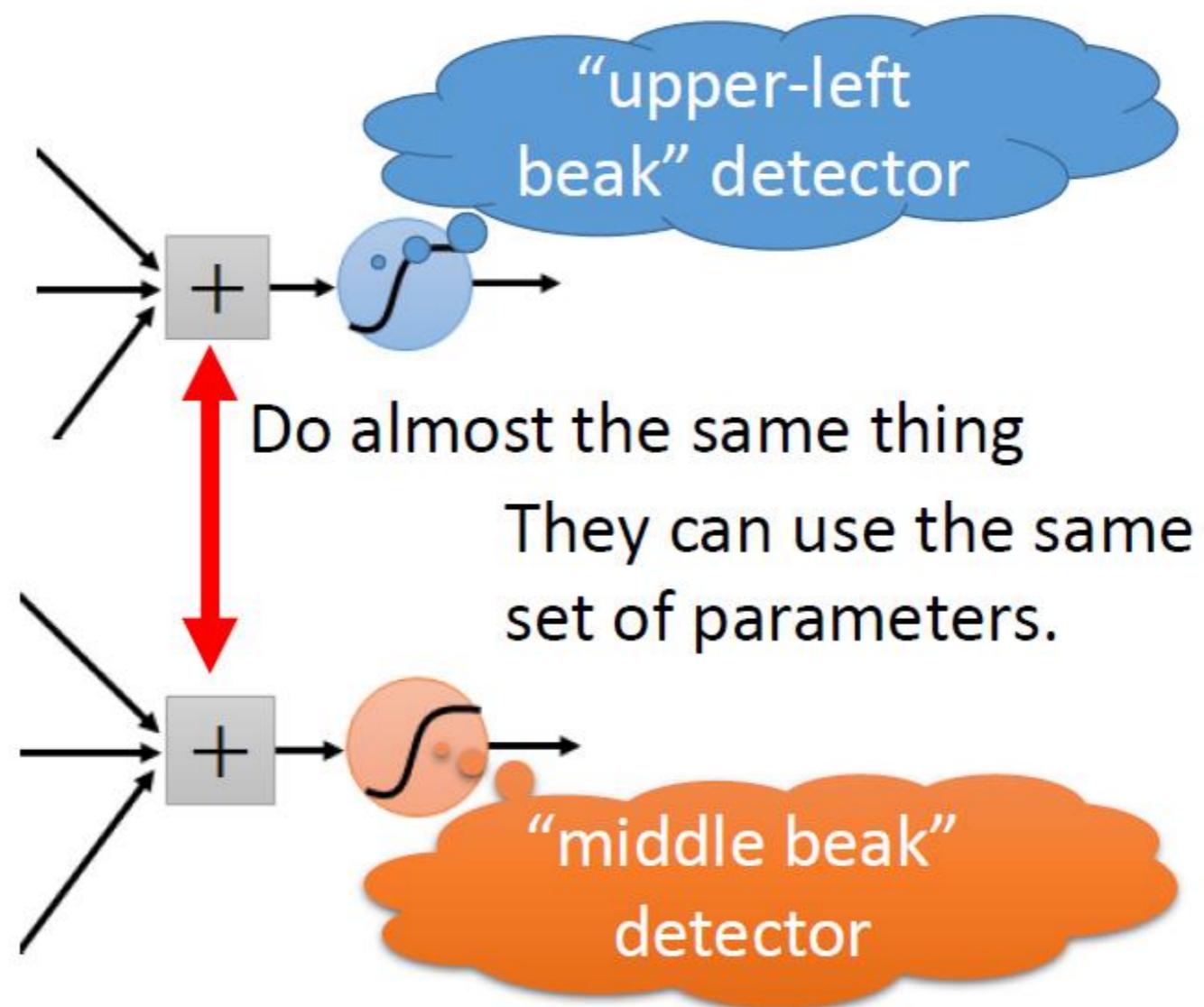
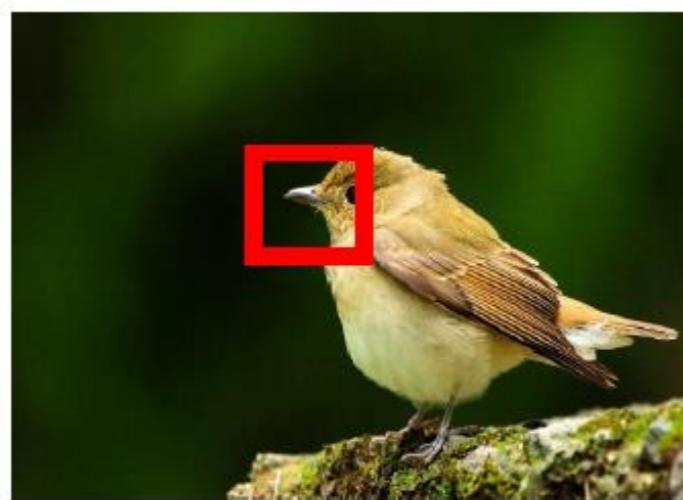
A neuron does not have to see the whole image to discover the pattern.

Connecting to small region with less parameters



Why CNN for Image

- The same patterns appear in different regions.



Why CNN for Image

- Subsampling the pixels will not change the object



We can subsample the pixels to make image smaller

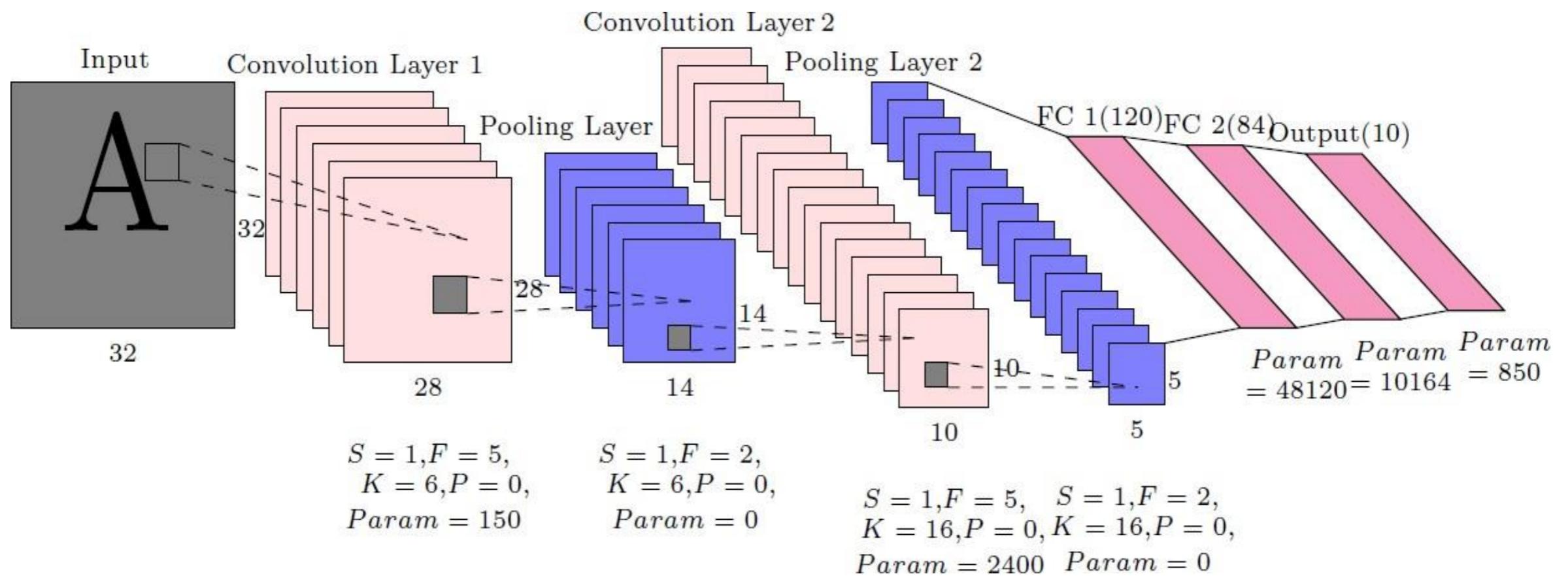
→ Less parameters for the network to process the image

Convolutional Neural Network

- A typical CNN looks as follows:

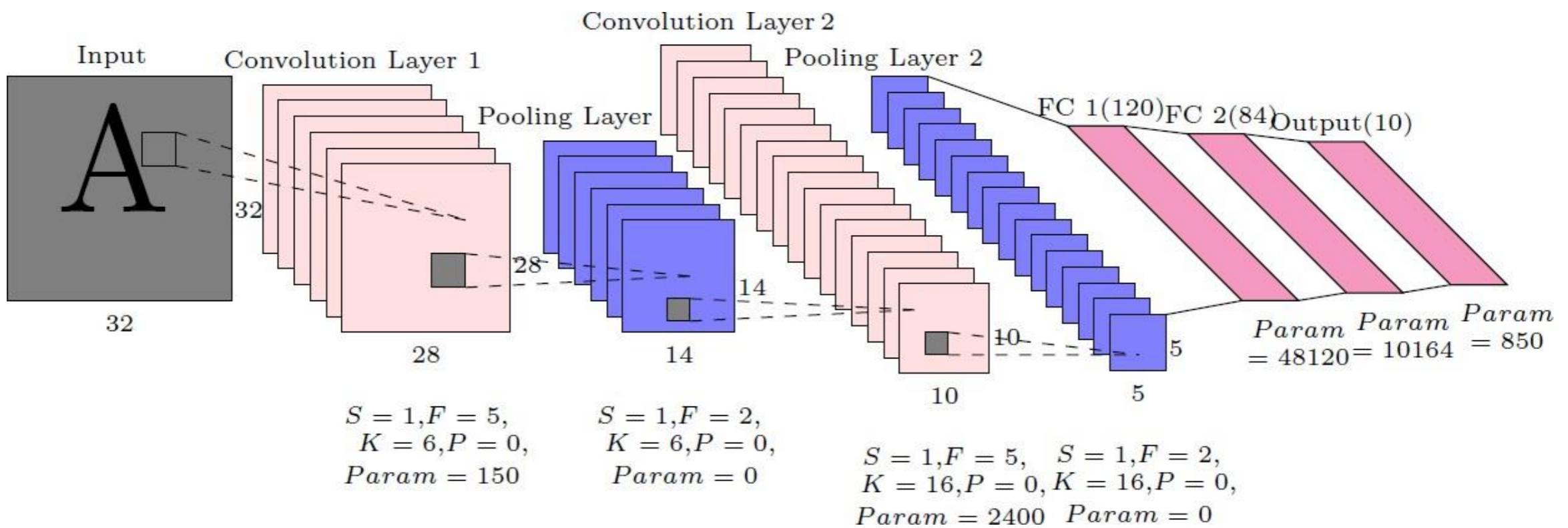
Convolutional Neural Network

- A typical CNN looks as follows:



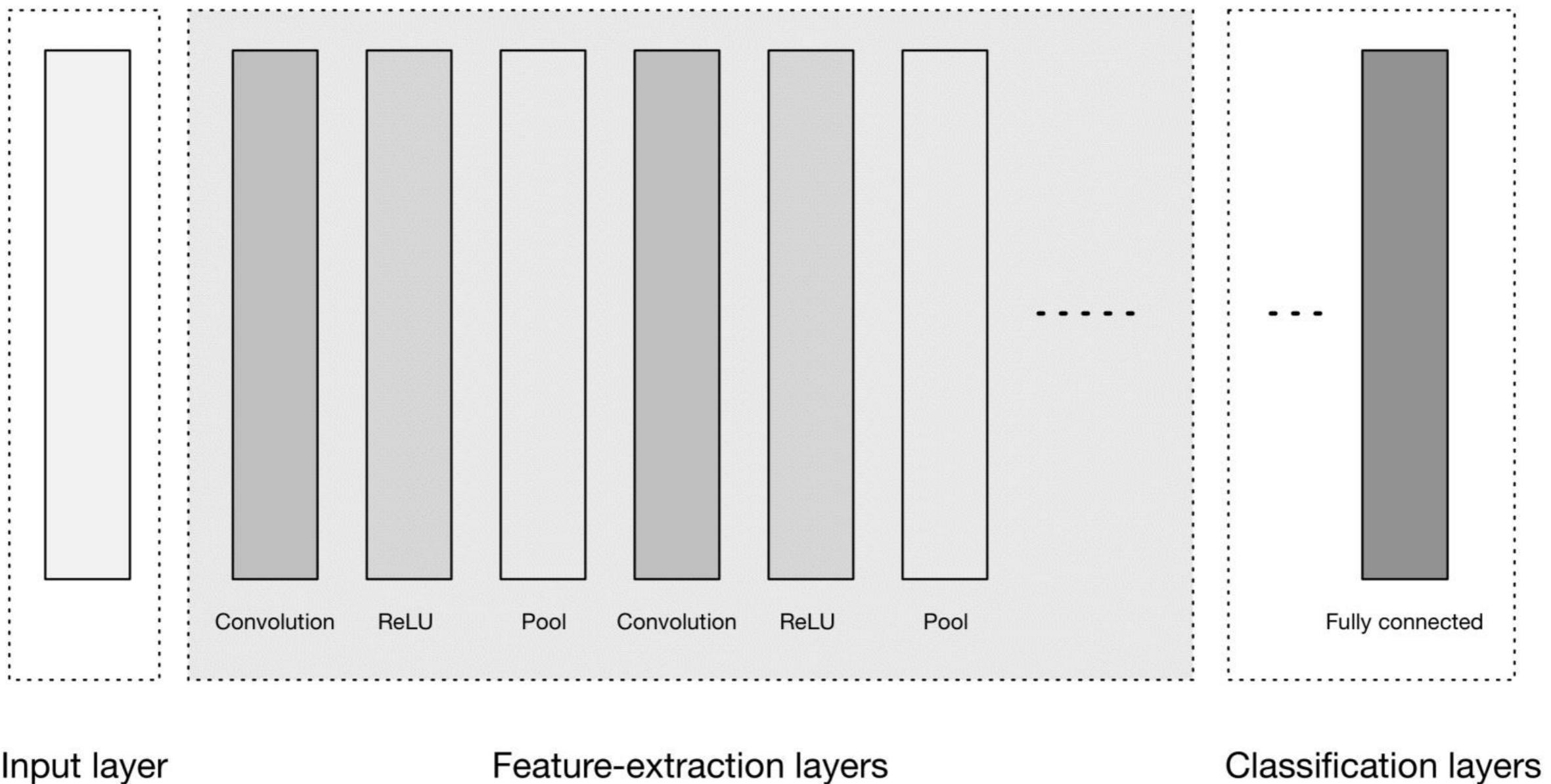
Convolutional Neural Network

- A typical CNN looks as follows:

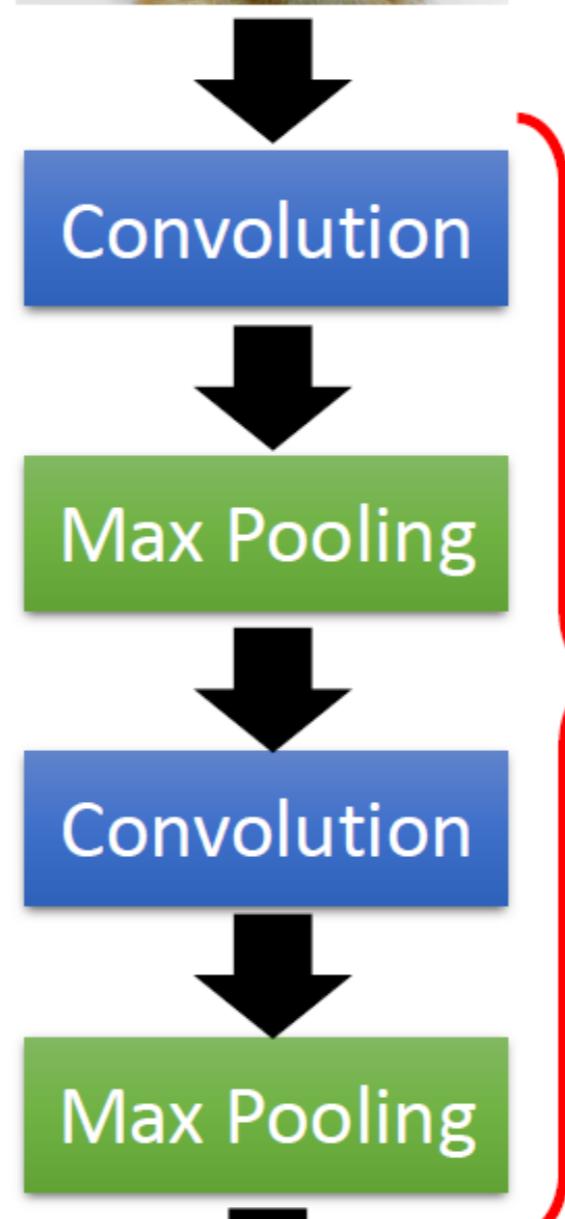
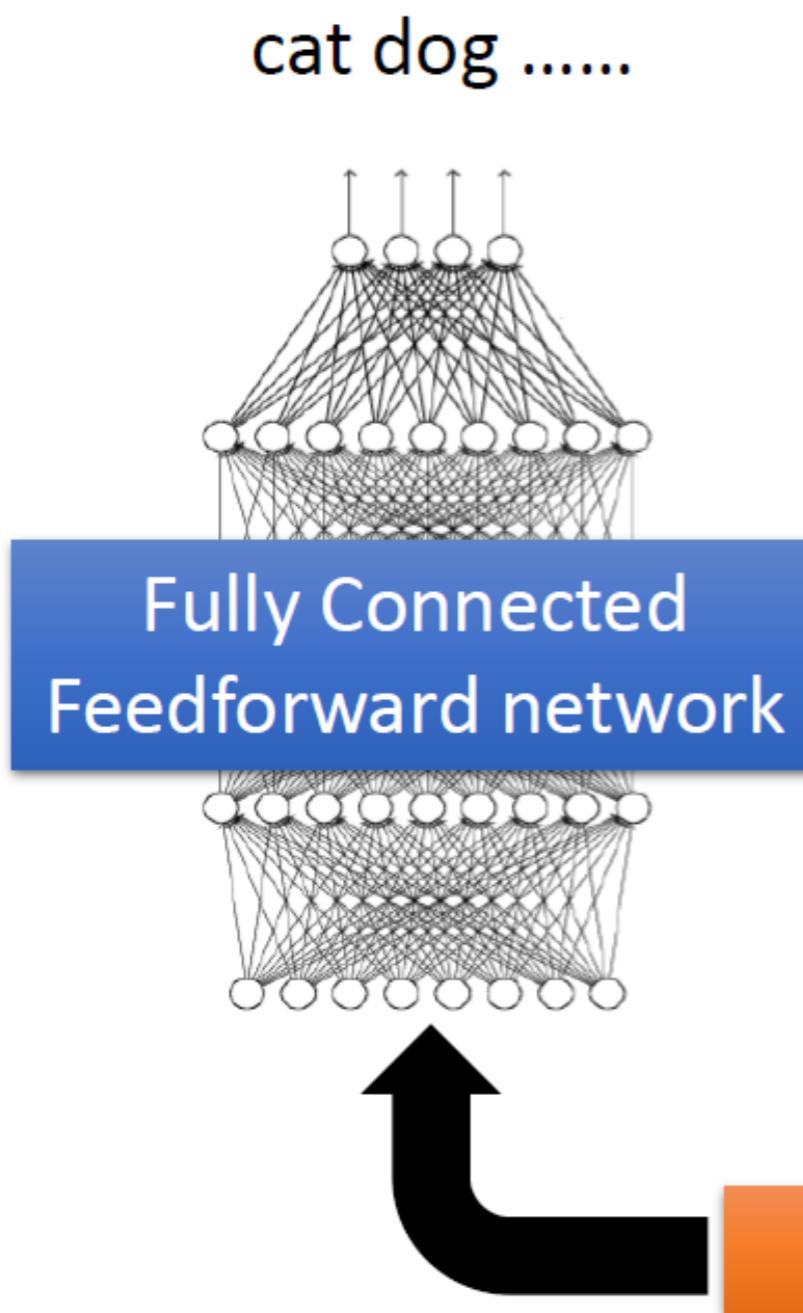


- It has alternate convolution and pooling layers. What do pooling layers do?

CNN Architecture Overview



The whole CNN



Flatten

The whole CNN



Property 1

- Some patterns are much smaller than the whole image

Property 2

- The same patterns appear in different regions.

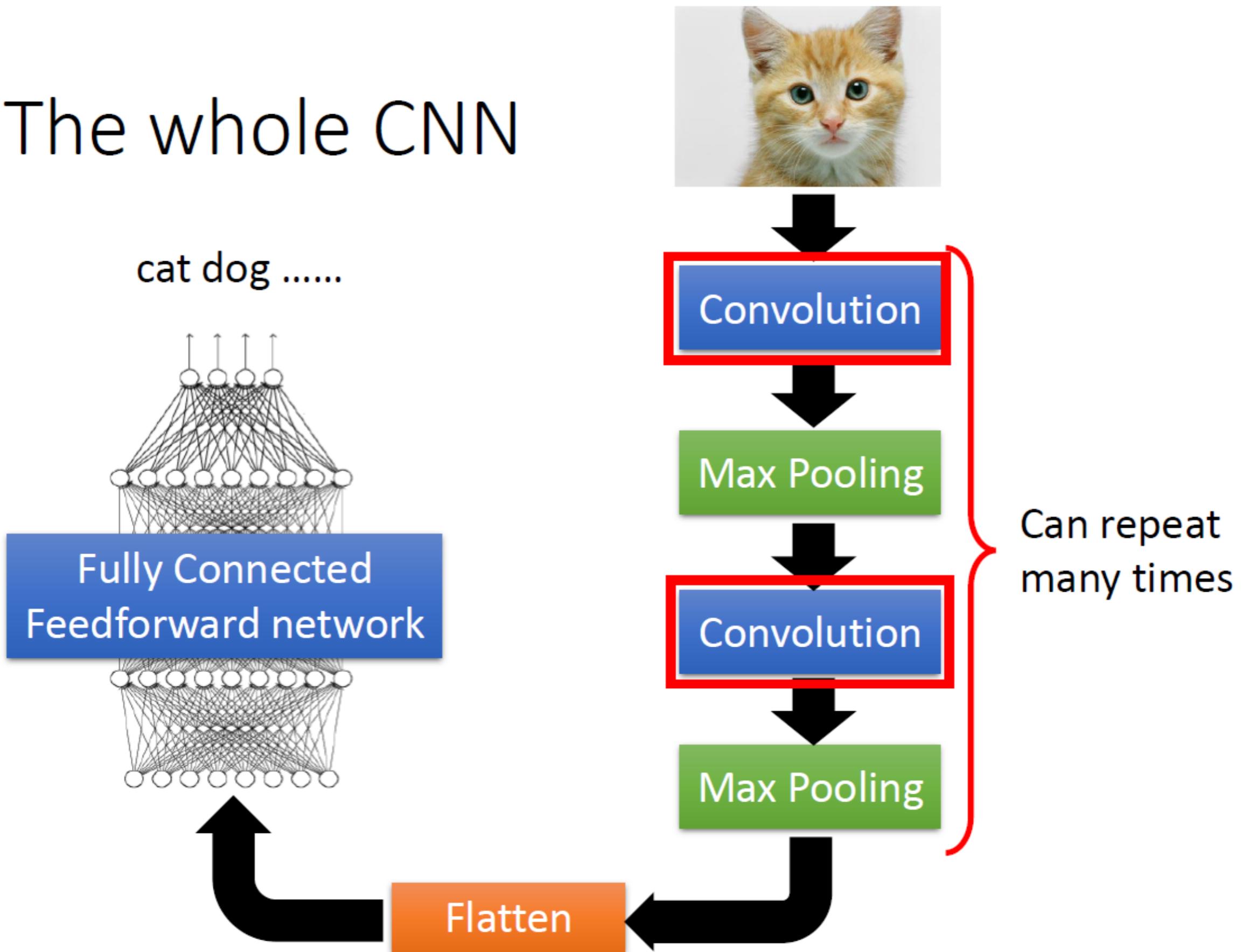
Property 3

- Subsampling the pixels will not change the object



Can repeat
many times

The whole CNN



CNN Architecture Overview

■ Convolution-

◦ Major components of convolutional layers:

- **Filters** These are small matrices (kernels) that slide over the input image or feature map.
They detect specific features like edges, textures, or patterns.
- **Activation maps** The output produced after applying the filters to the input.
Each filter produces an activation map showing where that feature is detected.
- **Parameter sharing** The same filter (set of weights) is used across different spatial locations.
This reduces the number of parameters and helps recognize features regardless of position.
- **Layer-specific hyperparameters**

These are settings specific to each convolutional layer that affect its behavior, such as:

Filter size (e.g., 3x3)

Number of filters

Stride (step size when sliding filters)

Padding (adding borders to preserve spatial size)

CNN Architecture Overview

Three major groups:

1. Input layer
2. Feature-extraction (learning) layers
3. Classification layers

CNN Architecture Overview

1. Input layer-

- Accepts three-dimensional input generally in the form spatially of the size (width × height) of the image and has a depth representing the color channels (generally three for RGB color channels).

2. Feature-extraction layers-

- have a general repeating pattern of the sequence:
 - 1. Convolution layer- Rectified Linear Unit (ReLU) activation function as a layer in the diagram
 - 2. Pooling layer

CNN Architecture Overview

3. Classification layers –

- one or more fully connected layers to take the higher-order features and produce class probabilities or scores.
- Fully connected to all of the neurons in the previous layer.
- The output of these layers produces typically a two-dimensional output of the dimensions $[b \times N]$, where b is the number of examples in the mini-batch and N is the number of classes we're interested in scoring.

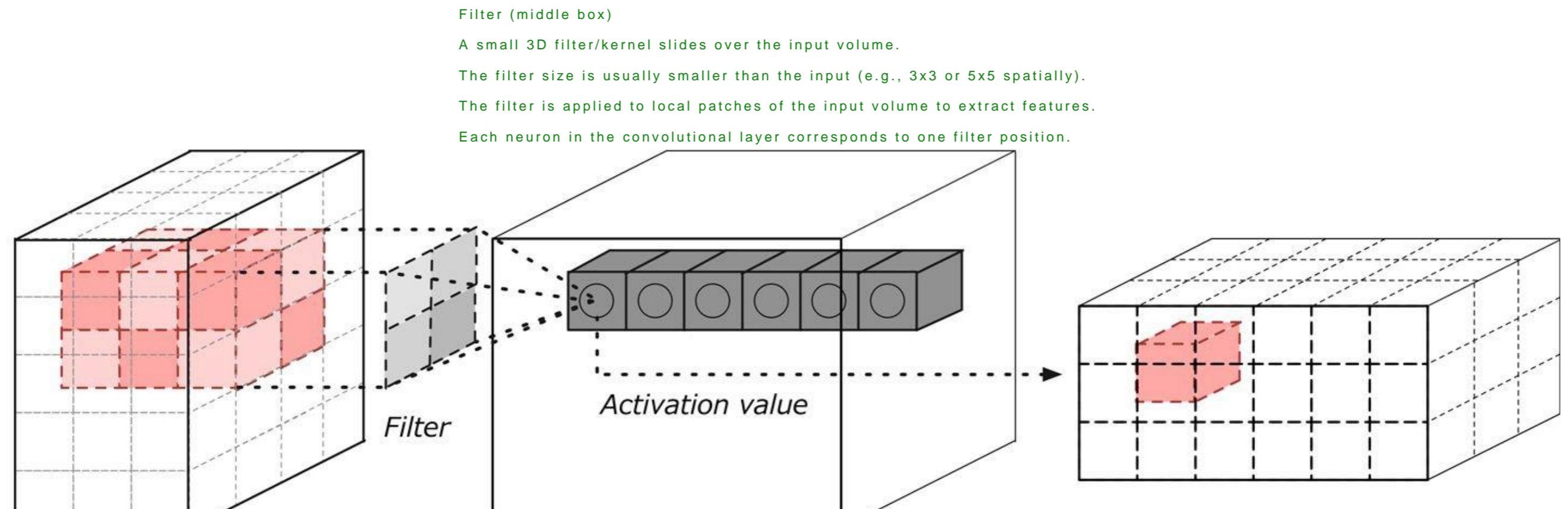
CNN Architecture Overview

- **Input Layers-**

- **Convolutional Layers-**

- Core building blocks of CNN architectures.
- Transforms the input data by using a patch of locally connecting neurons from the previous layer.
- The layer will compute a dot product between the region of the neurons in the input layer and the weights to which they are locally connected in the output layer.
- The resulting output generally has the same spatial dimensions (or smaller spatial dimensions) but sometimes increases the number of elements in the third dimension of the output (depth dimension).

CNN Architecture Overview



*Input volume
(with highlighted focus input)*

Input Volume (left box)

This represents the input data to the convolutional layer.

It's a 3D volume with width, height, and depth (e.g., RGB image channels).

The highlighted red cube is the local patch (small region) of the input the filter focuses on at a given time.

Convolution layer with input and output volumes

Activation Value (middle box)

The filter and the local input patch produce a single activation value (shown as a vector).

This activation is a result of element-wise multiplication and summation of the filter and the input patch values, often followed by an activation function.

Output activation volume

Output Activation Volume (right box)

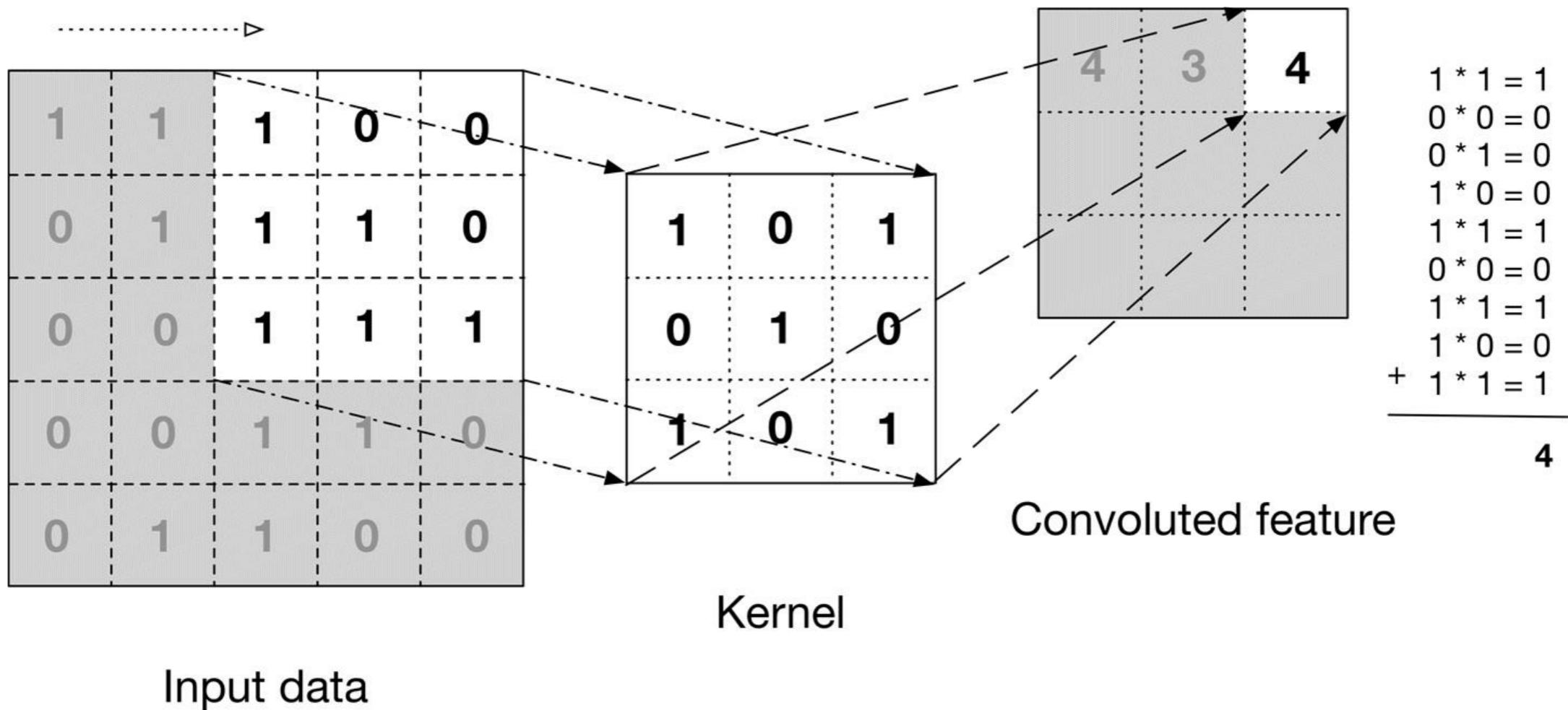
As the filter slides over the entire input volume, it produces a 3D output volume.

Each position in this output volume corresponds to the activation from one local patch.

This output volume is the input to the next layer.

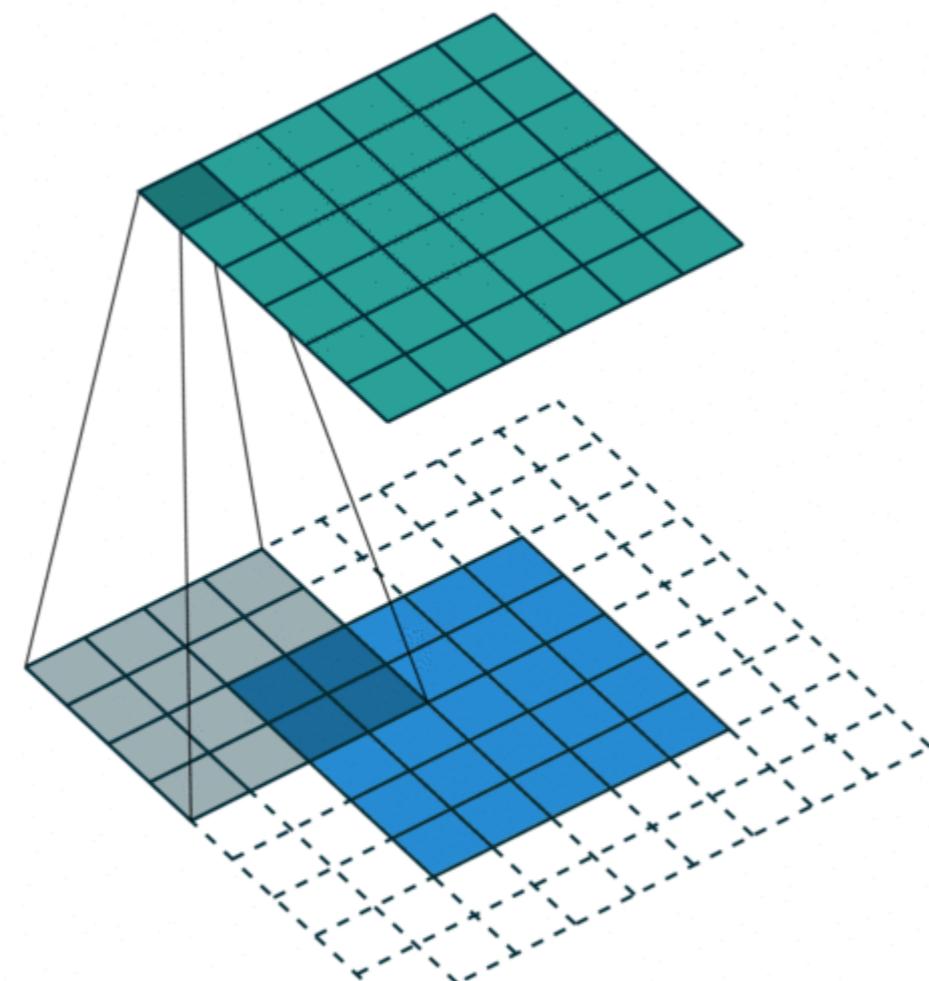
CNN Architecture Overview

■ Convolution-



The convolution operation

Convolutional kernel



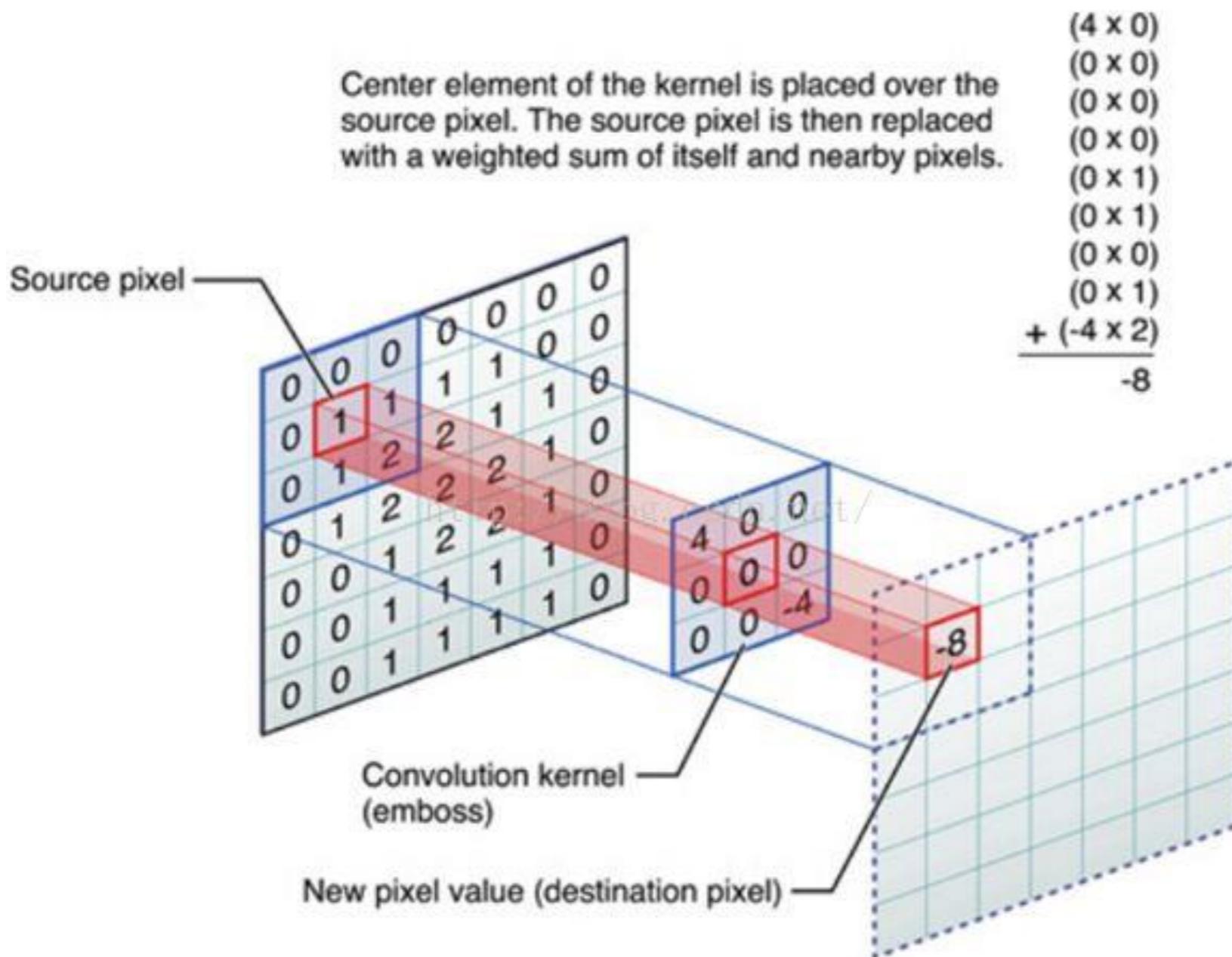
This is a gif image

The top grid represents the input feature map (part of an image or previous layer output).

The blue square on the bottom represents the output feature map (activation map).

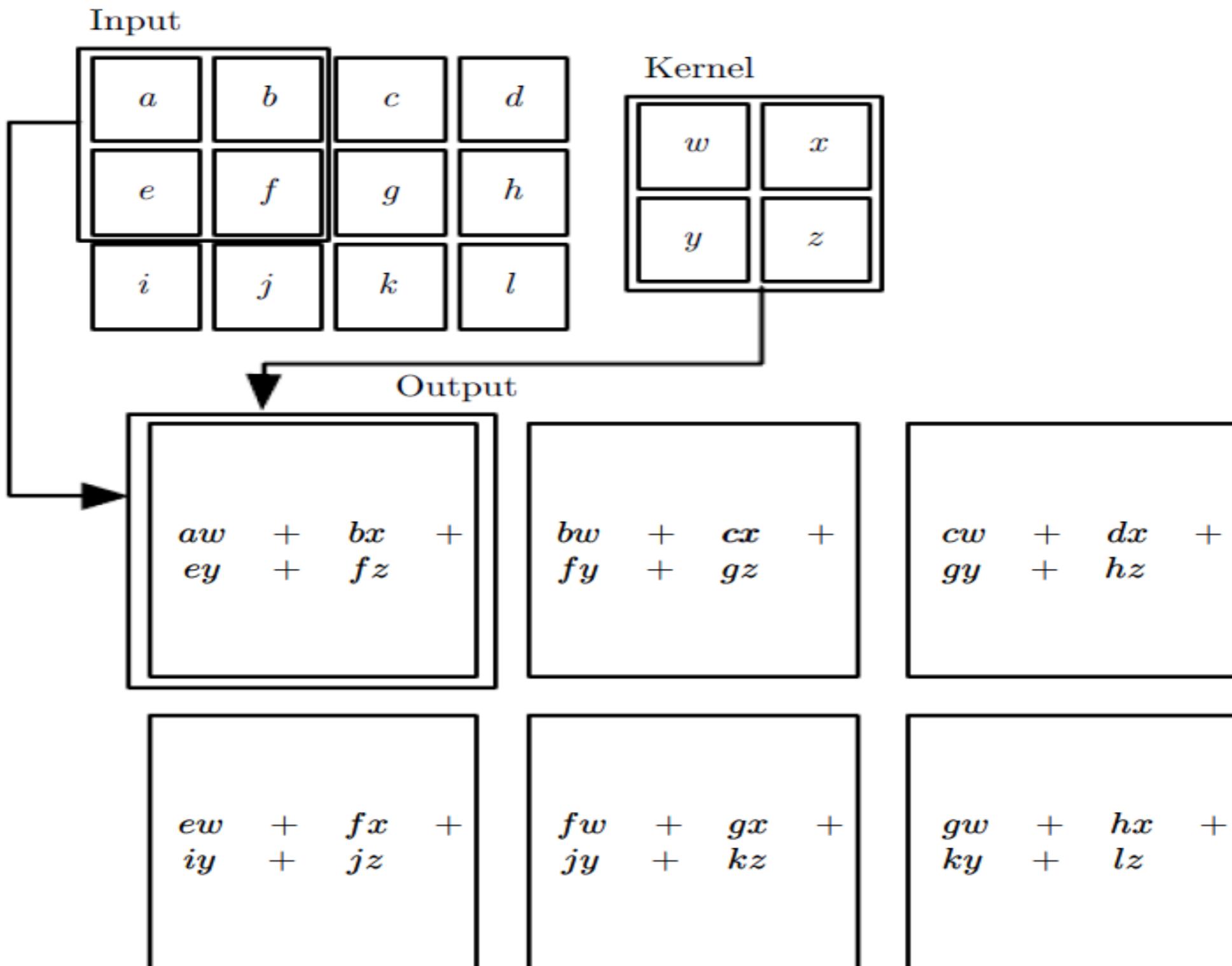
The gray square is the kernel (filter) — a small window (like 3x3 or 5x5) that slides over the input.

Convolutional kernel



Padding on the input volume with zeros in such way that the conv layer does not alter the spatial dimensions of the input

Convolutional kernel



CNN – Convolution

Those are the network parameters to be learned.

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

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Matrix

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

Matrix

⋮

Property 1

Each filter detects a small pattern (3 x 3).

CNN – Convolution

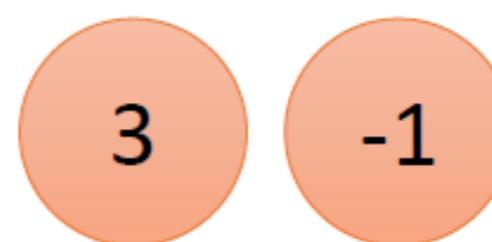
stride=1

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

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



CNN – Convolution

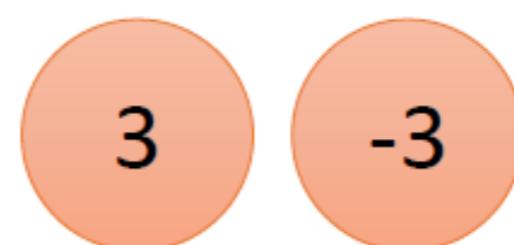
If stride=2

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

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



We set stride=1 below

CNN – Convolution

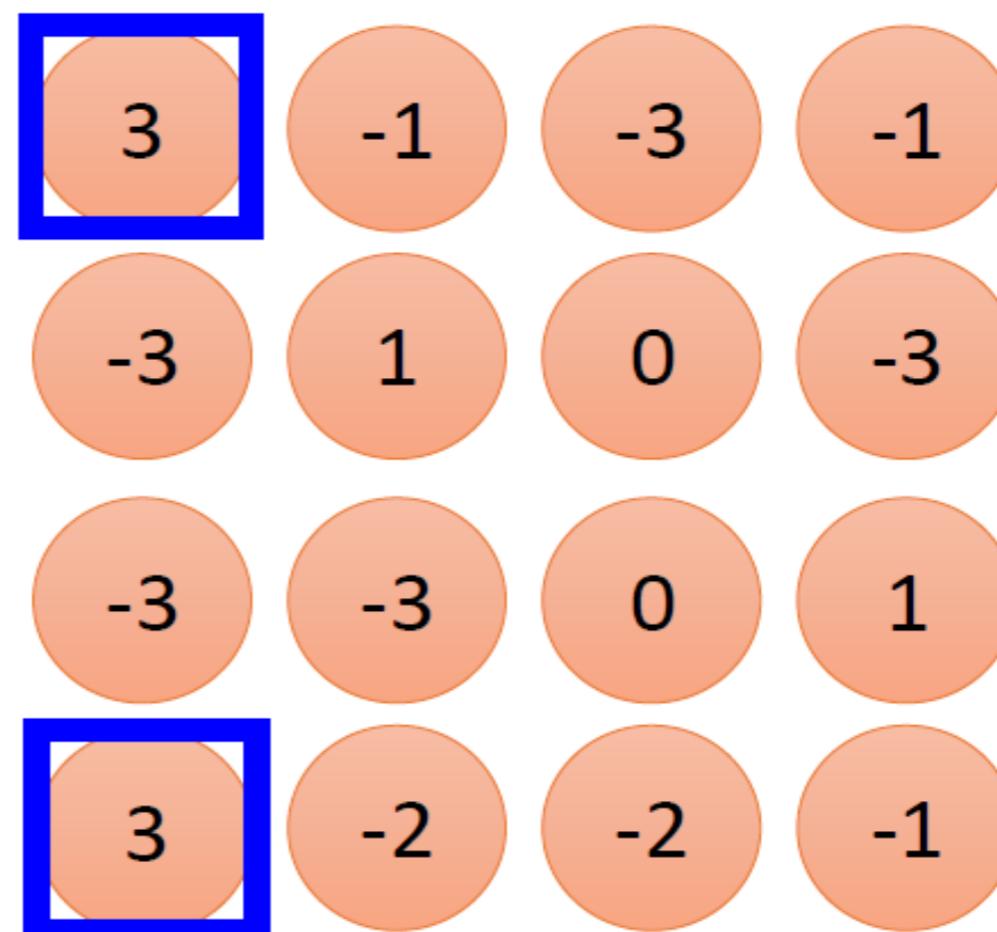
stride=1

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

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



Property 2

CNN – Convolution

stride=1

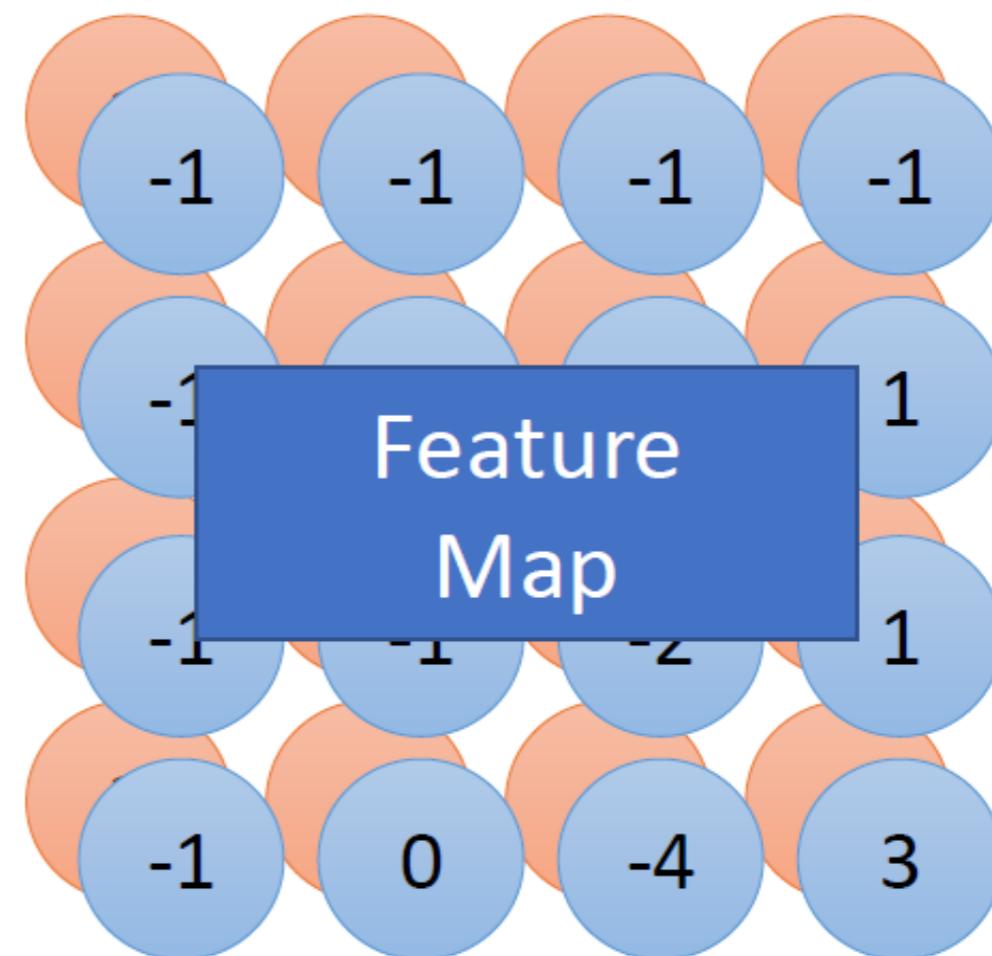
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

-1	1	-1
-1	1	-1
-1	1	-1

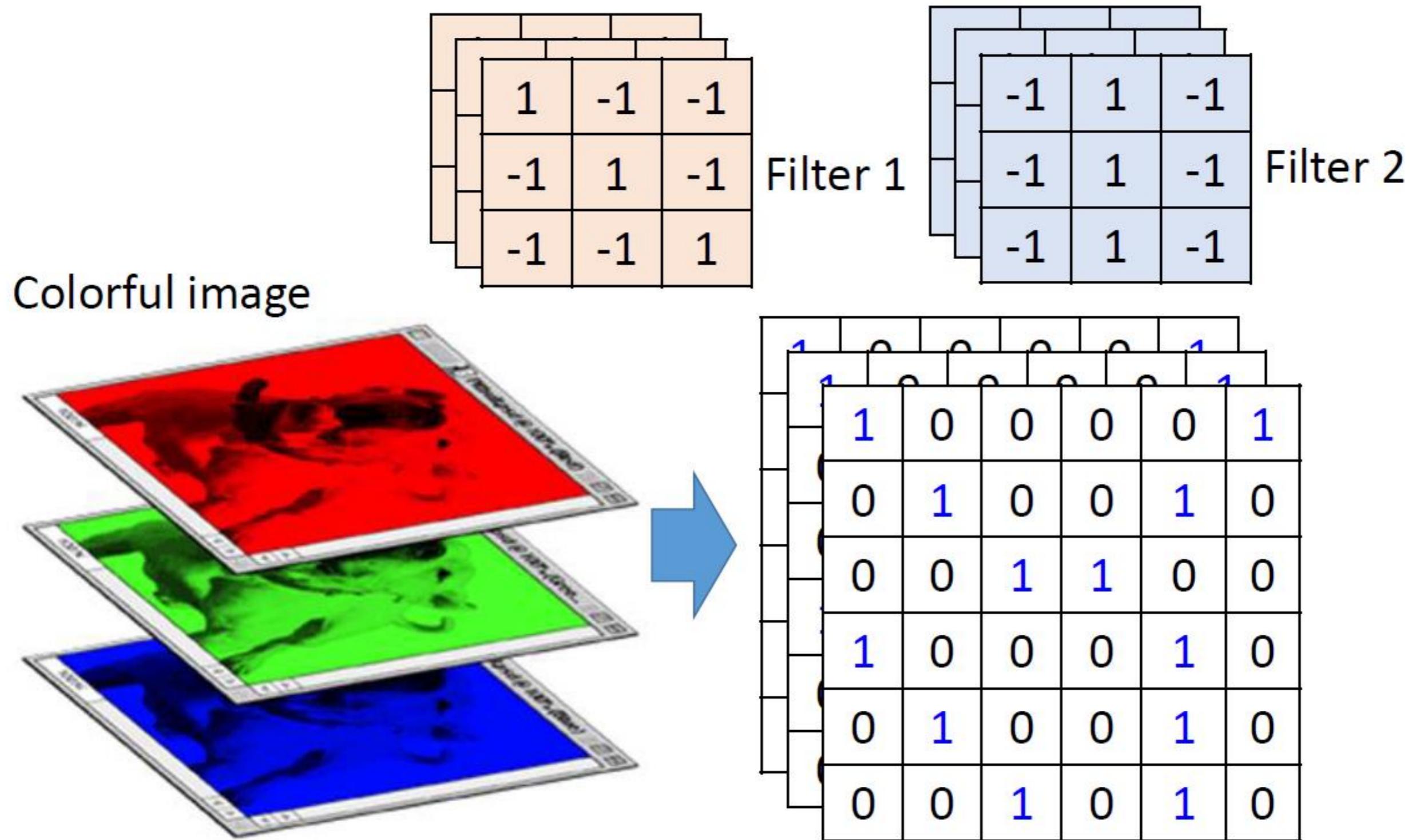
Filter 2

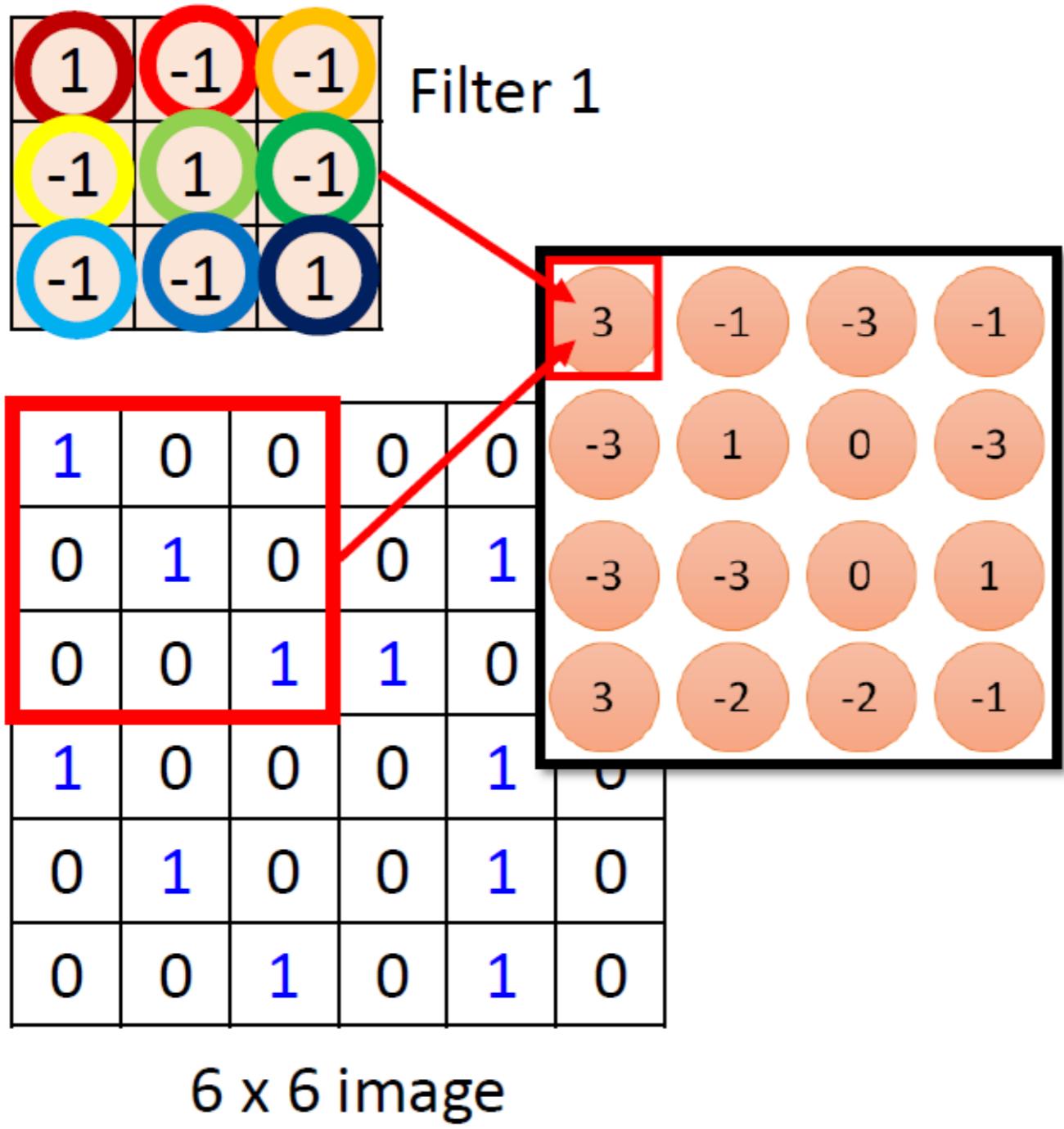
Do the same process for
every filter



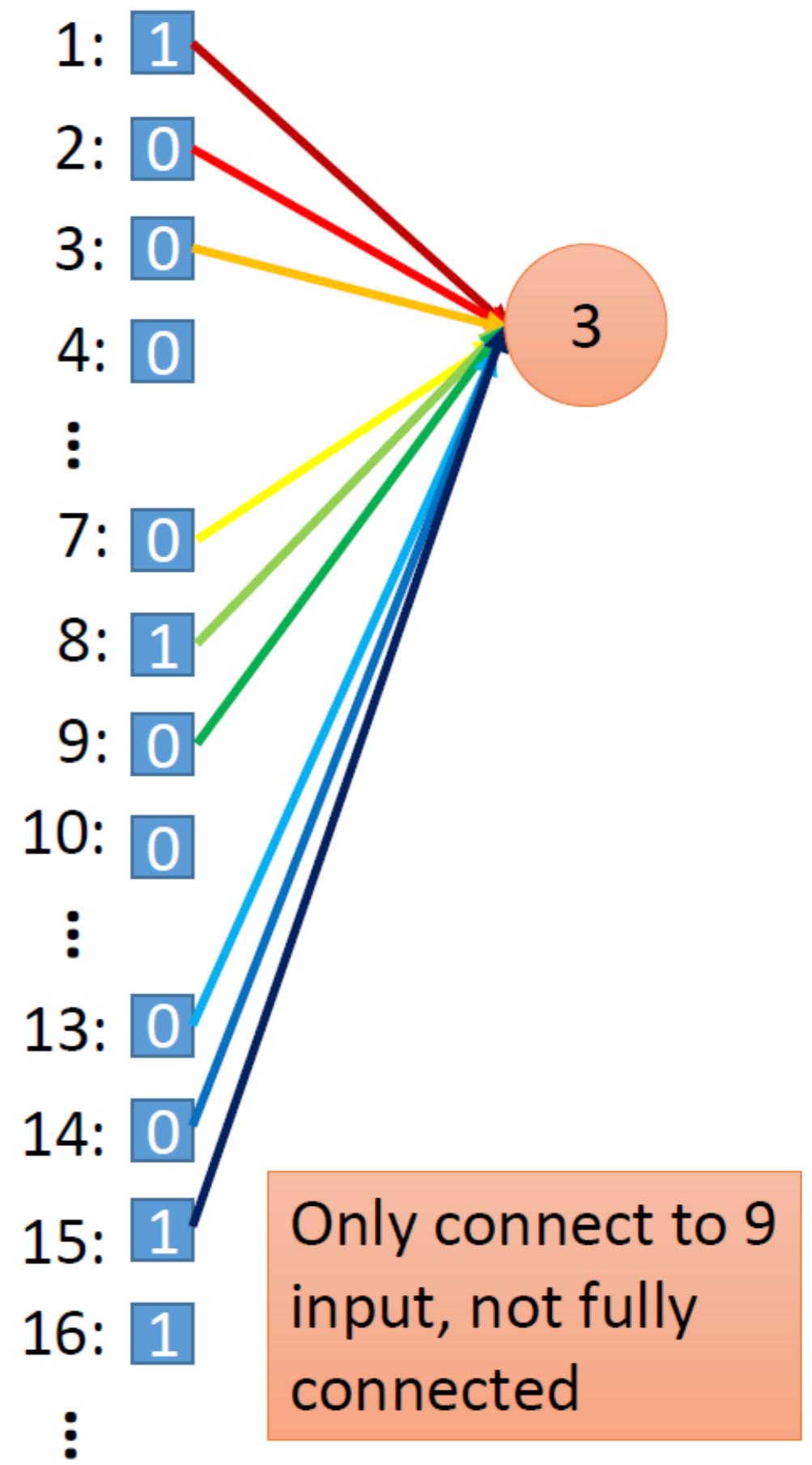
4 x 4 image

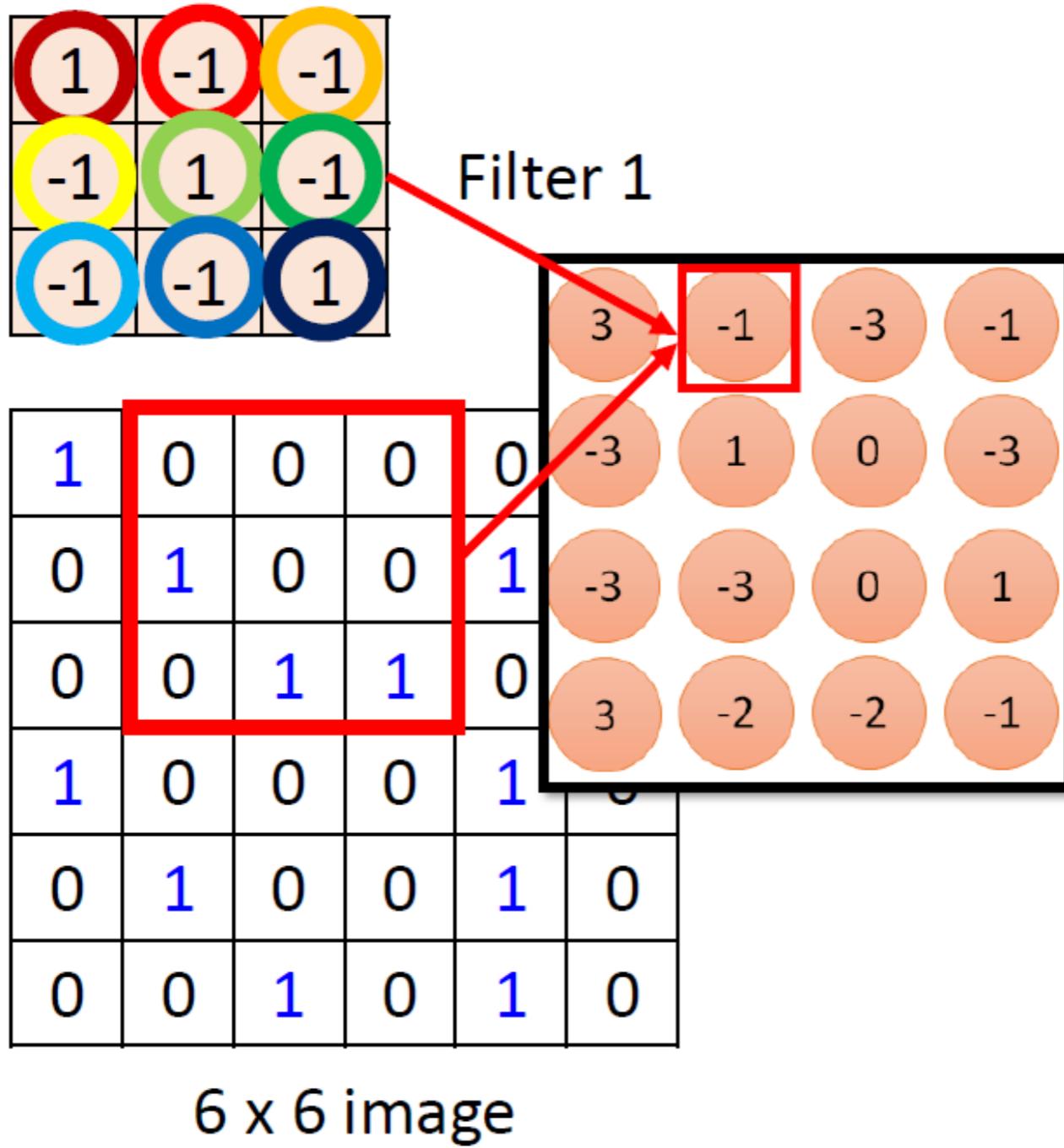
CNN – Colorful image





Less parameters!

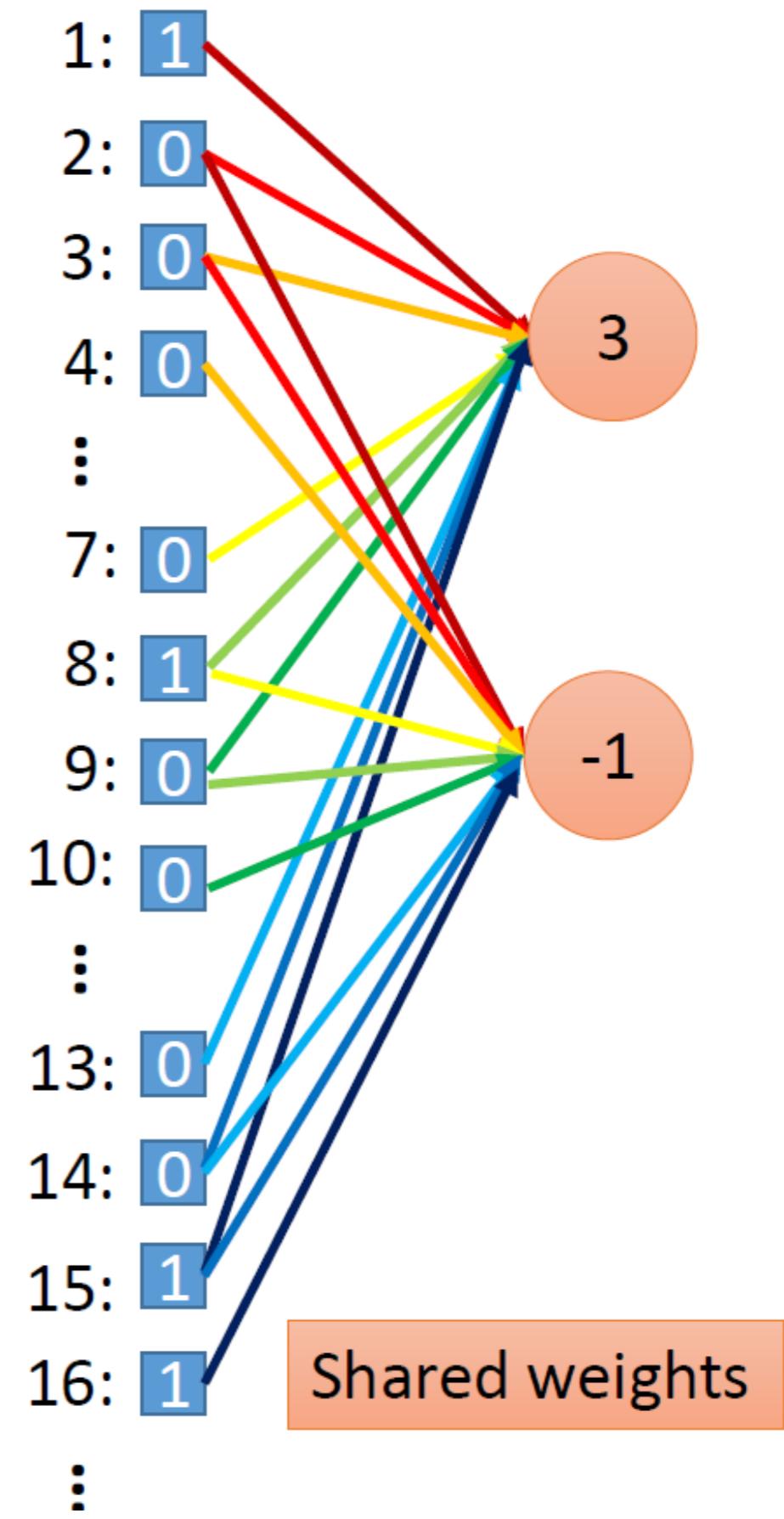




6×6 image

Less parameters!

Even less parameters!



Convolution

Valid and Same convolutions

“Valid”:

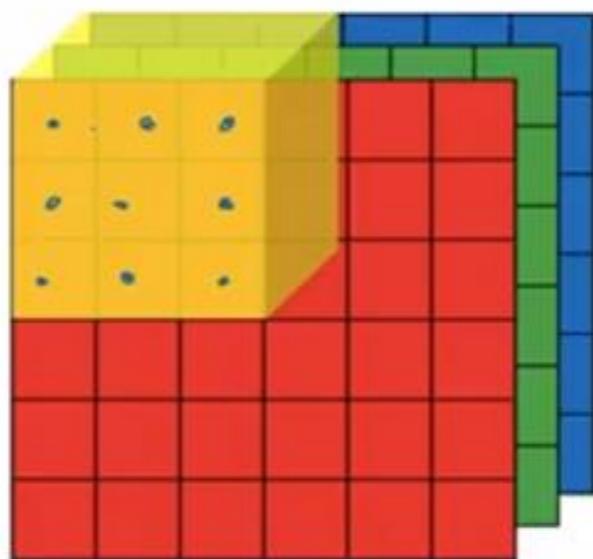
- No zero-padding is used whatsoever, and the convolution kernel is only allowed to visit positions where the entire kernel is contained entirely within the image.
- In MATLAB terminology, this is called **valid** convolution

“Same”:

Pad so that output size is the same as the input size.

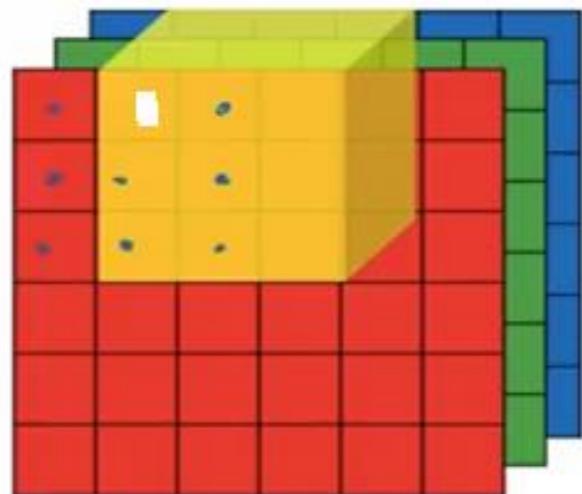
Optimal amount of zero padding (in terms of test set classification accuracy) lies somewhere between “valid” and “same” convolution.

Convolutions on RGB image



$$\begin{matrix} * & \begin{matrix} \text{Input Tensor} \\ \text{4x4} \end{matrix} & = & \begin{matrix} \text{Output Tensor} \\ \text{4x4} \end{matrix} \end{matrix}$$

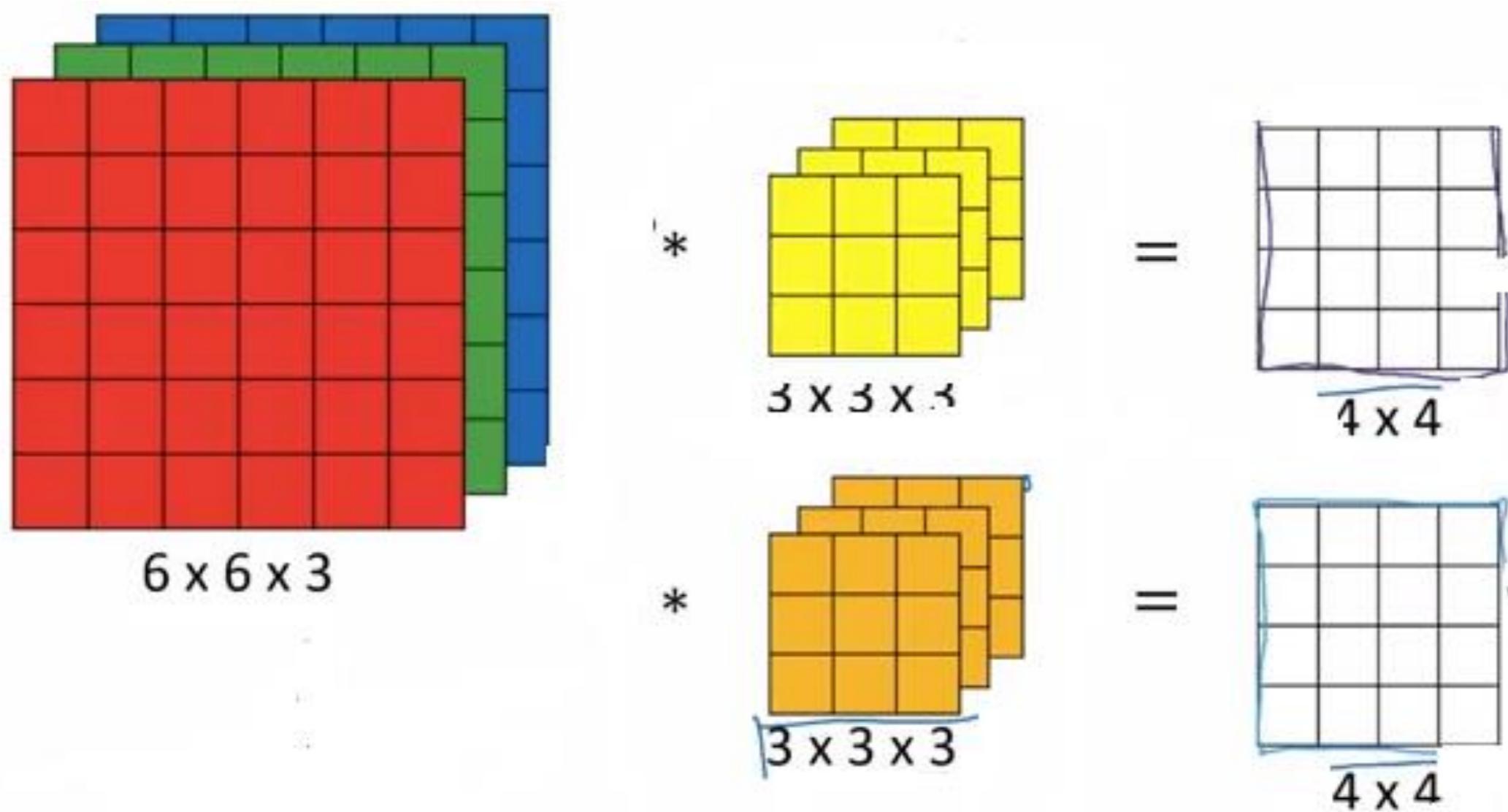
The input tensor is multiplied by a convolutional kernel. The output tensor is a 4x4 matrix where each element is the result of the convolution operation. The output tensor is shown as a yellow cube.



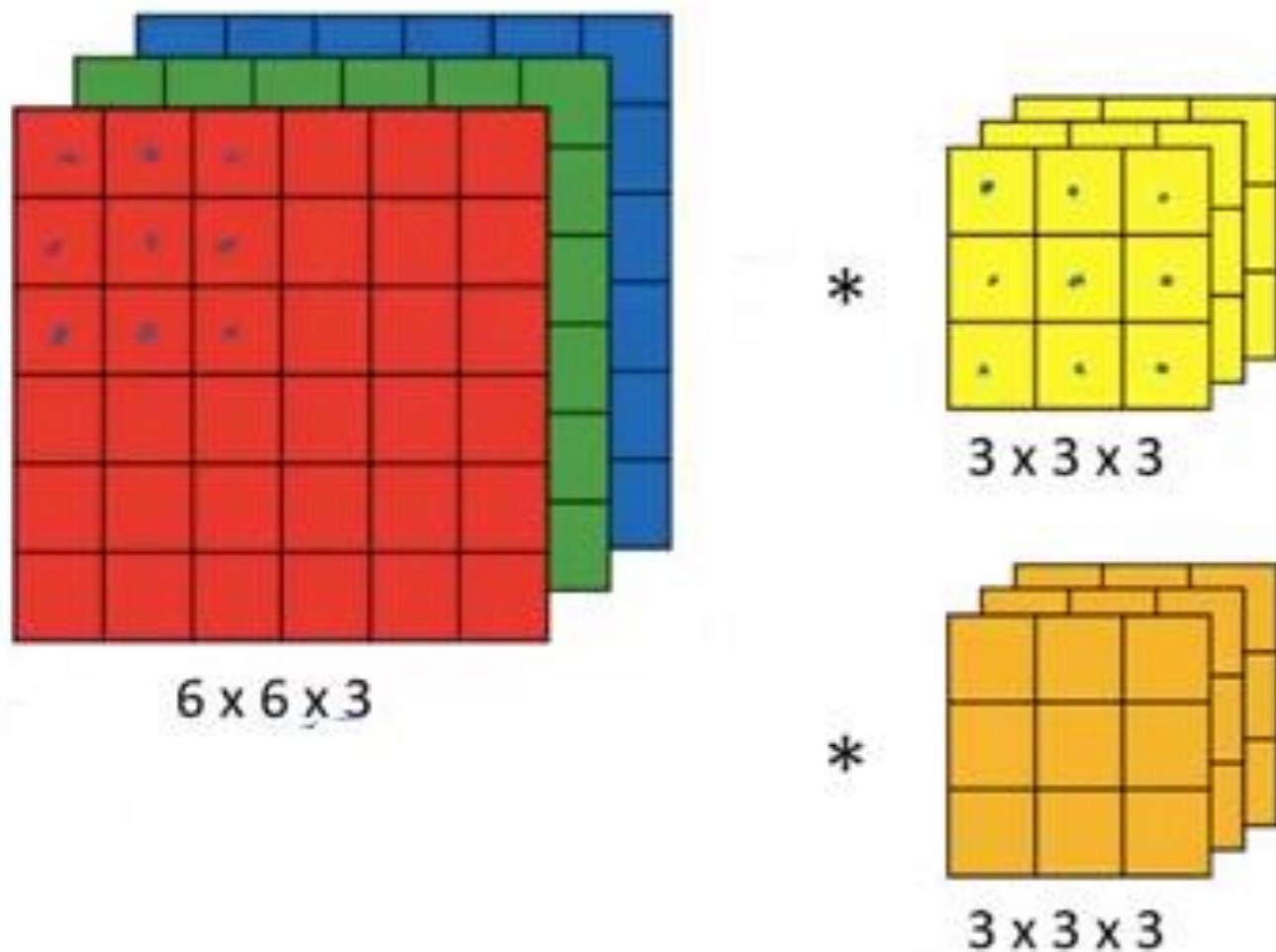
$$\begin{matrix} * & \begin{matrix} \text{Input Tensor} \\ \text{4x4} \end{matrix} & = & \begin{matrix} \text{Output Tensor} \\ \text{4x4} \end{matrix} \end{matrix}$$

The input tensor is multiplied by a convolutional kernel. The output tensor is a 4x4 matrix where each element is the result of the convolution operation. The output tensor is shown as a yellow cube.

Convolution-Multiple Filters



Convolution-Layer example



Convolution- no. of parameters in one layer

If you have 10 filters that are $3 \times 3 \times 3$ in one layer of a neural network, how many parameters does that layer have?

Given:

Number of filters = 10

Each filter size = $3 \times 3 \times 3$

3×3 is the spatial size

3 is the depth (e.g., RGB image with 3 channels)

Parameters per filter: $3 \times 3 \times 3 = 27$ weights

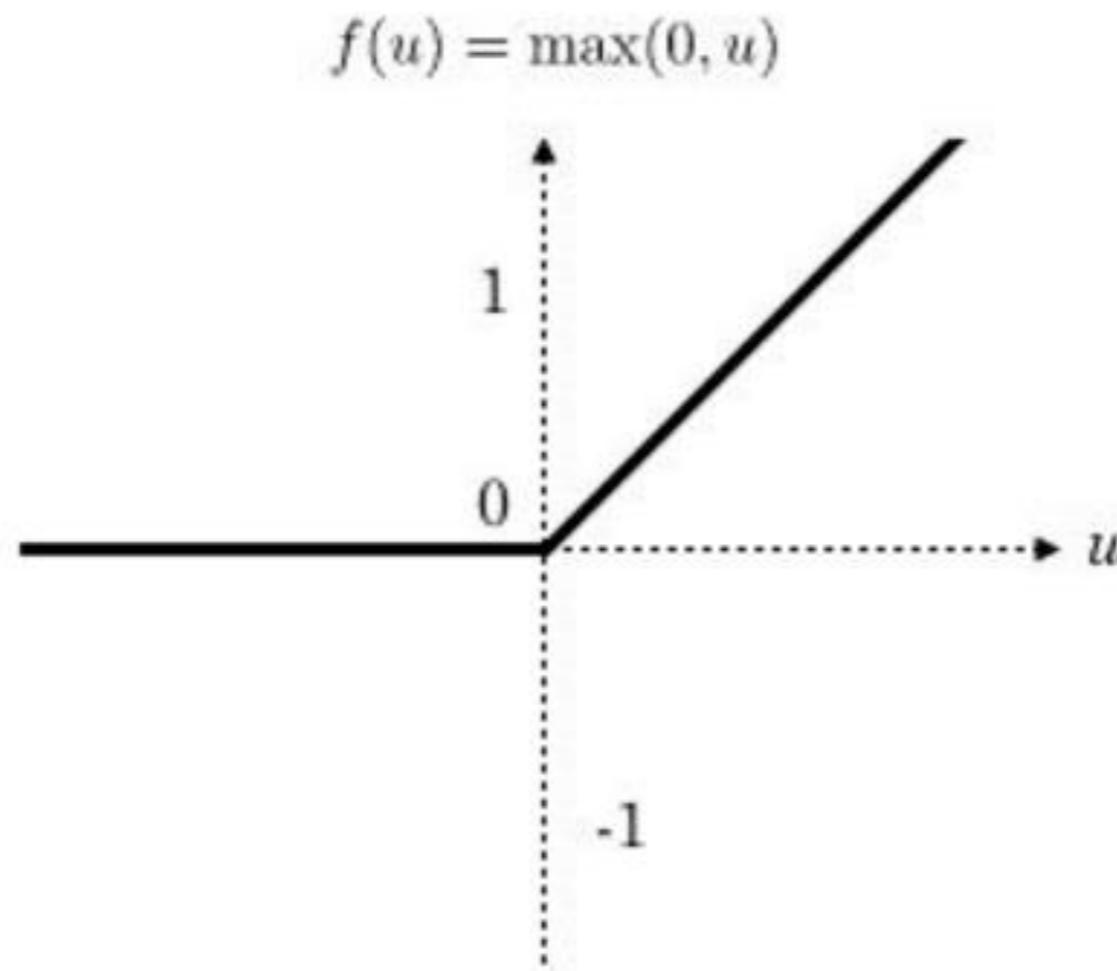
Each filter also has 1 bias term

So, total parameters per filter = $27 + 1 = 28$

Total parameters for 10 filters: $28 \times 10 = 280$ parameters

Rectified linear unit, ReLU

rectified linear function, $f(x) = \max(0, x)$



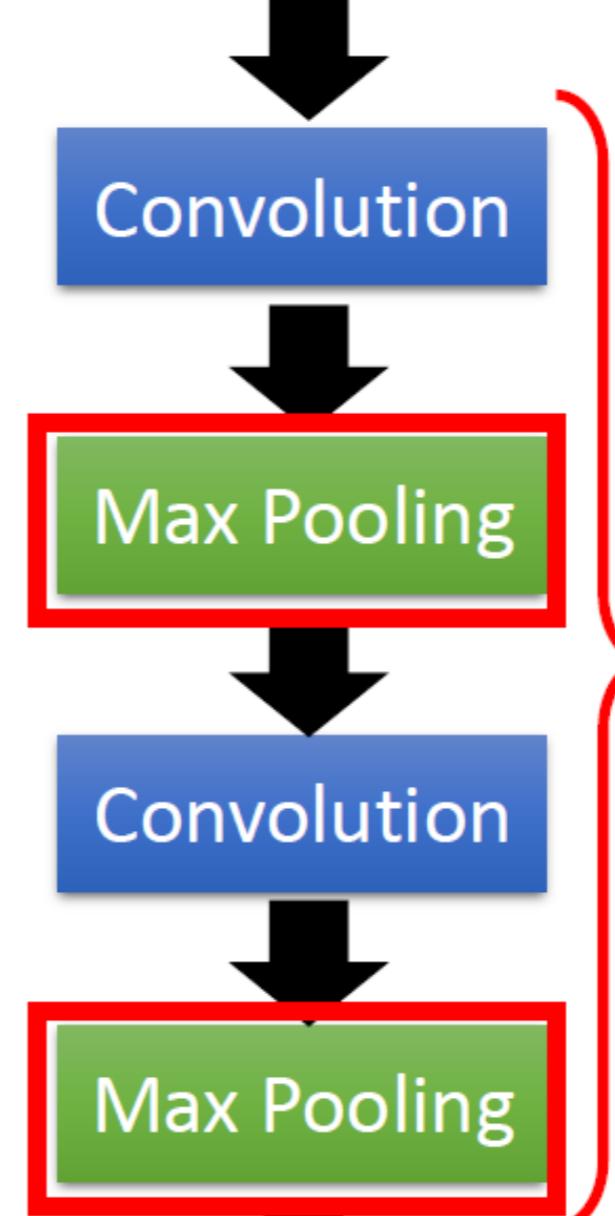
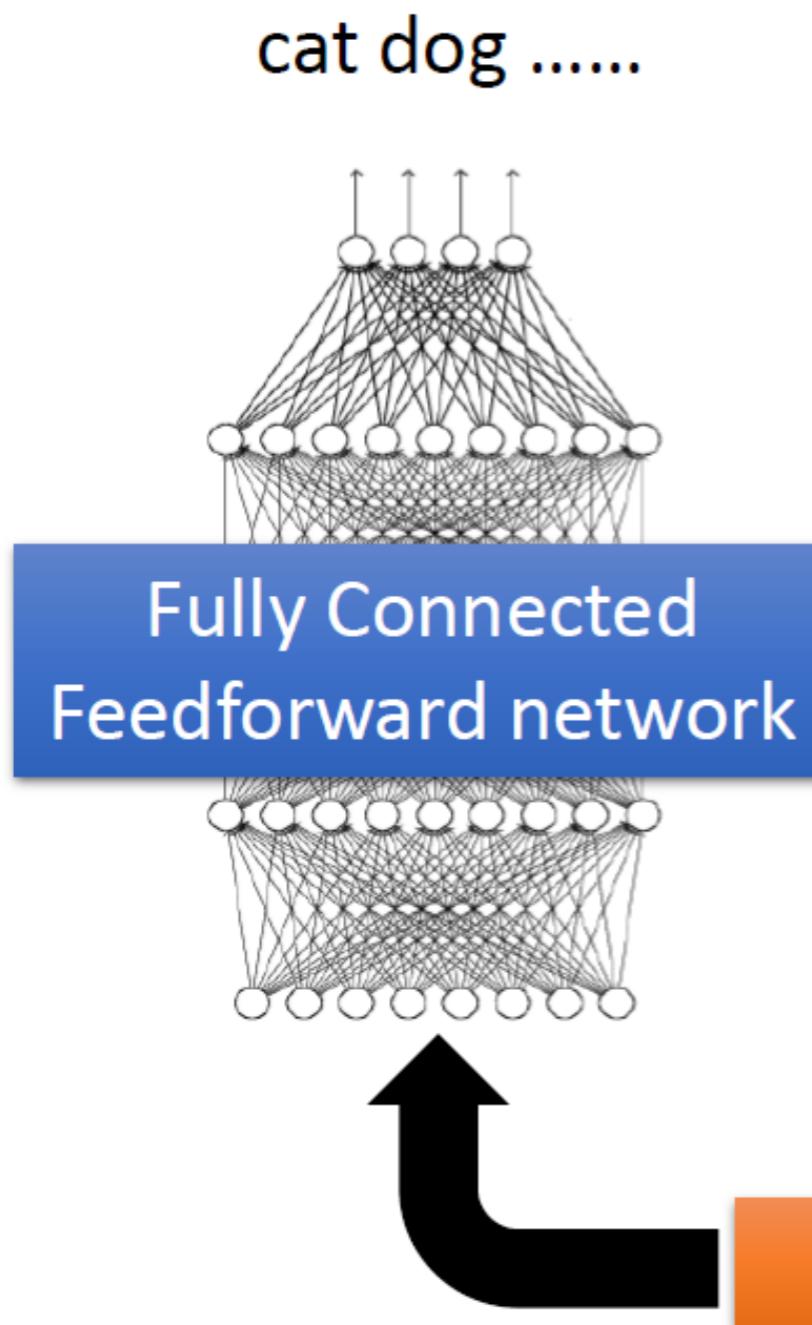
Convolution

If the 2d convolutional layer has 10 filters of 3×3 shape and the input to the convolutional layer is $24 \times 24 \times 3$, then this actually means that the filters will have shape $3 \times 3 \times 3$, i.e. each filter will have the 3rd dimension that is equal to the 3rd dimension of the input. So, the 3rd dimension of the kernel is not given because it can be determined from the 3rd dimension of the input.

Example of ConvNet

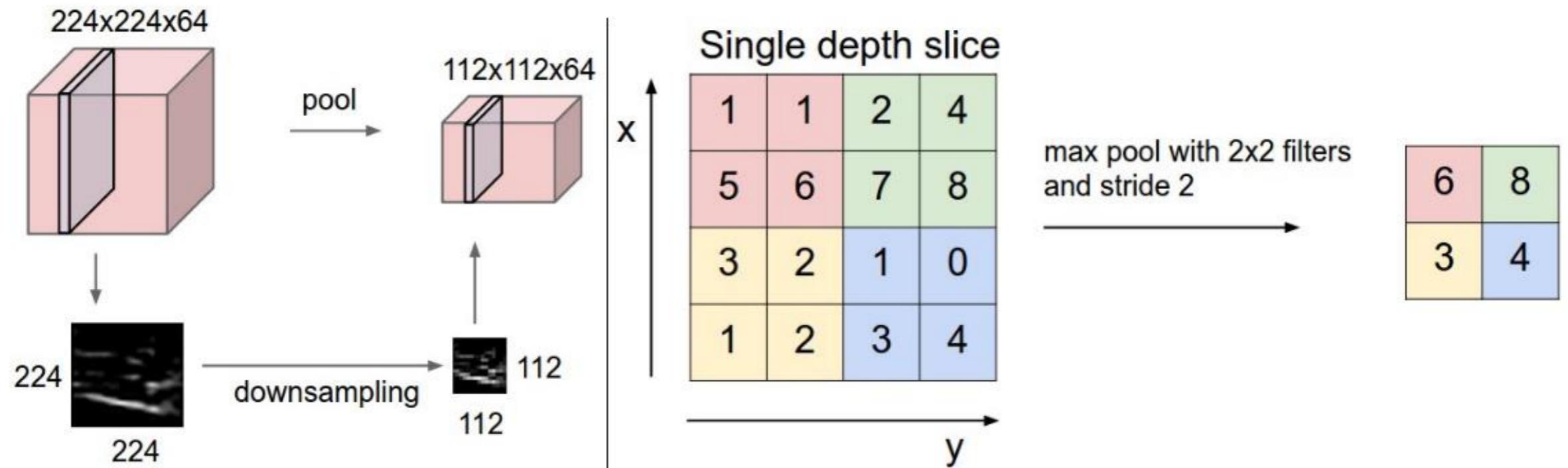
CNN-Pooling layer

The whole CNN



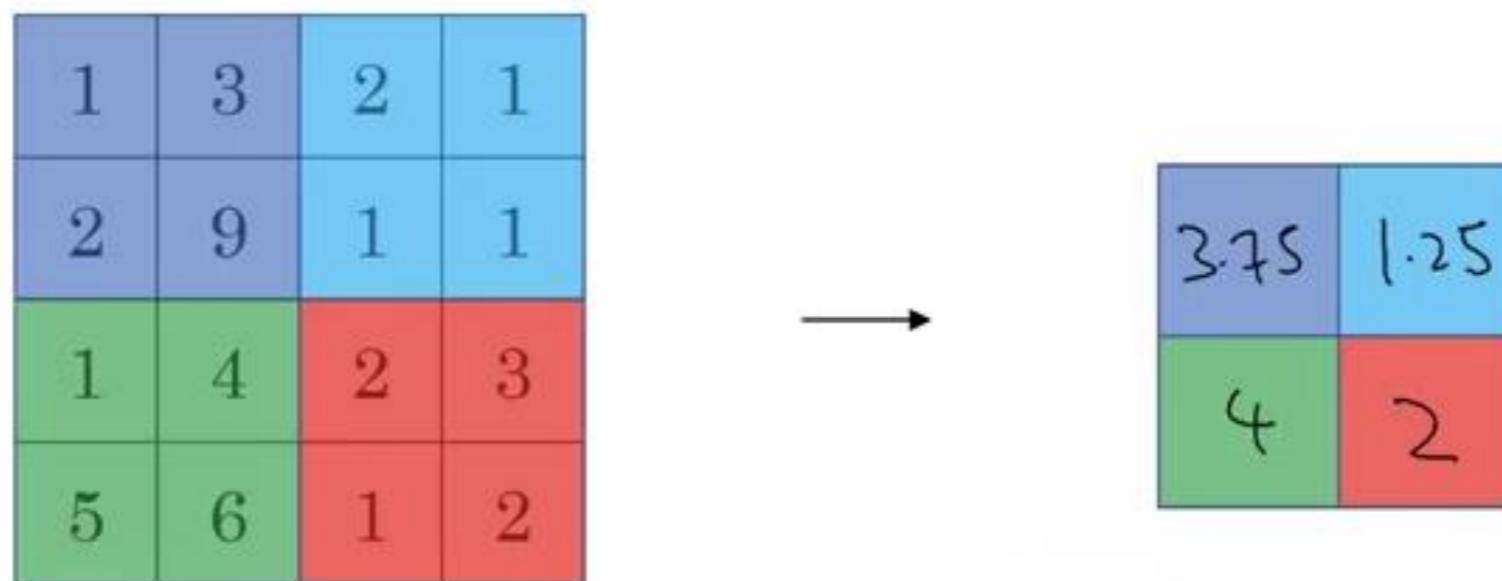
Can repeat
many times

Pooling layer



Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size $[224 \times 224 \times 64]$ is pooled with filter size 2, stride 2 into output volume of size $[112 \times 112 \times 64]$. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2×2 square).

Pooling layer-avg pooling



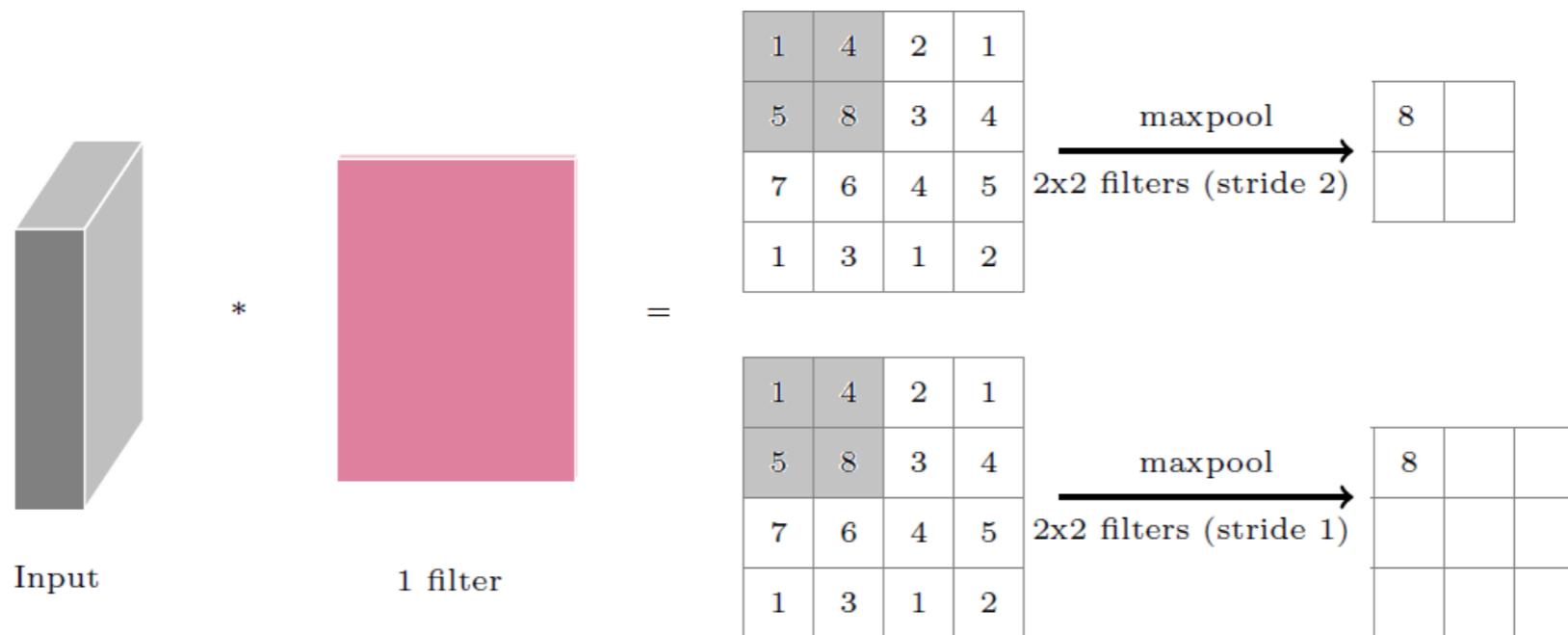
Hyperparameters:

f : filter size

s : stride

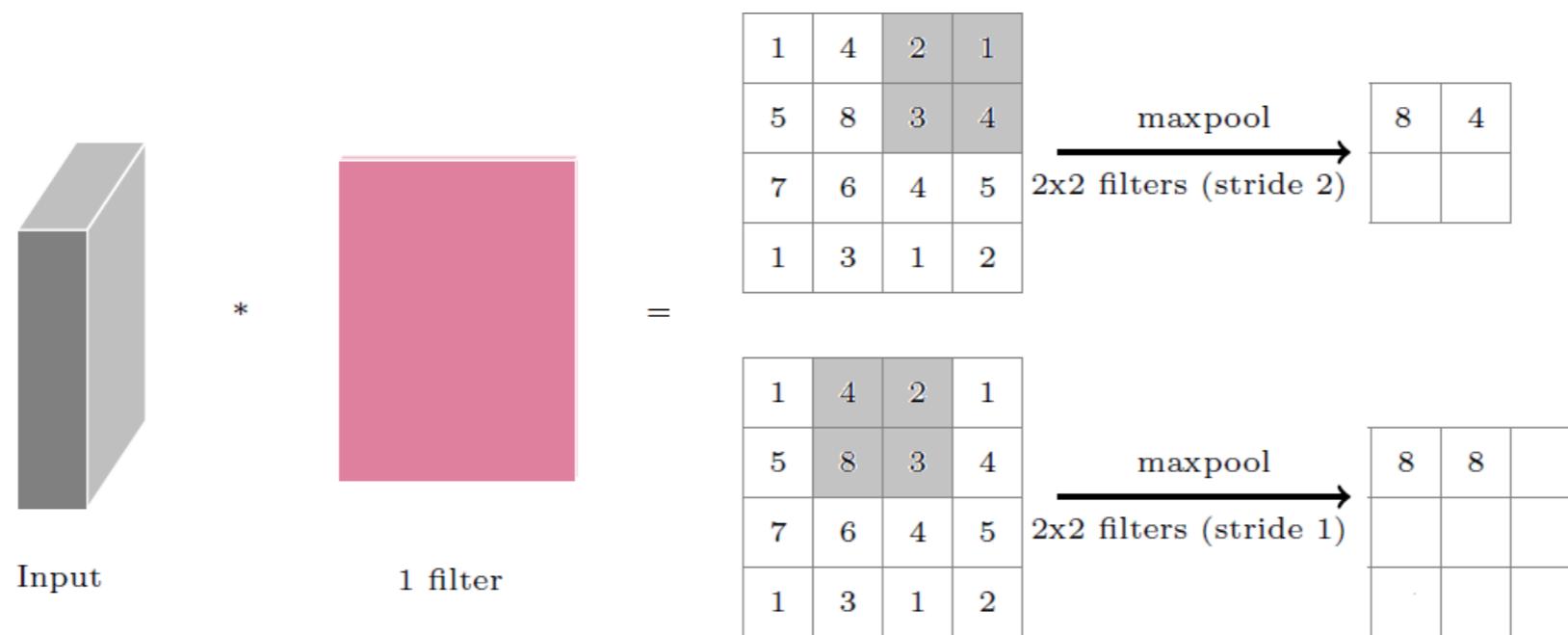
Max or average pooling

Pooling Layer



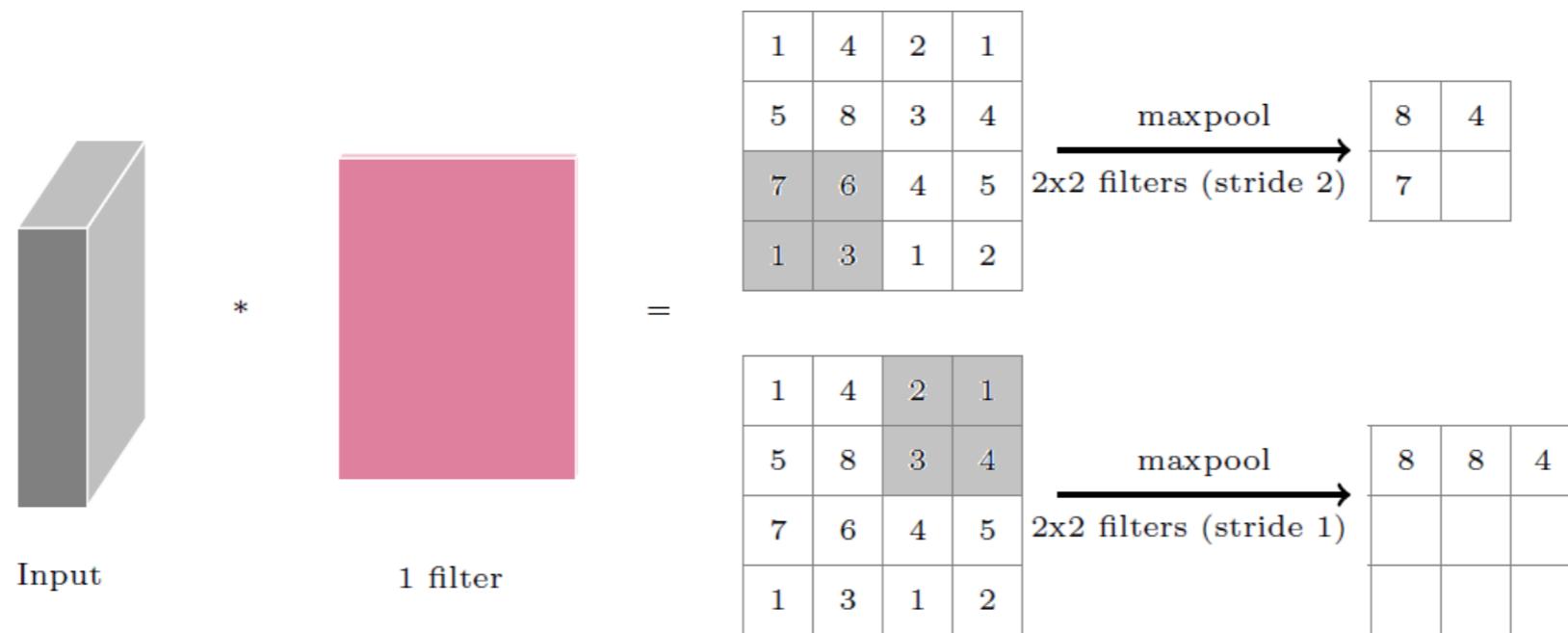
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



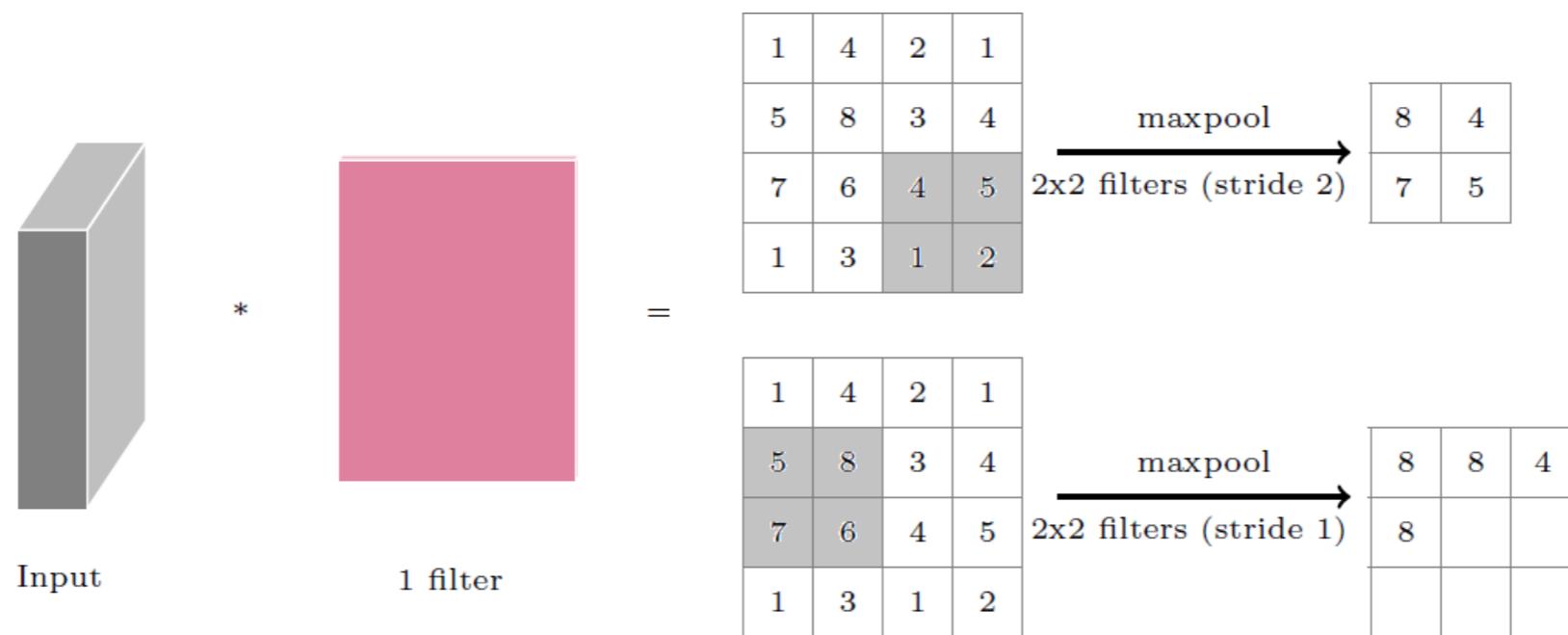
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



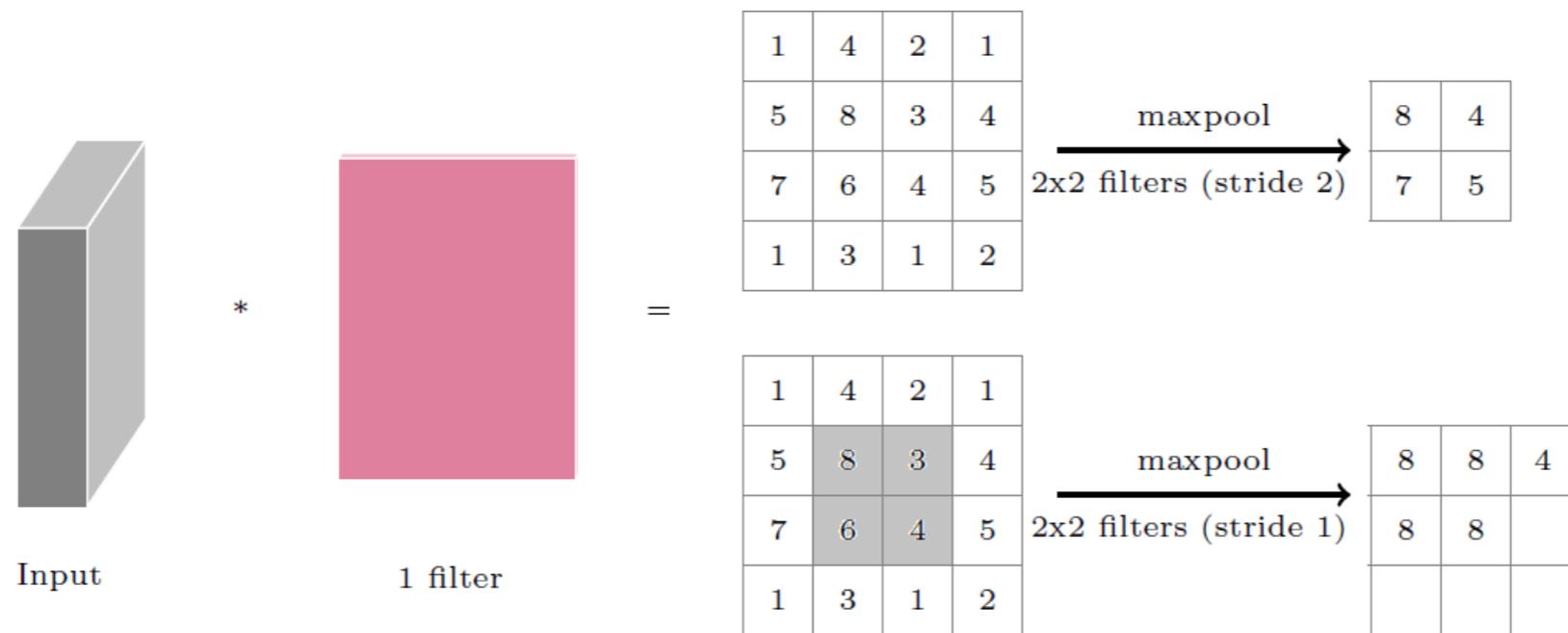
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



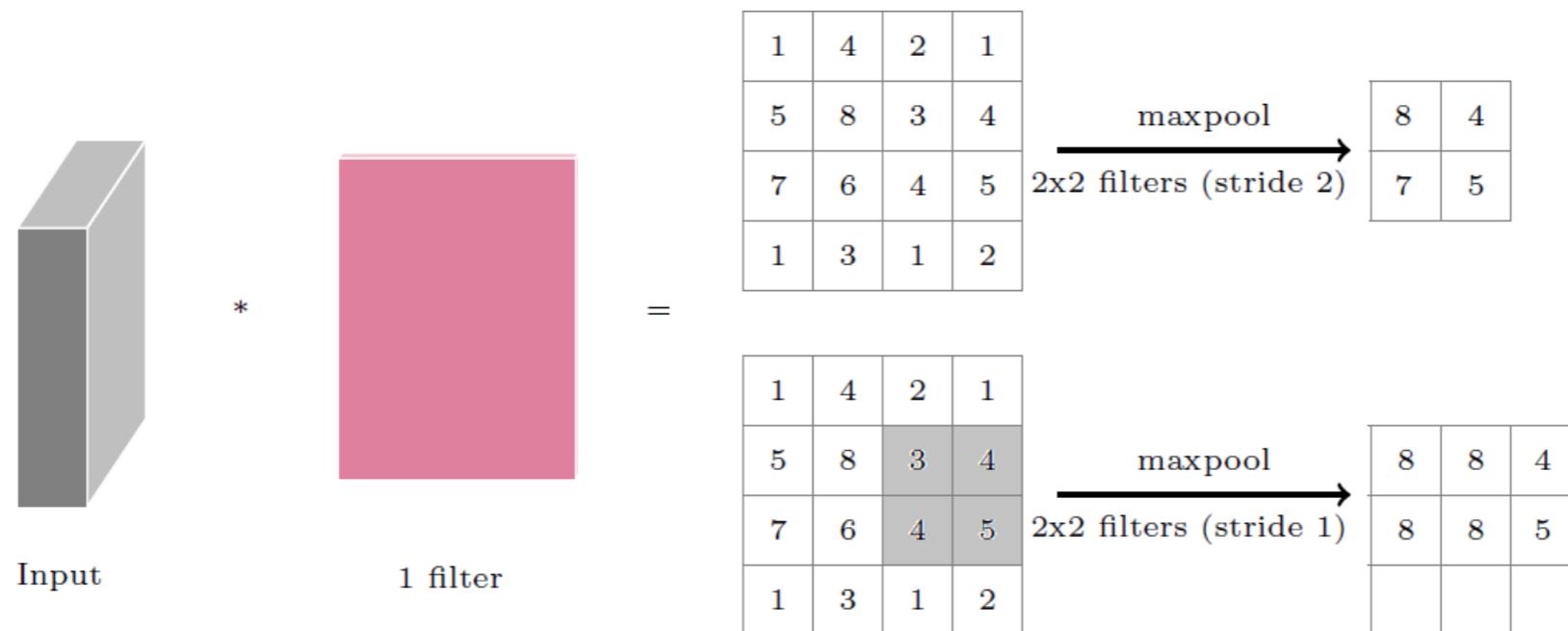
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



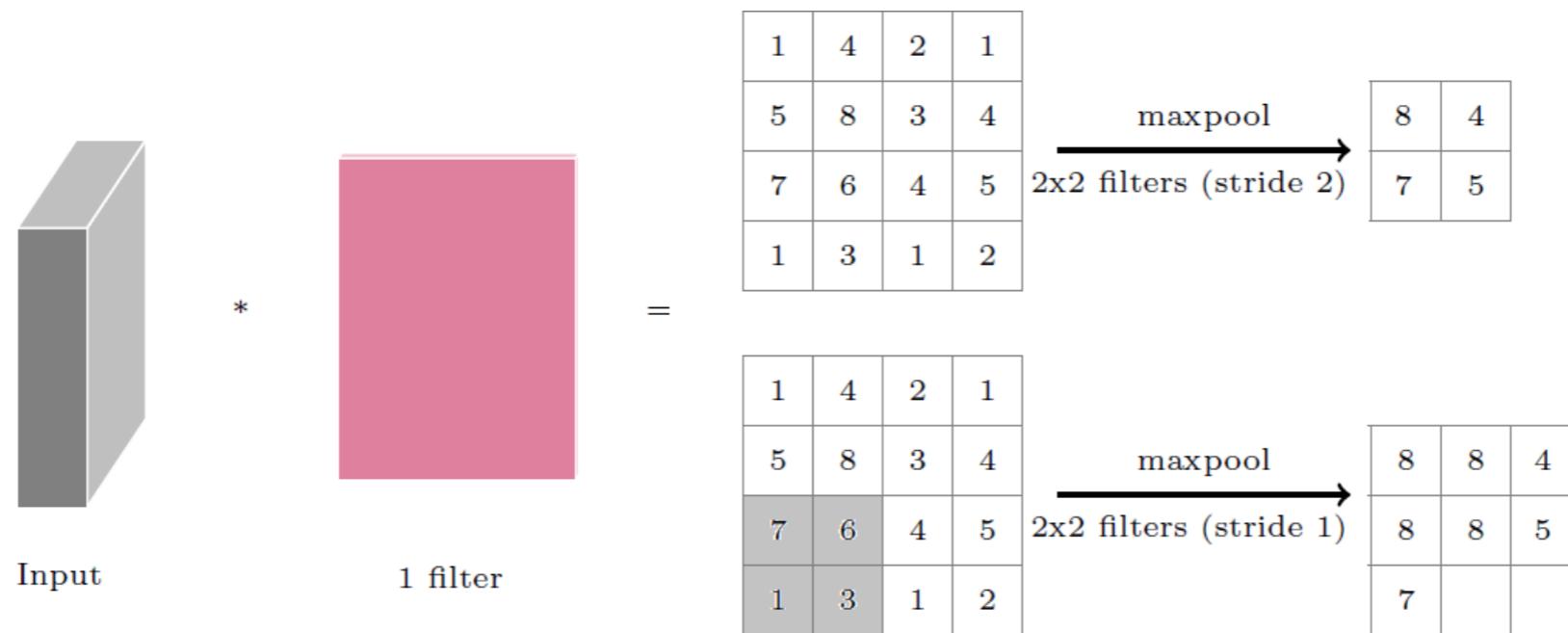
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



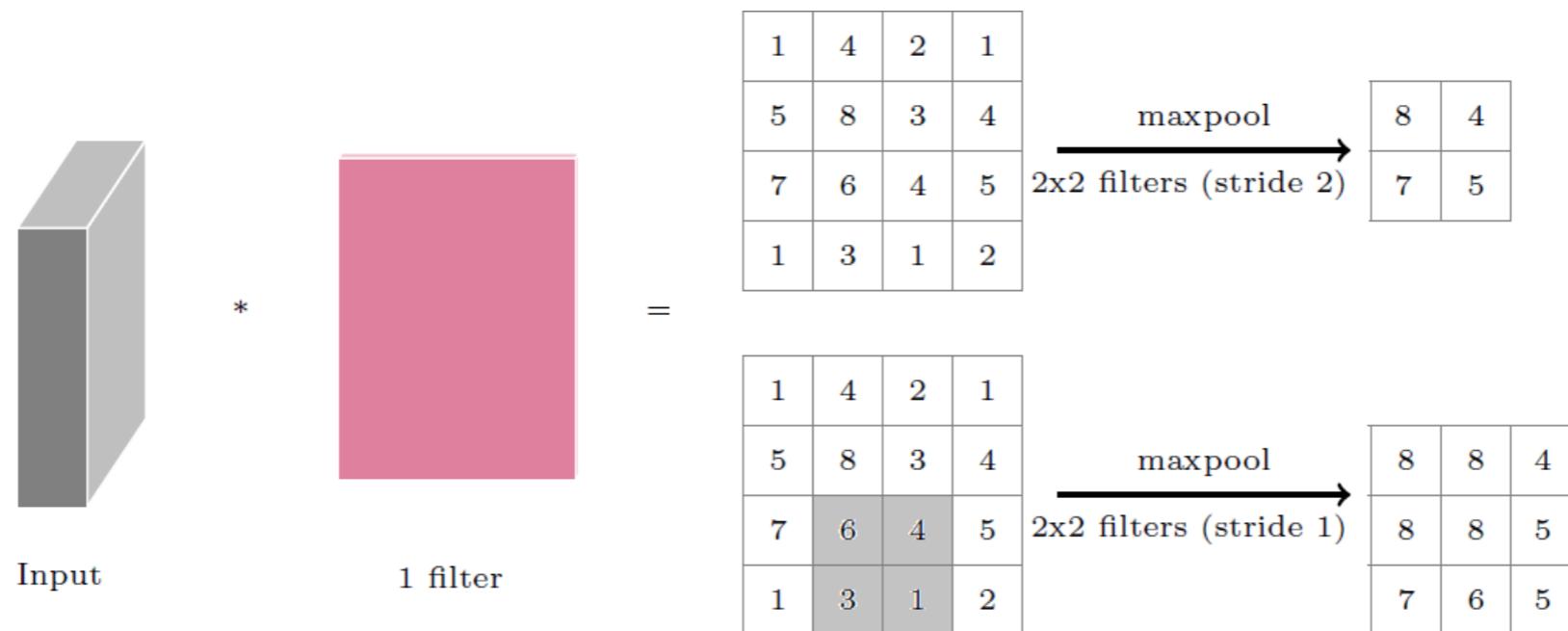
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



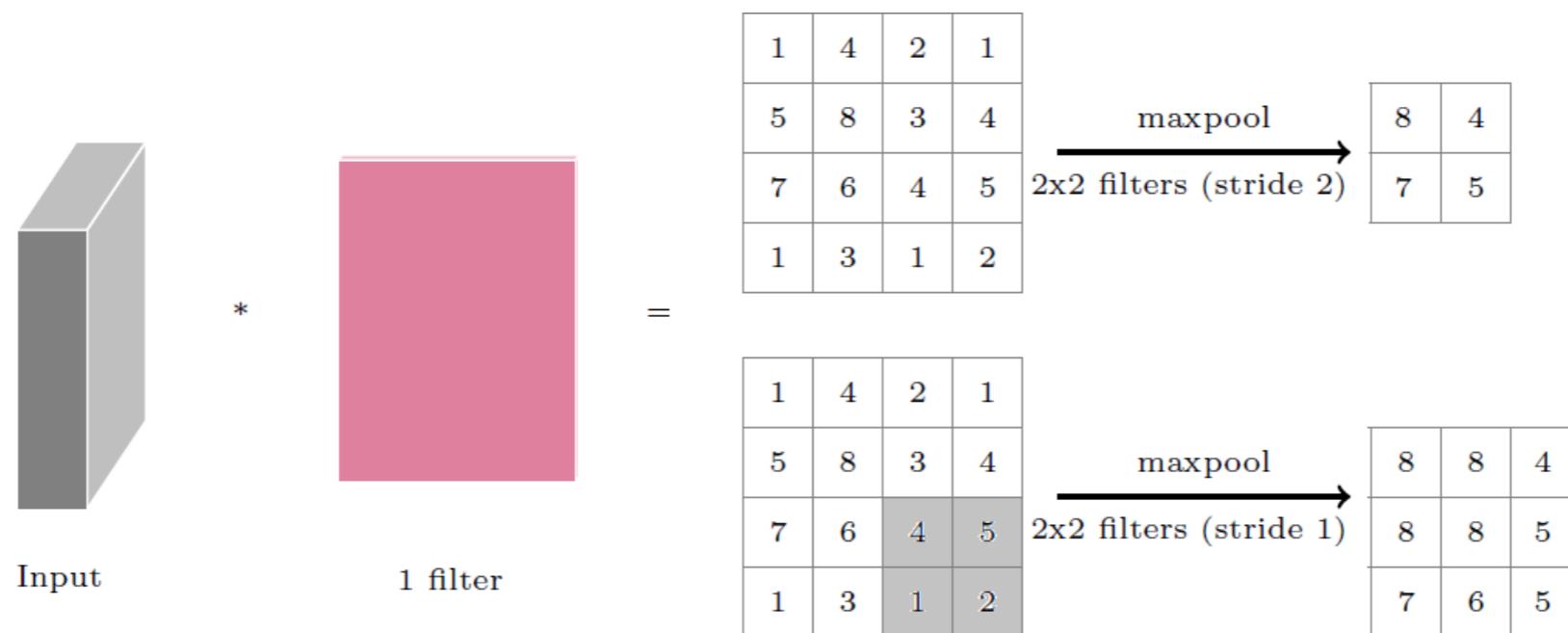
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



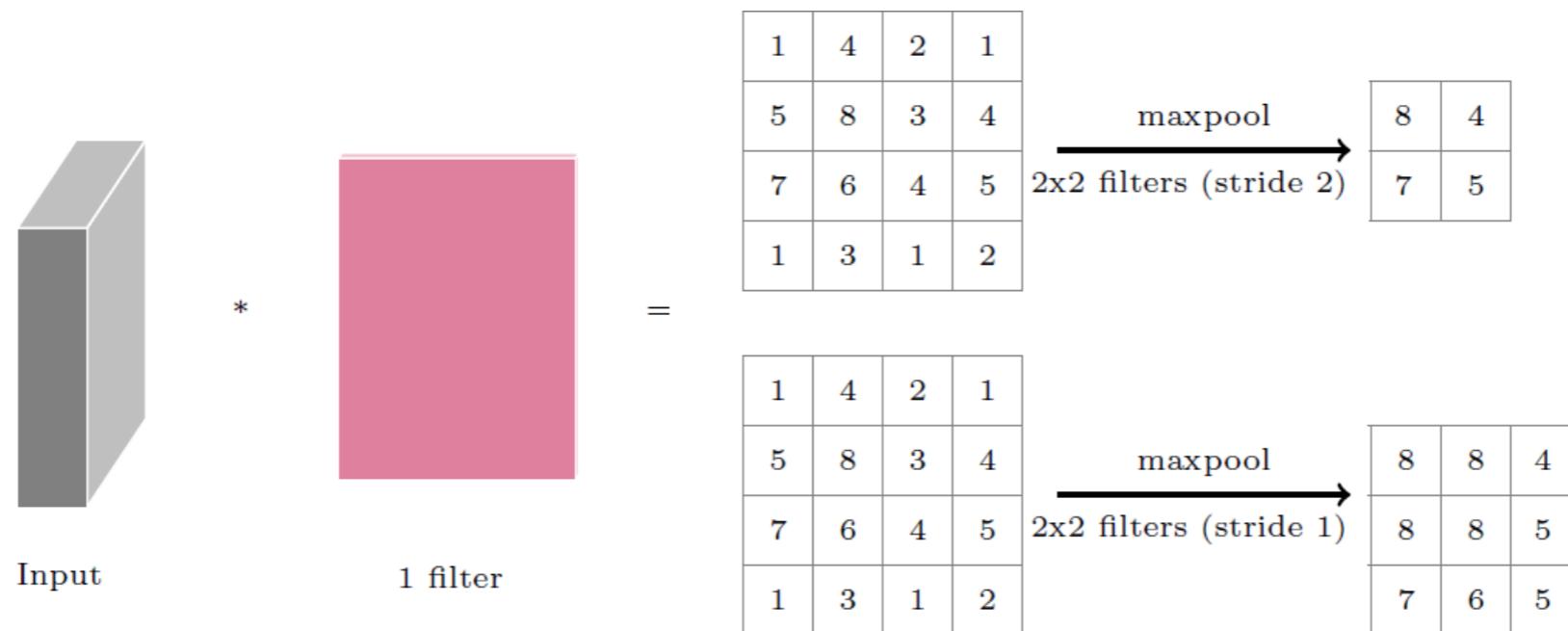
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



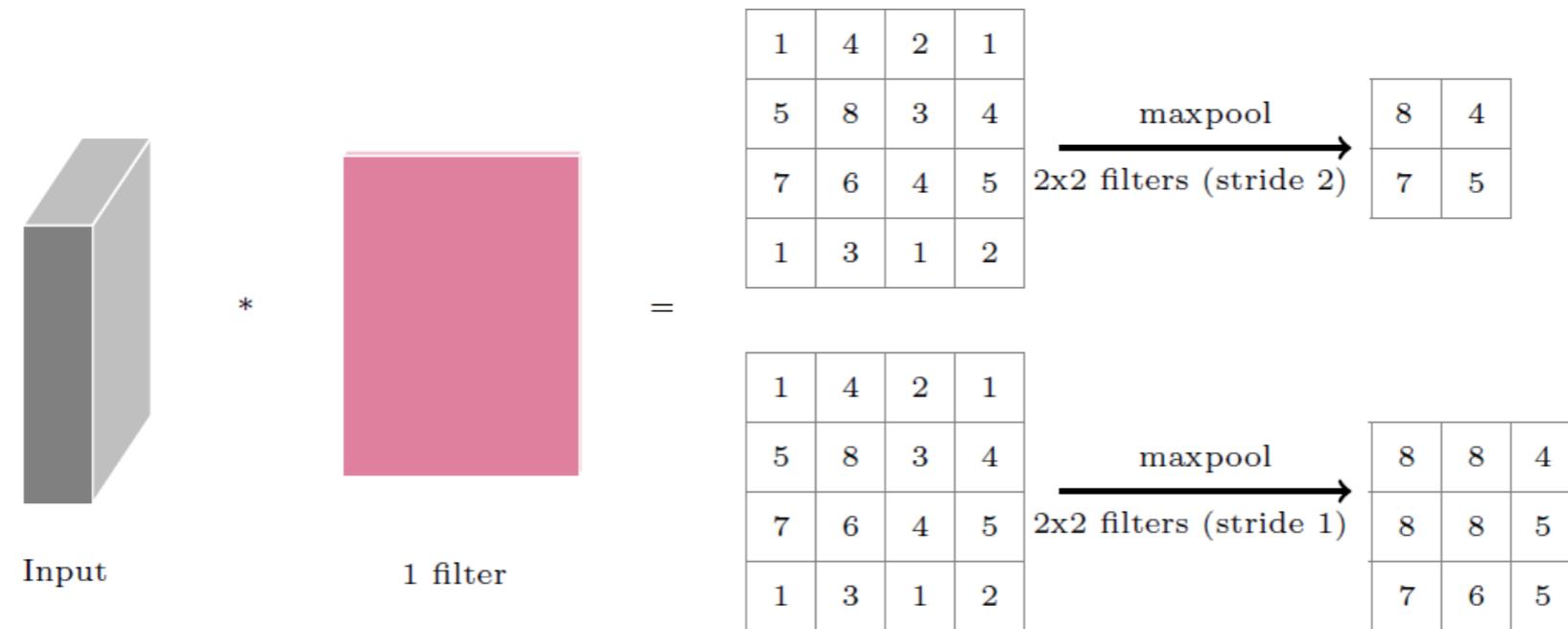
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



- **Pooling** is a parameter-free down sampling operation

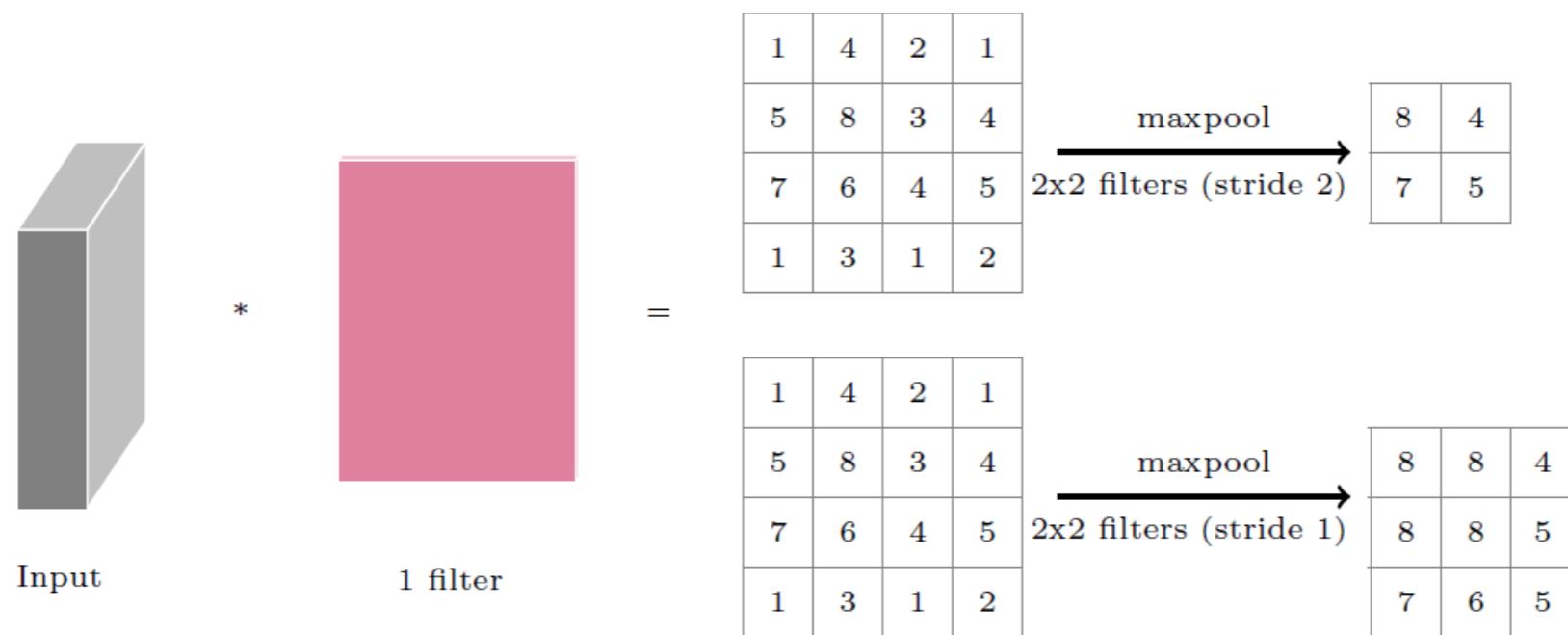
Pooling Layer



- **Pooling** is a parameter-free down sampling operation
- Instead of **Max Pooling**, we can also do **Average Pooling**, L_2 **Pooling**, etc

Pooling Layer

L2 pooling, also known as root mean square (RMS) pooling, computes the square root of the sum of squares within the window.



- **Pooling** is a parameter-free down sampling operation
- Instead of **Max Pooling**, we can also do **Average Pooling**, L_2 **Pooling**, etc
- Other notable mentions: Mixed Pooling (combines max and average pooling), Spatial Pyramid Pooling, Spectral Pooling

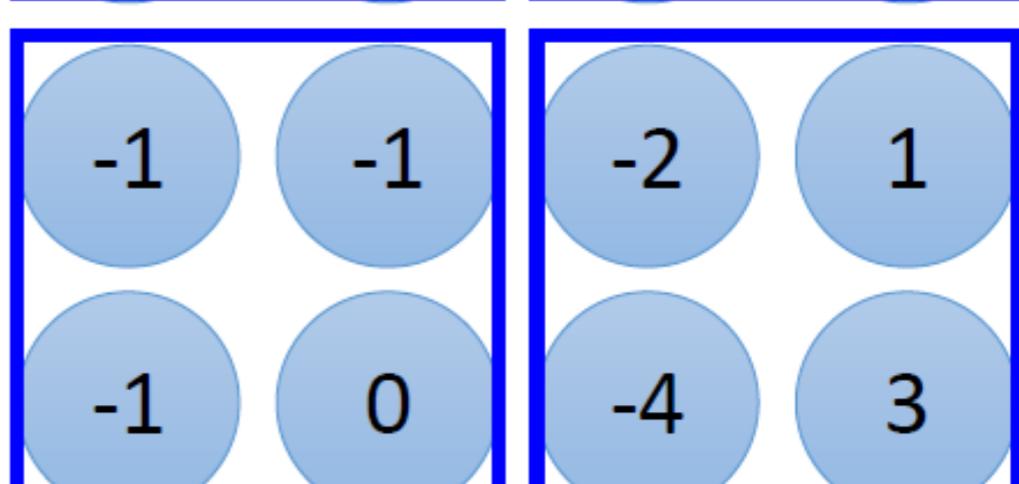
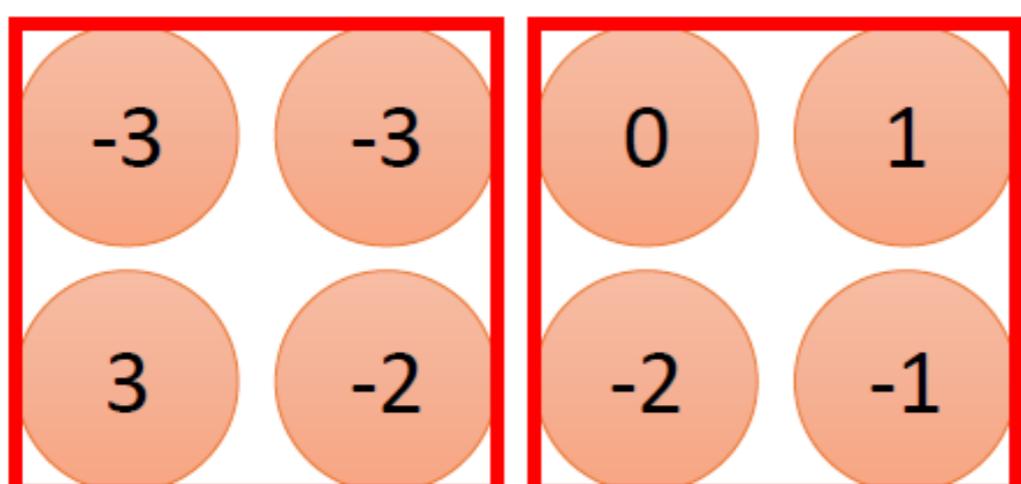
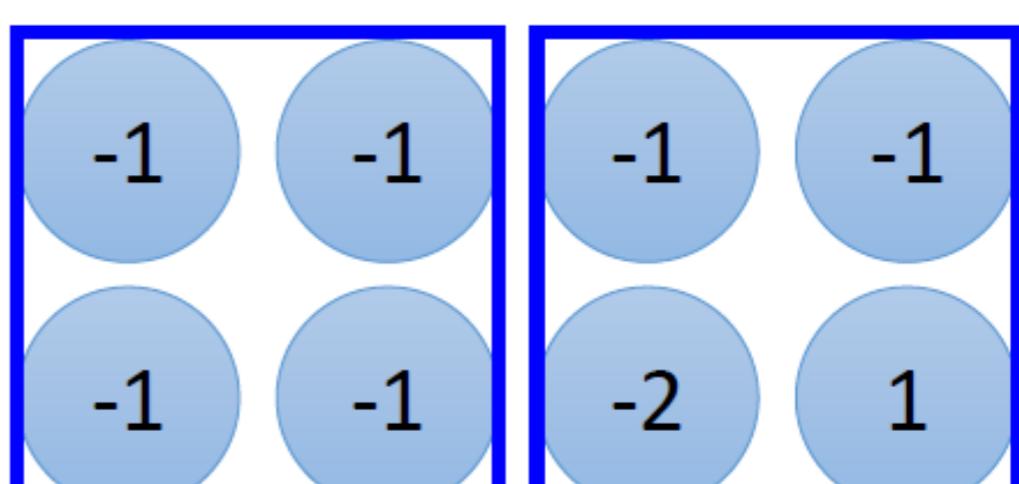
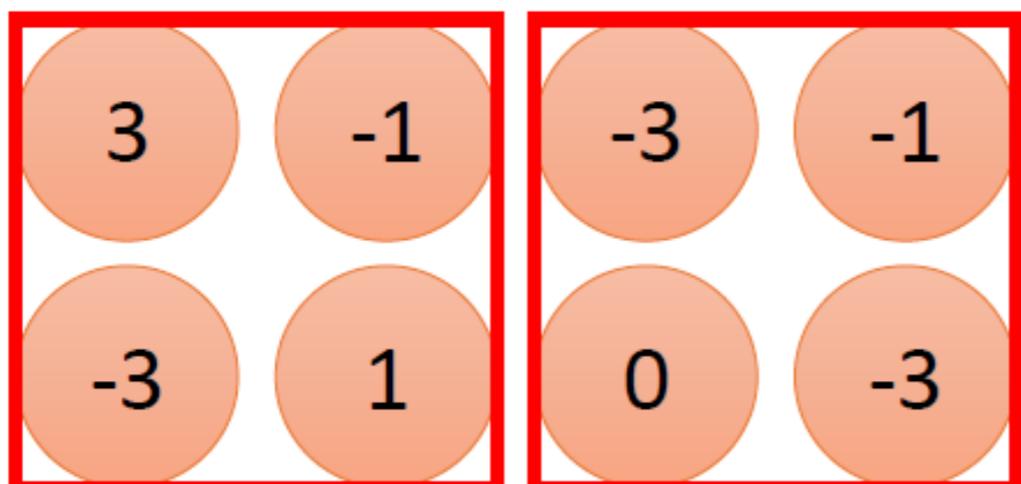
CNN – Max Pooling

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

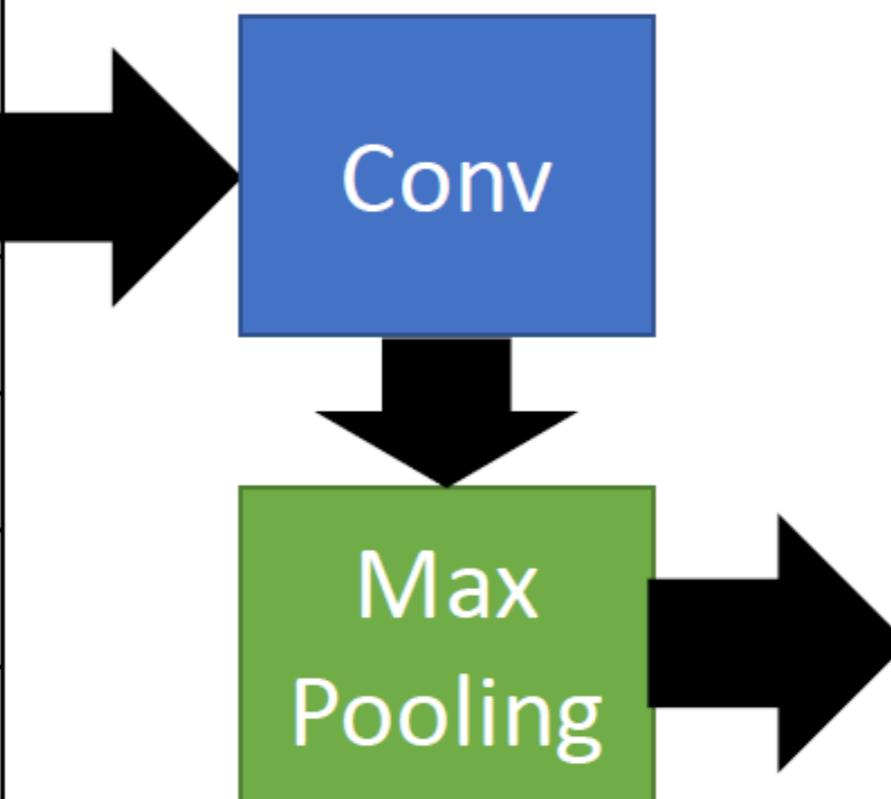
Filter 2



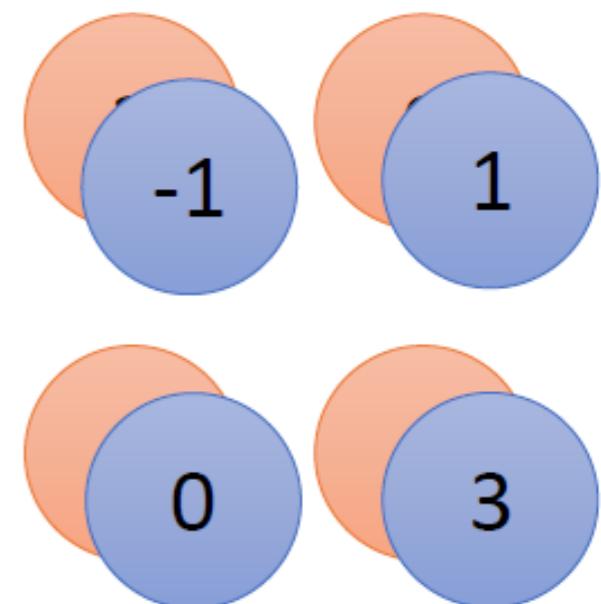
CNN – Max Pooling

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

6 x 6 image



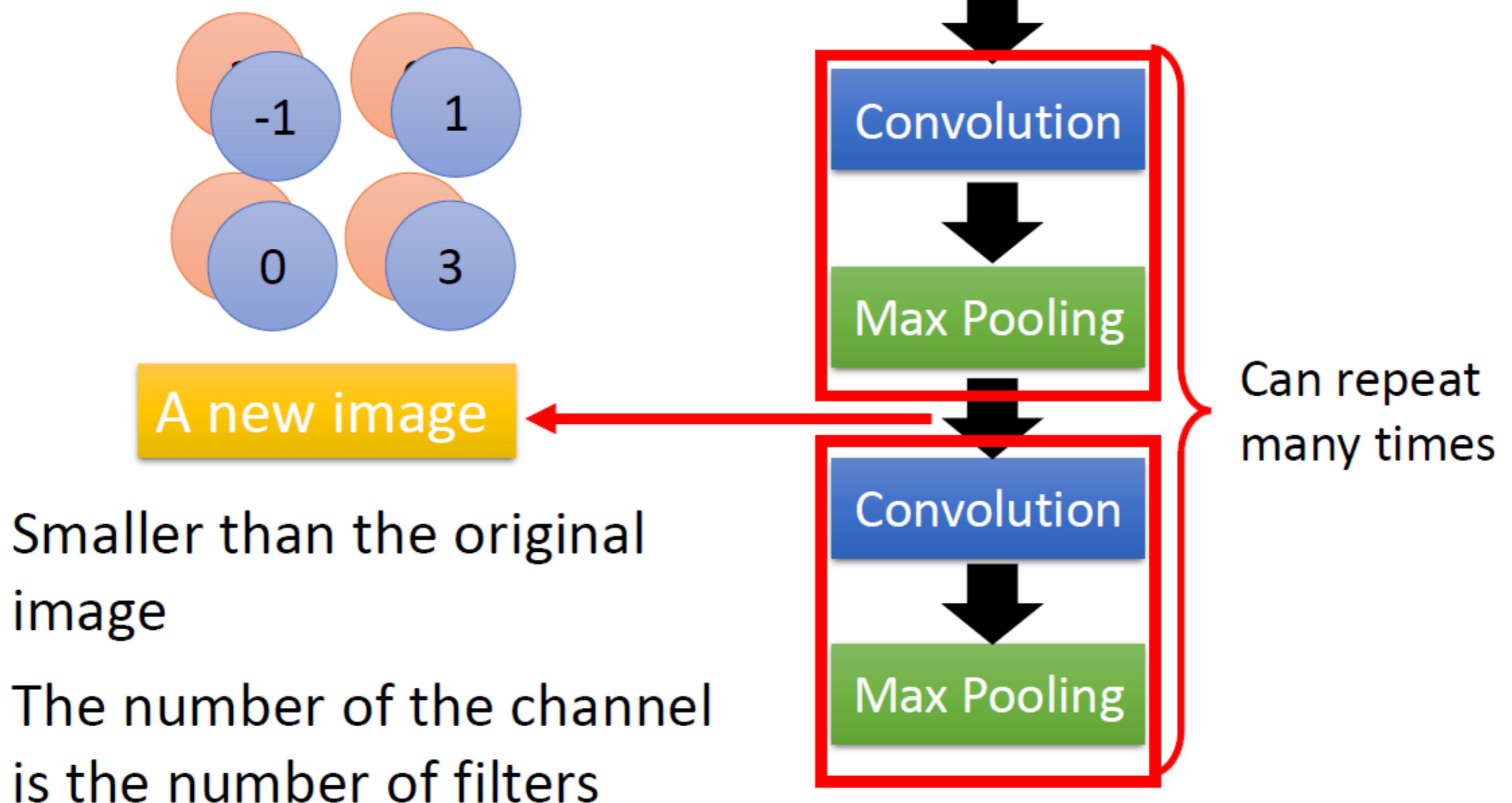
New image
but smaller



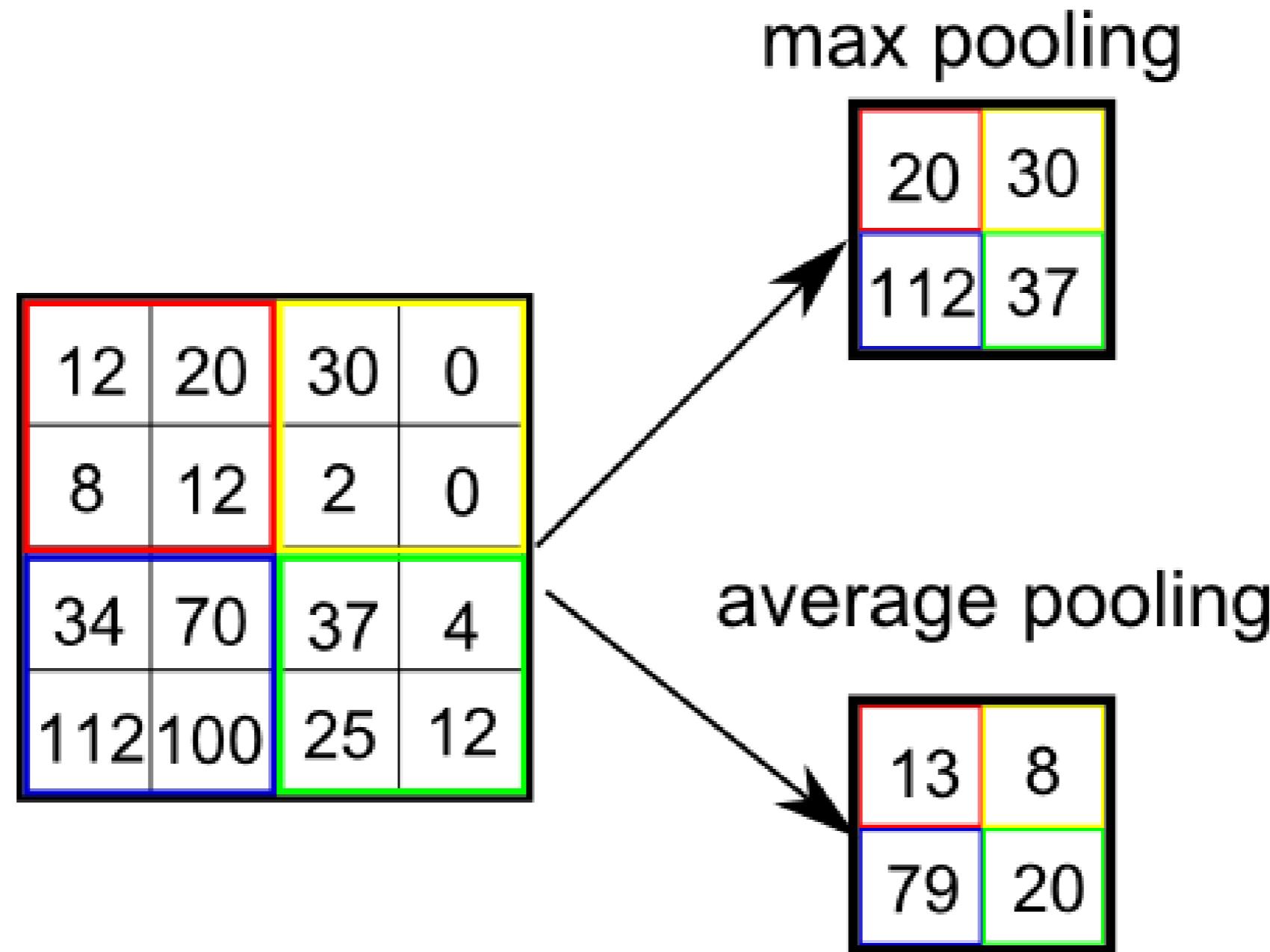
2 x 2 image

Each filter
is a channel

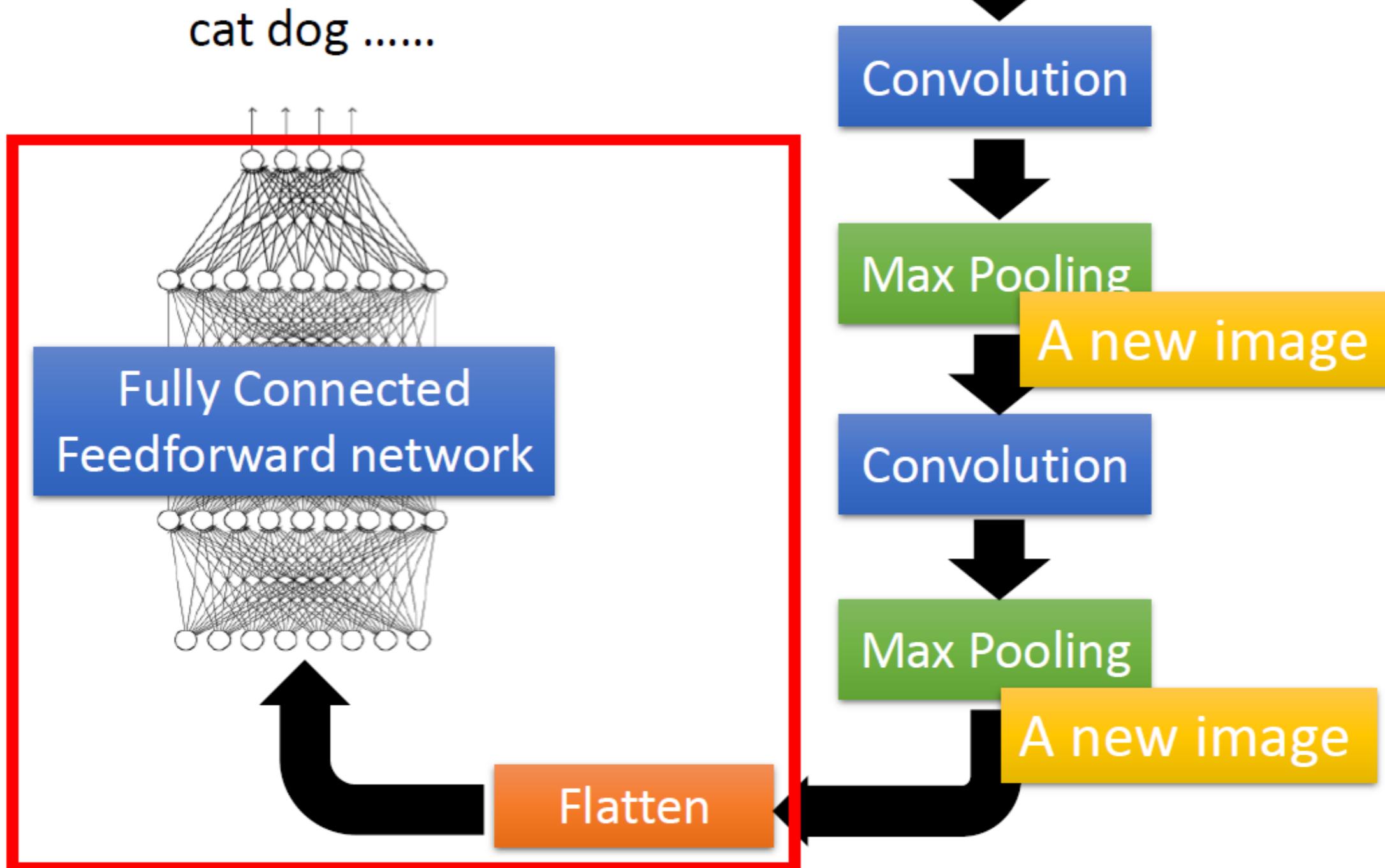
The whole CNN



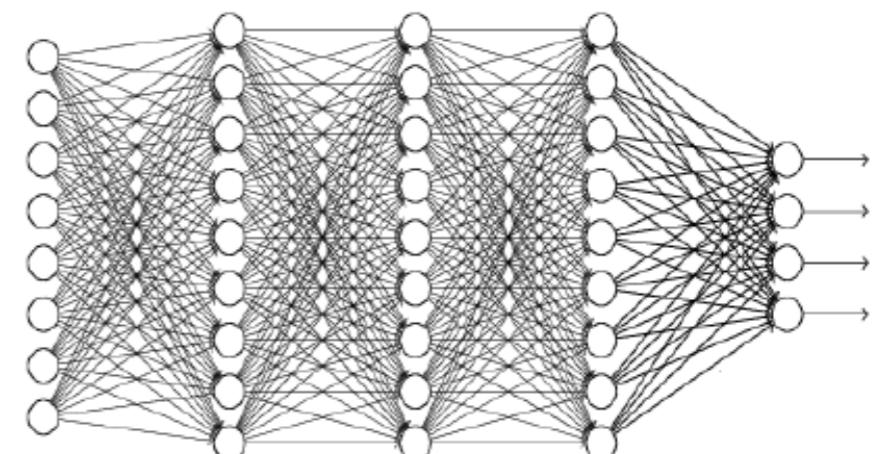
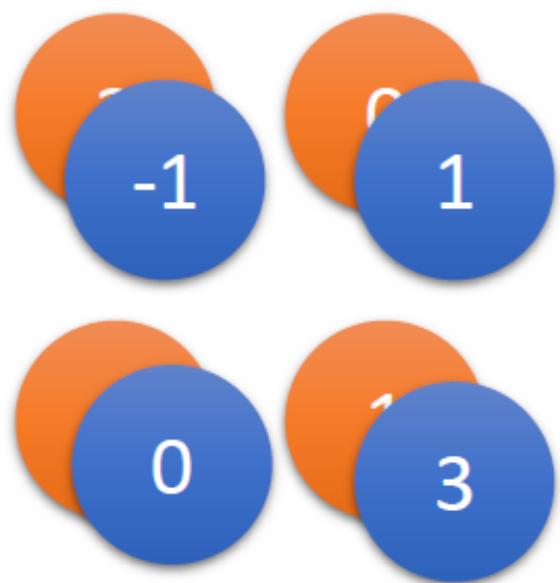
Pooling



The whole CNN



Flatten



Fully Connected
Feedforward network

Other Variants of Convolution: Dilated Convolution

- Introduces another parameter to convolutional layer called **dilation rate**

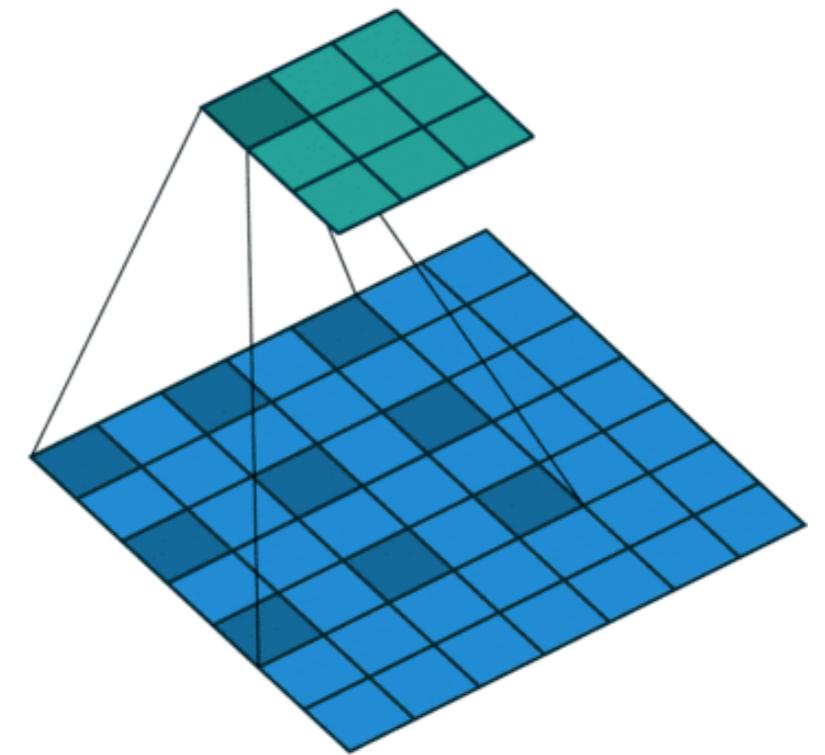


Image Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Dilated Convolution

- Introduces another parameter to convolutional layer called **dilation rate**
- Controls spacing between values in a kernel

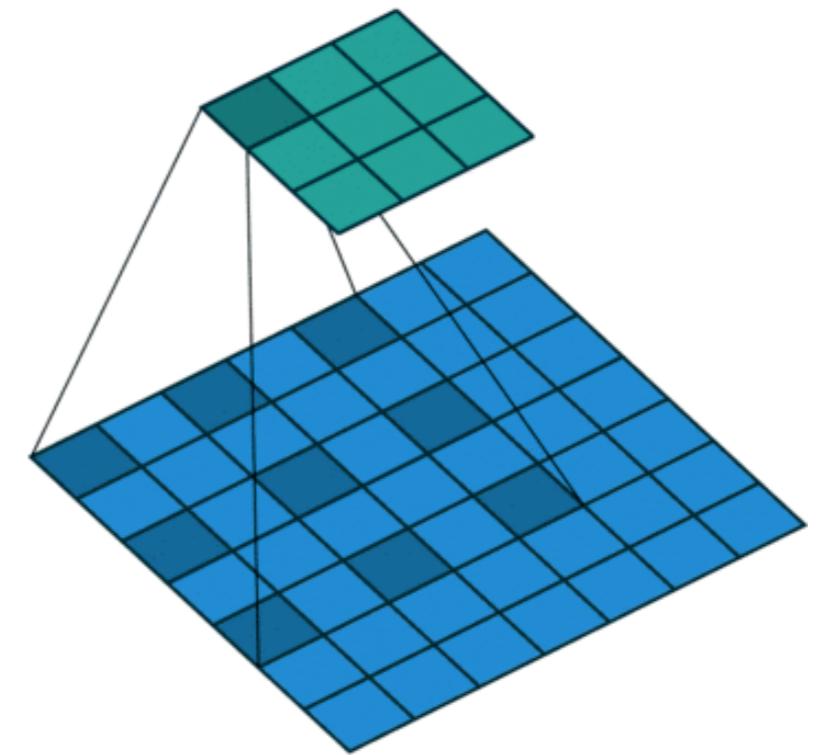


Image Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Dilated Convolution

- Introduces another parameter to convolutional layer called **dilation rate**
- Controls spacing between values in a kernel
- Figure shows 3×3 kernel with dilation rate 2

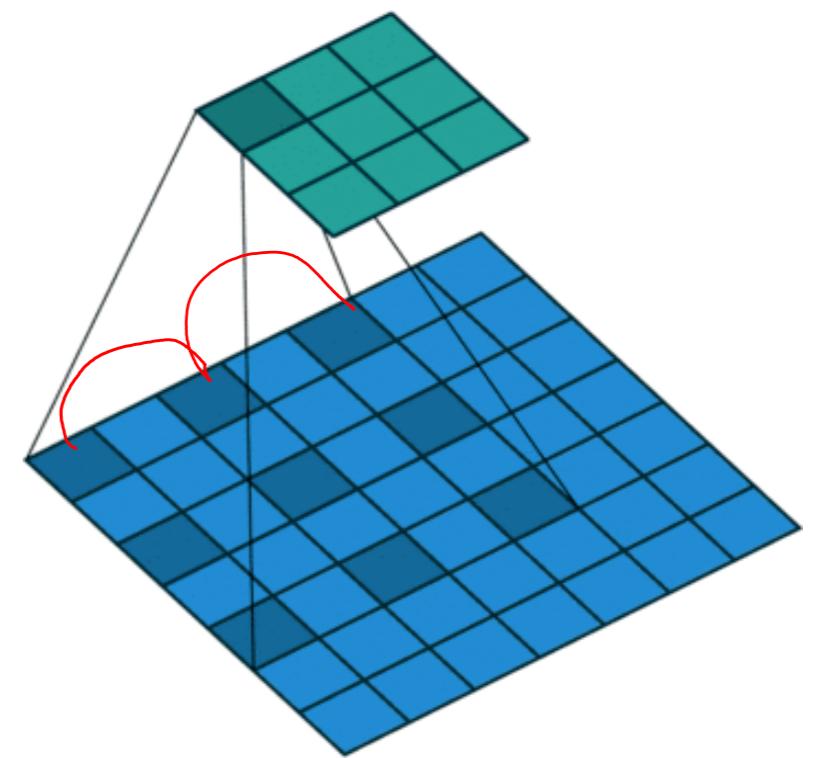


Image Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Dilated Convolution

- Introduces another parameter to convolutional layer called **dilation rate**
- Controls spacing between values in a kernel
- Figure shows 3×3 kernel with dilation rate 2

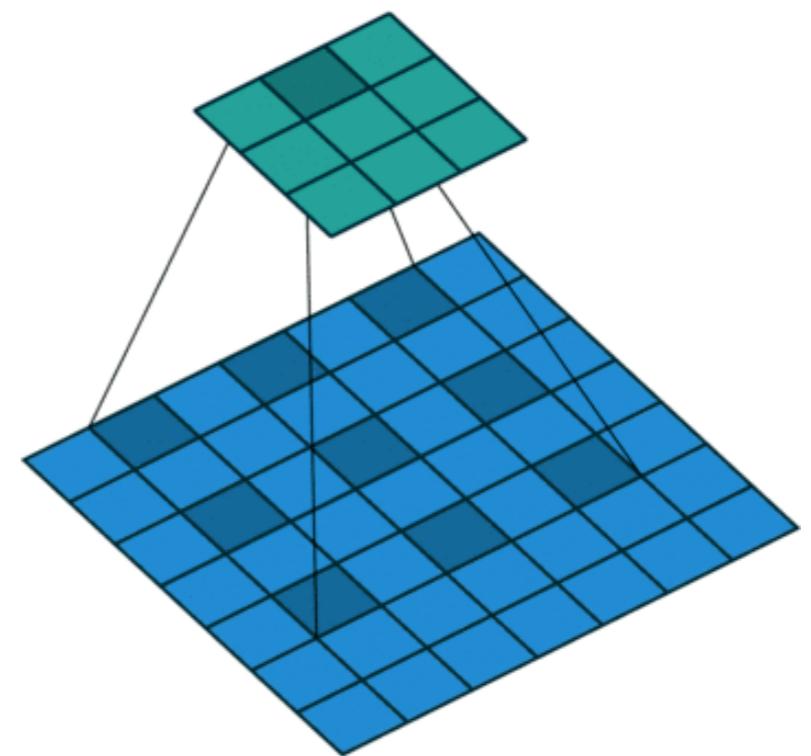


Image Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Dilated Convolution

- Introduces another parameter to convolutional layer called **dilation rate**
- Controls spacing between values in a kernel
- Figure shows 3×3 kernel with dilation rate 2

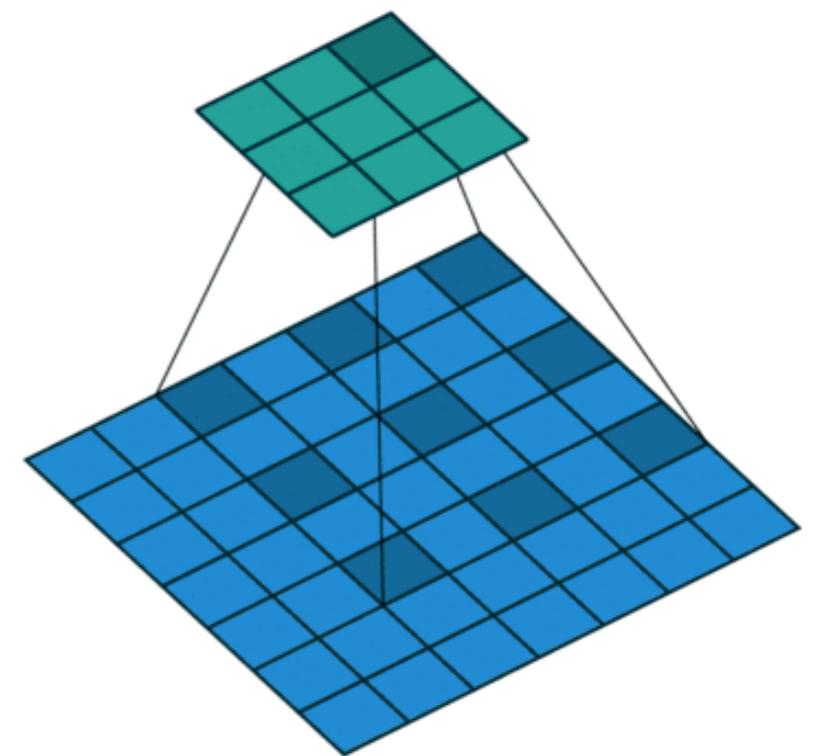


Image Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Dilated Convolution

- Introduces another parameter to convolutional layer called **dilation rate**
- Controls spacing between values in a kernel
- Figure shows 3×3 kernel with dilation rate 2

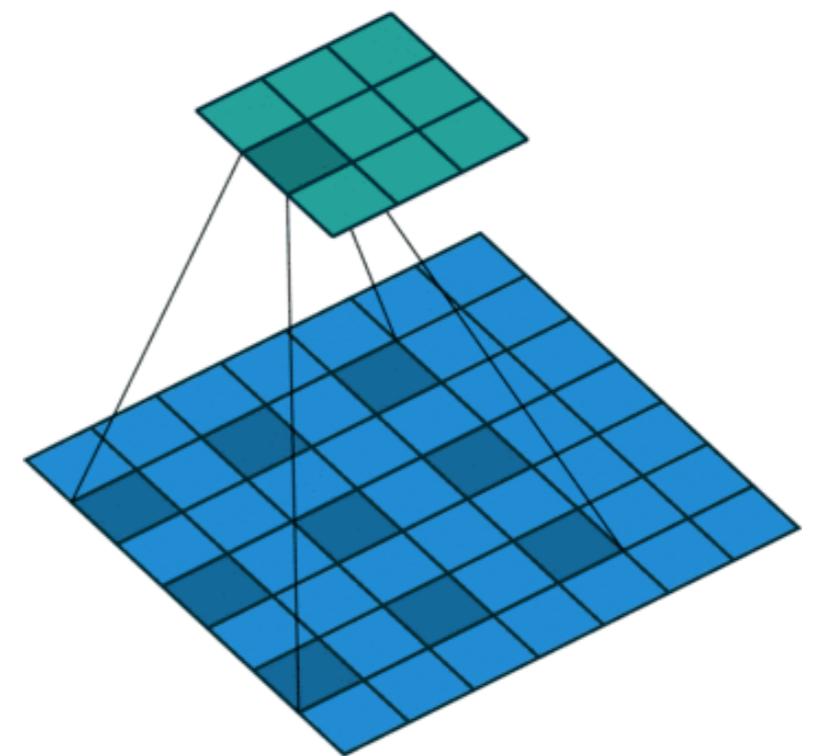


Image Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Dilated Convolution

- Introduces another parameter to convolutional layer called **dilation rate**
- Controls spacing between values in a kernel
- Figure shows 3×3 kernel with dilation rate 2

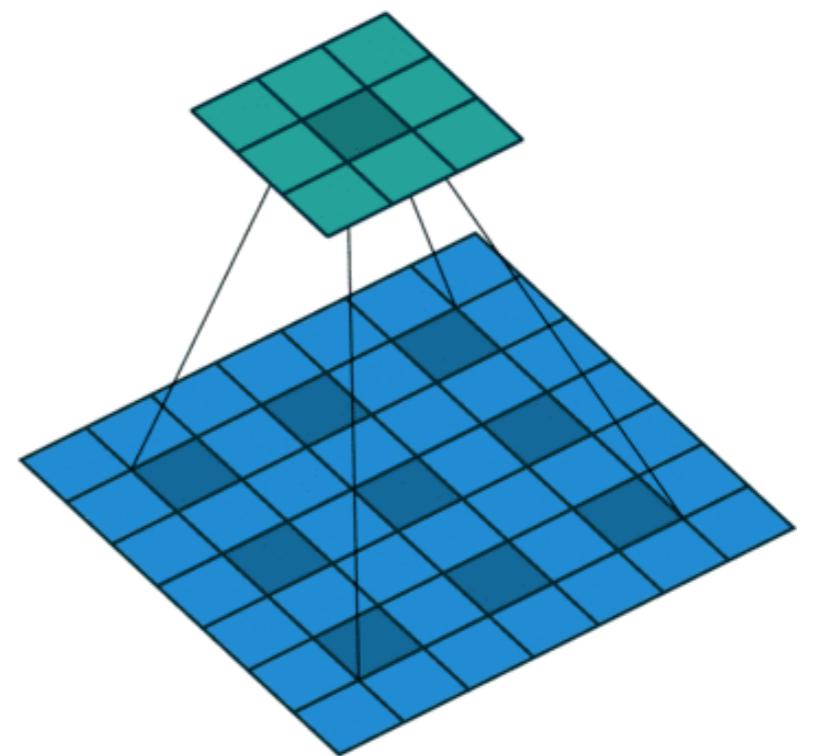


Image Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Dilated Convolution

- Introduces another parameter to convolutional layer called **dilation rate**
- Controls spacing between values in a kernel
- Figure shows 3×3 kernel with dilation rate 2

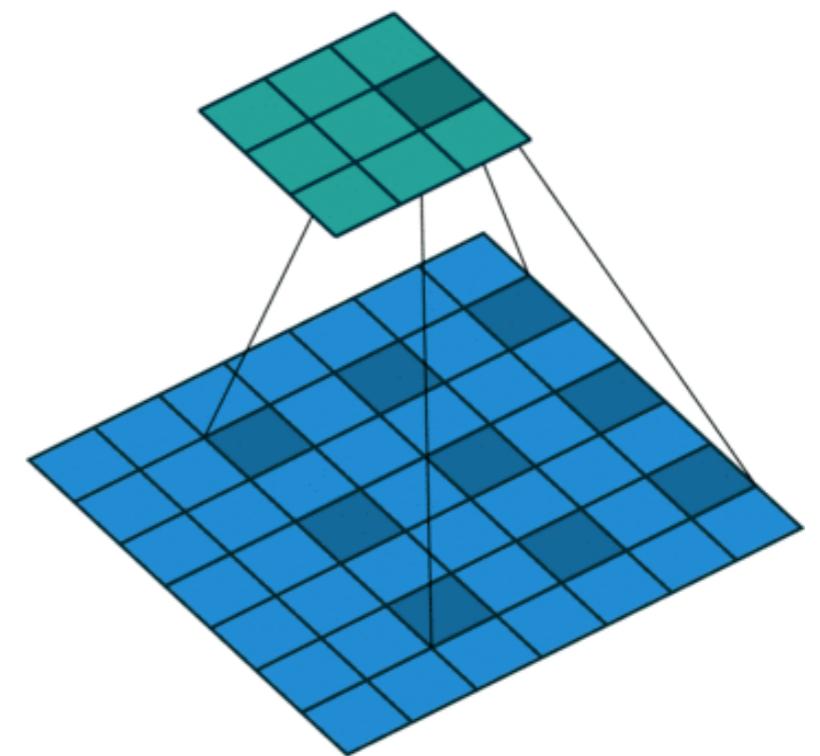


Image Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Dilated Convolution

- Introduces another parameter to convolutional layer called **dilation rate**
- Controls spacing between values in a kernel
- Figure shows 3×3 kernel with dilation rate 2

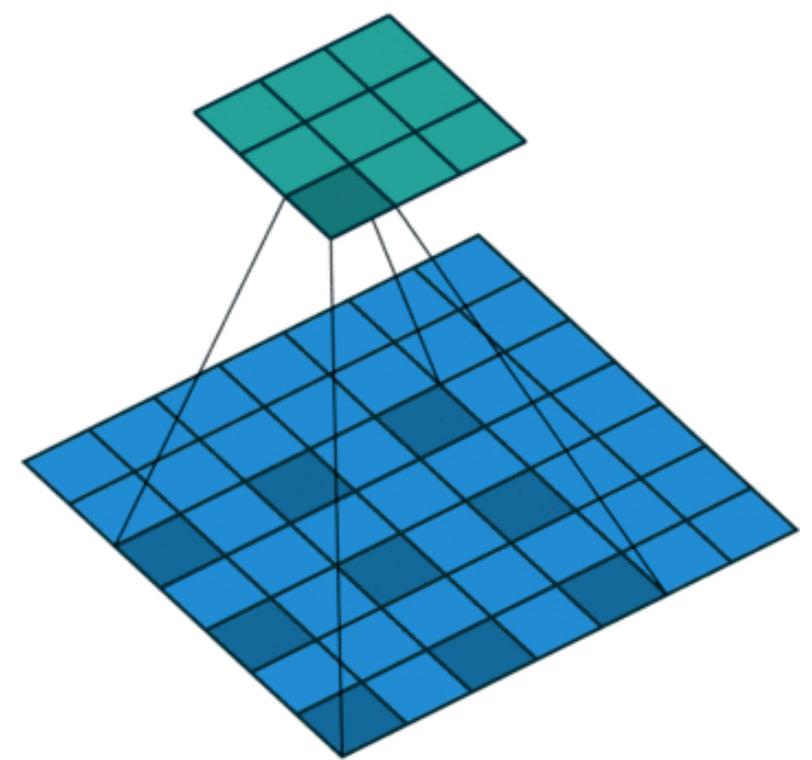


Image Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Dilated Convolution

- Introduces another parameter to convolutional layer called **dilation rate**
- Controls spacing between values in a kernel
- Figure shows 3×3 kernel with dilation rate 2

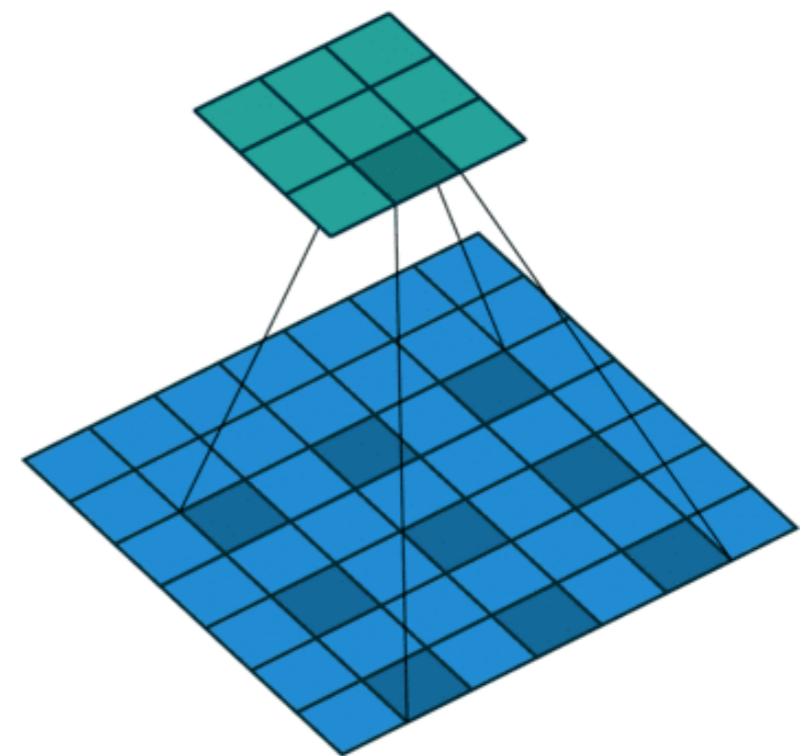


Image Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Dilated Convolution

- Introduces another parameter to convolutional layer called **dilation rate**
- Controls spacing between values in a kernel
- Figure shows 3×3 kernel with dilation rate 2

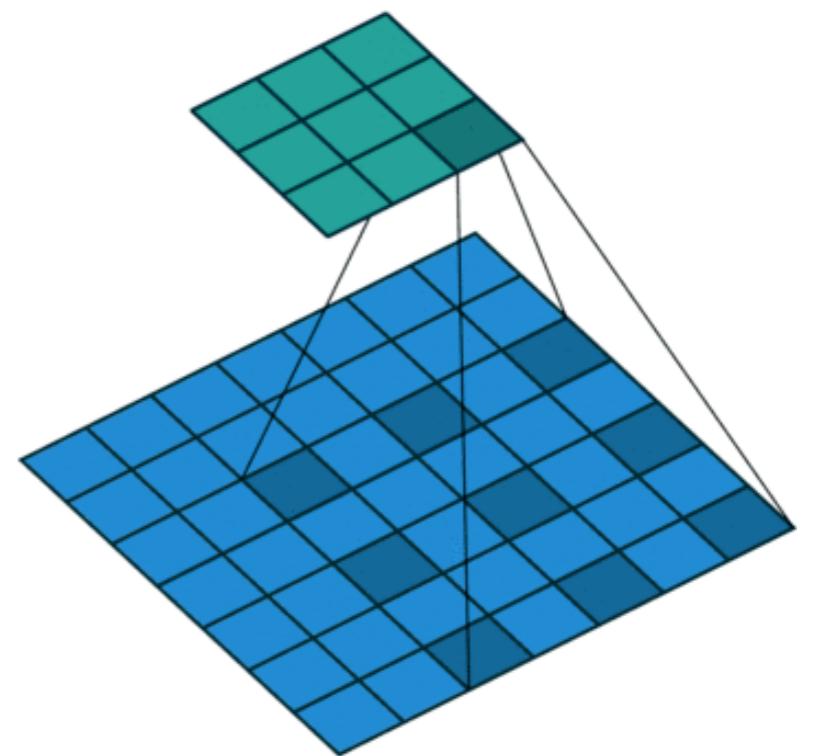


Image Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Dilated Convolution

- Introduces another parameter to convolutional layer called **dilation rate**
- Controls spacing between values in a kernel
- Figure shows 3×3 kernel with dilation rate 2
- Notice that dilated rate 1 is standard convolution

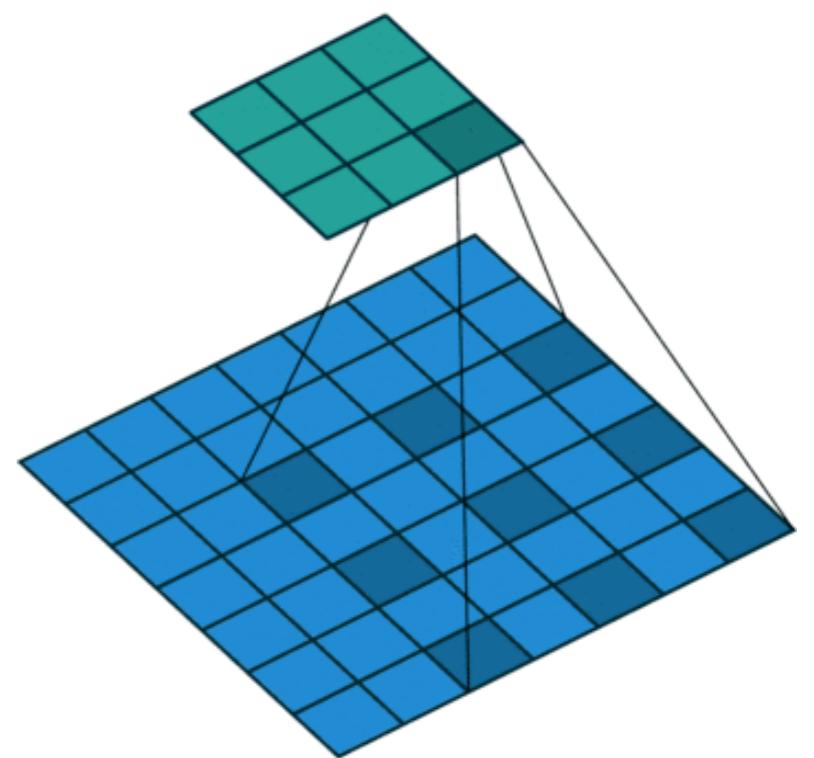


Image Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Dilated Convolution

- Introduces another parameter to convolutional layer called **dilation rate**
- Controls spacing between values in a kernel
- Figure shows 3×3 kernel with dilation rate 2
- Notice that dilated rate 1 is standard convolution
- A subtle difference between dilated convolution and standard convolution with stride > 1

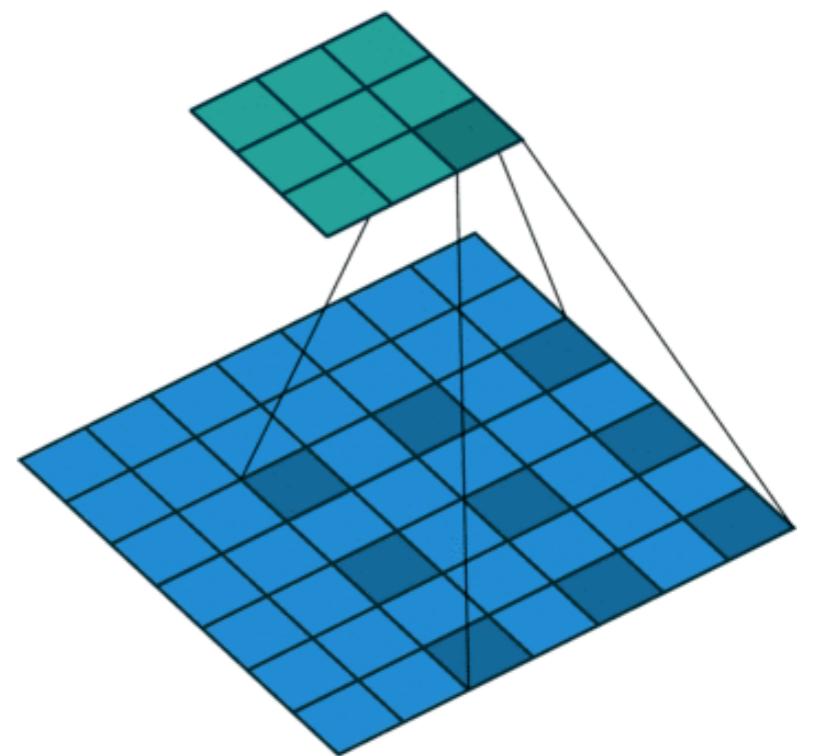


Image Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution

- Allows for learnable upsampling
- Also known as Deconvolution (bad) or Upconvolution

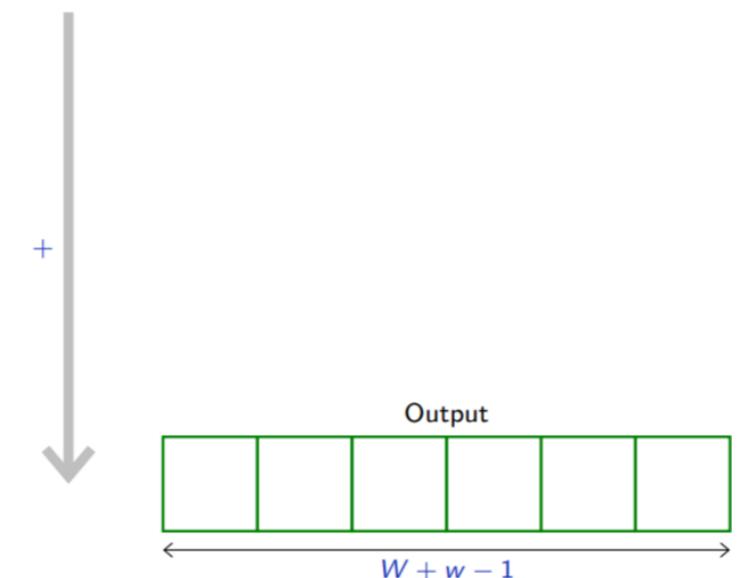
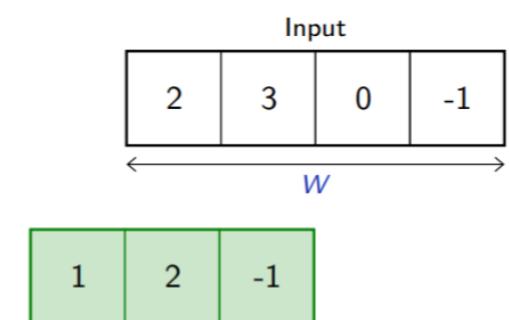
Other Variants of Convolution: Transpose Convolution

- Allows for learnable upsampling
- Also known as Deconvolution (bad) or Upconvolution
- Traditionally, we could achieve upsampling through interpolation or similar rules
- Why not allow the network to learn the rules by itself?

Other Variants of Convolution: Transpose Convolution

- Allows for learnable upsampling
- Also known as Deconvolution (bad) or Upconvolution
- Traditionally, we could achieve upsampling through interpolation or similar rules
- Why not allow the network to learn the rules by itself?
- Let us see a 1D example

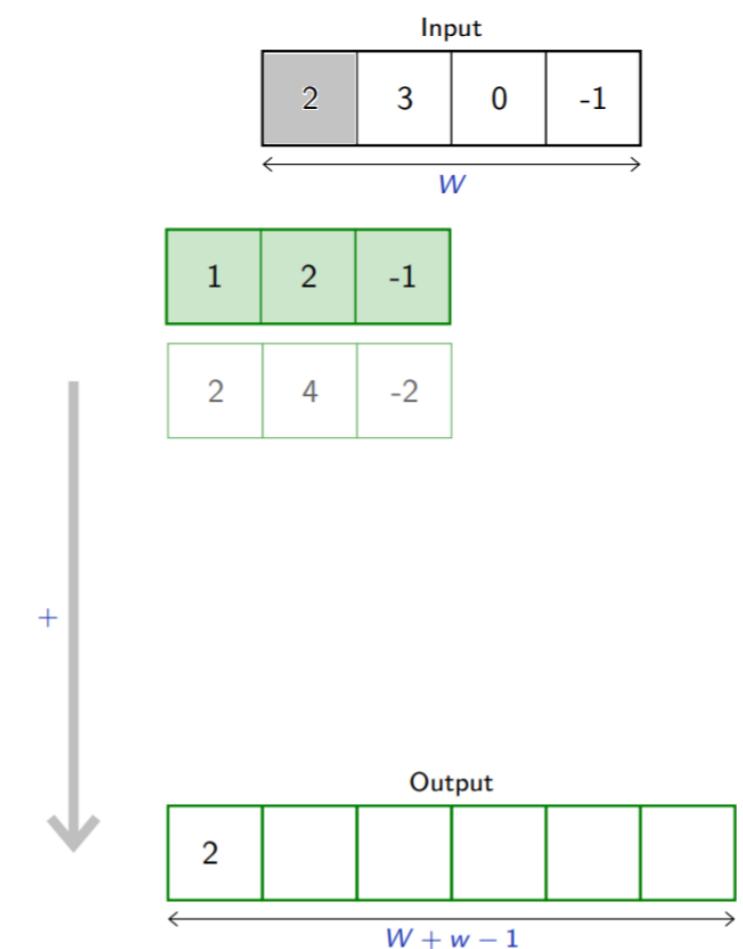
Transposed convolution layer



Other Variants of Convolution: Transpose Convolution

- Allows for learnable upsampling
- Also known as Deconvolution (bad) or Upconvolution
- Traditionally, we could achieve upsampling through interpolation or similar rules
- Why not allow the network to learn the rules by itself?
- Let us see a 1D example

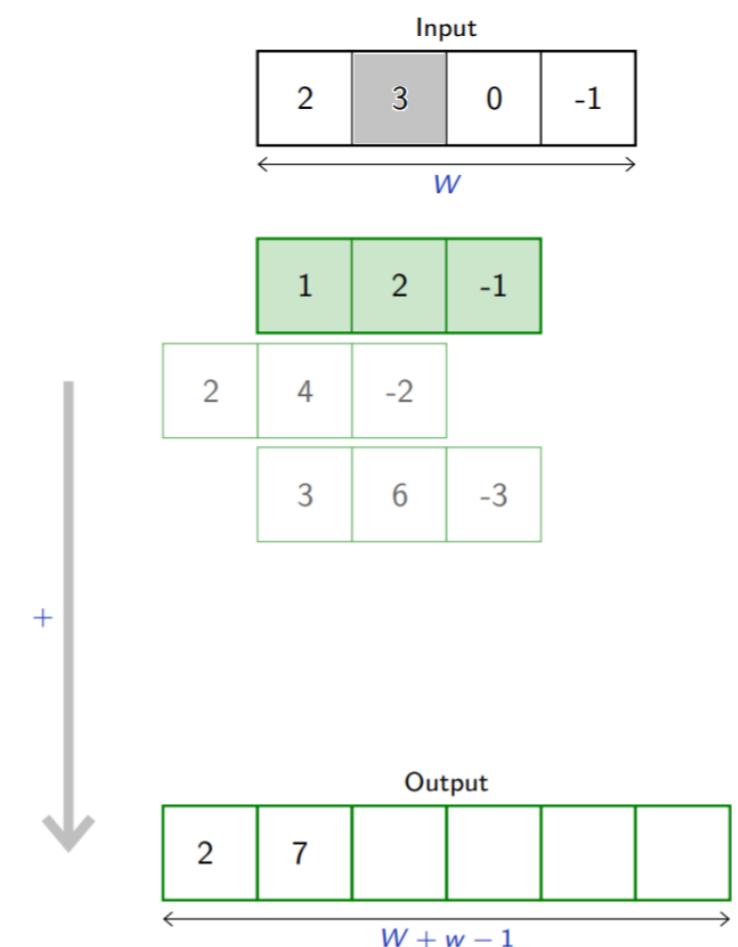
Transposed convolution layer



Other Variants of Convolution: Transpose Convolution

- Allows for learnable upsampling
- Also known as Deconvolution (bad) or Upconvolution
- Traditionally, we could achieve upsampling through interpolation or similar rules
- Why not allow the network to learn the rules by itself?
- Let us see a 1D example

Transposed convolution layer

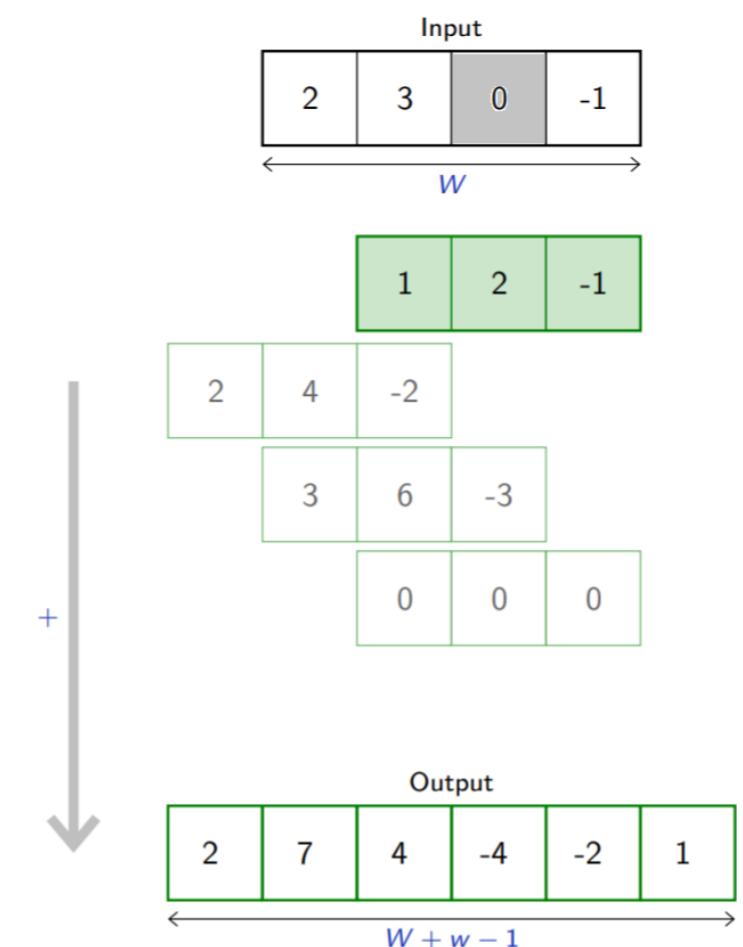


Credit: Francois Fleuret

Other Variants of Convolution: Transpose Convolution

- Allows for learnable upsampling
- Also known as Deconvolution (bad) or Upconvolution
- Traditionally, we could achieve upsampling through interpolation or similar rules
- Why not allow the network to learn the rules by itself?
- Let us see a 1D example

Transposed convolution layer

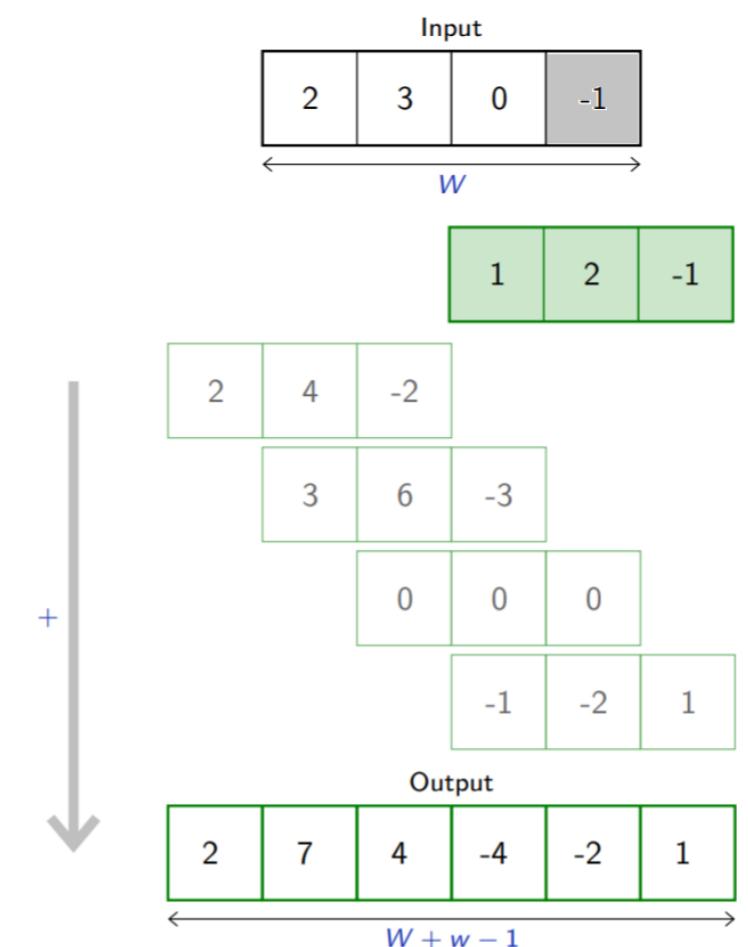


Credit: Francois Fleuret

Other Variants of Convolution: Transpose Convolution

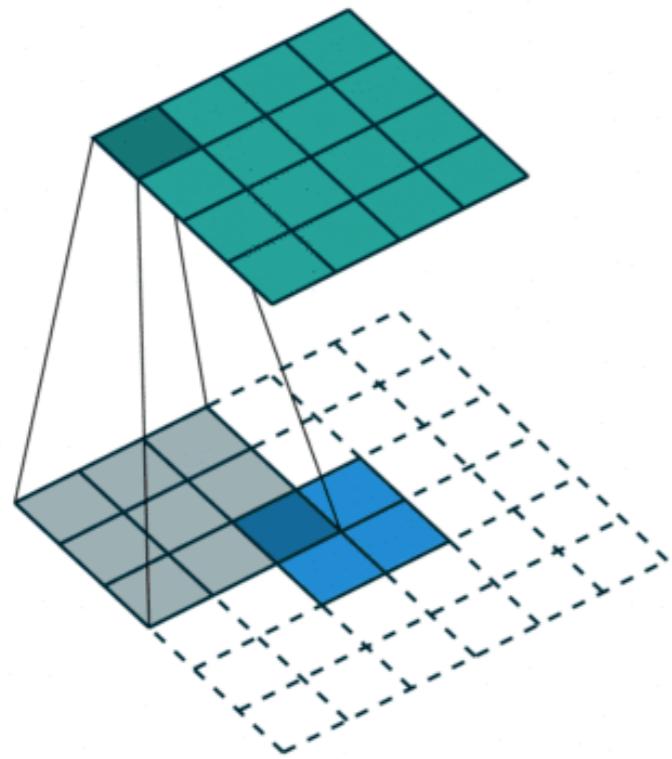
- Allows for learnable upsampling
- Also known as Deconvolution (bad) or Upconvolution
- Traditionally, we could achieve upsampling through interpolation or similar rules
- Why not allow the network to learn the rules by itself?
- Let us see a 1D example

Transposed convolution layer

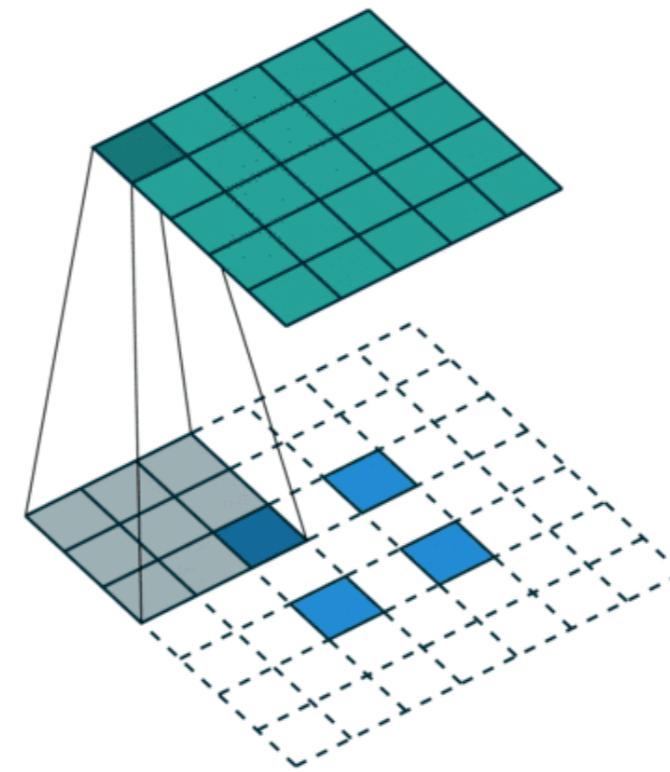


Credit: Francois Fleuret

Other Variants of Convolution: Transpose Convolution



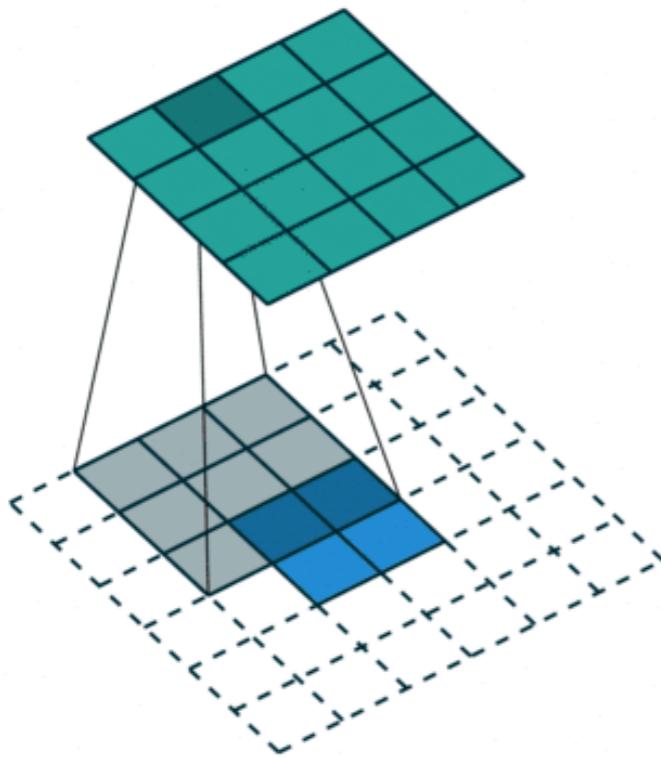
Upsampling 2×2 input to a 4×4 output



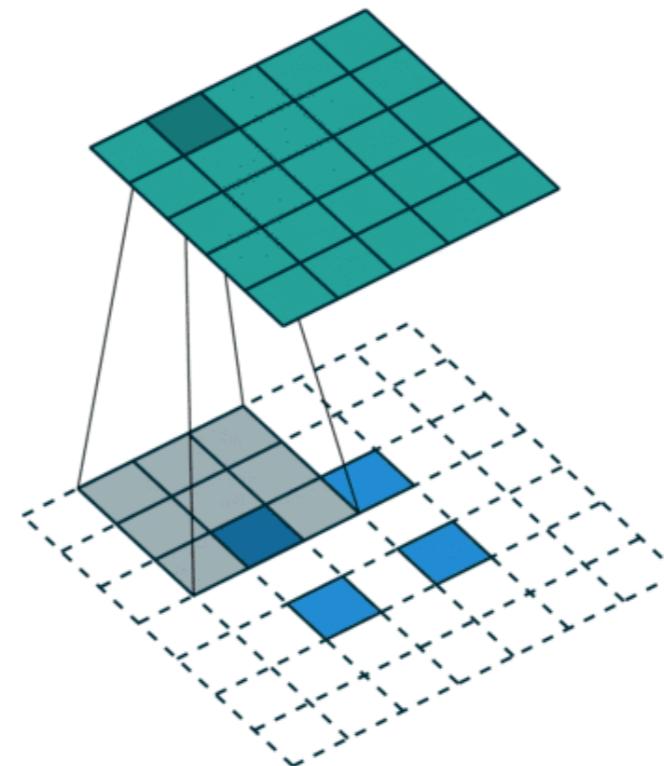
Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution



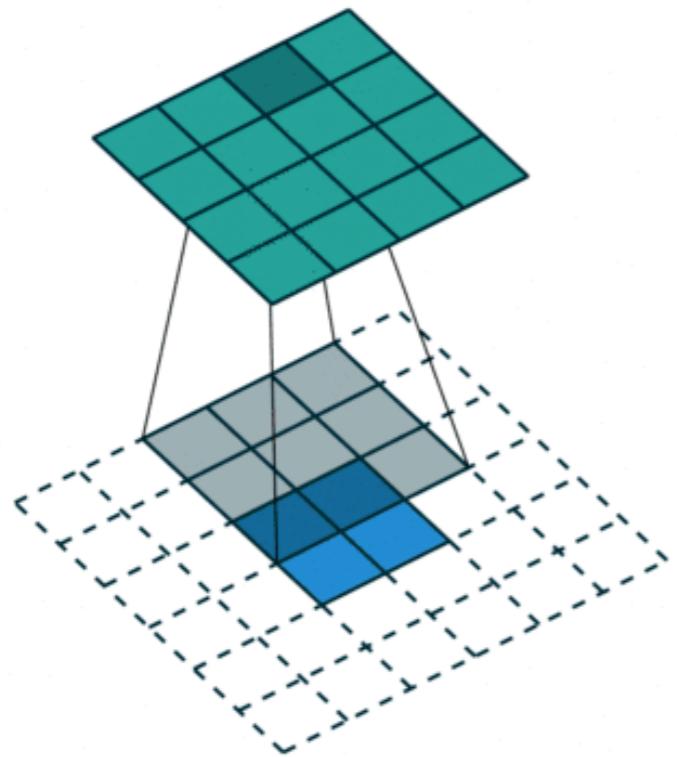
Upsampling 2×2 input to a 4×4 output



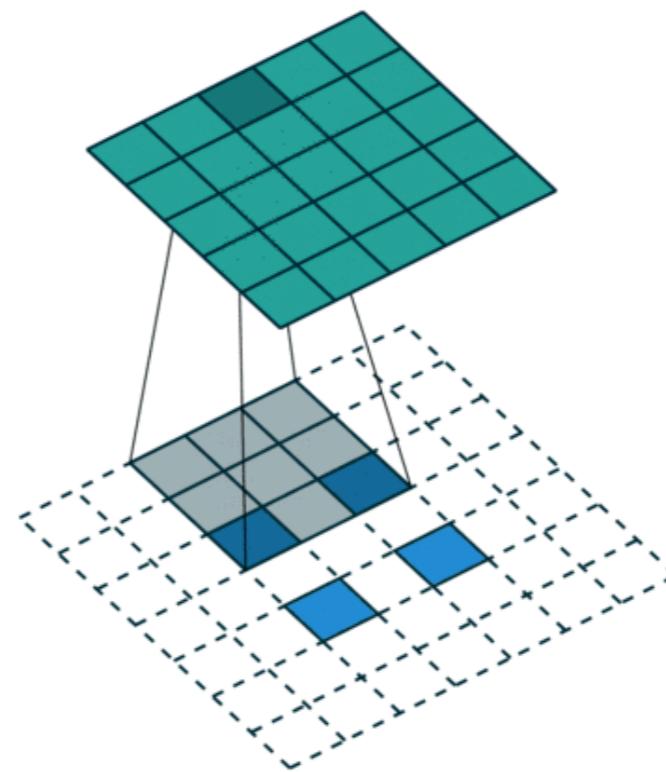
Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution



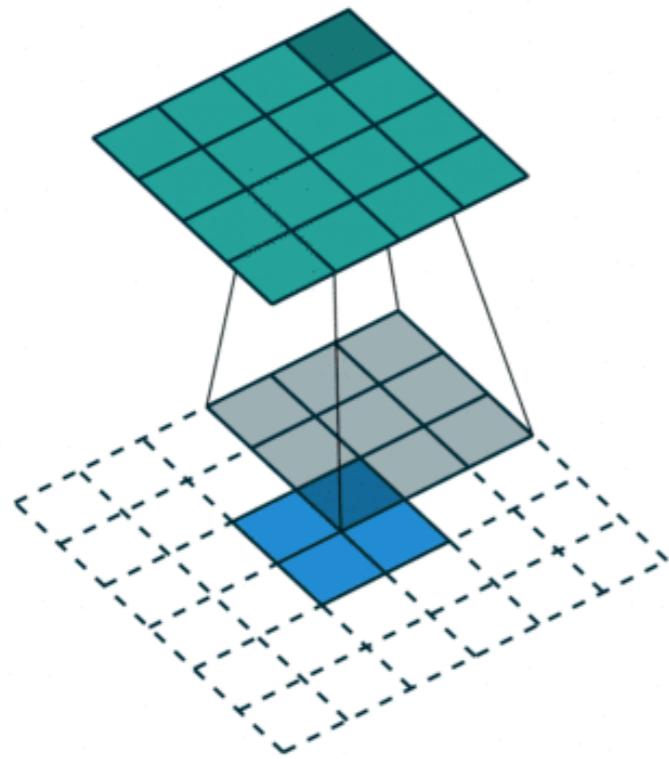
Upsampling 2×2 input to a 4×4 output



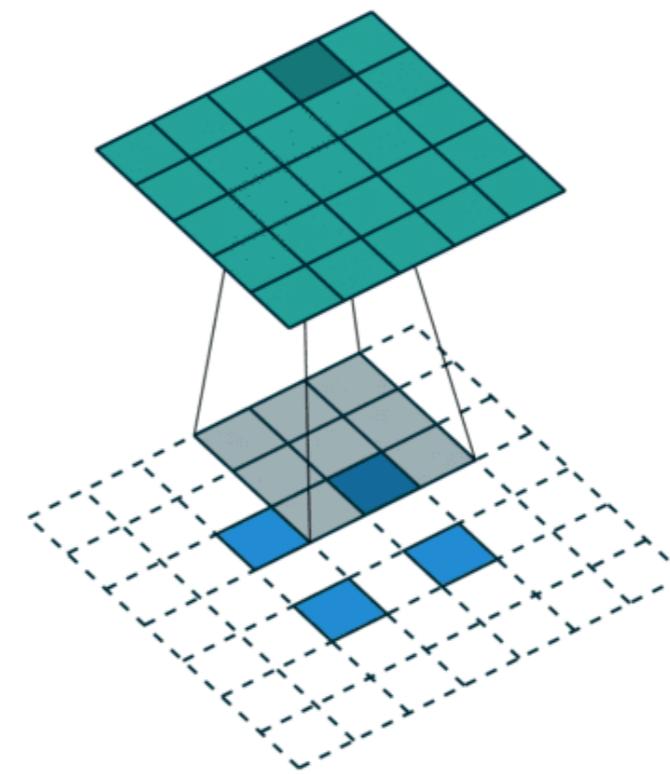
Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution



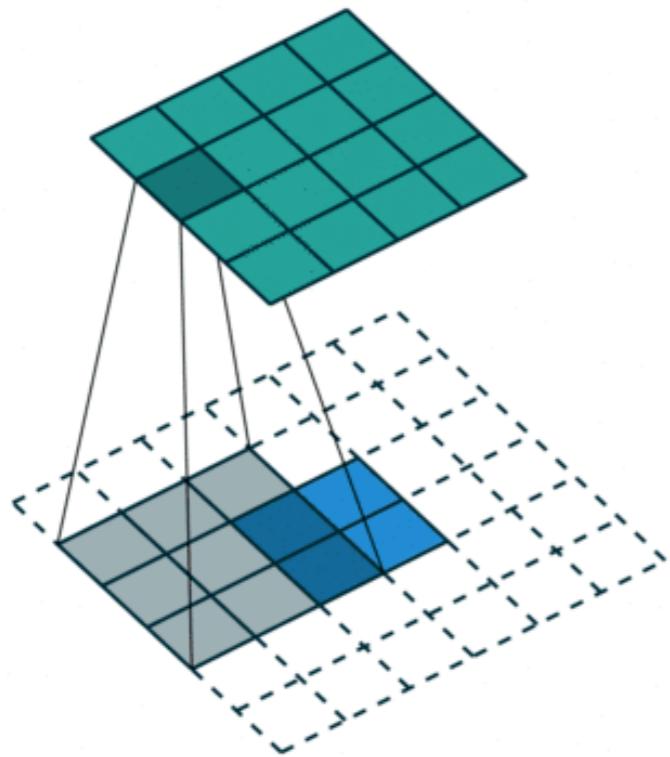
Upsampling 2×2 input to a 4×4 output



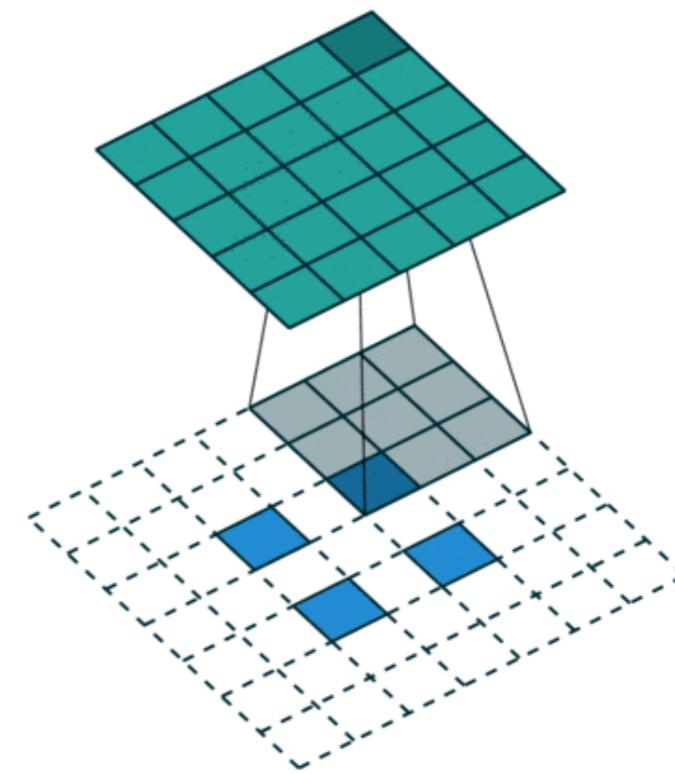
Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution



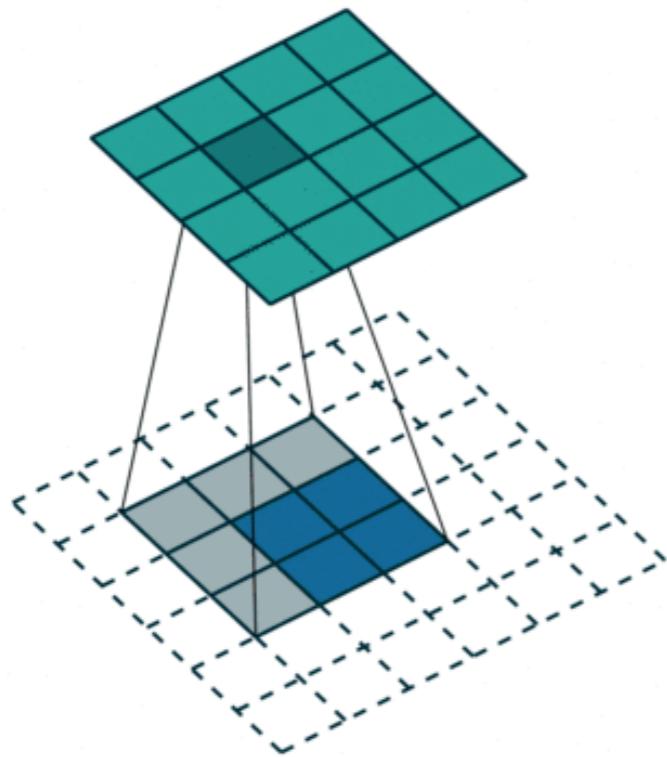
Upsampling 2×2 input to a 4×4 output



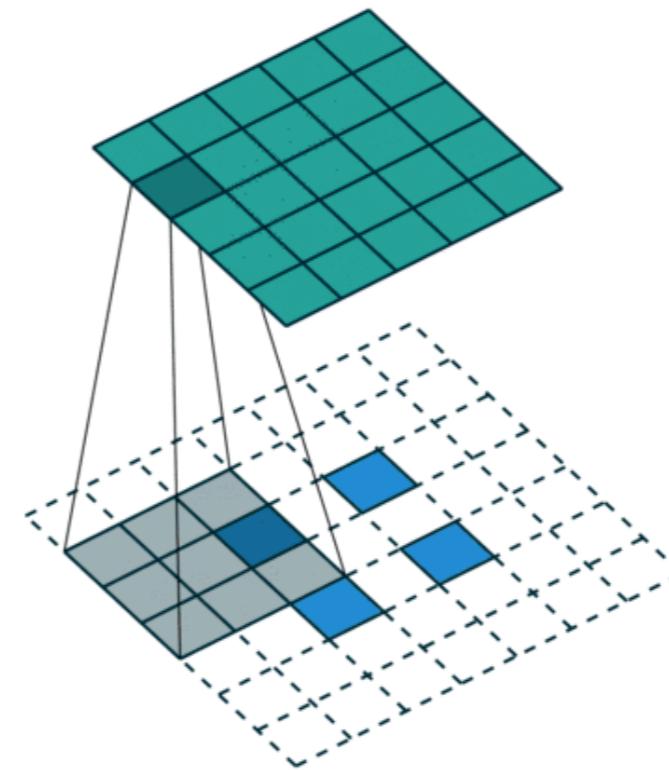
Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution



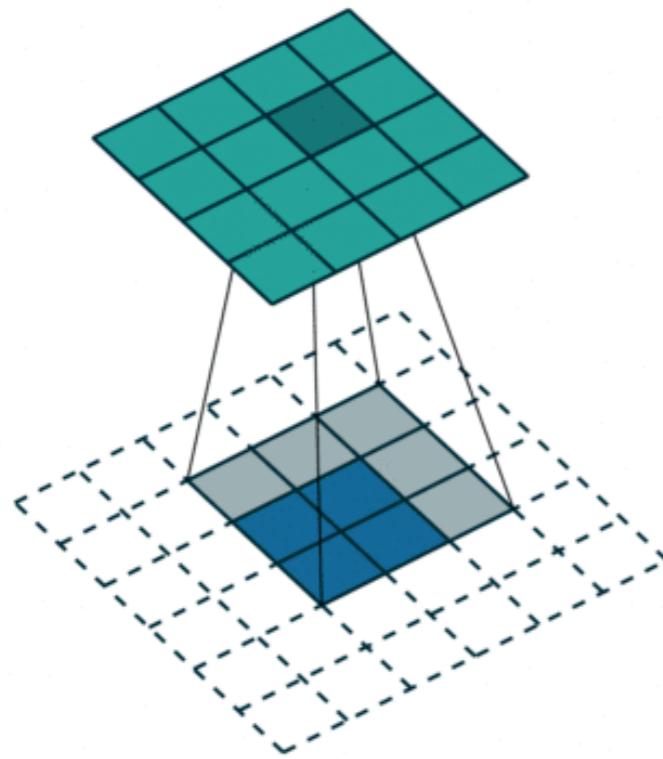
Upsampling 2×2 input to a 4×4 output



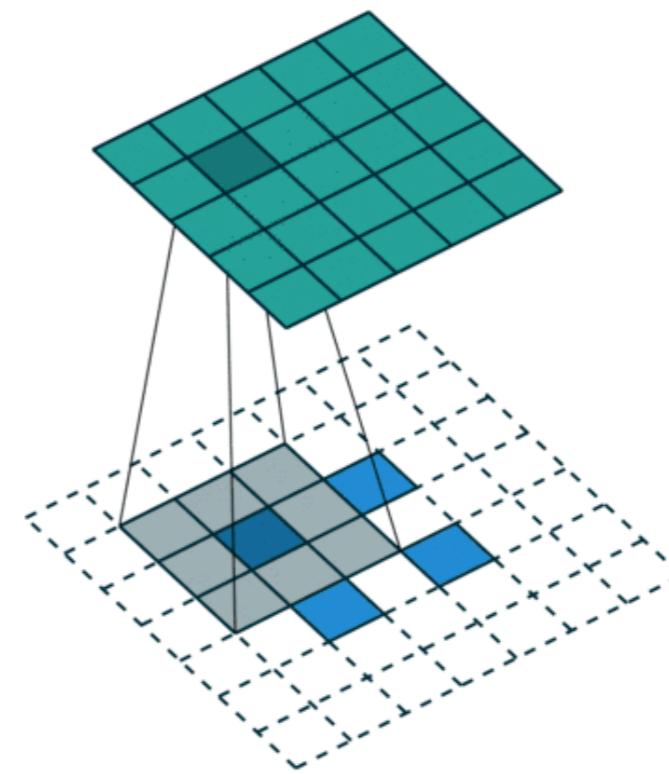
Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution



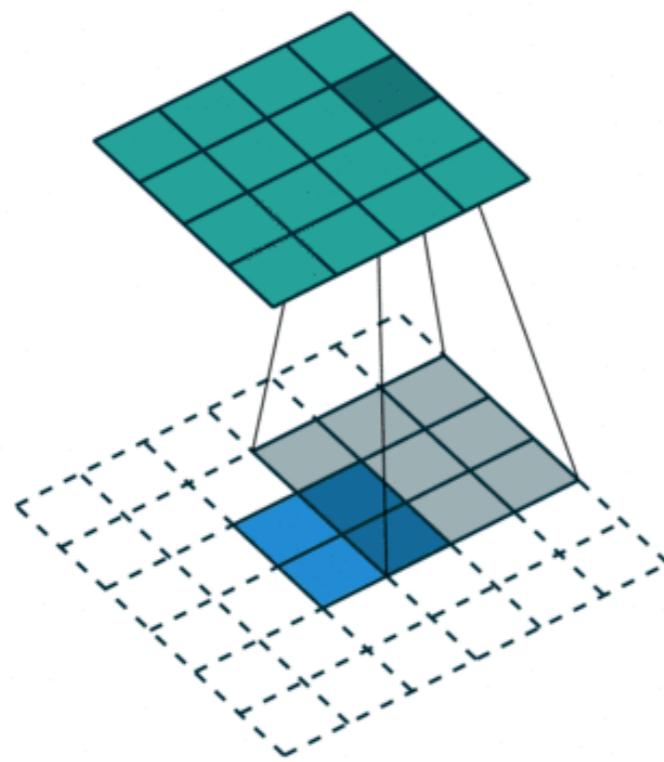
Upsampling 2×2 input to a 4×4 output



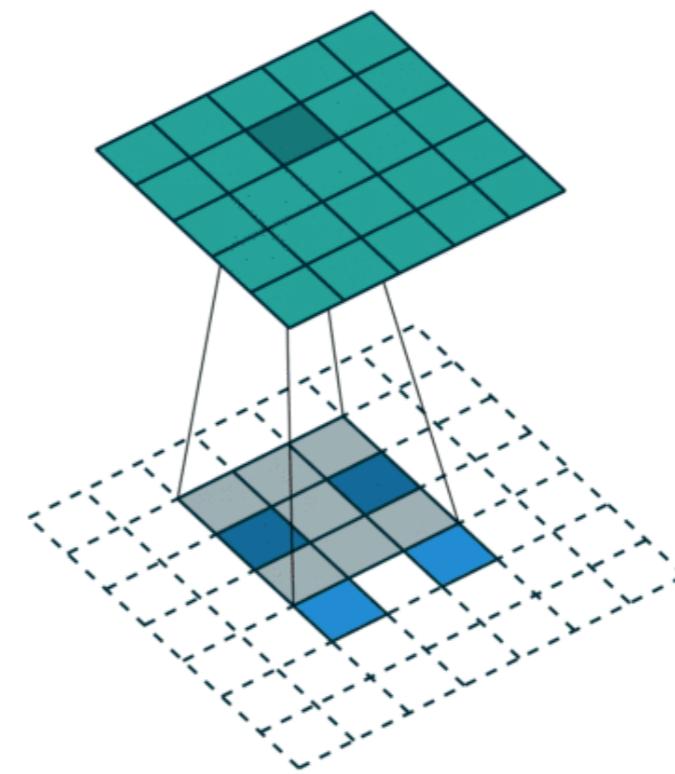
Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution



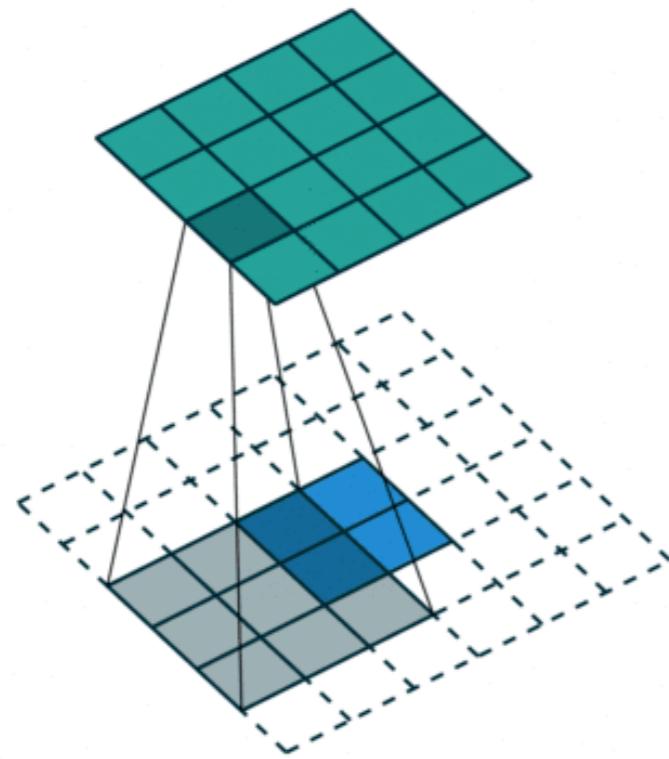
Upsampling 2×2 input to a 4×4 output



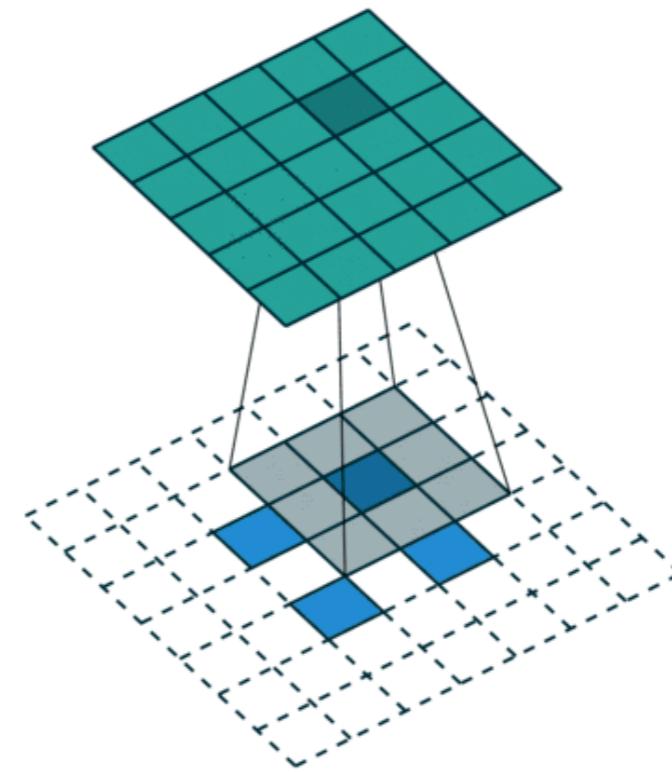
Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution



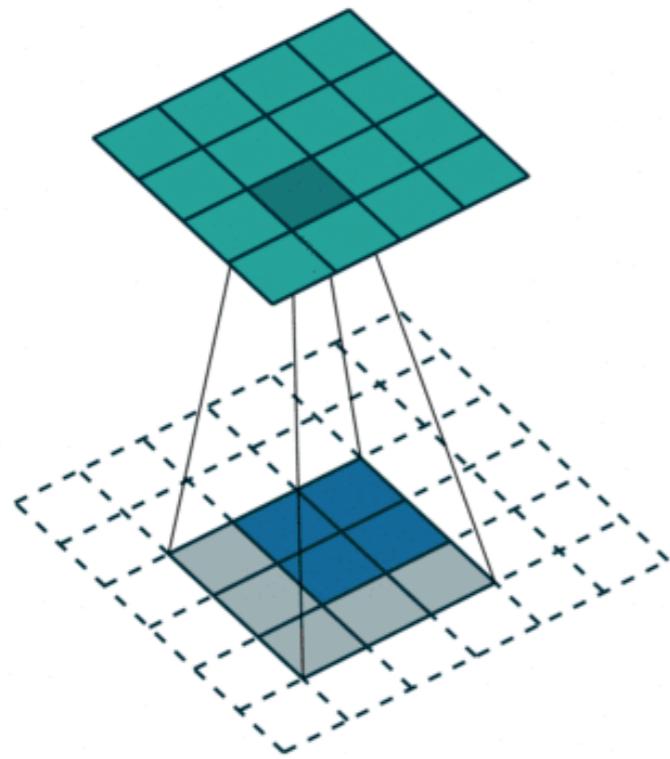
Upsampling 2×2 input to a 4×4 output



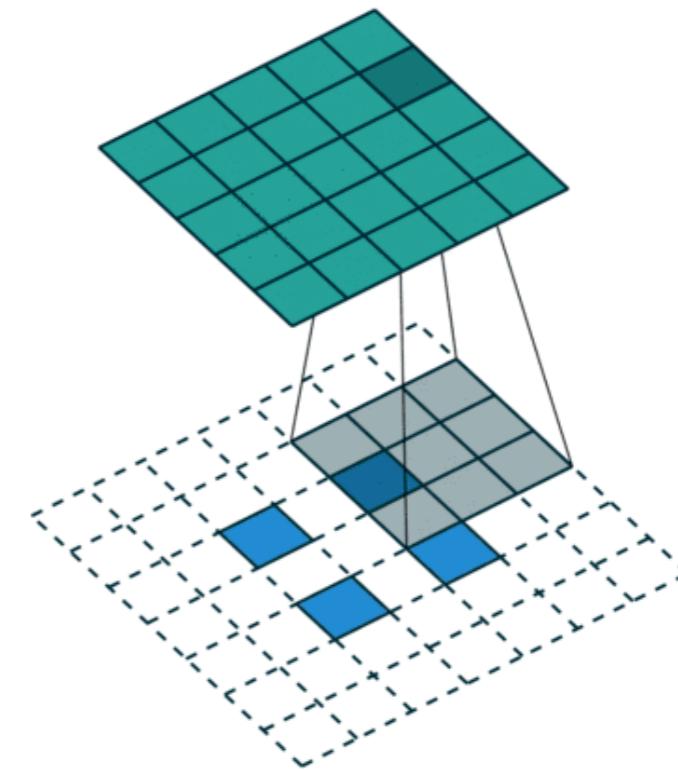
Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution



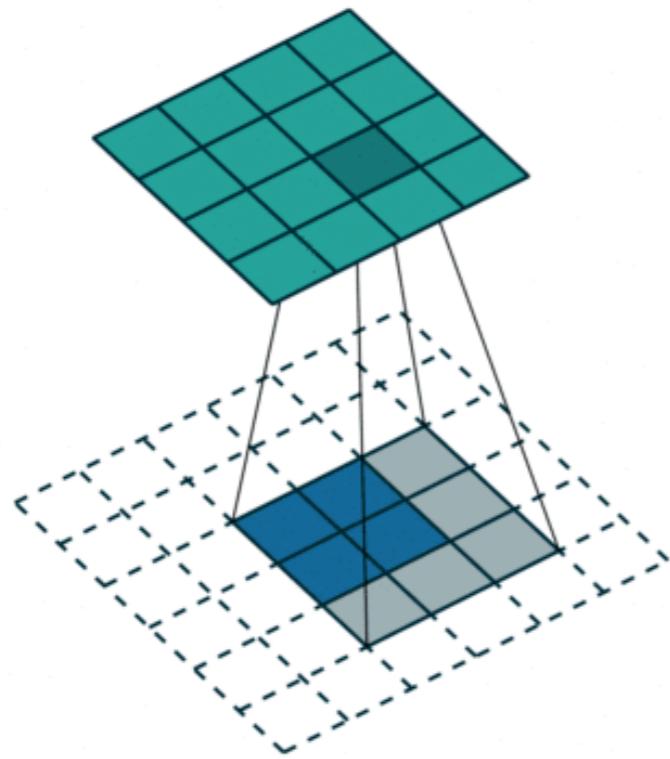
Upsampling 2×2 input to a 4×4 output



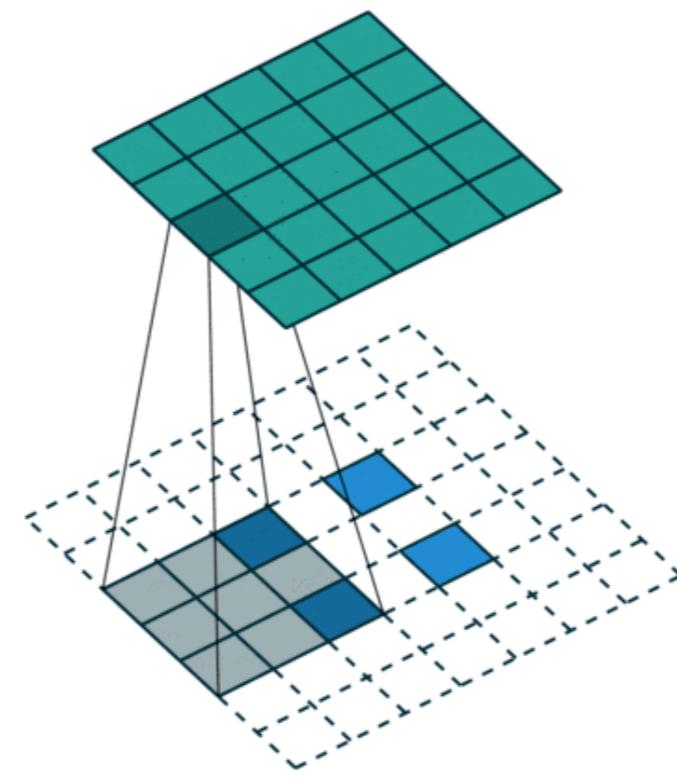
Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution



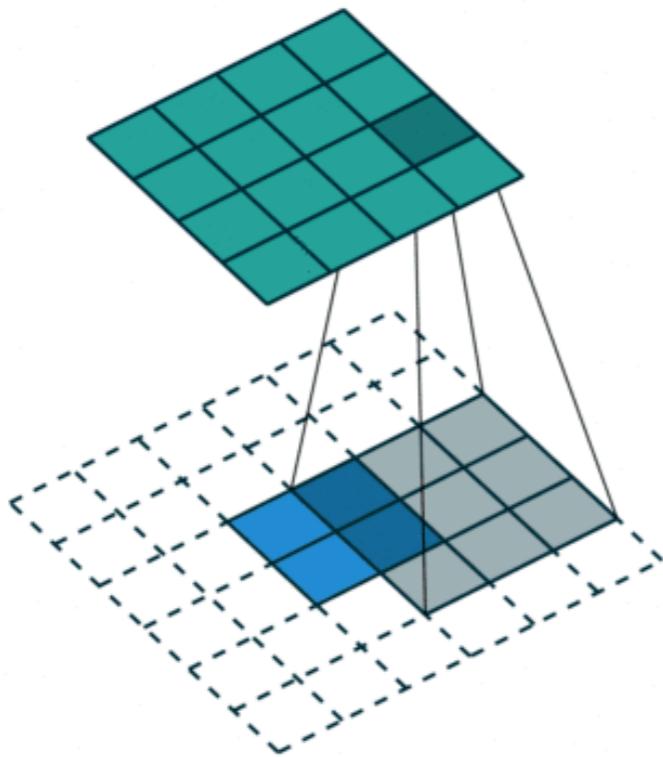
Upsampling 2×2 input to a 4×4 output



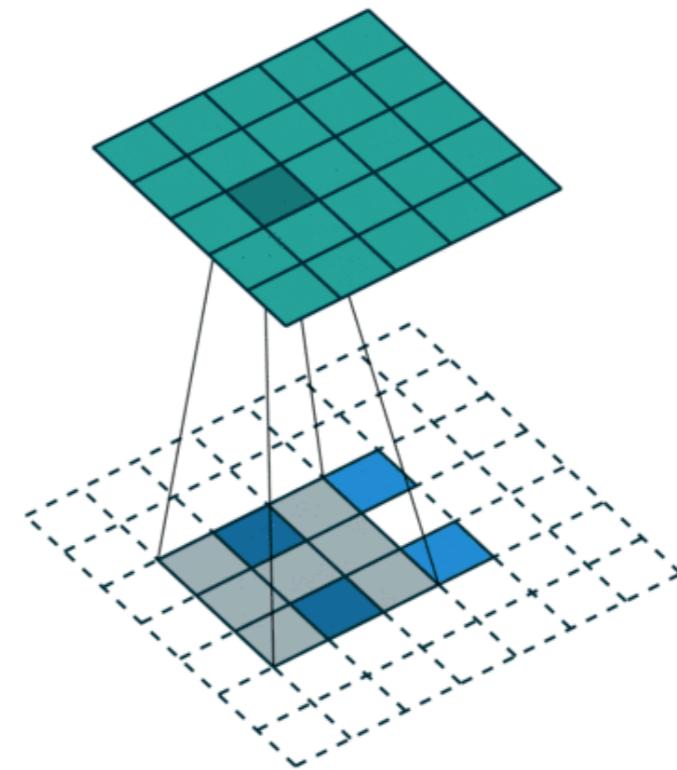
Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution



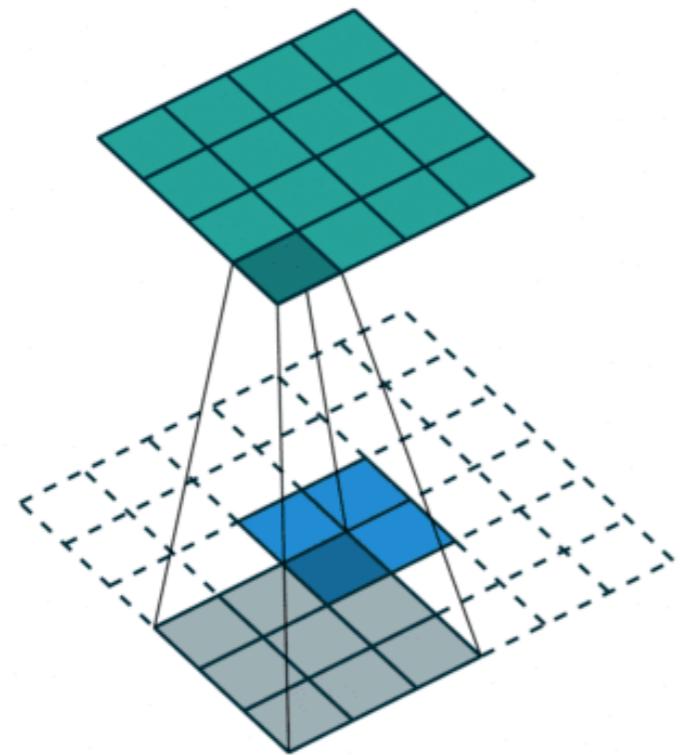
Upsampling 2×2 input to a 4×4 output



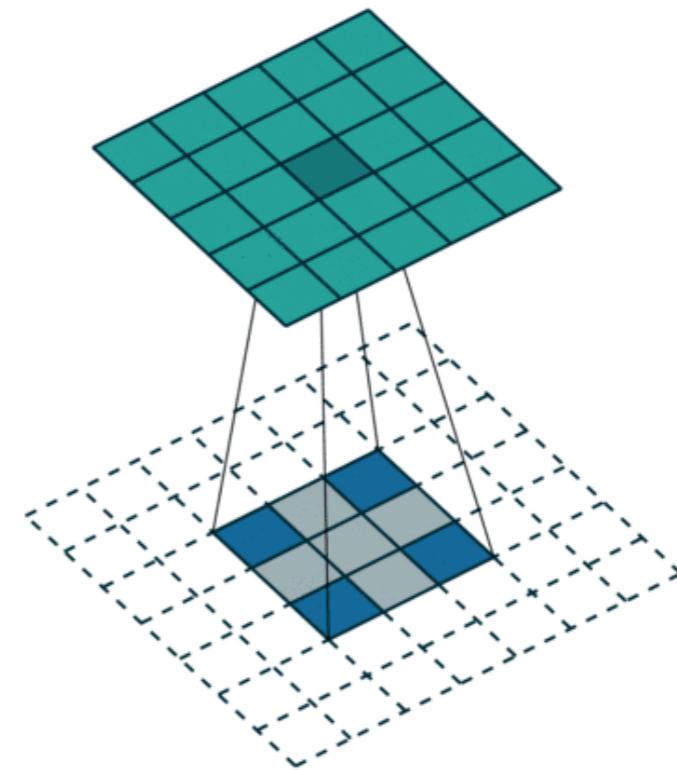
Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution



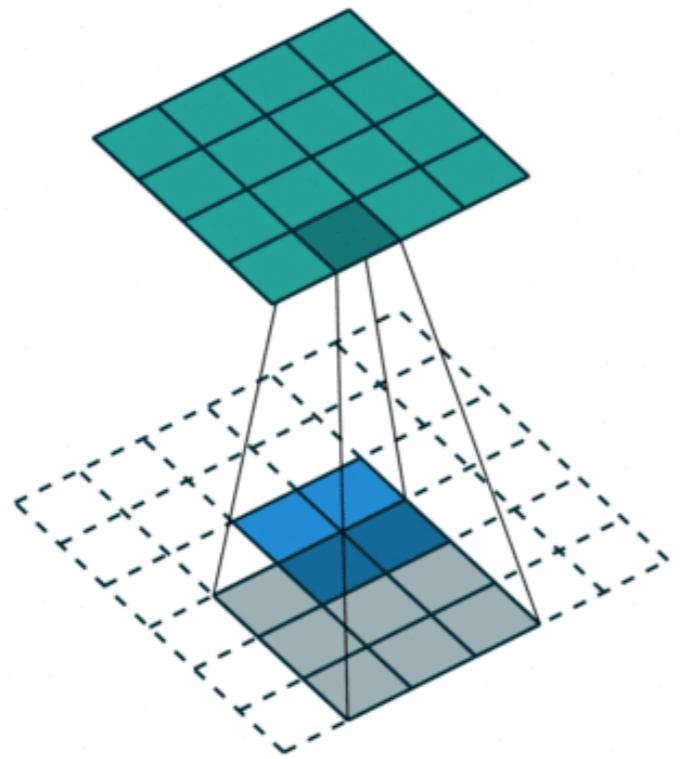
Upsampling 2×2 input to a 4×4 output



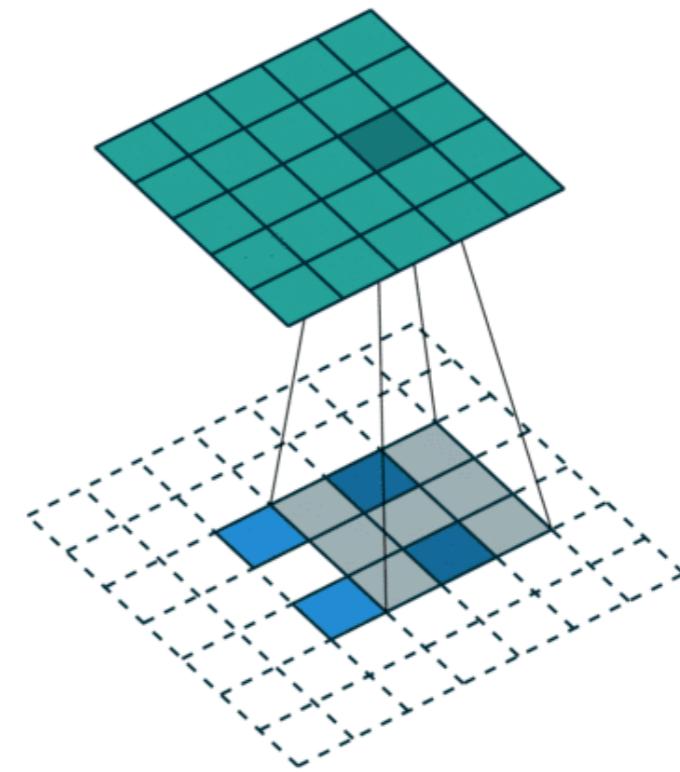
Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution



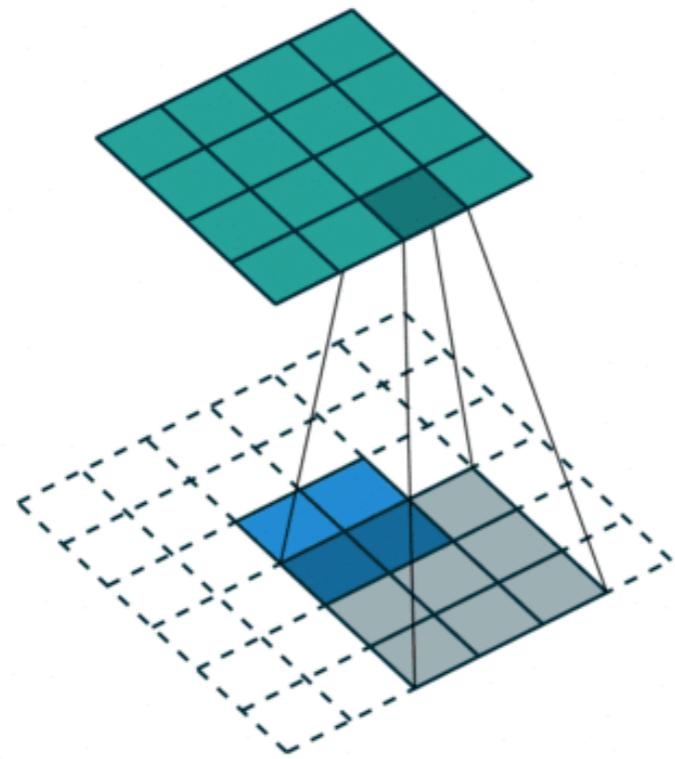
Upsampling 2×2 input to a 4×4 output



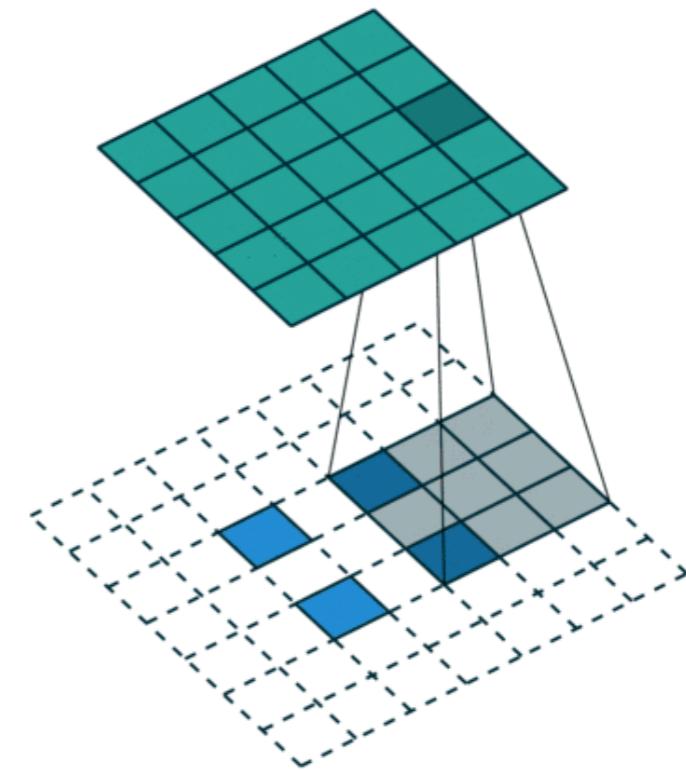
Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution



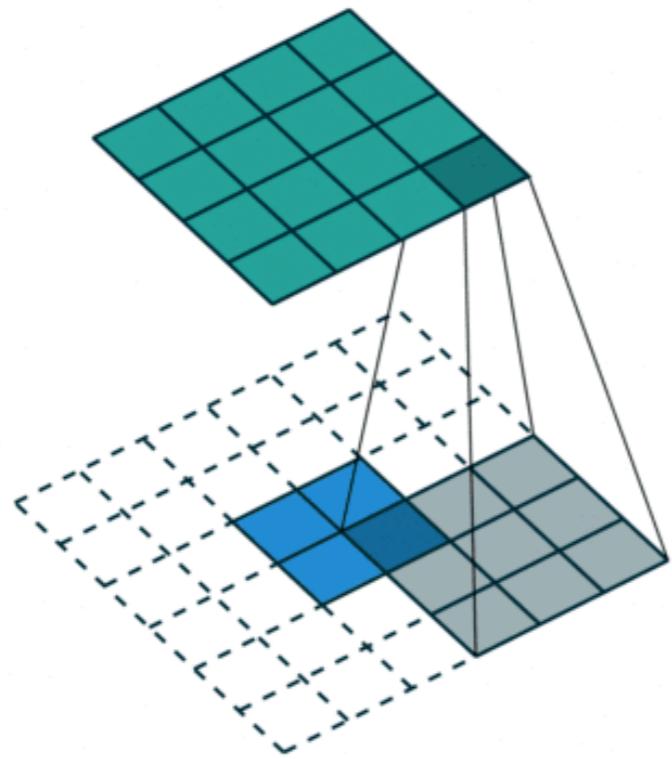
Upsampling 2×2 input to a 4×4 output



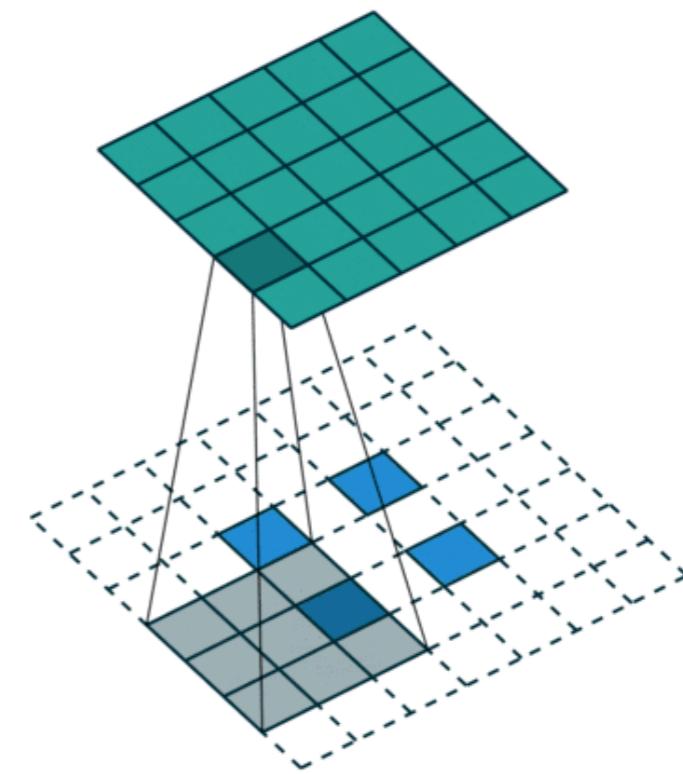
Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution: Transpose Convolution



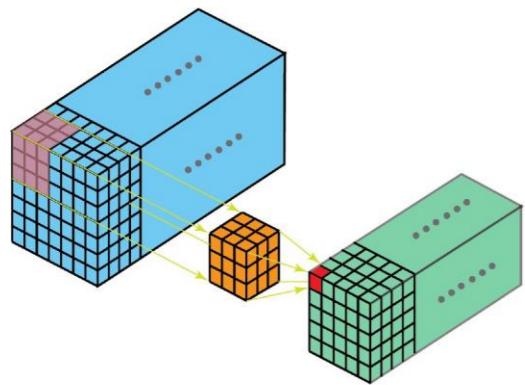
Upsampling 2×2 input to a 4×4 output



Upsampling 2×2 input to a 5×5 output

GIF Credit: [Vincent Dumoulin](#)

Other Variants of Convolution

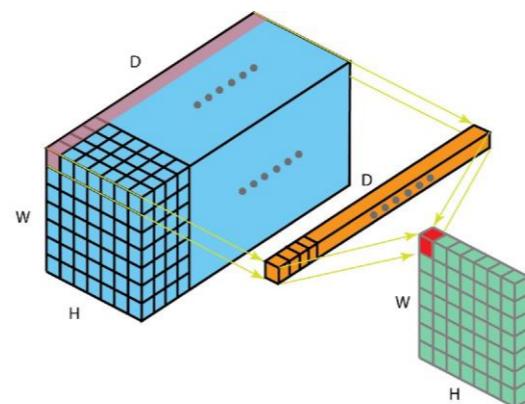


3D Convolution

Extends convolution into three dimensions (width, height, depth).

Commonly used in video data or volumetric data (e.g., medical imaging).

The filter moves in 3D across the input volume to extract spatiotemporal or volumetric features.



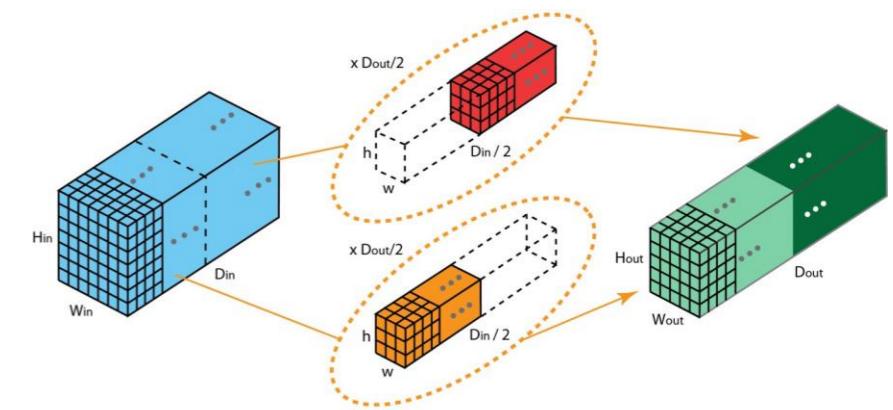
1×1 Convolution Pointwise Convolution

Uses filters of size 1×1 (but full depth of input).

It doesn't capture spatial context but helps to combine features across channels.

Often used for dimensionality reduction or expansion between layers.

Great for reducing computation and adding non-linearity.



Grouped Convolution

Splits input channels into groups, and convolutions are performed separately on each group.

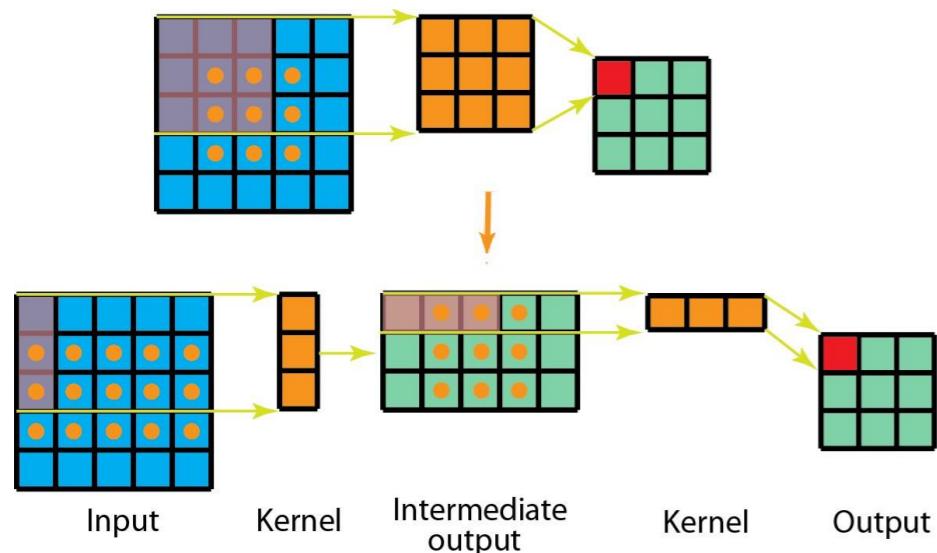
Outputs from groups are concatenated.

Reduces computation and parameters.

Popularized by architectures like AlexNet and ResNeXt.

Credit: Illarion Khlestov, Chi-Feng Wang

Other Variants of Convolution



Spatial Separable Convolution

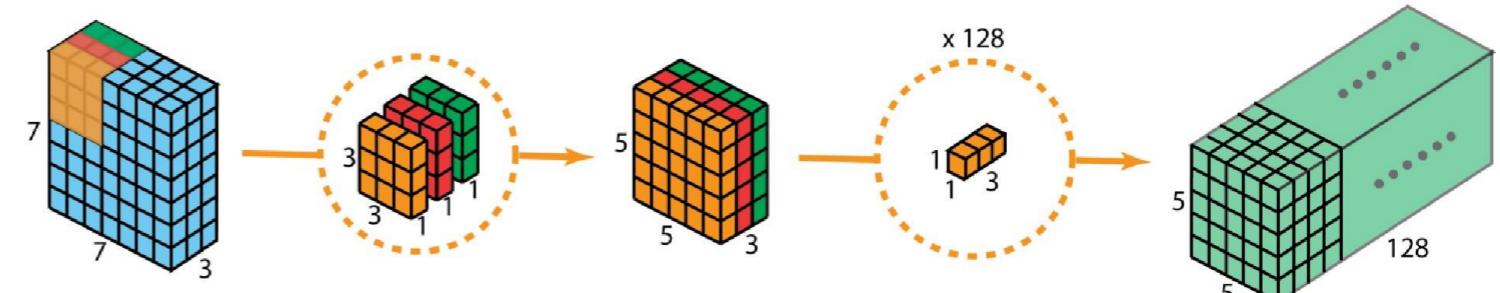
Breaks a convolution operation into two simpler steps:

- Apply a 1D filter horizontally across the input.
- Apply a 1D filter vertically on the output of the first step.

This reduces the number of parameters and computation compared to a full 2D convolution.

The intermediate output helps split the process into smaller parts without losing much information.

Credit: Chi-Feng Wang



Depthwise Separable Convolution

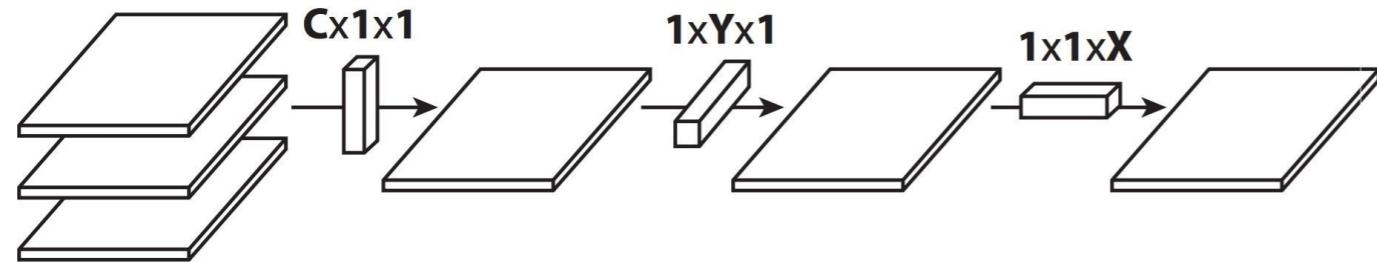
Splits convolution into two parts:

- Depthwise convolution: Applies a single filter per input channel (independently).
- Pointwise convolution (1×1): Combines outputs of the depthwise step across channels.

This dramatically reduces computation and parameters compared to standard convolution.

Widely used in efficient CNN architectures like MobileNet.

Other Variants of Convolutions



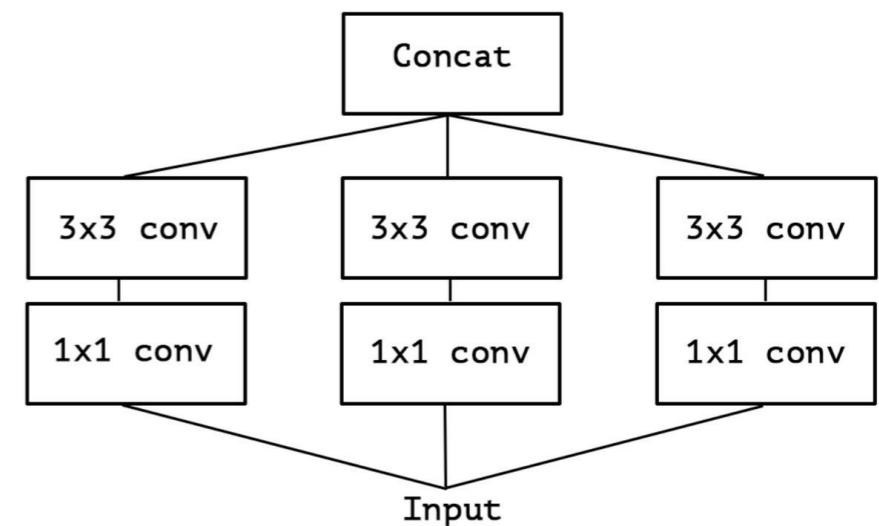
Flattened Convolutions

This breaks down convolutions into separate steps across each dimension. Instead of applying a 3D convolution all at once, it factorizes the process into consecutive convolutions of shape:

$C \times 1 \times 1$ (channel-wise),
 $1 \times Y \times 1$ (height-wise),
 $1 \times 1 \times X$ (width-wise).

This decomposition reduces computational cost and parameters. Flattened convolutions can be seen as factorizing the 3D convolution kernel into simpler 1D convolutions.

Credit: Illarion Khlestov



Spatial and Cross-Channel Convolutions

This involves separating the spatial convolution (e.g., 3×3) from channel-wise (1×1) convolutions.

The input is processed through multiple branches:
- Each branch applies a 3×3 convolution followed by a 1×1 convolution.

The outputs from all branches are concatenated.
This design improves model capacity and efficiency by decoupling spatial and channel-wise feature extraction.

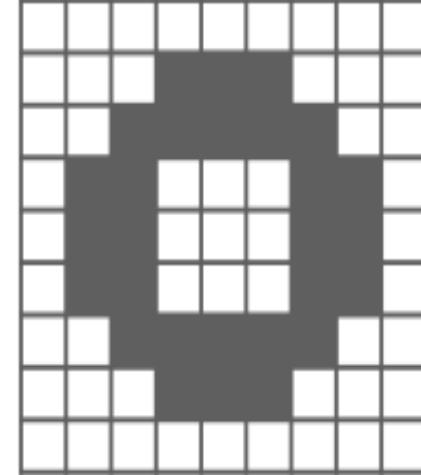
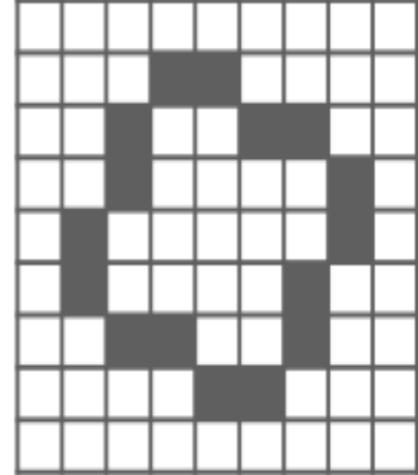
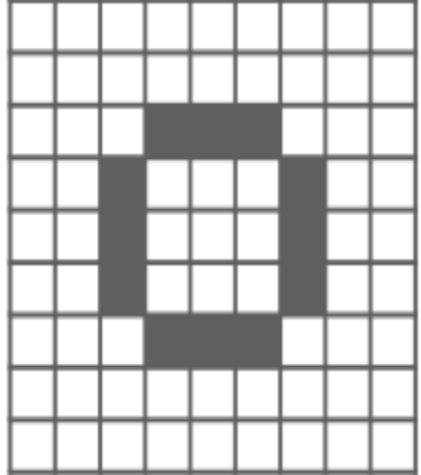
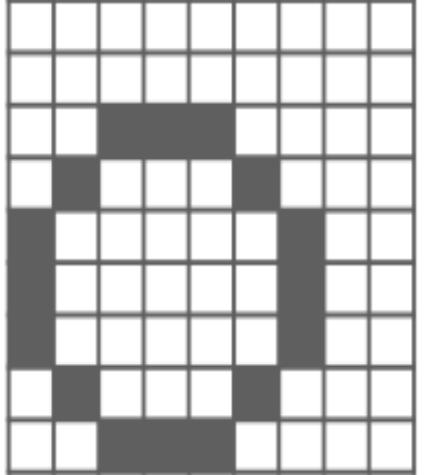
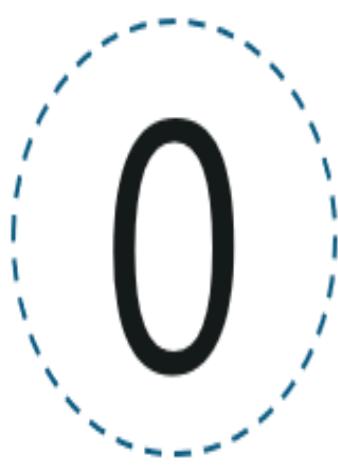
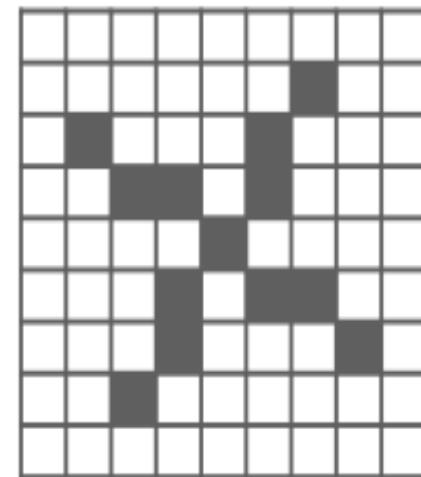
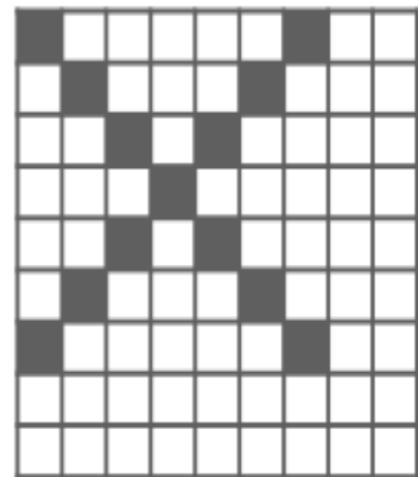
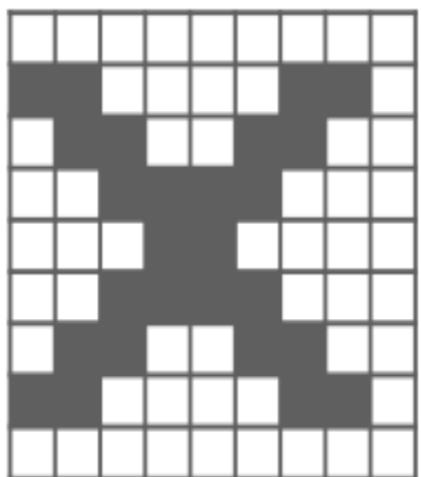
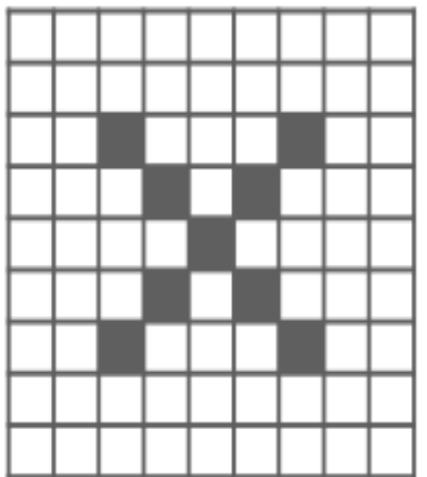
CNN Architecture Overview

Fully Connected Layers-

- Used to compute class scores used as output of the network (e.g., the output layer at the end of the network).
- The dimensions of the output volume is [$1 \times 1 \times N$], where N is the number of output classes.
- This layer has a connection between all of its neurons and every neuron in the previous layer.
- Fully connected layers have the normal parameters for the layer and hyperparameters.
- Fully connected layers perform transformations on the input data volume that are a function of the activations in the input volume and the parameters (weights and biases of the neurons).
- **MULTIPLE FULLY CONNECTED LAYERS**- some CNN architectures use multiple fully connected layers at the end of the network. E.g. AlexNet- two fully connected layers followed by a softmax layer at the end.

CNN-Example

Example of CNN:





ReLU Function and Pooling Layer

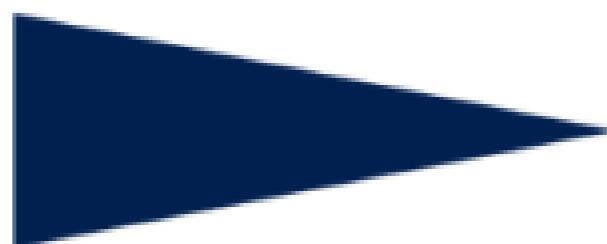
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.0	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.0	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77



0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	1.77

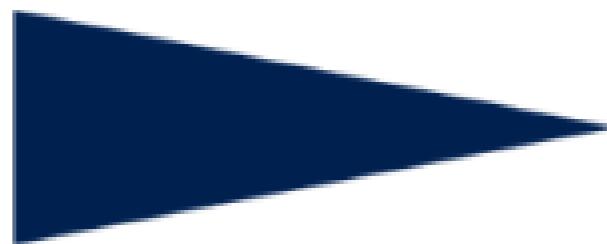
Pooling Layer

0.77	0	0.11	0.11	0.55	0	0.33
0	1.00	0	0.11	0	0.11	0
0.11	0	1.00	0	0.11	0	0.44
0.33	0.33	0	0.55	0	0.33	0.33
0.66	0	0.11	0	1.00	0	0.11
0	0.11	0	0.11	0	1.00	0
0.33	0	0.33	0.33	0.11	0	1.22



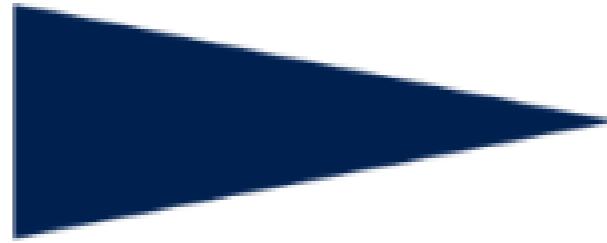
1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.33	0	0.11	0	0.11	0	0.33
0	0.33	0	0.11	0	0.11	0
0.11	0	0.33	0	0.11	0	0.11
0	0.33	0	1.00	0	0.11	0
0.11	0	0.33	0	0.33	0	0.11
0	0.33	0	0.11	0	0.33	0
0.33	0	0.11	0	0.11	0	0.33



0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

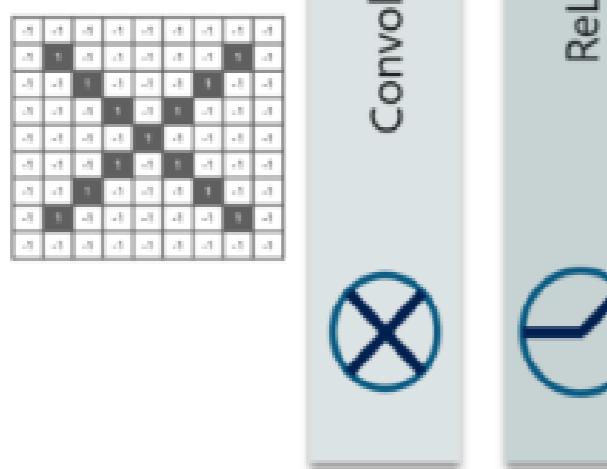
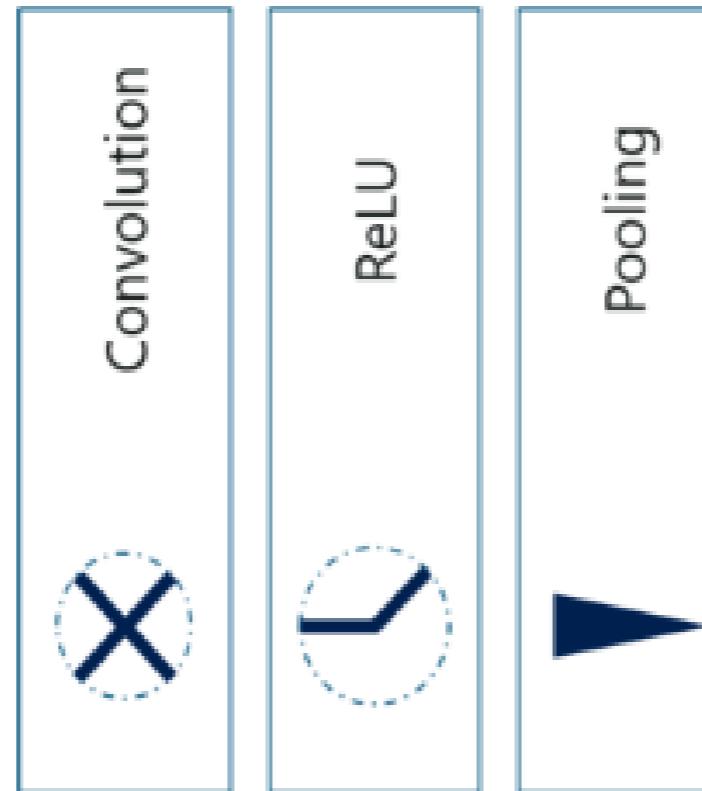
0.33	0	0.33	0.33	0.33	0	0.33
0	0.33	0	0.33	0	1.00	0
0.33	0	0.33	0	0.33	0	0.33
0.33	0.33	0	0.33	0	0.33	0.33
0.33	0	0.33	0	0.33	0	0.33
0	0.33	0	0.33	0	0.33	0
0.33	0	0.33	0.33	0.33	0	0.33



0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

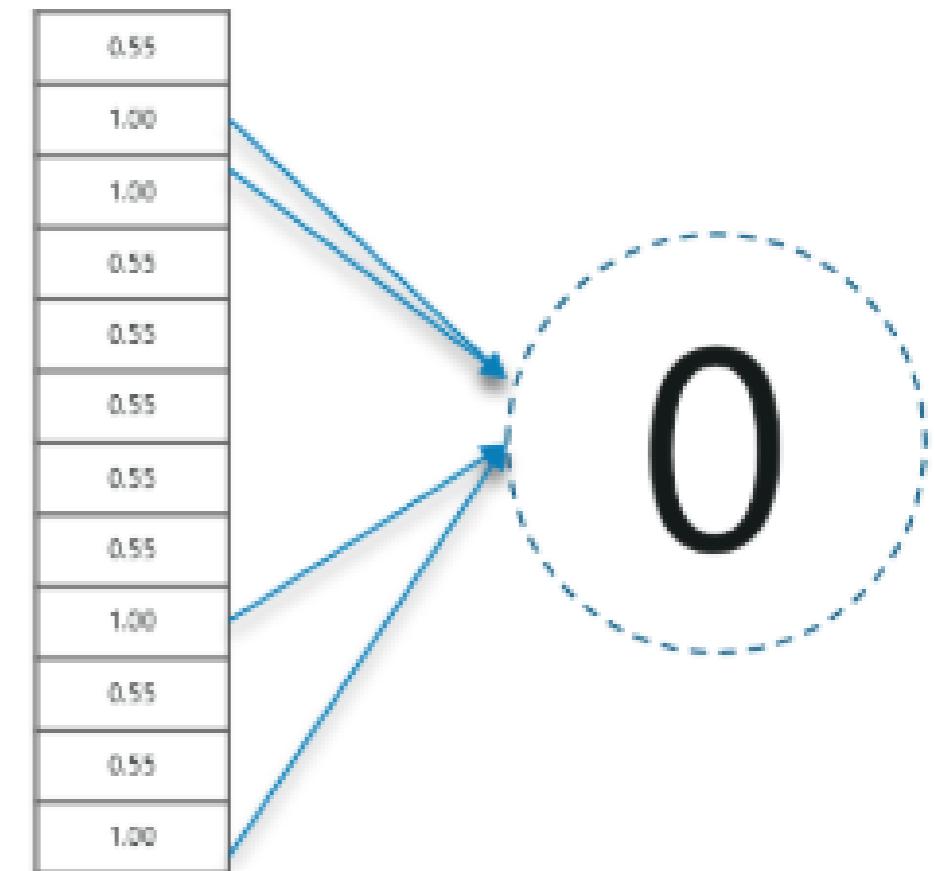
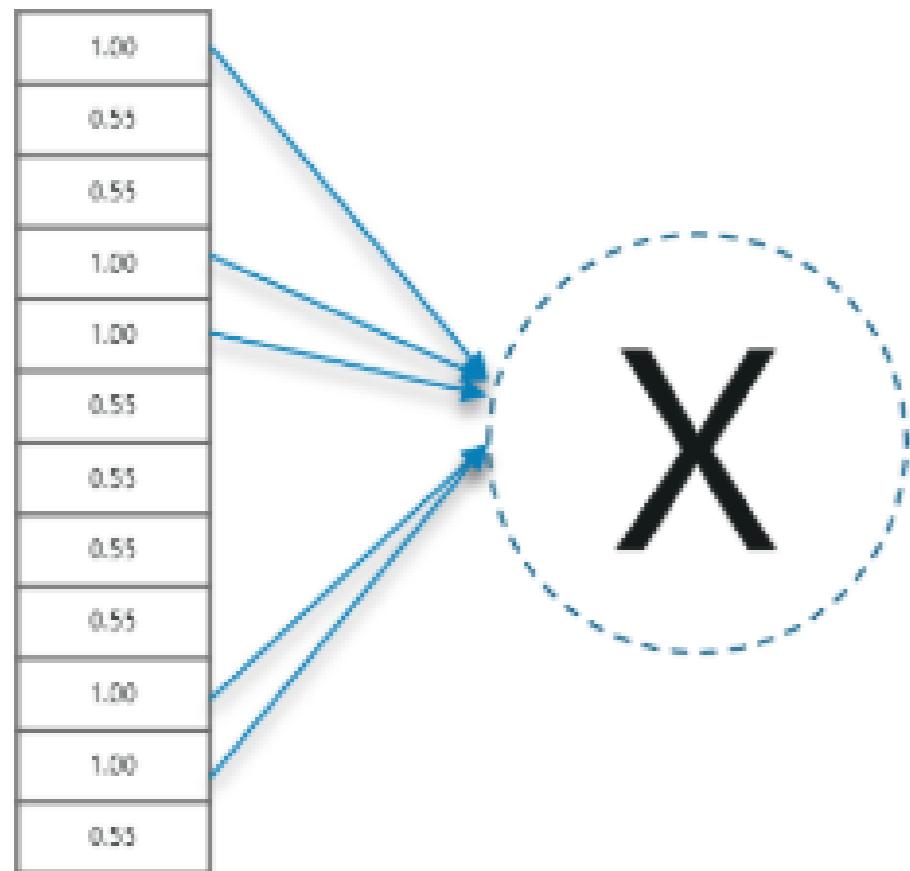
Stacking Up The Layers

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1



1.00
0.55
0.55
1.00
1.00
0.55
0.55
0.55
0.55
1.00
1.00
0.55

Stacking Up The Layers



Stacking Up The Layers

0.9
0.65
0.45
0.87
0.96
0.73
0.23
0.63
0.44
0.89
0.94
0.53

Sum



4.56



5

Sum



0.91

1.00
0.55
0.55
1.00
1.00
0.55
0.55
0.55
1.00
1.00
0.55

Input Image

Vector for 'X'

Stacking Up The Layers

0.9
0.65
0.45
0.87
0.96
0.73
0.23
0.63
0.44
0.89
0.94
0.53

Sum



2.07



4

Sum



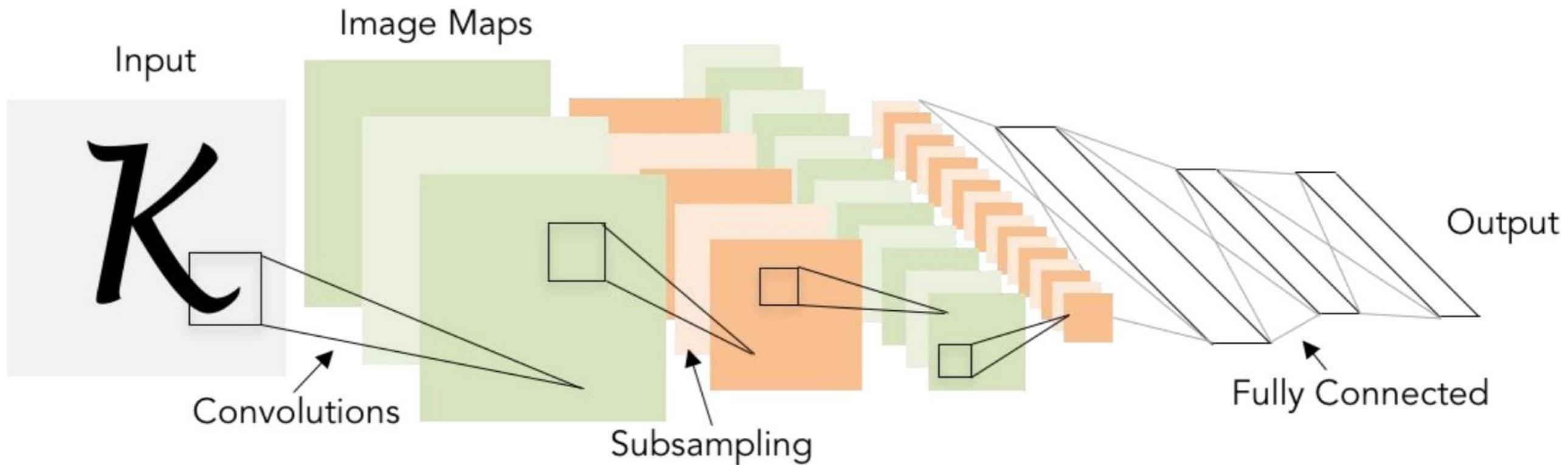
0.51

0.55
1.00
1.00
0.55
0.55
0.55
0.55
0.55
1.00
0.55
0.55
1.00

Input Image

Vector for 'O'

LeNet-5 (1989-1998)

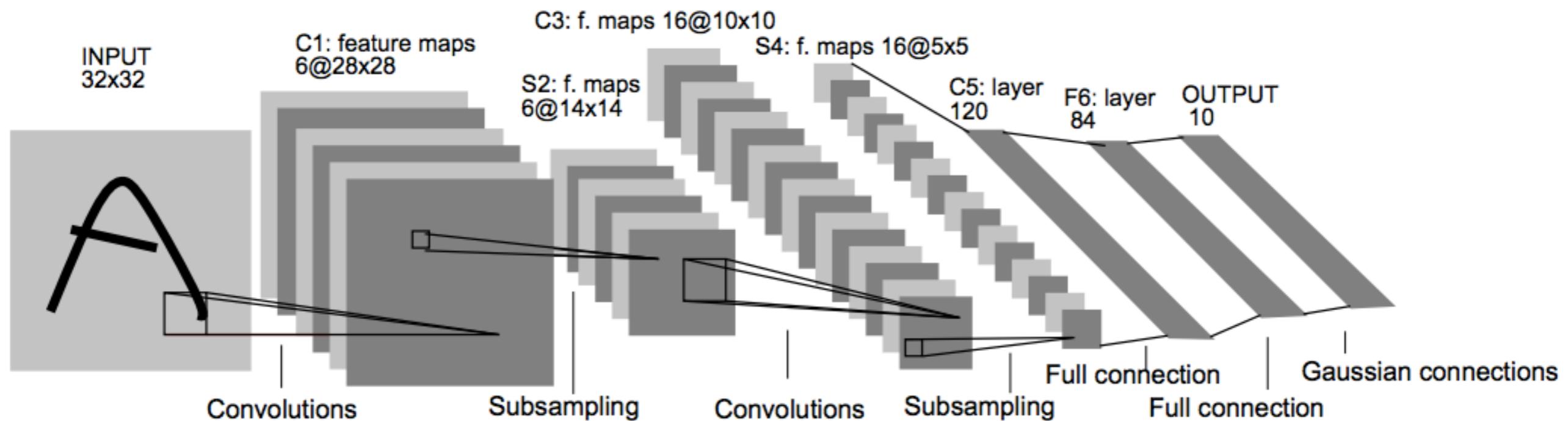


- Conv filters were 5×5 , applied at stride 1
- Subsampling (Pooling) layers were 2×2 applied at stride 2
- Overall Architecture:** [CONV-POOL-CONV-POOL-FC-FC]

Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

LeNet-5 (1989-1998)

LeNet-5 Architecture. Credit: [LeCun et al., 1998](#)



LeNet-5 (1989-1998)

Architecture

- **Convolutional #1** outputs $28 \times 28 \times 6$
 - **Activation** any activation function, we will `relu`
- **Pooling #1** The output shape should be $14 \times 14 \times 6$.
- **Convolutional #2** outputs $10 \times 10 \times 16$.
 - **Activation** any activation function, we will `relu`
- **Pooling #2** outputs $5 \times 5 \times 16$.
 - **Flatten** Flatten the output shape of the final pooling layer
- **Fully Connected #1** outputs 120
 - **Activation** any activation function, we will `relu`
- **Fully Connected #2** outputs 84
 - **Activation** any activation function, we will `relu`
- **Fully Connected (Logits) #3** output 10

ImageNet Classification Challenge

- Image database organized according to WordNet hierarchy (currently only nouns)
- Currently, over five hundred images per node
- Started the ImageNet LSVRC in 2010, for benchmarking of methods for image classification
- Performance measure in Top-1 error and Top-5 error
- <http://www.image-net.org/>



Loss Functions: Beyond Mean Square Error

- **Cross-Entropy Loss Function:** Most popular for classification
- Given by:

$$\begin{aligned} L &= -\frac{1}{C} \sum_{i=1}^C y_i \log \hat{y}_i \\ &= -y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}) \text{(binary case)} \end{aligned}$$

- When activation function is sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$, derivative of cross-entropy loss function, $\frac{\partial L}{\partial w_j}$, w.r.t. a weight in last layer, w_j , is:

Loss Functions: Beyond Mean Square Error

- **Cross-Entropy Loss Function:** Most popular for classification

- Given by:

$$L = -\frac{1}{C} \sum_{i=1}^C y_i \log \hat{y}_i$$

$$= -y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}) \text{(binary case)}$$

- When activation function is sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}, \text{ derivative of}$$

cross-entropy loss function, $\frac{\partial L}{\partial w_j}$, w.r.t. a weight in last layer, w_j , is:

$$\begin{aligned}\frac{\partial L}{\partial w_j} &= -\frac{1}{n} \sum_x \frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)} \frac{\partial \sigma}{\partial w_j} \\ &= -\frac{1}{n} \sum_x \frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)} \sigma'(z)x_j \\ &= \frac{1}{n} \sum_x \frac{\sigma'(z)x_j}{\sigma(z)(1-\sigma(z))} (\sigma(z) - y) \\ &= \frac{1}{n} \sum_x x_j (\sigma(z) - y)\end{aligned}$$

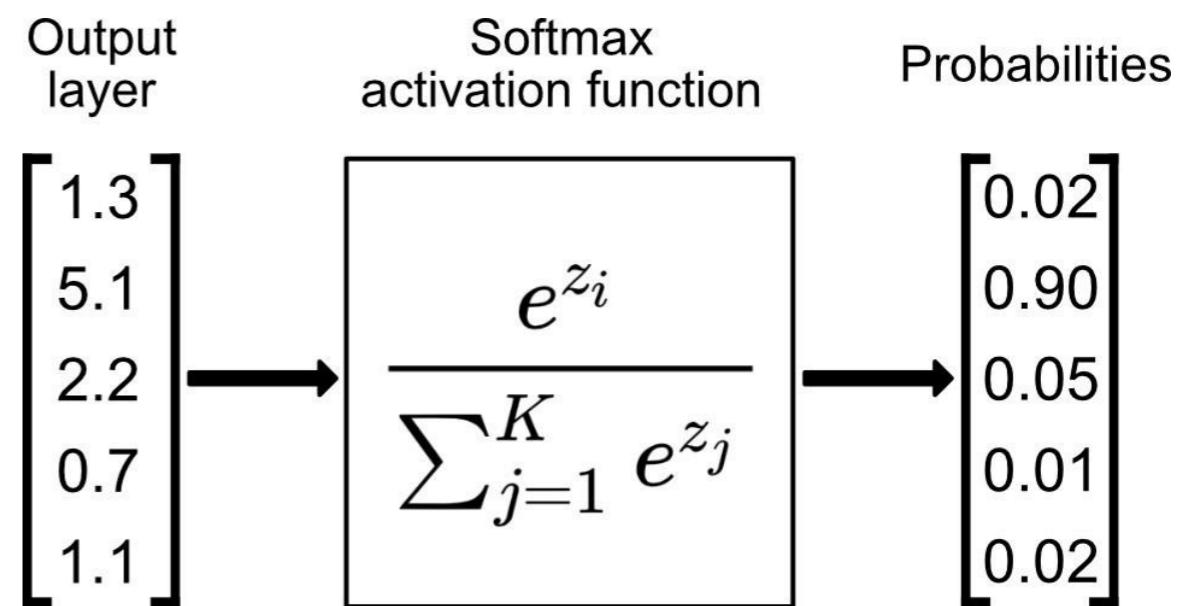
Note the last term in the final expression, very similar to gradient of MSE loss function

Activation Function in Output Layer

Softmax activation function

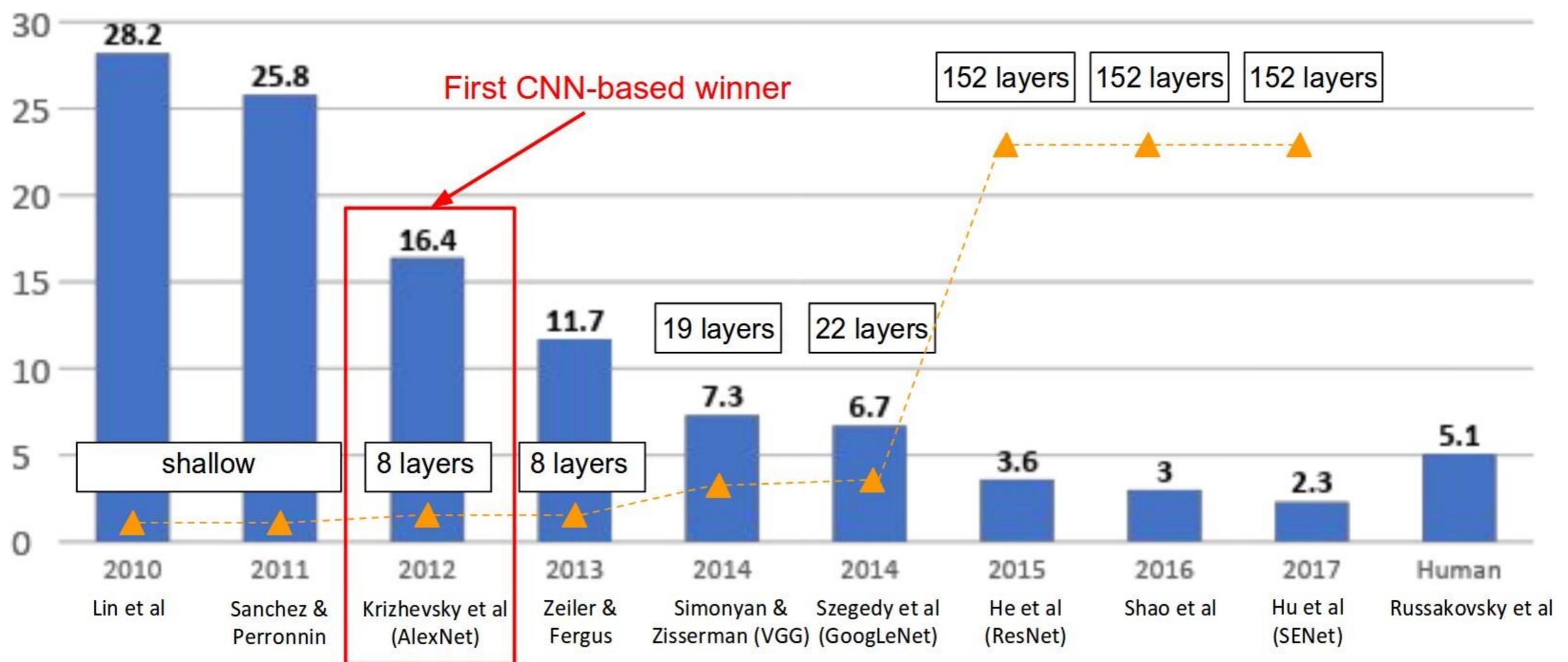
$$a_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Helps convert output layer values (also called **logits**) to probability scores



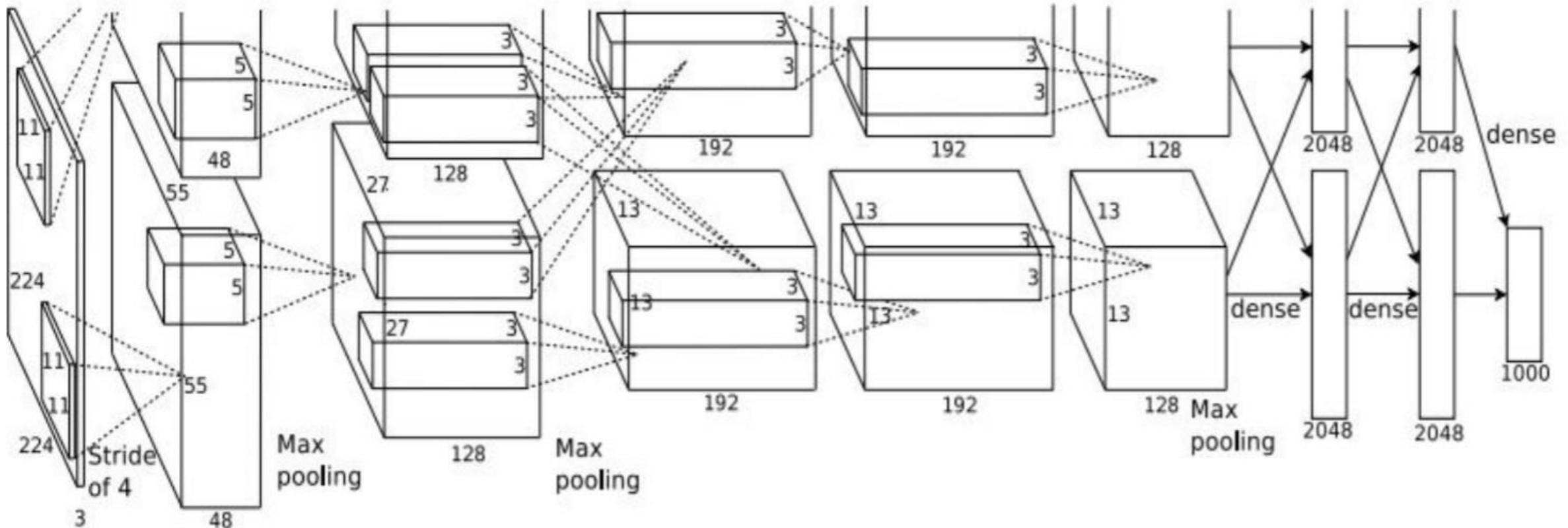
Credit: [Dario Redicic, TowardsDataScience blog](#)

Winners of ImageNet Classification Challenge



Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

AlexNet¹



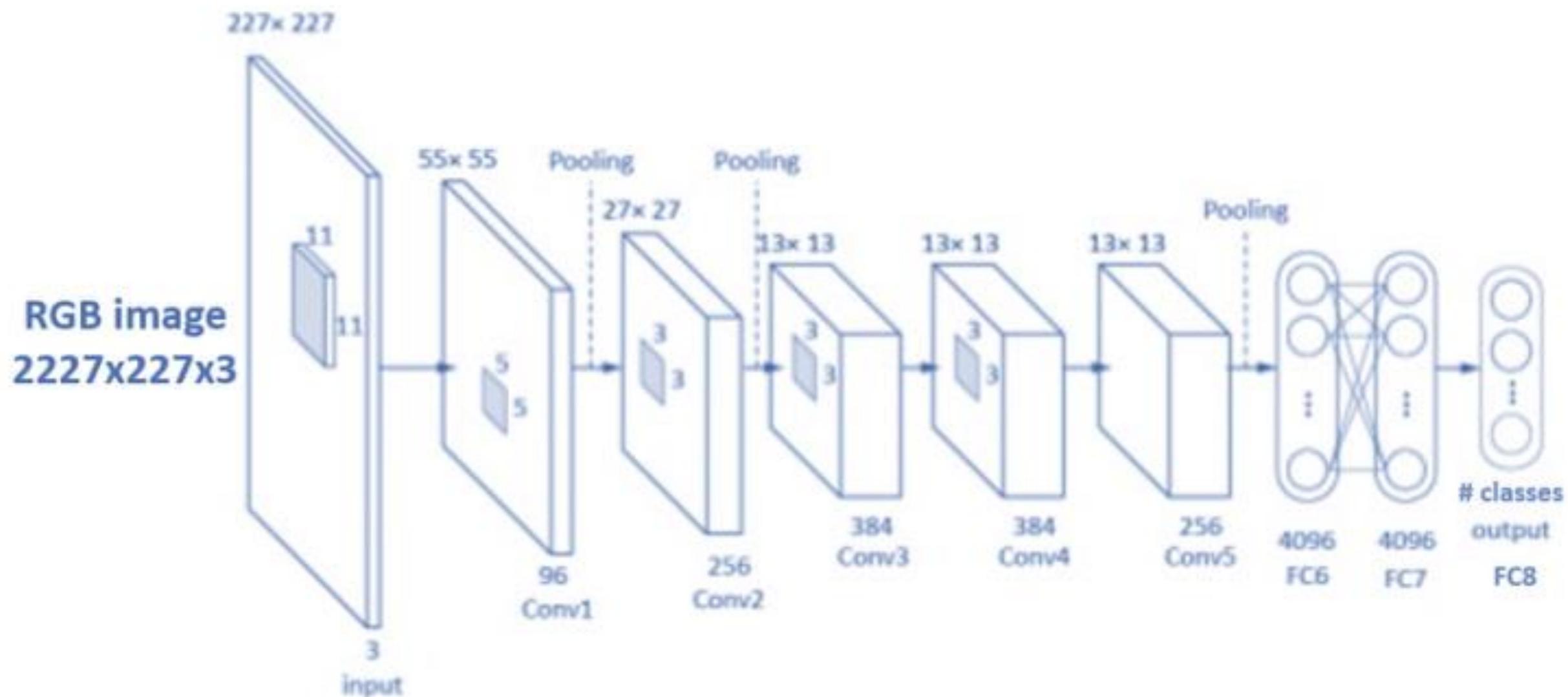
- Winner of ImageNet LSVRC-2012
- Overall architecture design similar to LeNet; but deeper with conv layers stacked on top of each other
- Trained over 1.2M images using SGD with regularization

¹Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

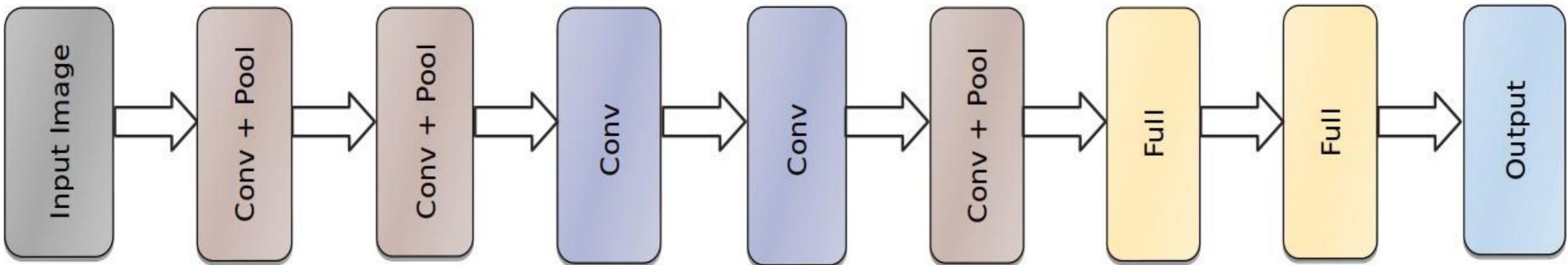
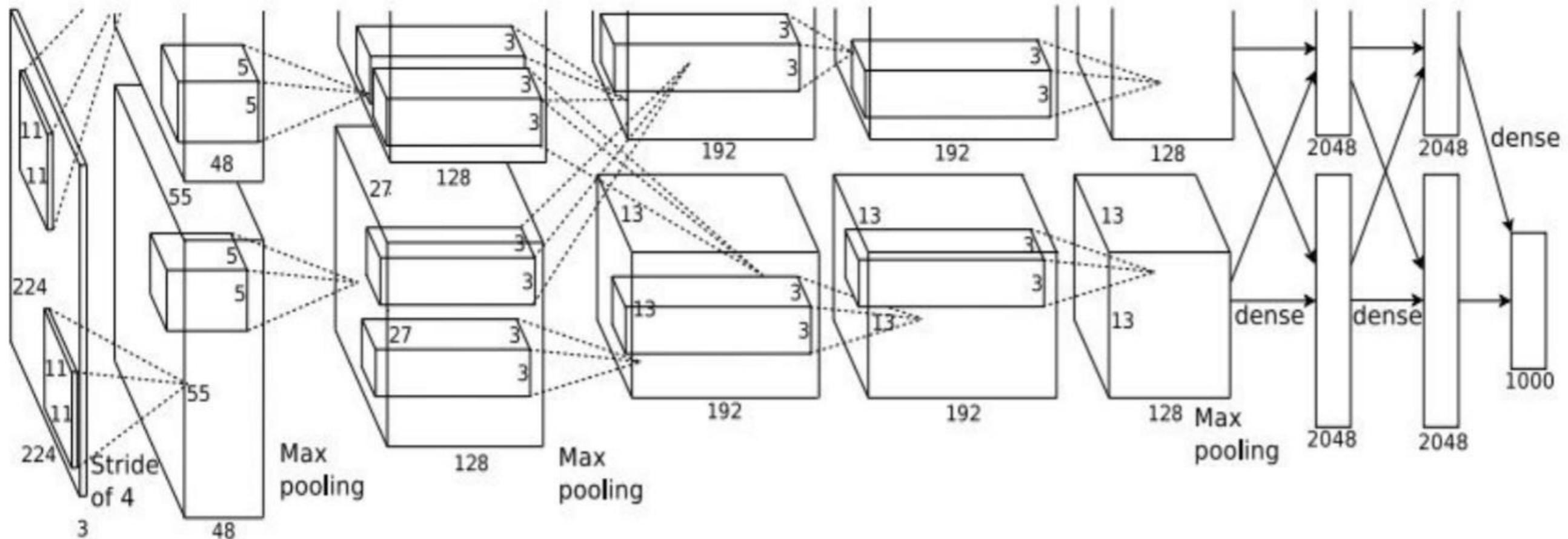
AlexNet¹

Layer	Type	Maps	Size	Kernel Size	Stride	Padding	Activation
Out	Fully Connected	—	1000	—	—	—	Softmax
F10	Fully Connected	—	4096	—	—	—	ReLU
F9	Fully Connected	—	4096	—	—	—	ReLU
S8	Max pooling	256	6X6	3X3	2	valid	—
C7	Convolution	256	13X13	3X3	1	same	ReLU
C6	Convolution	384	13X13	3X3	1	same	ReLU
C5	Convolution	384	13X13	3X3	1	same	ReLU
S4	Max pooling	256	13X13	3X3	2	valid	—
C3	Convolution	256	27X27	5X5	1	same	ReLU
S2	Max pooling	96	27X27	3X3	2	valid	—
C1	Convolution	96	55X55	11X11	4	valid	ReLU
In	Input	3(RGB)	227X227	—	—	—	—

AlexNet

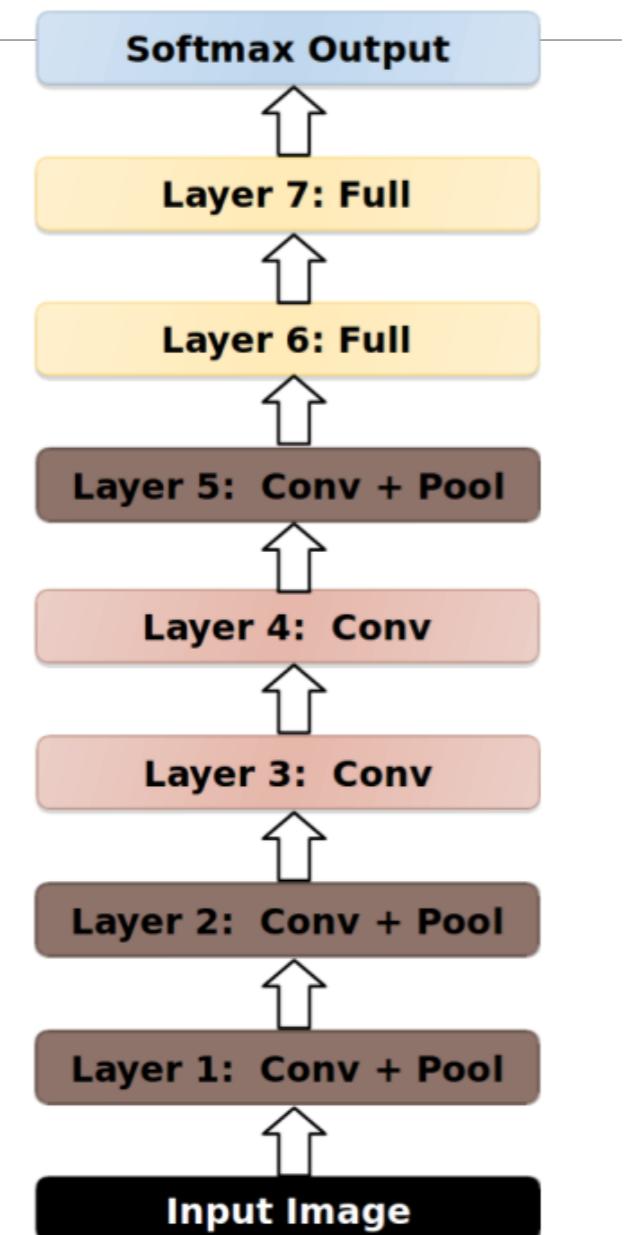


AlexNet

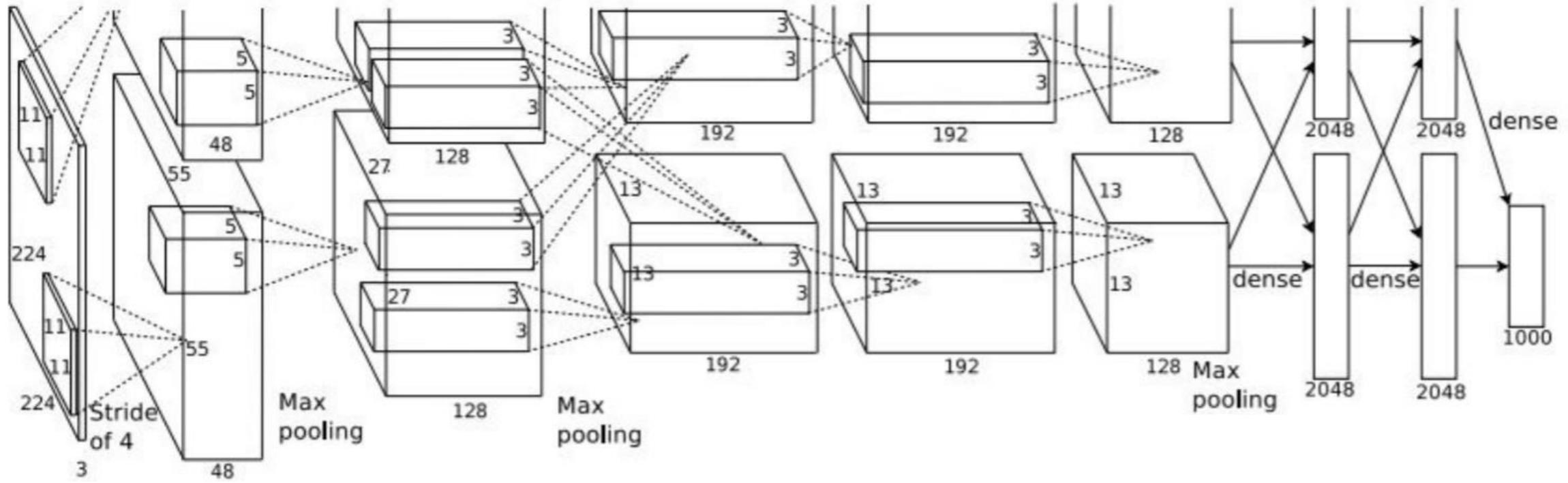


AlexNet

- 8 layers in total (5 convolutional layers, 3 fully connected layers)
- Trained on ImageNet Dataset
- Response normalization layers** follow the first and second convolutional layers.
- Max-pooling follow first, second and the fifth convolutional layers
- The ReLU non-linearity is applied to the output of every layer



AlexNet



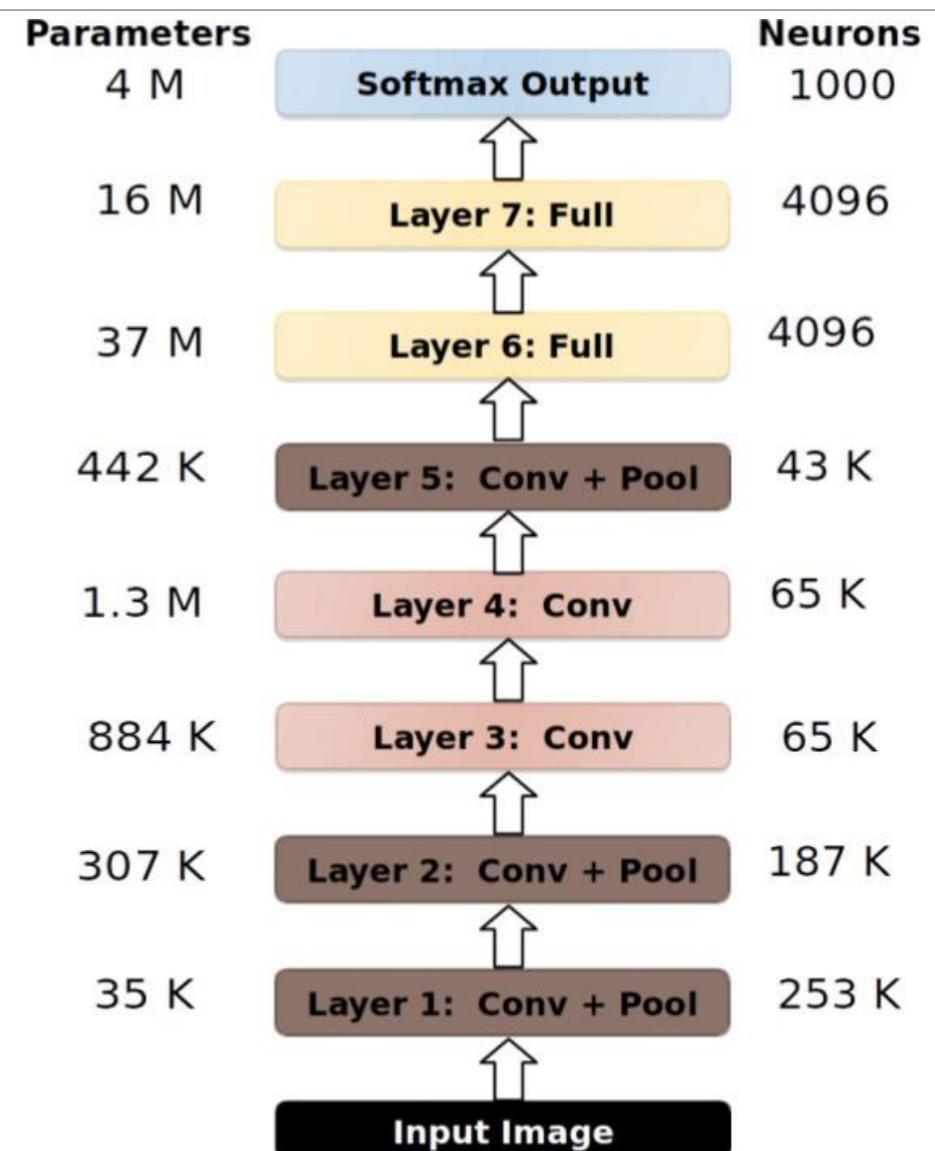
About 57 M parameters are in
the fully connected layers

Parameters :	$[(11 \times 11 \times 3) + 1] \times 96 = 35 \text{ K}$	$[5 \times 5 \times 48] \times 256 = 307 \text{ K}$	$[3 \times 3 \times 256] \times 384 = 884 \text{ K}$	663 K	442 K	37 M	16 M	4 M	60 M
Neurons :	253,440	$27 \times 27 \times 256 = 186,624$	$13 \times 13 \times 384 = 64,896$	$13 \times 13 \times 384 = 64,896$	$13 \times 13 \times 256 = 43,264$	4096	4096	1000	0.63 M

Total

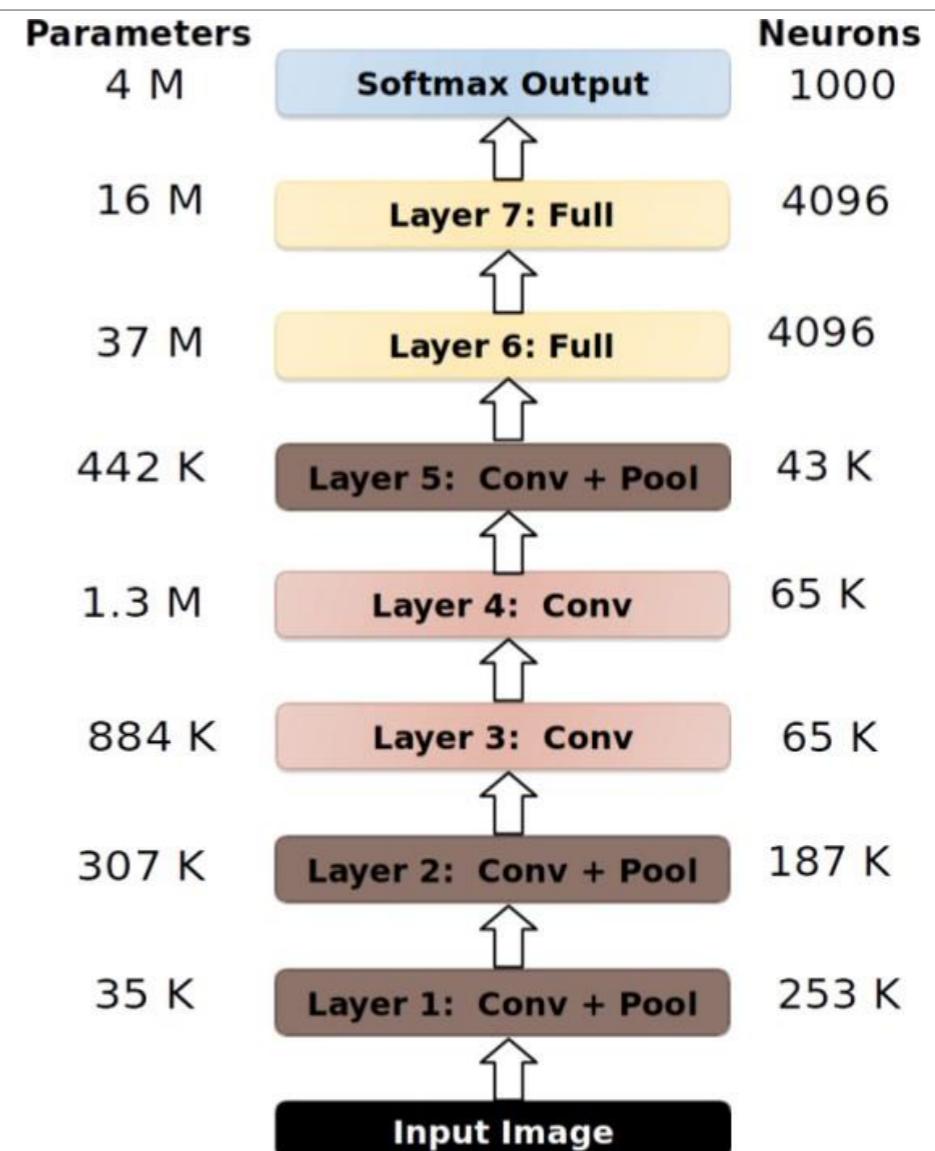
AlexNet

- Convolutional layers cumulatively contain about 90 – 95% of computation, only about 5% of the parameters
- Fully-connected layers contain about 95% of parameters.

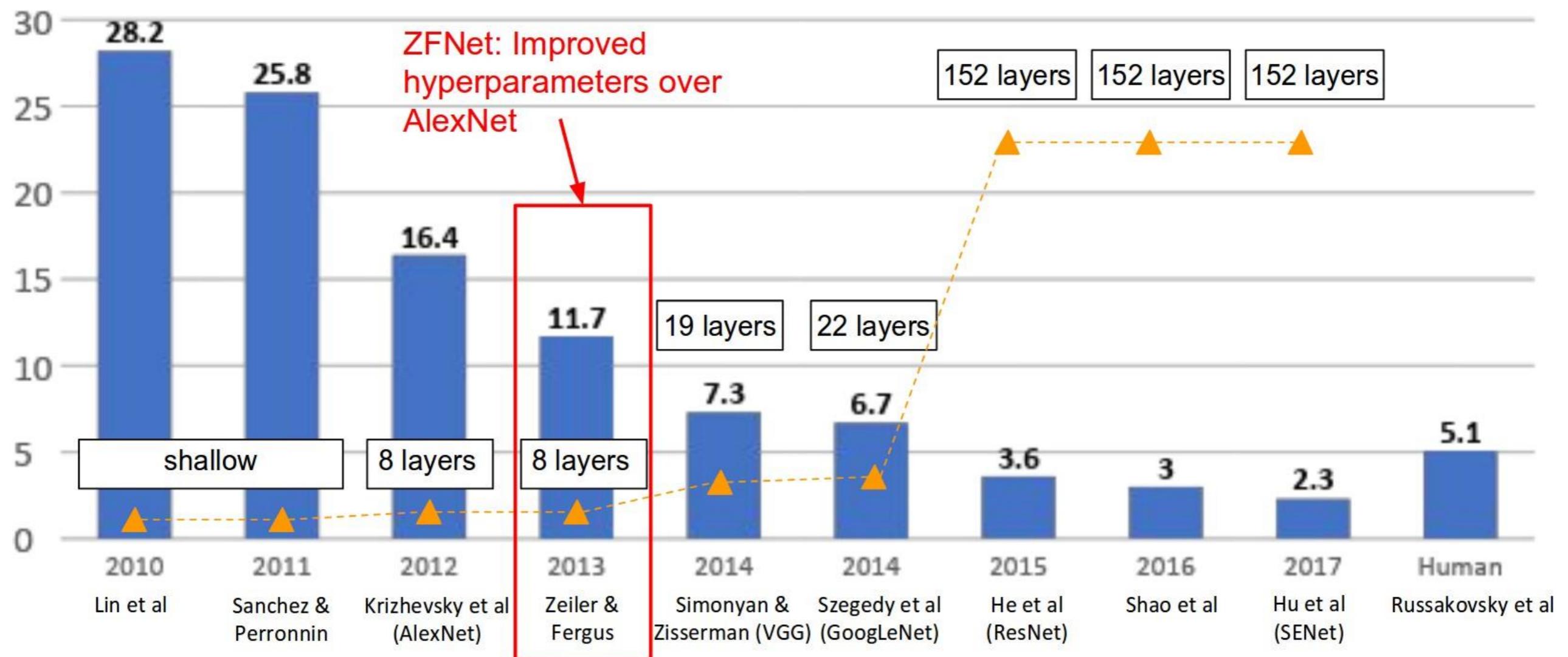


AlexNet

- Convolutional layers cumulatively contain about 90 – 95% of computation, only about 5% of the parameters
- Fully-connected layers contain about 95% of parameters.
- Trained with SGD
 - on **two** NVIDIA GTX 580 3GB GPUs
 - for about a week

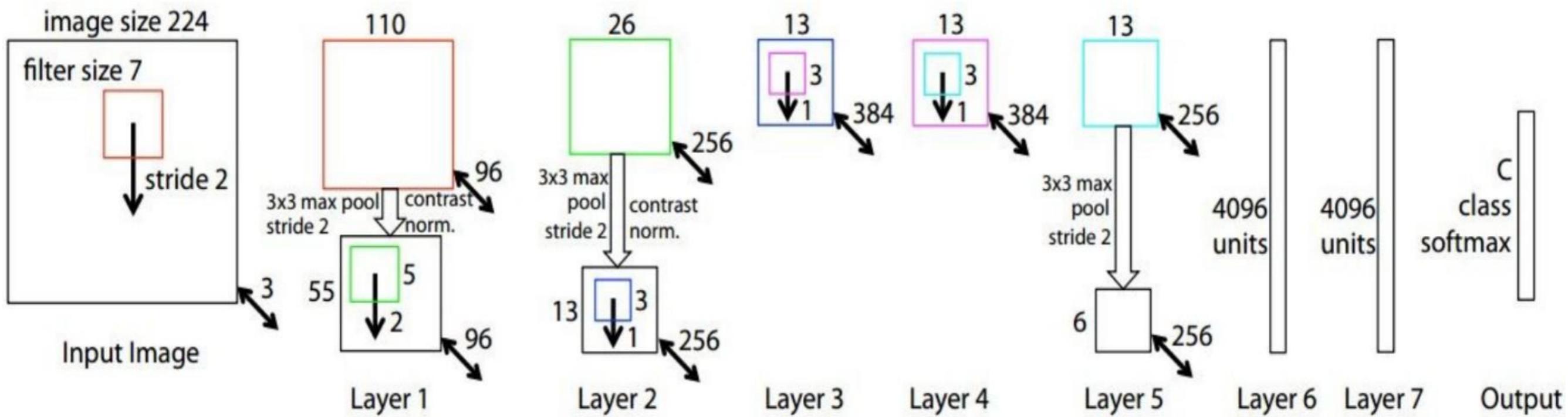


Winners of ImageNet Classification Challenge



Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

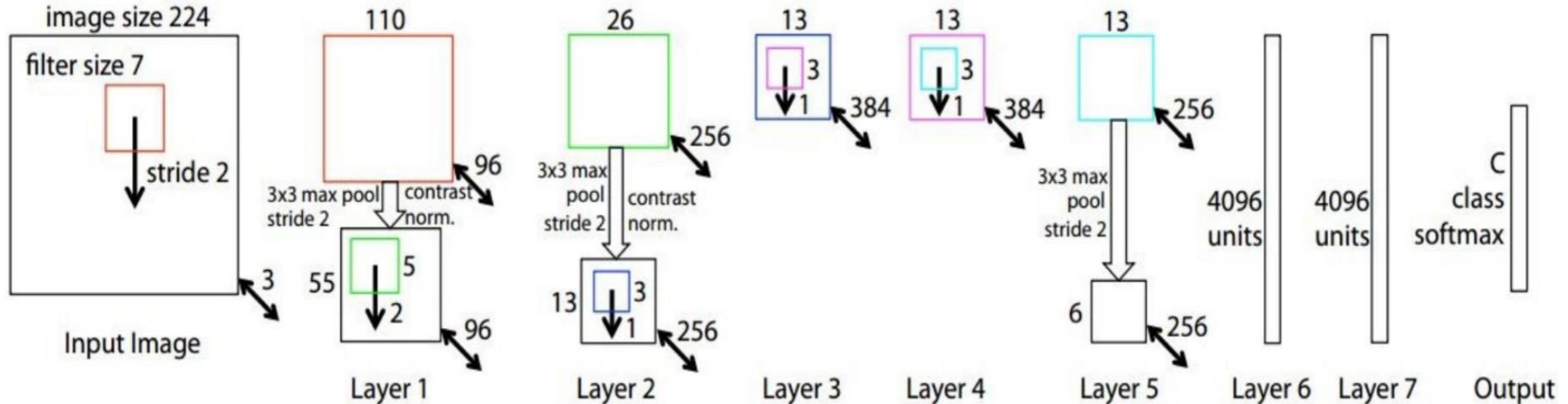
ZFNet (2013)²



- Similar to AlexNet but:

²Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

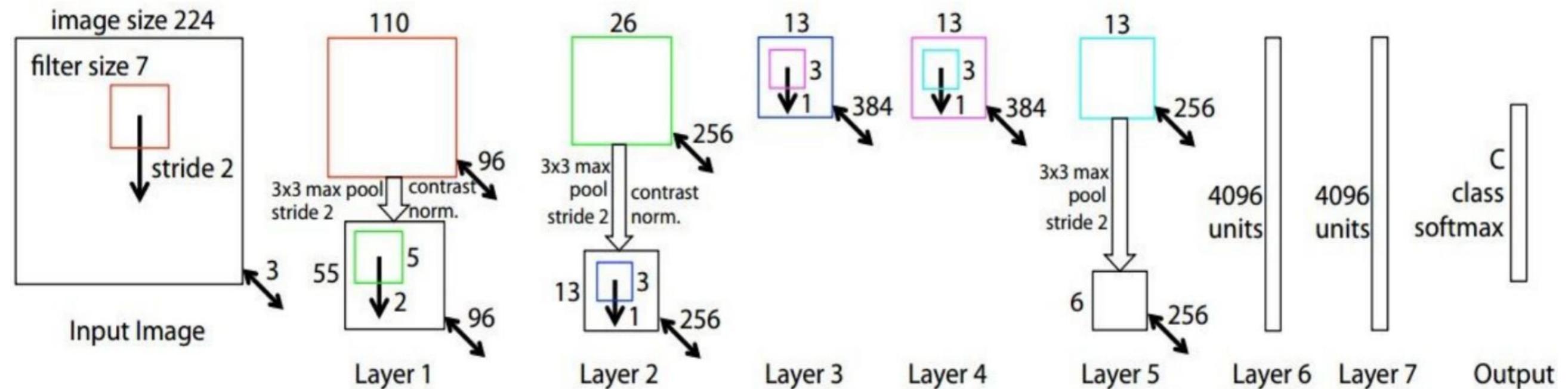
ZFNet (2013)²



- Similar to AlexNet but:

²Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

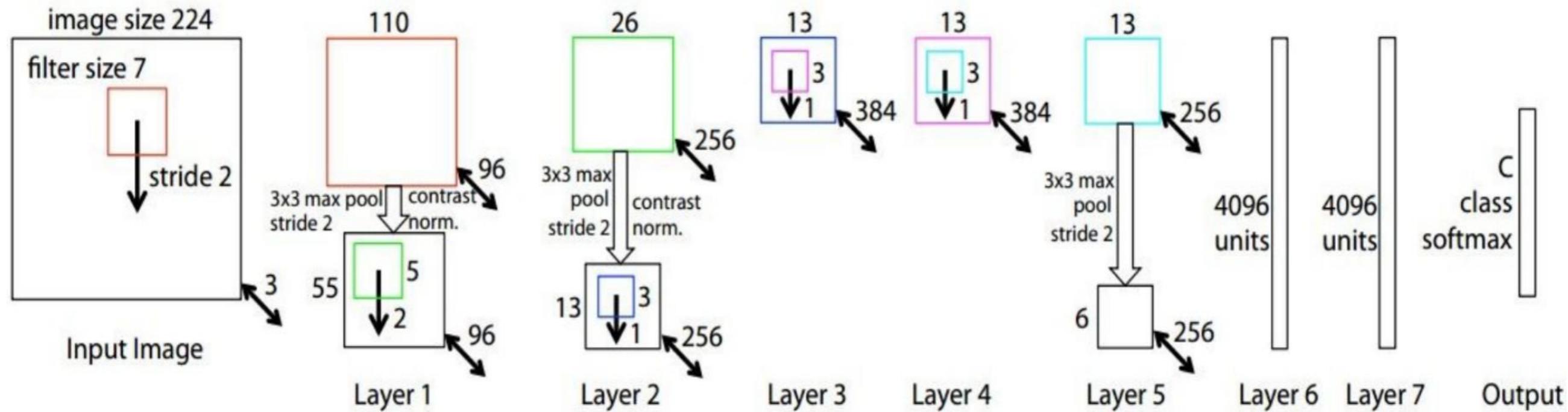
ZFNet (2013)²



- Similar to AlexNet but:
 - CONV1: change from $(11 \times 11 \text{ stride } 4)$ to $(7 \times 7 \text{ stride } 2)$

²Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

ZFNet (2013)²

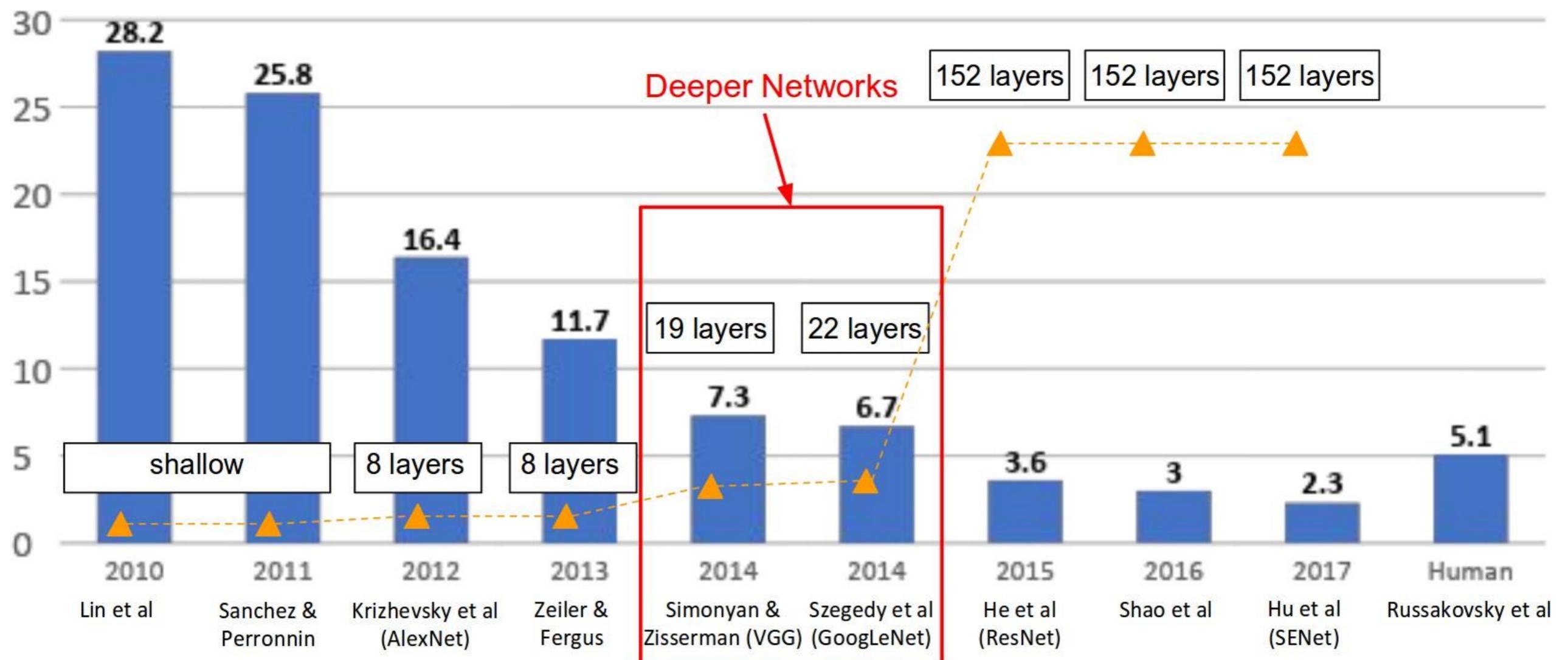


- Similar to AlexNet but:
 - CONV1: change from $(11 \times 11 \text{ stride } 4)$ to $(7 \times 7 \text{ stride } 2)$
 - CONV3,4,5: instead of 384, 384, 256 filters, use 512, 1024, 512
- ImageNet top-5 error: 16.4% \rightarrow 11.7%

Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

²Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

Winners of ImageNet Classification Challenge



Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

VGGNet

image
conv-64
maxpool

conv-128
maxpool

conv-256
conv-256
maxpool

conv-512
conv-512
maxpool

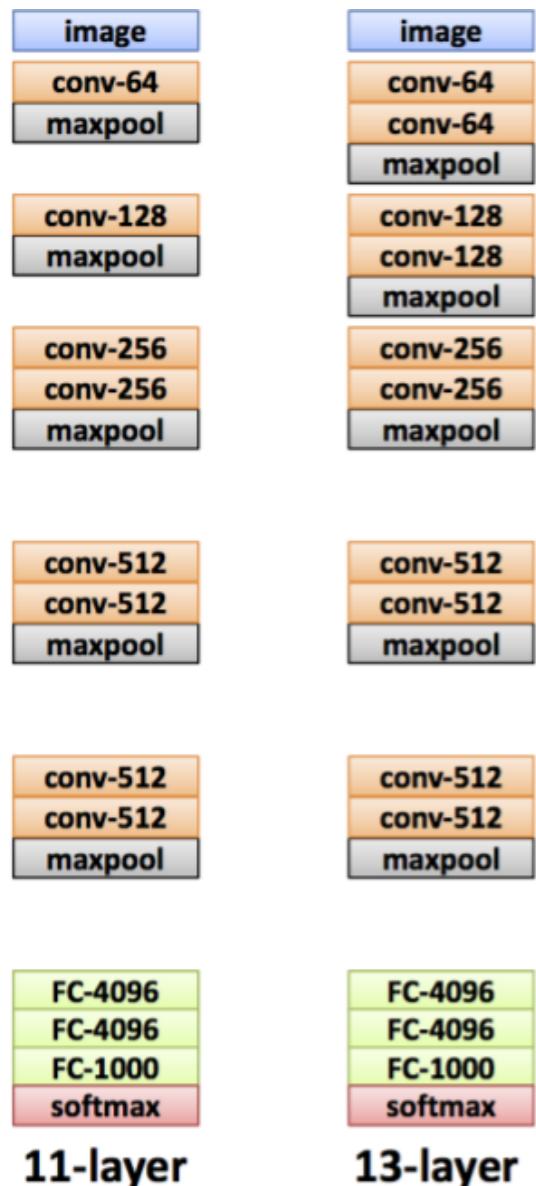
conv-512
conv-512
maxpool

FC-4096
FC-4096
FC-1000
softmax

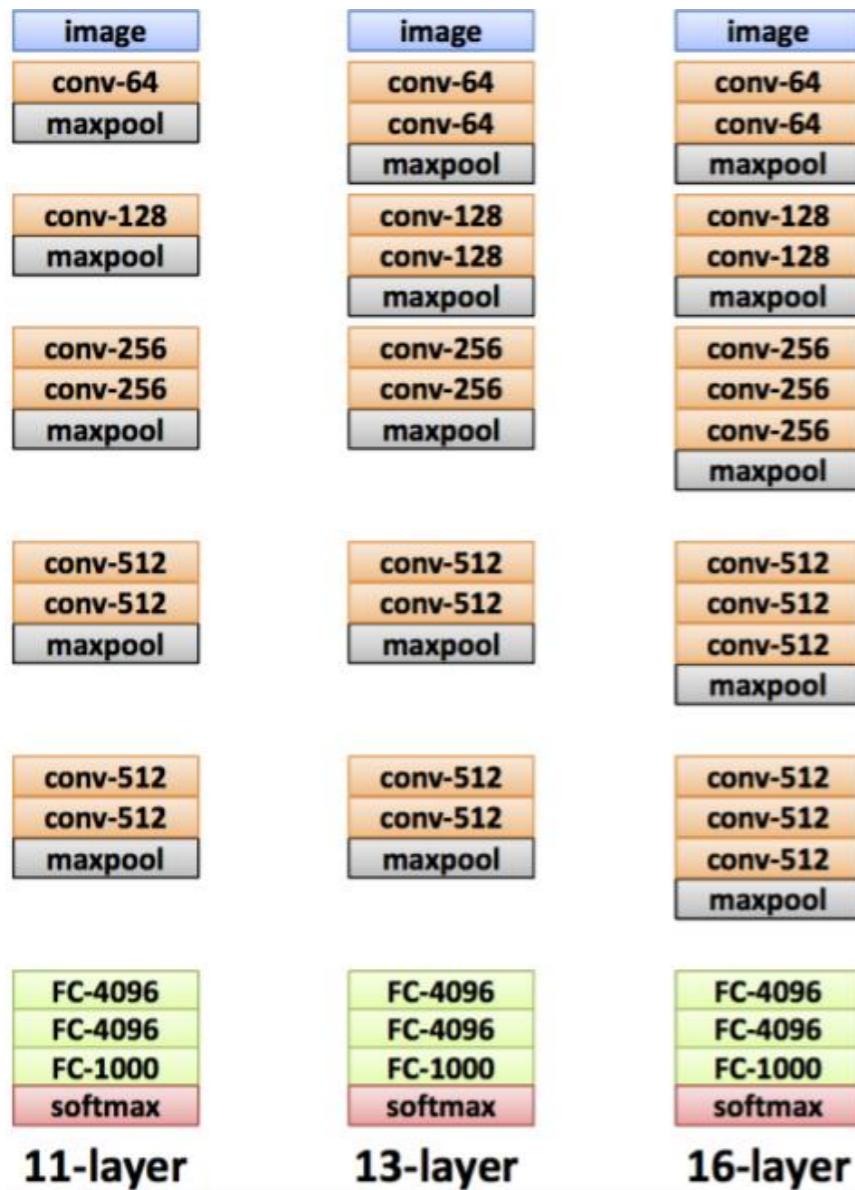
11-layer



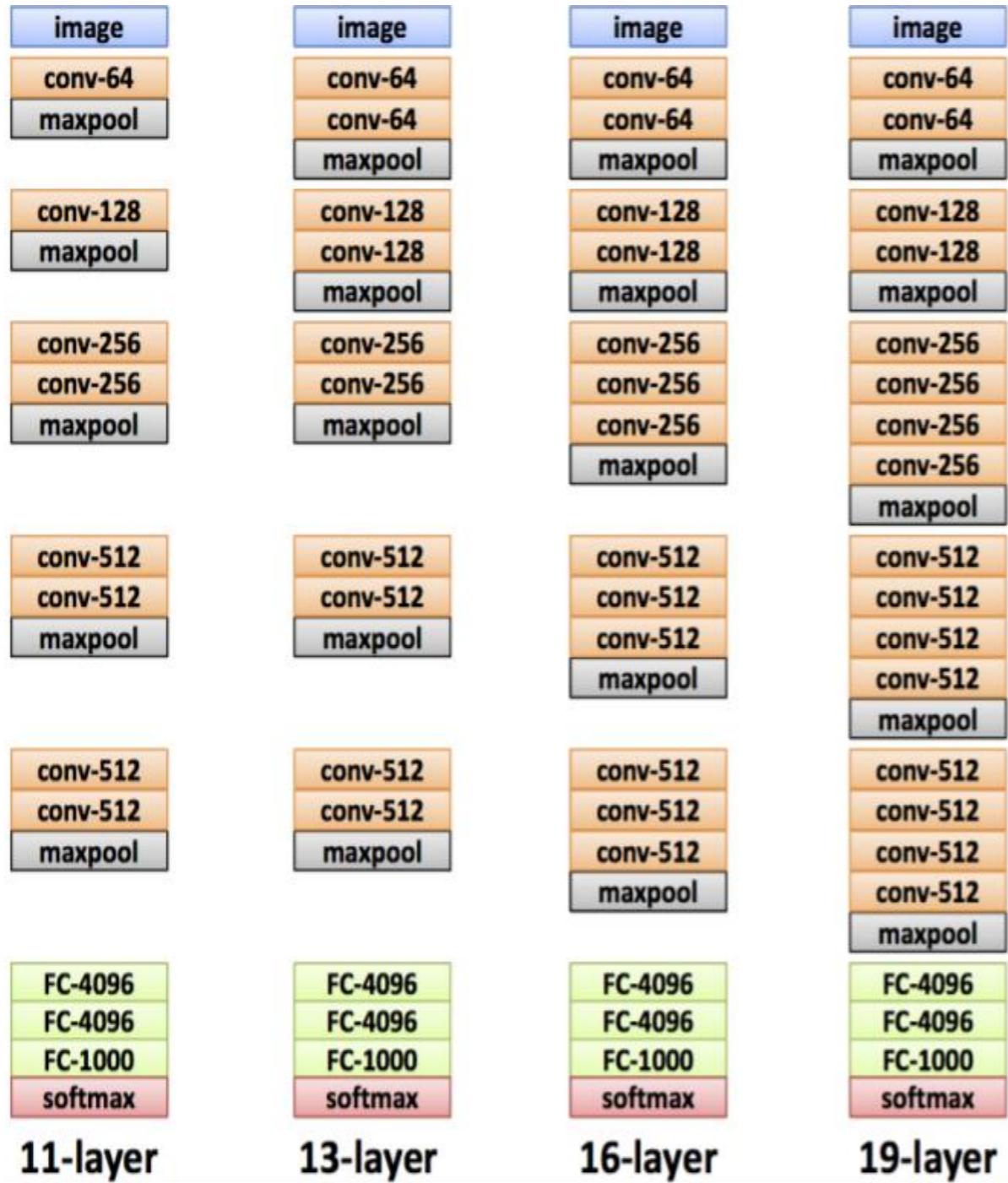
VGGNet



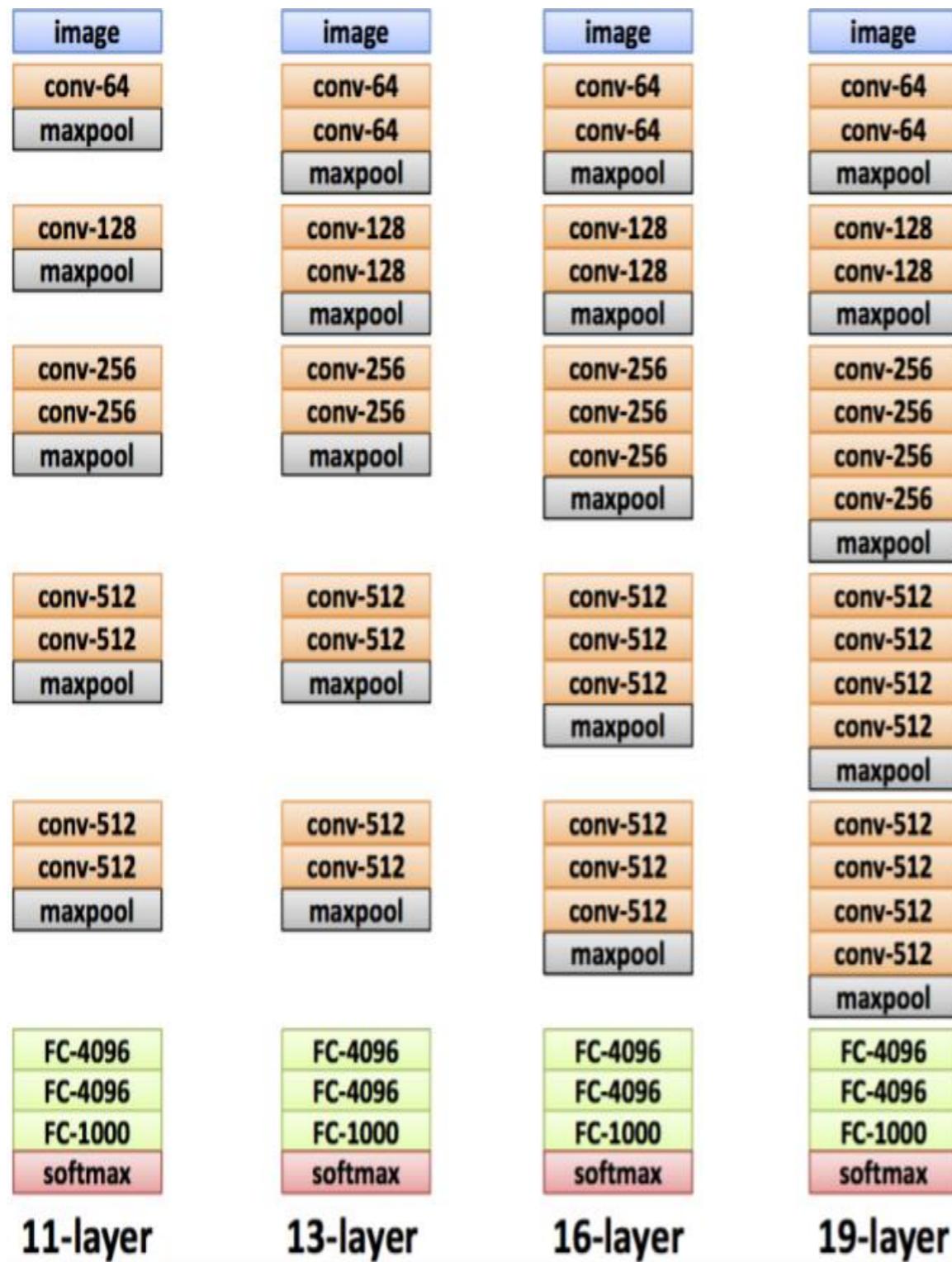
VGGNet



VGGNet

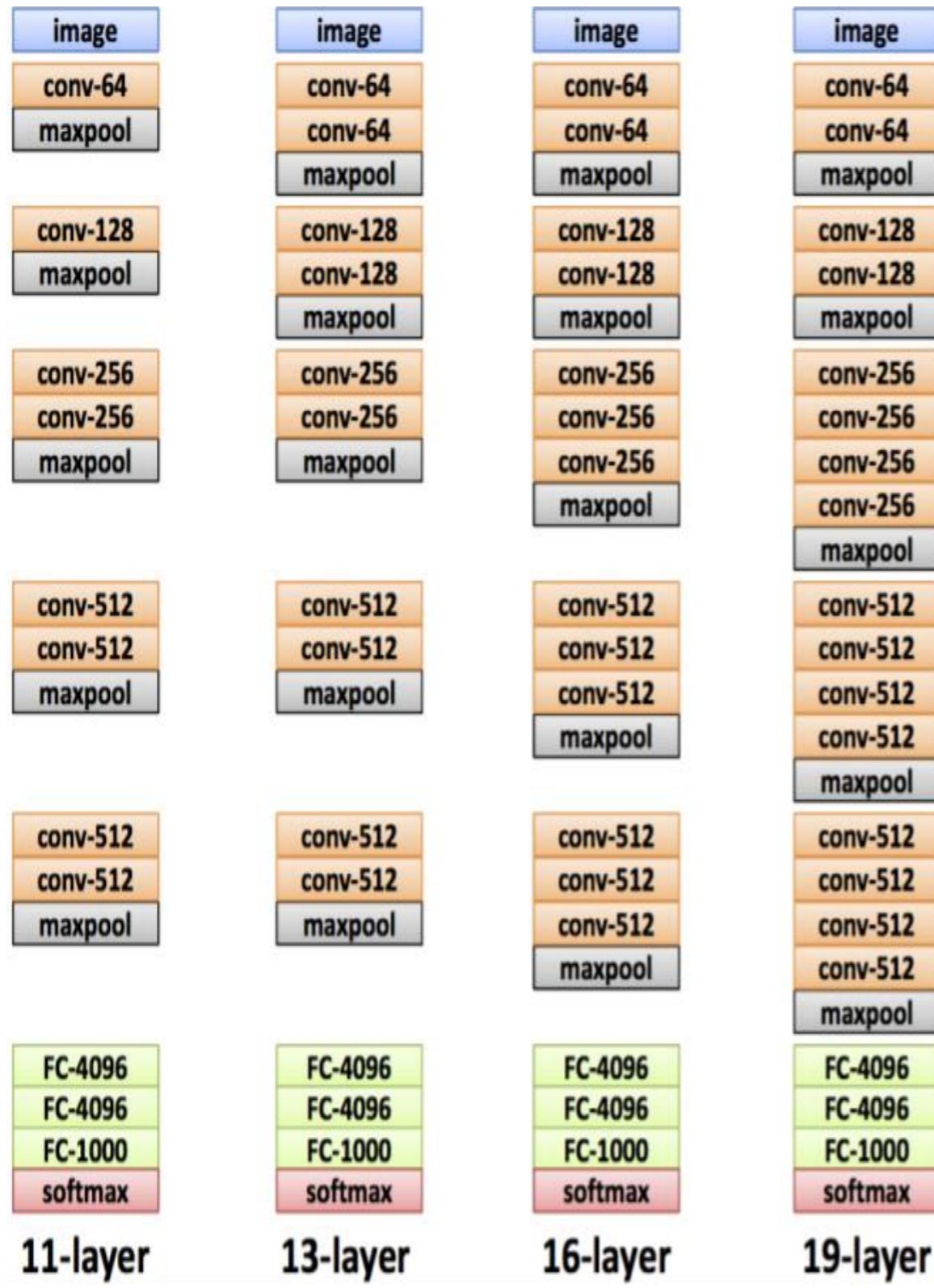


VGGNet



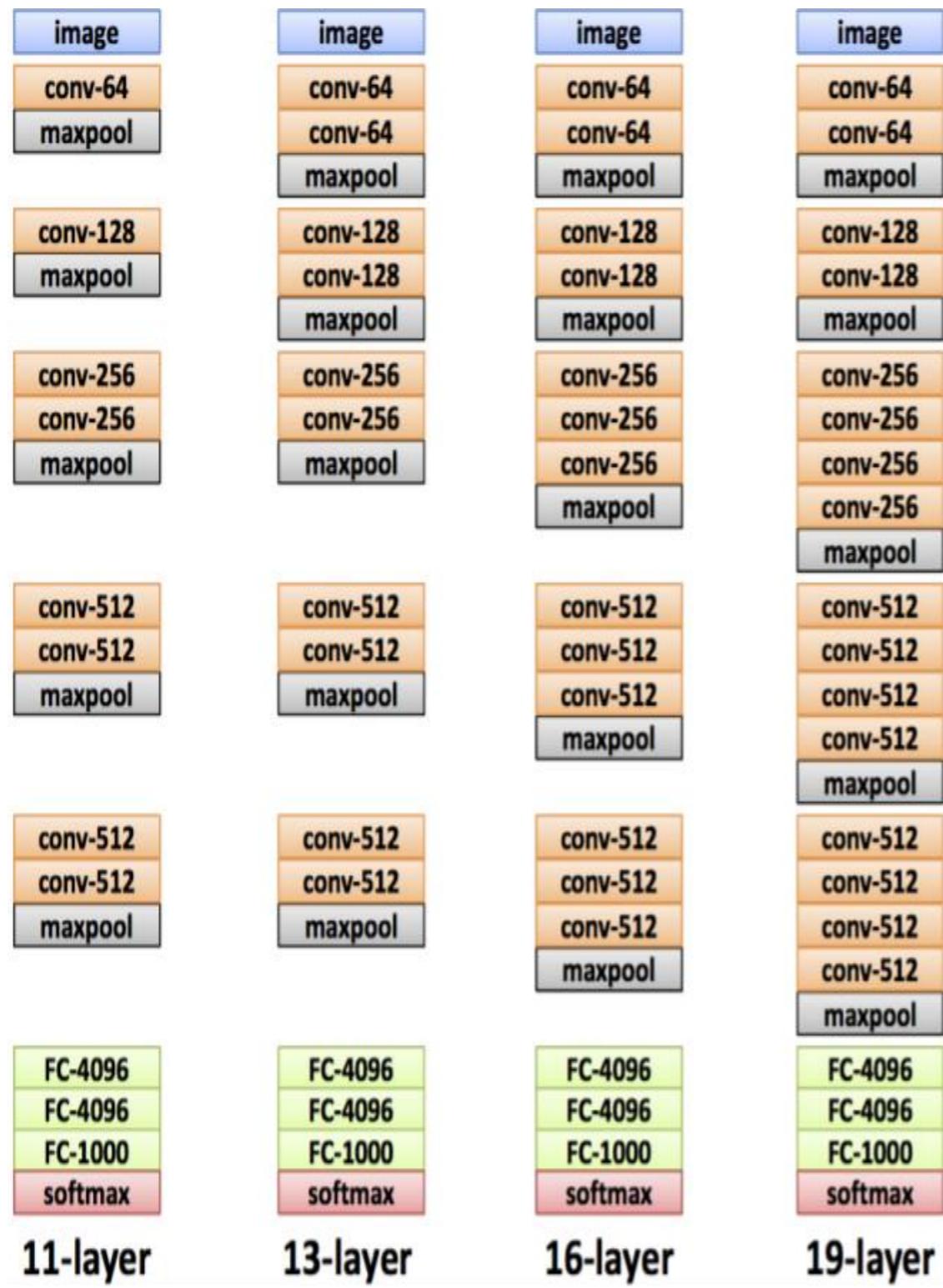
- Runner-up in ILSVRC 2014

VGGNet



- Runner-up in ILSVRC 2014
- More layers lead to more nonlinearities

VGGNet

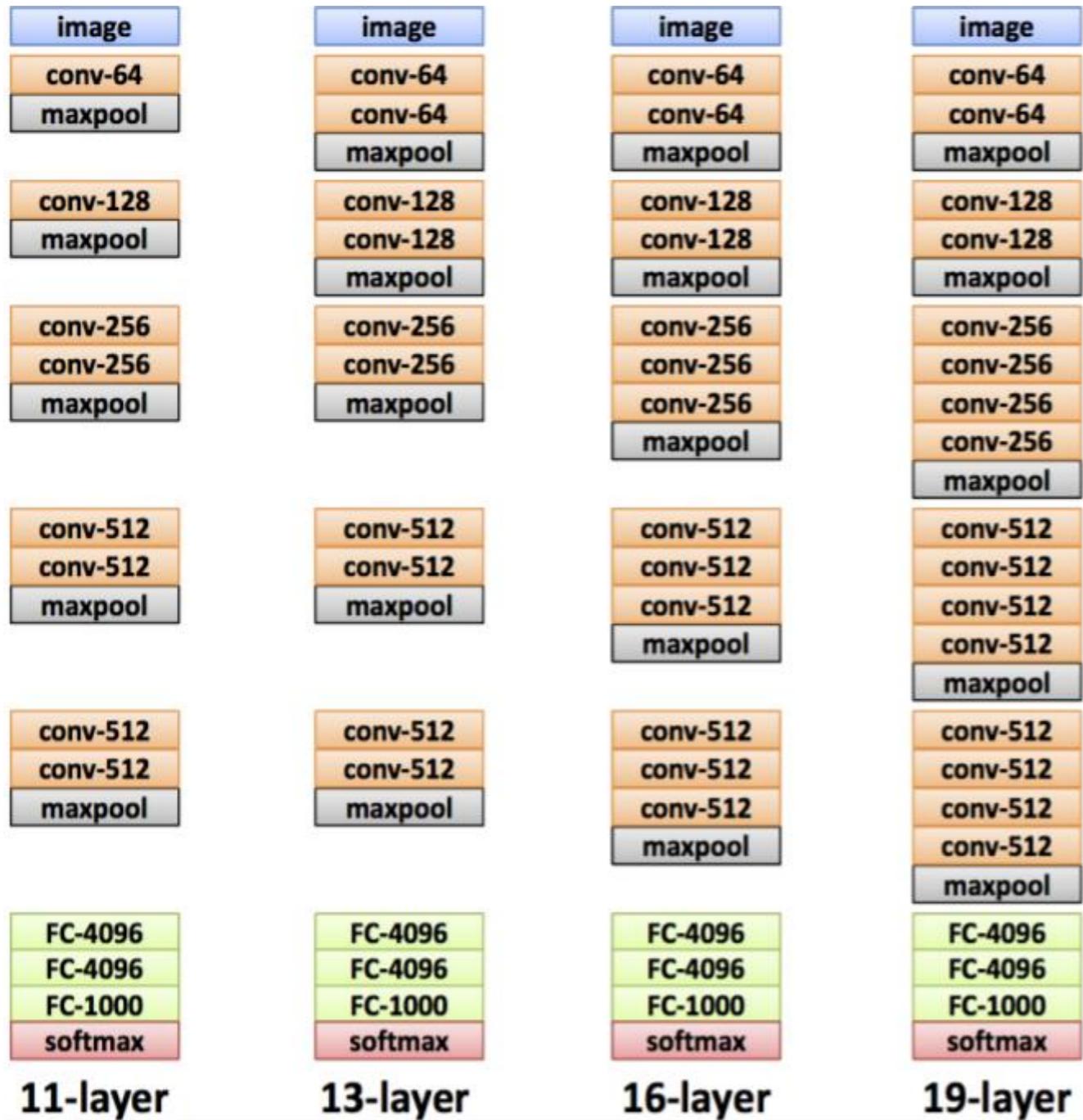


Runner-up in ILSVRC 2014

More layers lead to more nonlinearities

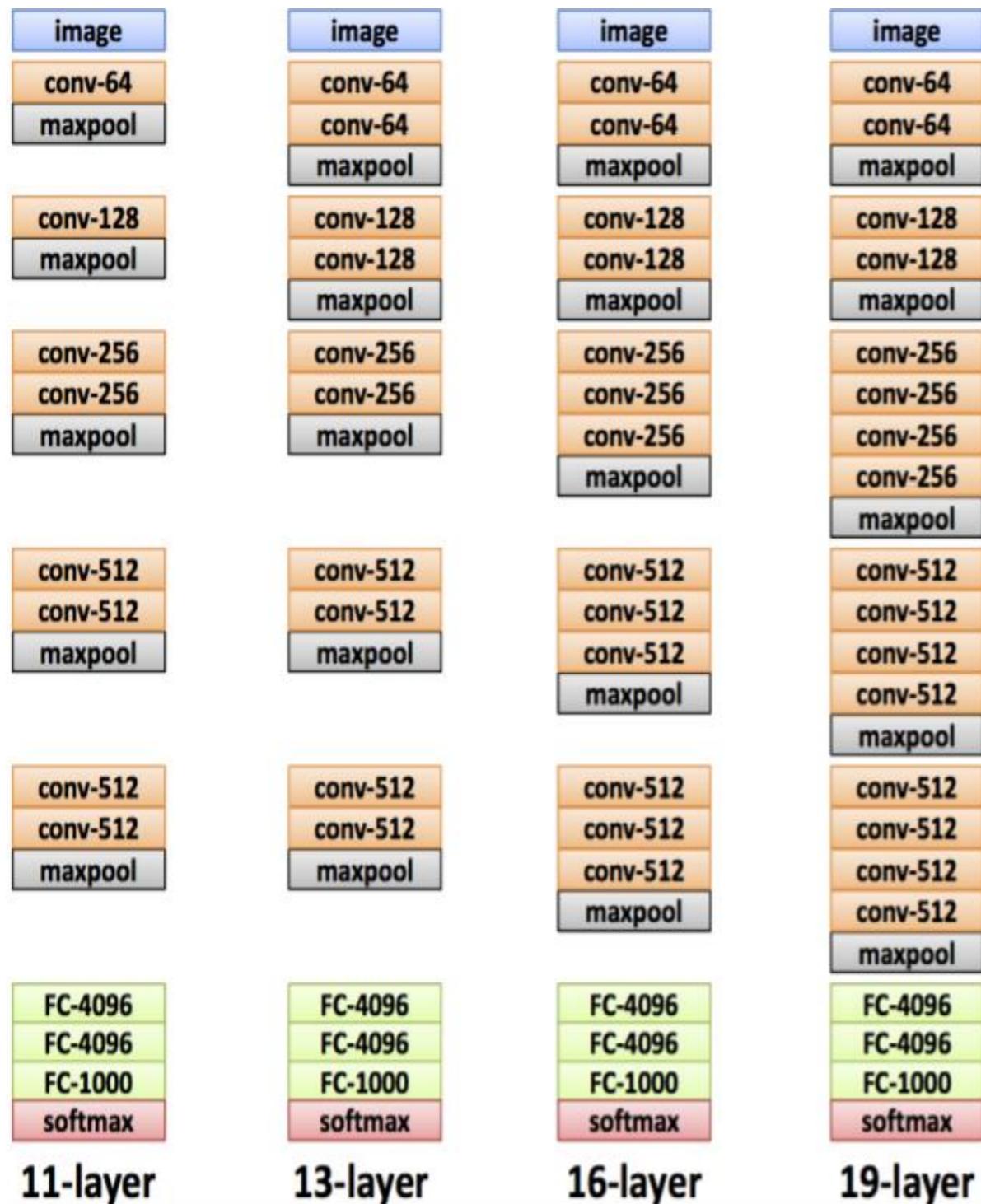
Key contribution: Depth of the network
is a critical component for good
performance

VGGNet



- Runner-up in ILSVRC 2014
- More layers lead to more nonlinearities
- **Key contribution:** Depth of the network is a critical component for good performance
- **Homogeneous Architecture:** From beginning to end:
 - 3 x 3 CONV stride 1 pad 1
 - 2 x 2 MAX POOL stride 2
- **Smaller receptive fields:**
 - less parameters; faster
 - two 3 x 3 conv has same receptive field as a single 5 x 5 conv; three 3 x 3 conv has same receptive field as a single 7 x 7 conv
 - Fewer parameters: $3 \times 3^2 C^2$ (vs) $7^2 C^2$

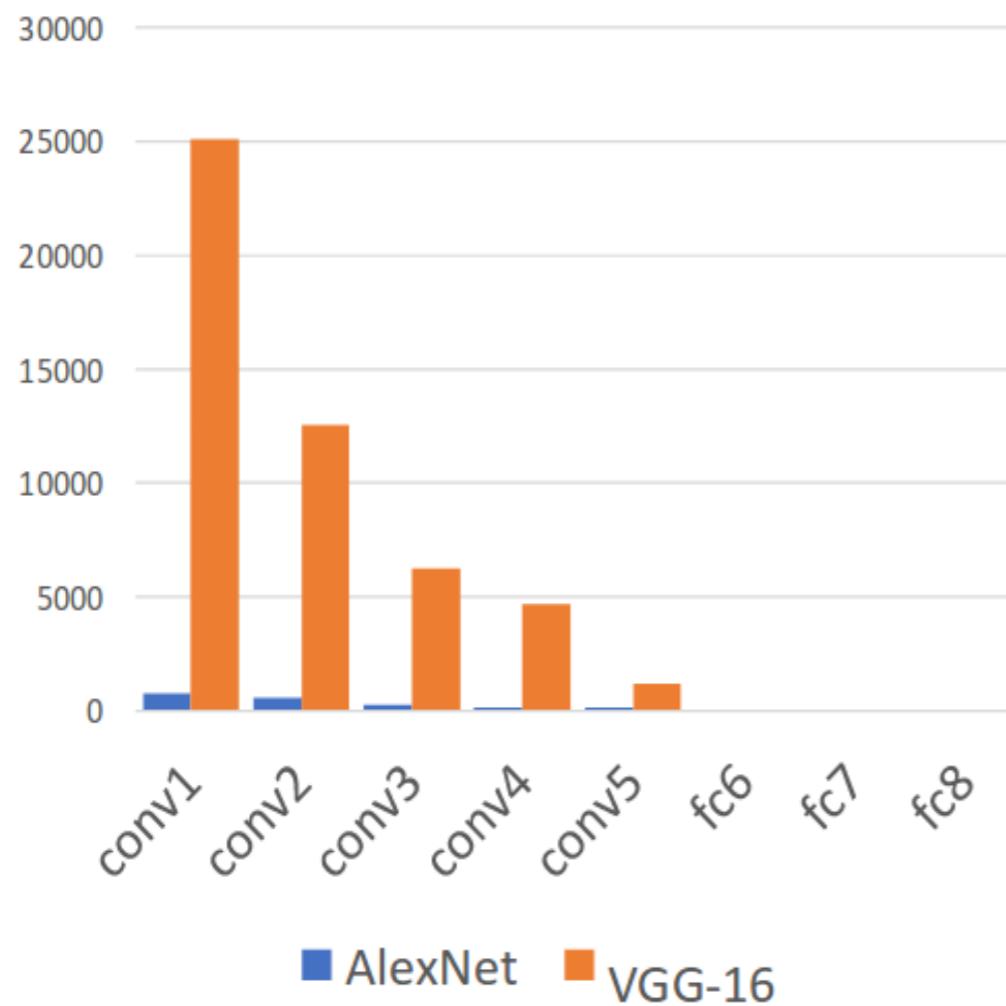
VGGNet



- VGG19 only slightly better than VGG16
- Used ensembles of networks for best results

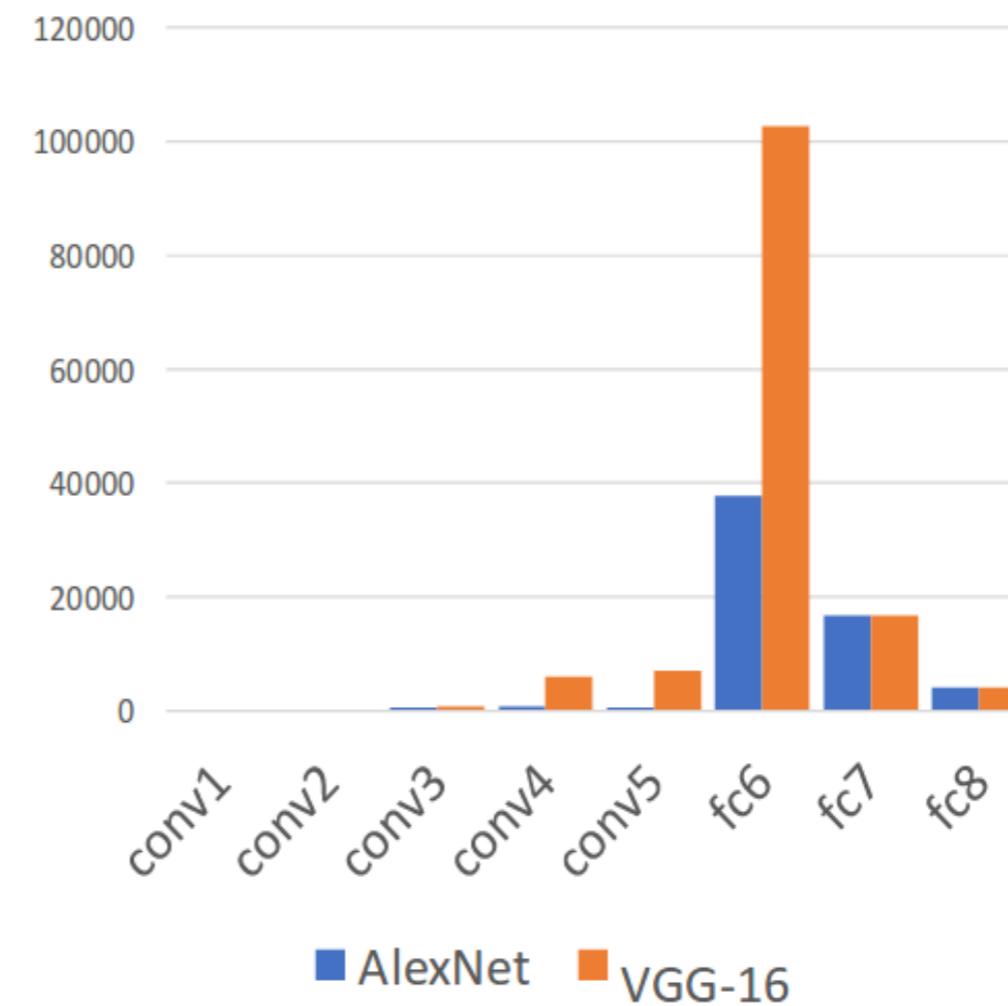
VGGNet

AlexNet vs VGG-16
(Memory, KB)



AlexNet total: 1.9 MB
VGG-16 total: 48.6 MB (25x)

AlexNet vs VGG-16
(Params, M)

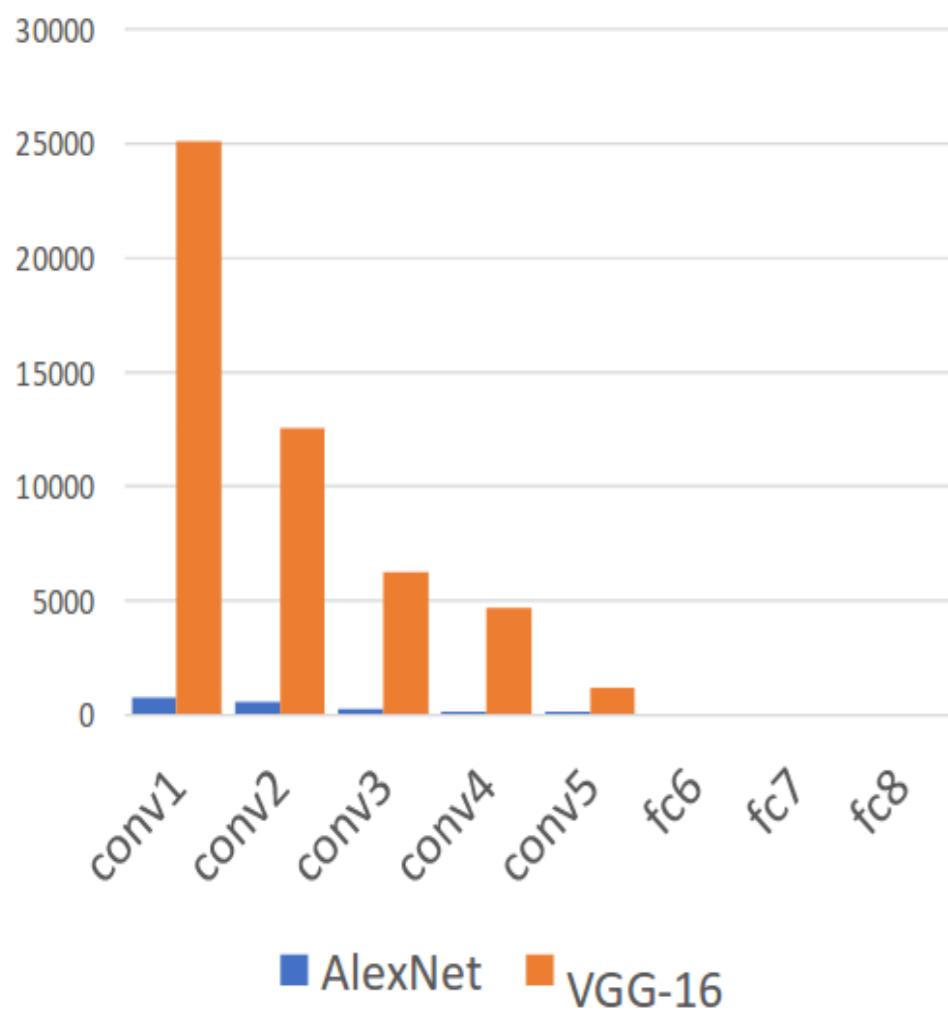


AlexNet total: 61M
VGG-16 total: 138M (2.3x)

Credit: Justin Johnson, Univ of Michigan

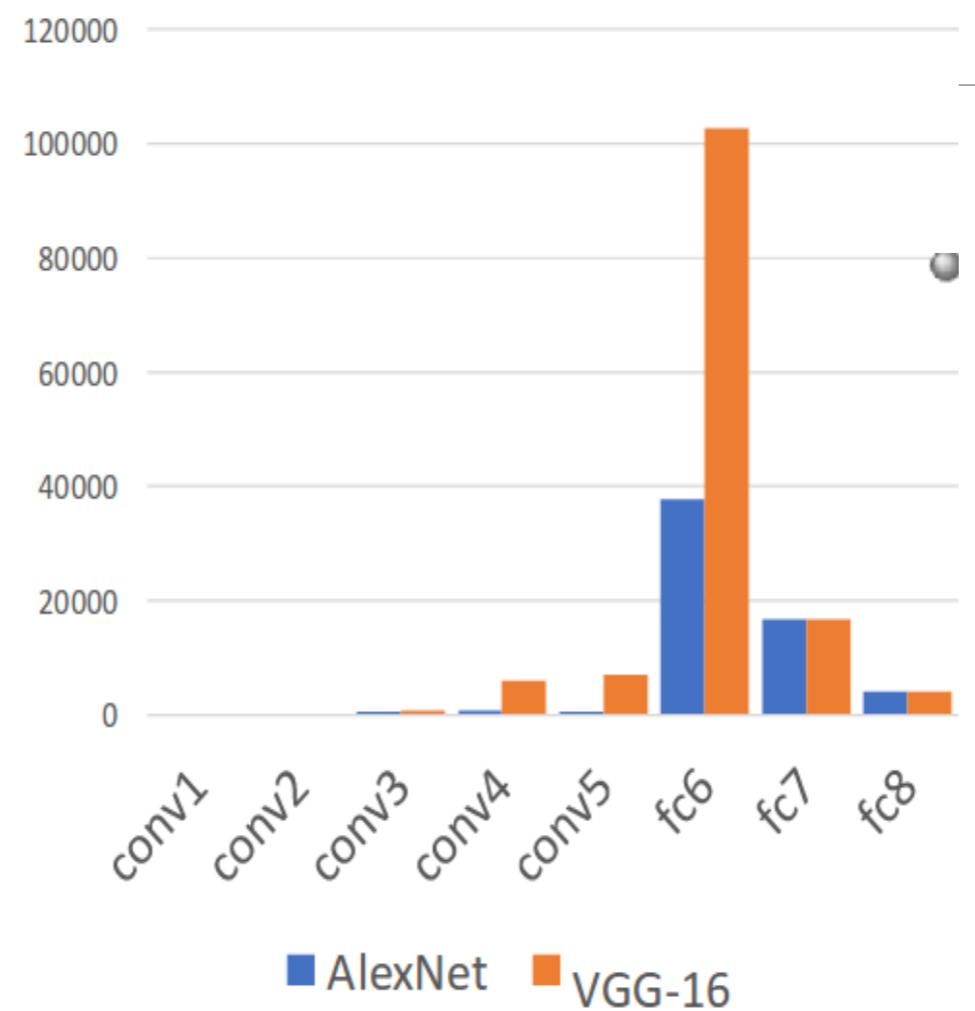
VGGNet

AlexNet vs VGG-16
(Memory, KB)



AlexNet total: 1.9 MB
VGG-16 total: 48.6 MB (25x)

AlexNet vs VGG-16
(Params, M)



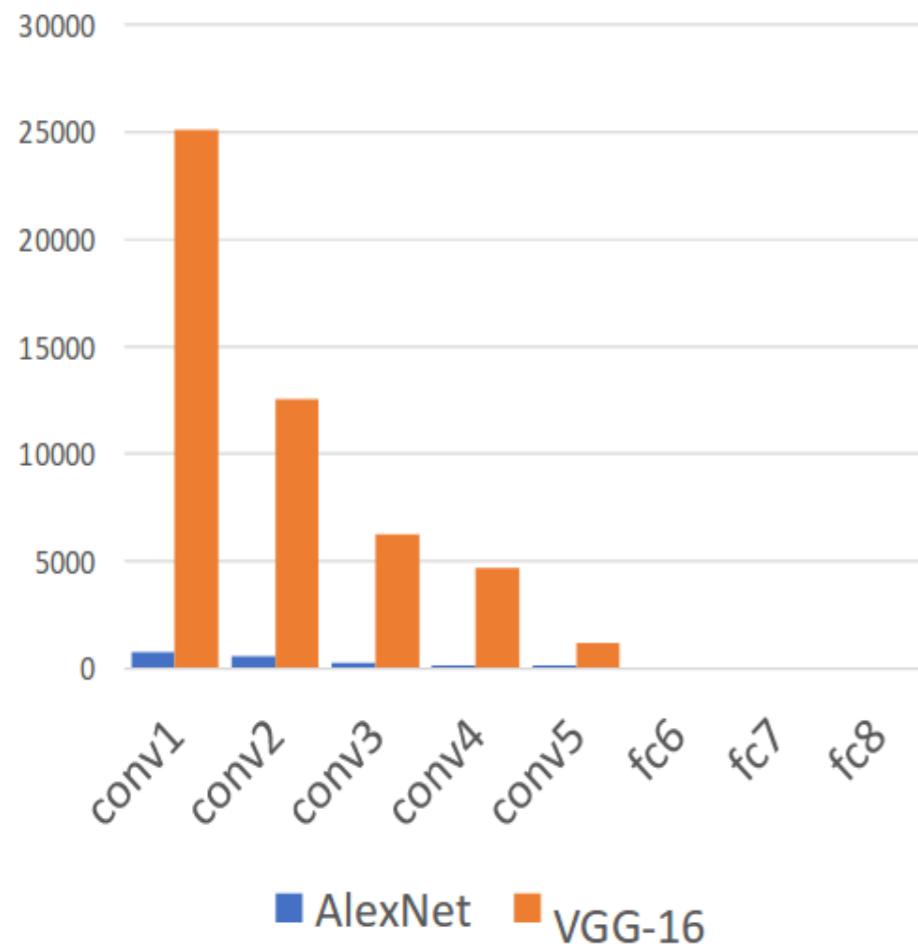
AlexNet total: 61M
VGG-16 total: 138M (2.3x)

- Uses a lot more memory and parameters

Credit: Justin Johnson, Univ of Michigan

VGGNet

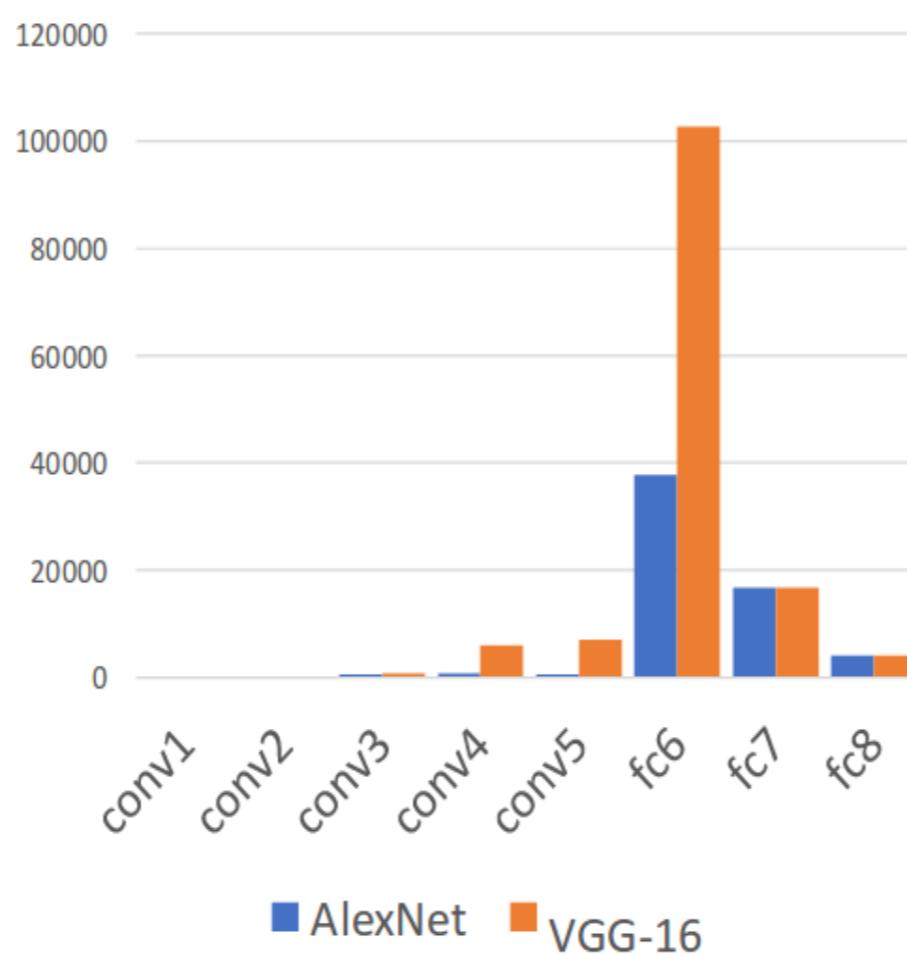
AlexNet vs VGG-16
(Memory, KB)



AlexNet total: 1.9 MB
VGG-16 total: 48.6 MB (25x)

Credit: Justin Johnson, Univ of Michigan

AlexNet vs VGG-16
(Params, M)

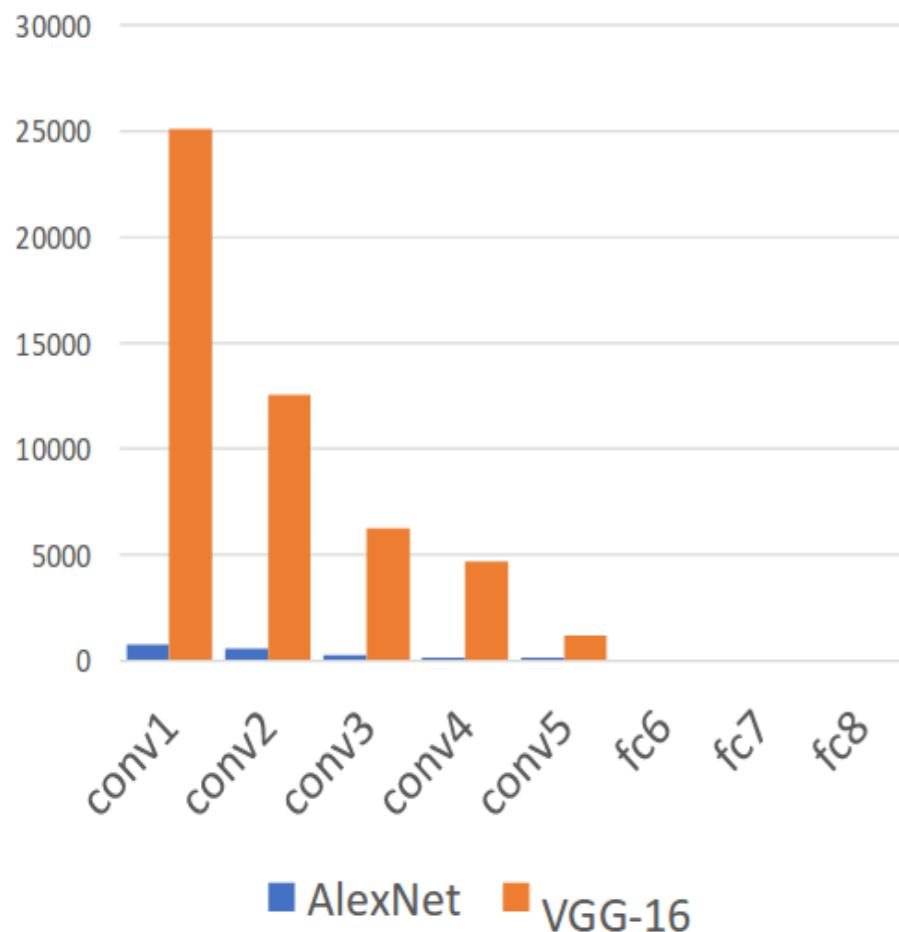


AlexNet total: 61M
VGG-16 total: 138M (2.3x)

- Uses a lot more memory and parameters
- Most of these parameters are in the first fully connected layer

VGGNet

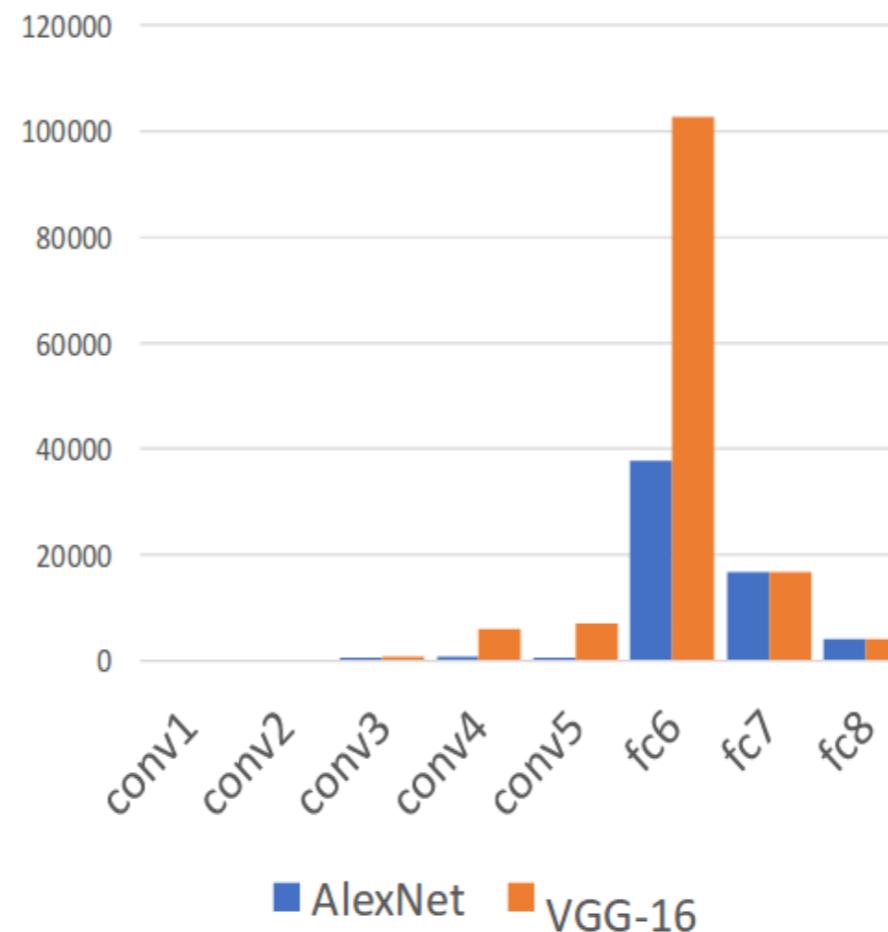
AlexNet vs VGG-16
(Memory, KB)



AlexNet total: 1.9 MB

VGG-16 total: 48.6 MB (25x)

AlexNet vs VGG-16
(Params, M)



AlexNet total: 61M

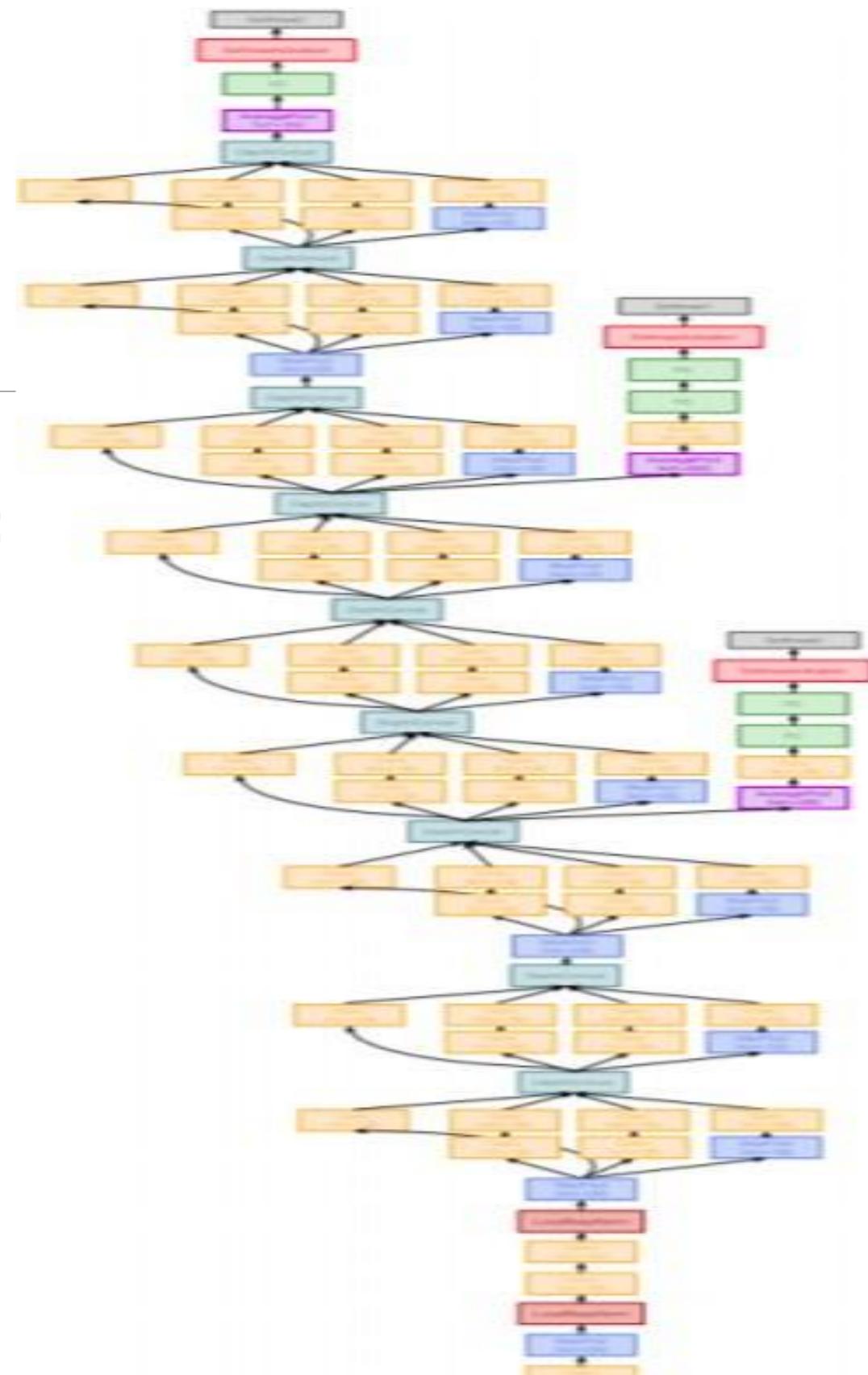
VGG-16 total: 138M (2.3x)

- Uses a lot more memory and parameters
- Most of these parameters are in the first fully connected layer
- Most of the memory is used in early CONV layer

Credit: Justin Johnson, Univ of Michigan

GoogleNet

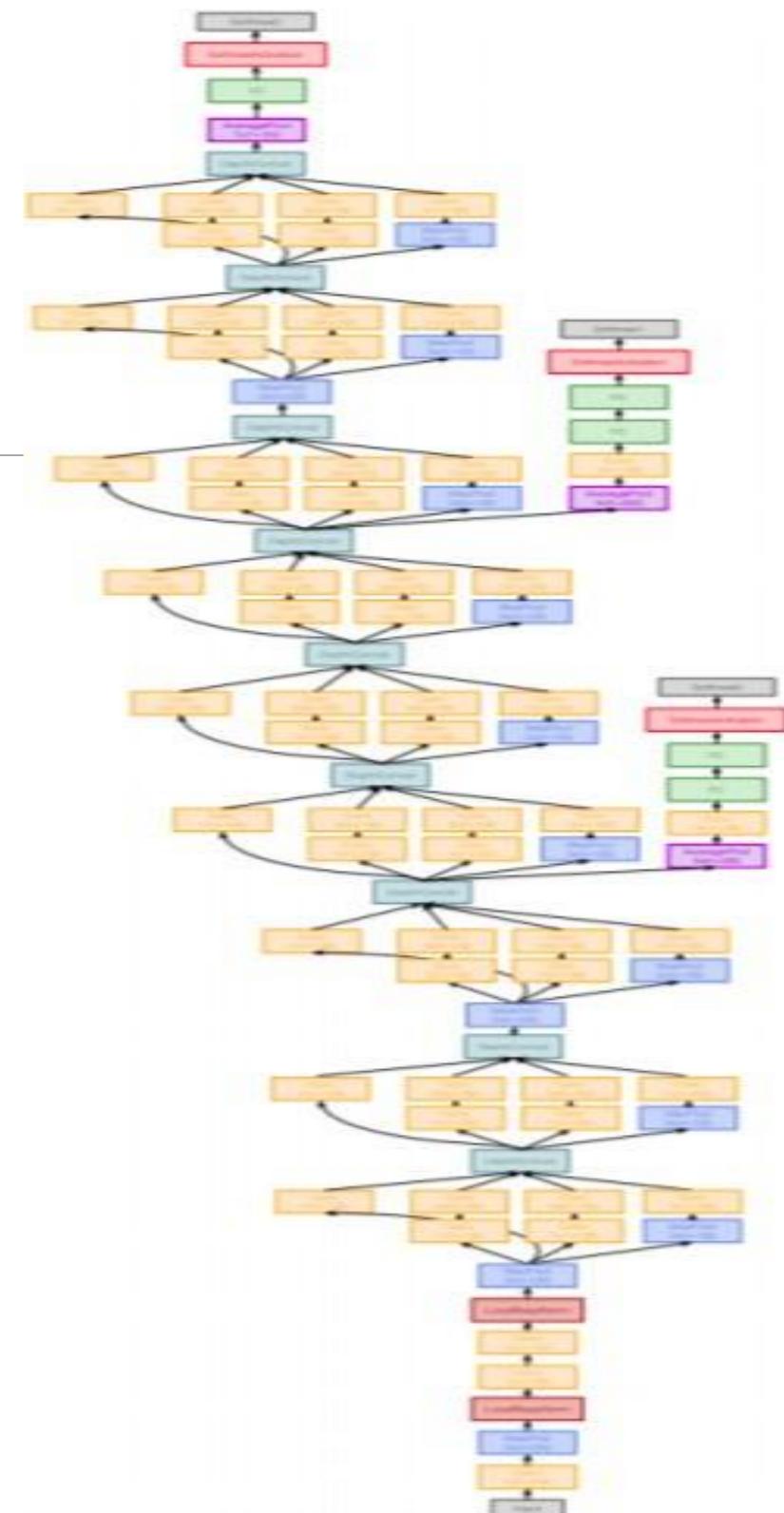
- Deeper networks with focus on efficiency:
reduce parameter count, memory usage, and
computation



Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford; Justin Johnson, Univ of Michigan

GoogleNet

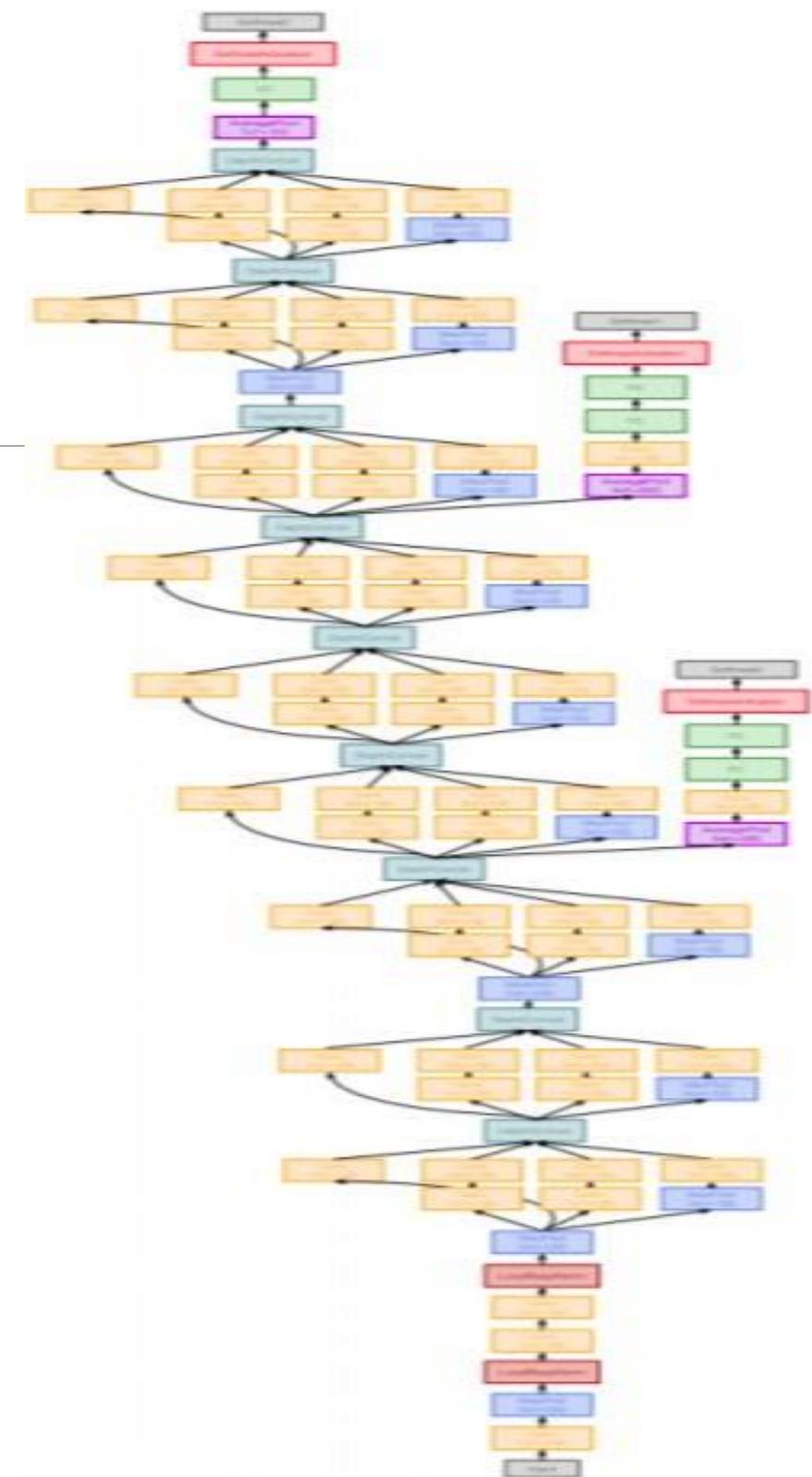
- Deeper networks with focus on efficiency:
reduce parameter count, memory usage, and
computation
- 22 layers



Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford; Justin Johnson, Univ of Michigan

GoogleNet

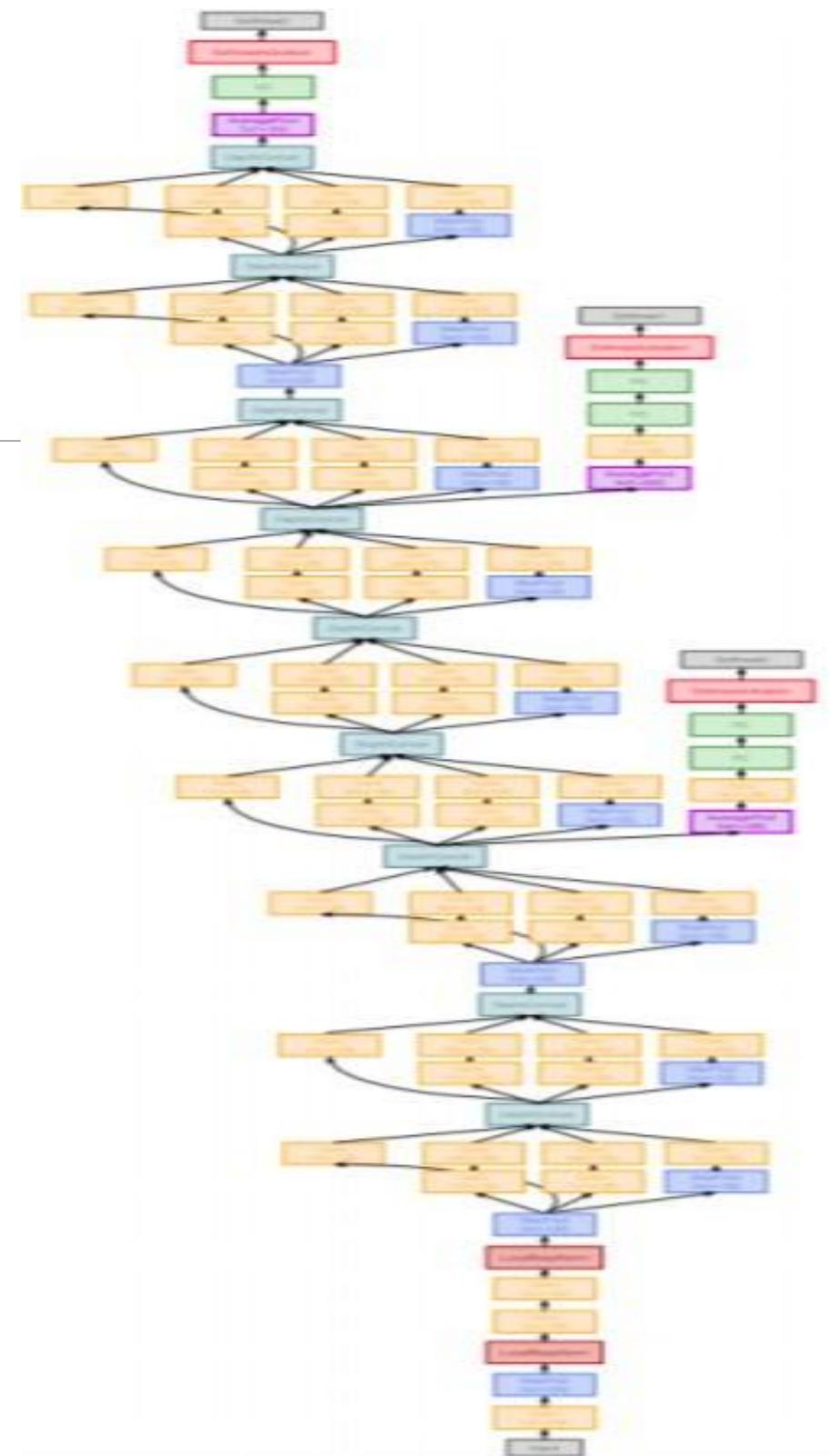
- Deeper networks with focus on efficiency: reduce parameter count, memory usage, and computation
- 22 layers
- No FC layers



Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford; Justin Johnson, Univ of Michigan

GoogleNet

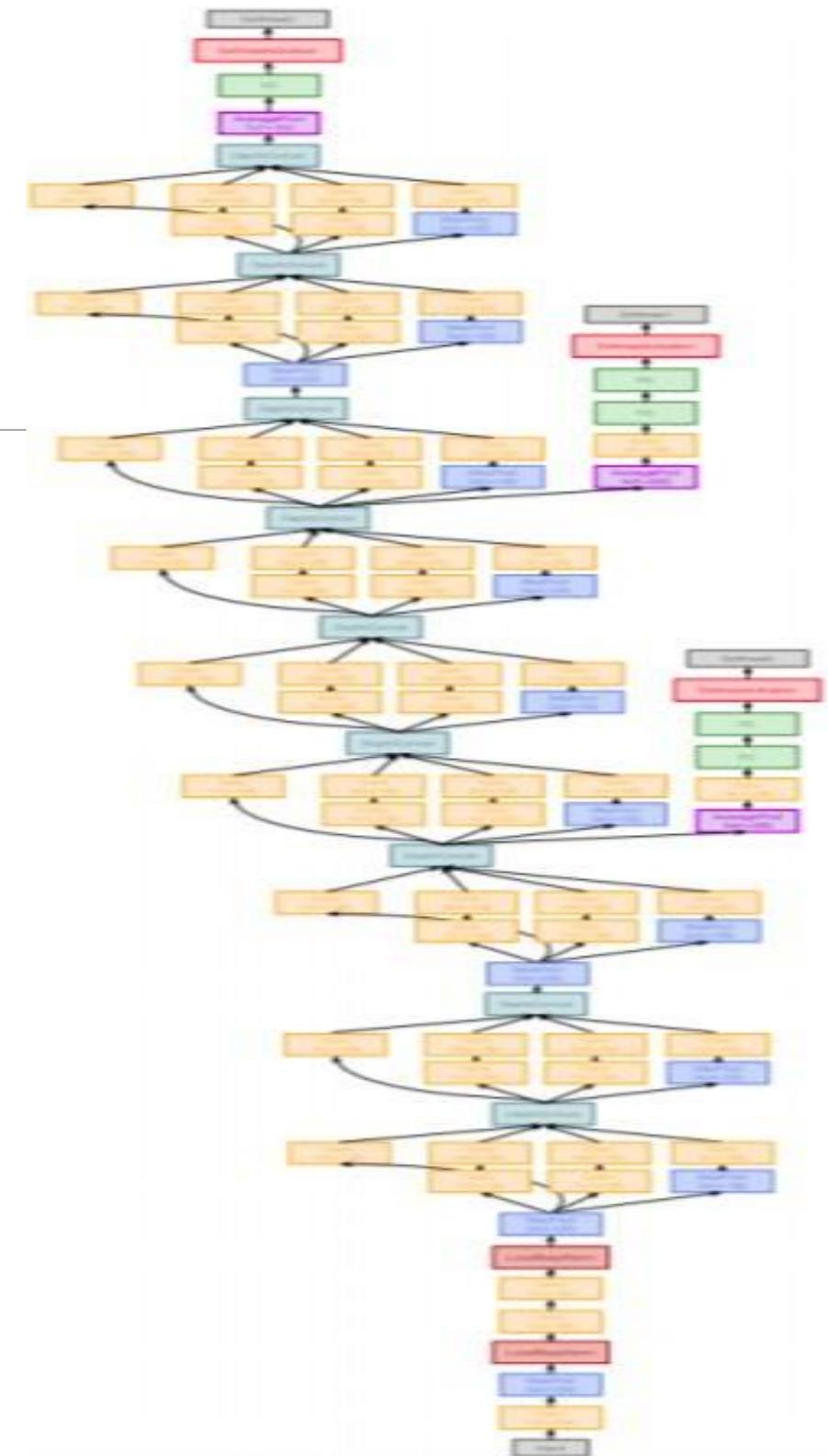
- Deeper networks with focus on efficiency:
reduce parameter count, memory usage, and
computation
- 22 layers
- No FC layers
- Efficient “Inception” module



Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford; Justin Johnson, Univ of Michigan

GoogleNet

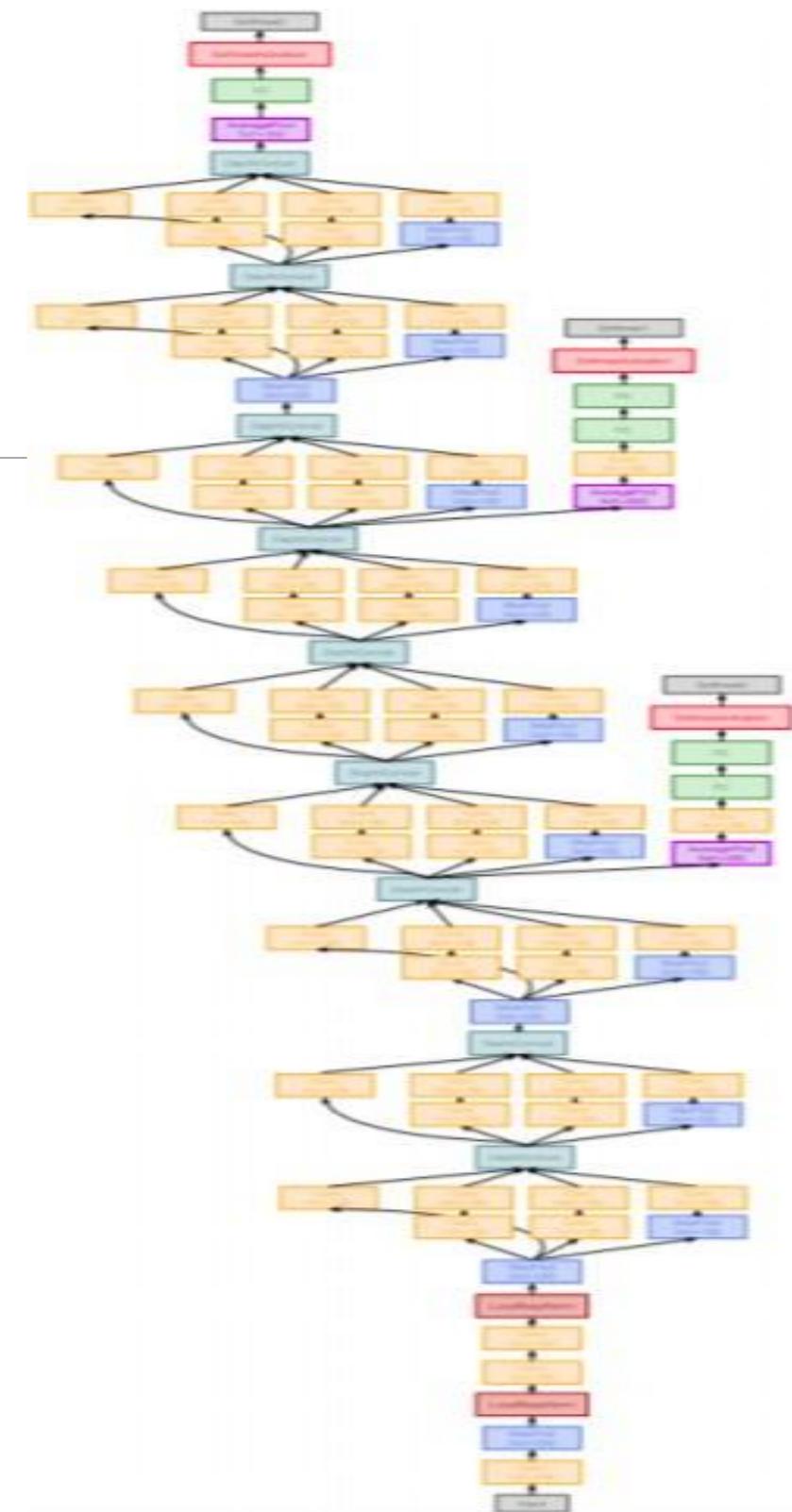
- Deeper networks with focus on efficiency: reduce parameter count, memory usage, and computation
- 22 layers
- No FC layers
- Efficient “Inception” module
- Only 5 million parameters! ([12x less than AlexNet](#))



Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford; Justin Johnson, Univ of Michigan

GoogleNet

- Deeper networks with focus on efficiency: reduce parameter count, memory usage, and computation
- 22 layers
- No FC layers
- Efficient “Inception” module
- Only 5 million parameters! ([12x less than AlexNet](#))
- ILSVRC’14 classification winner** (6.7% top-5 error)



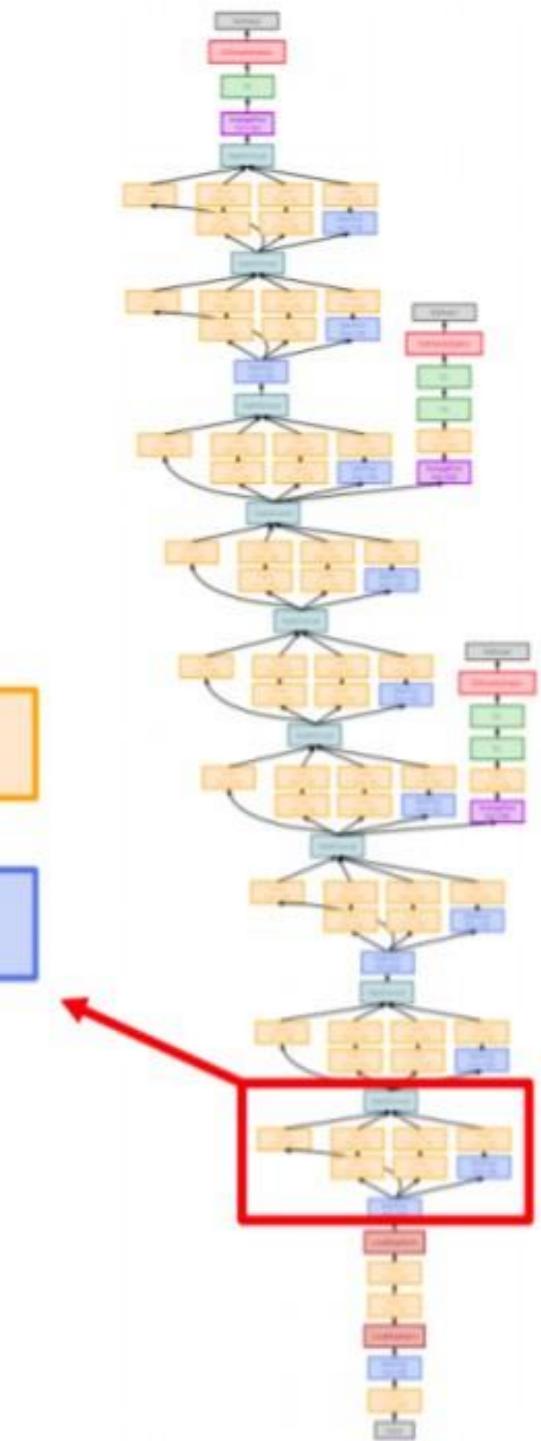
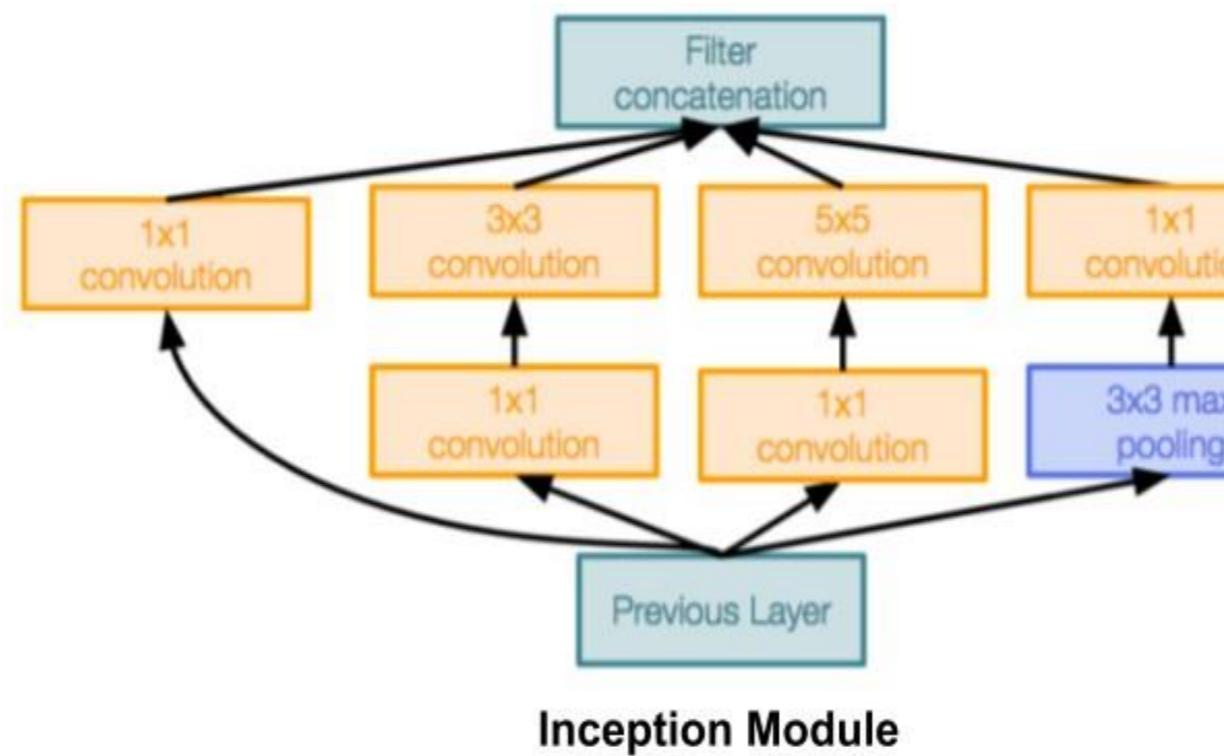
Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford; Justin Johnson, Univ of Michigan

GoogleNet

- **Inception module:**

- Local unit with parallel branches

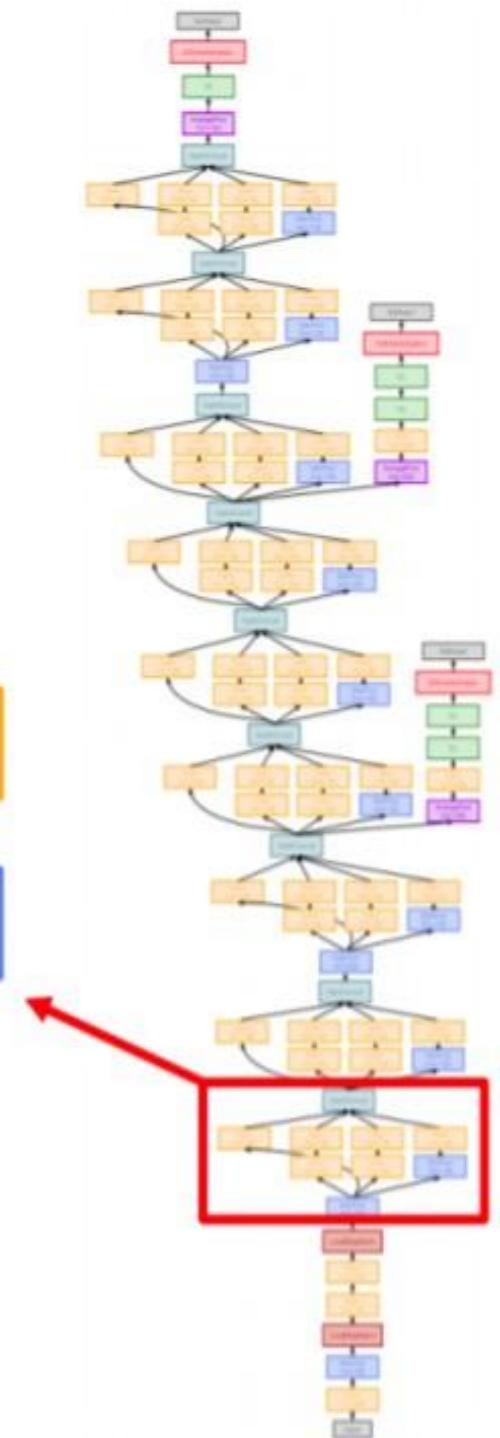
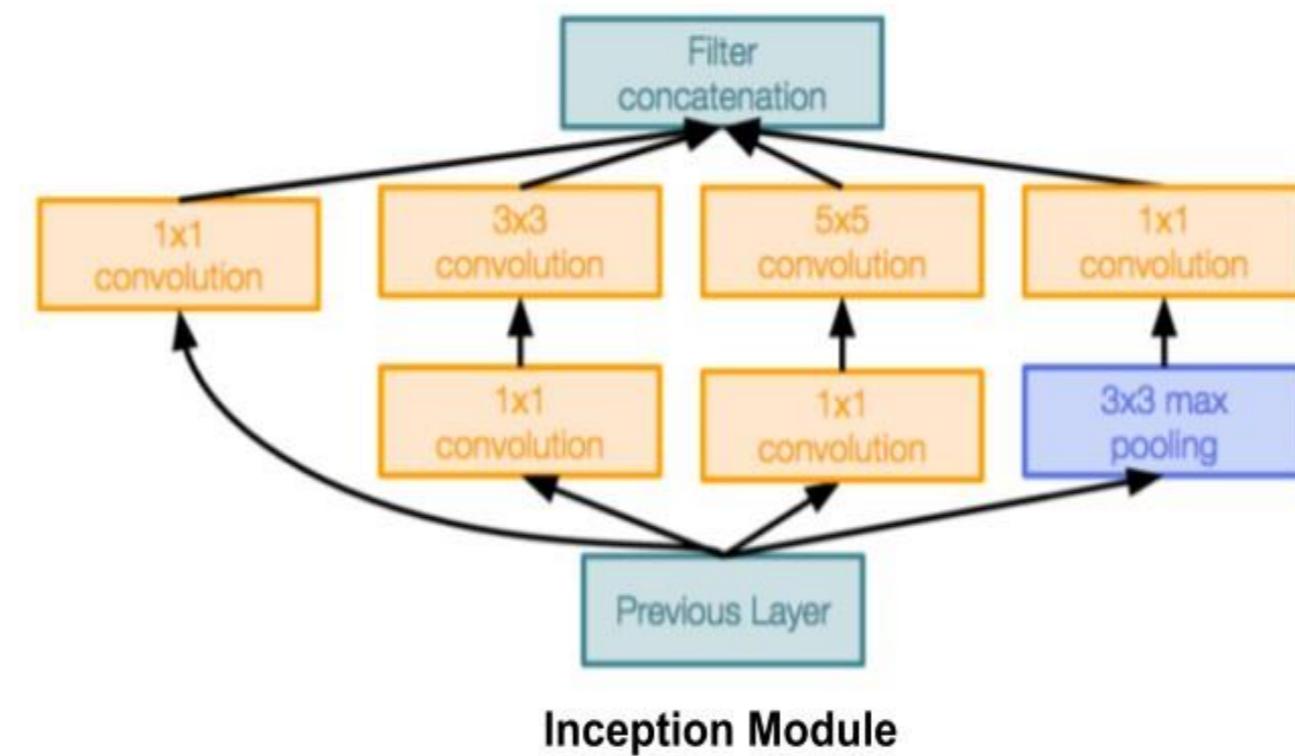
- Local structure repeated many times throughout the network



Credit: Justin Johnson, Univ of Michigan

GoogleNet

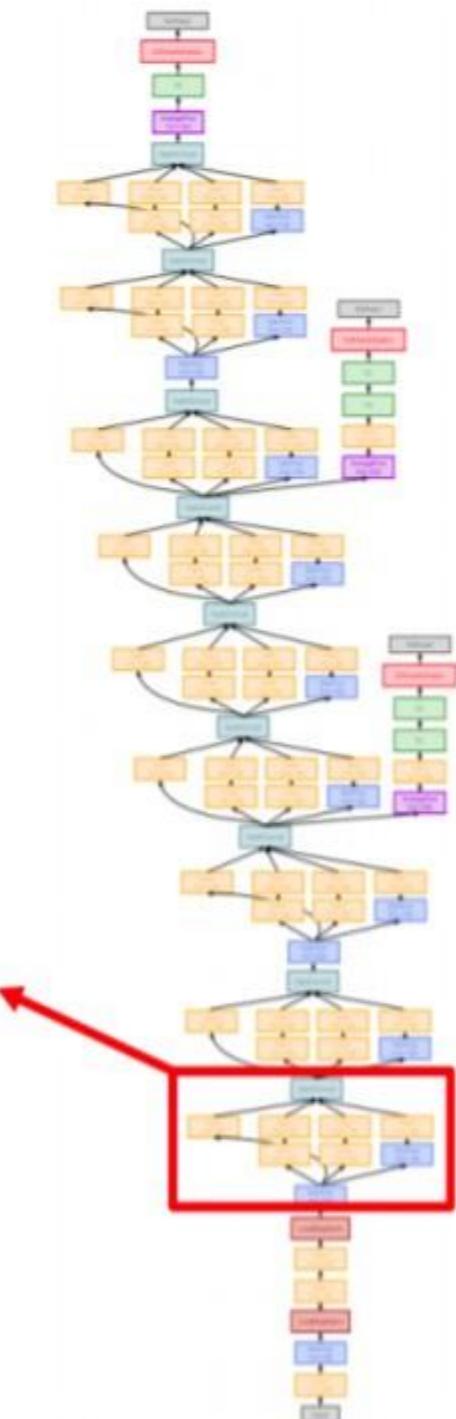
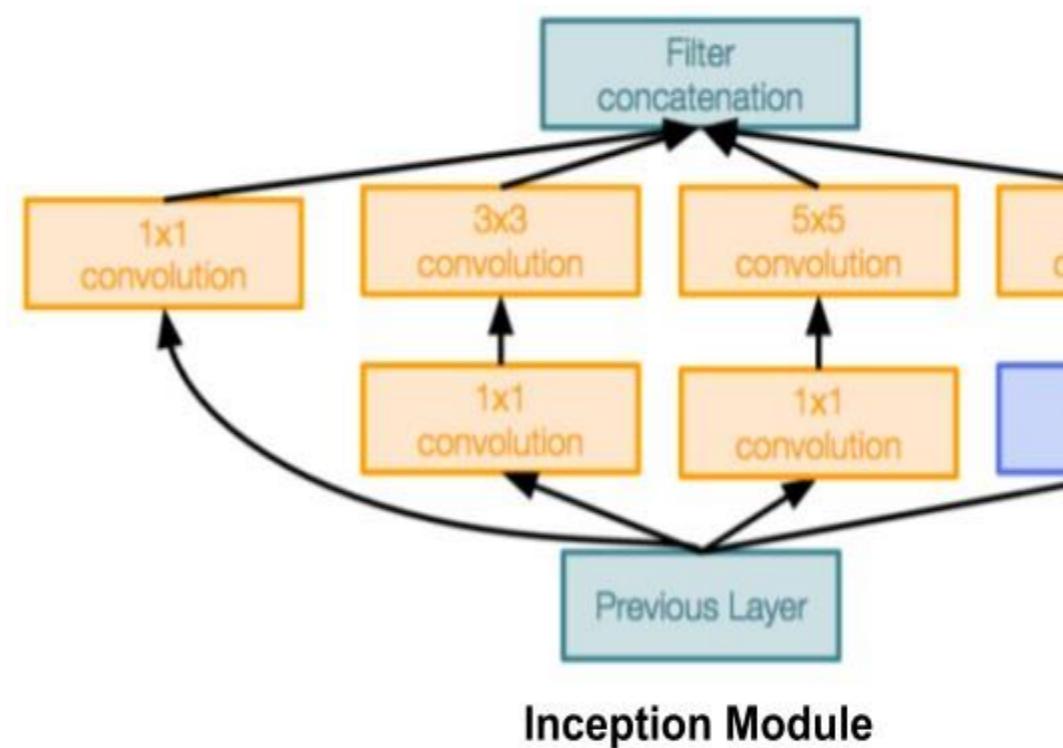
- **Inception module:**
Local unit with parallel branches
- Local structure repeated many times throughout the network



Credit: Justin Johnson, Univ of Michigan

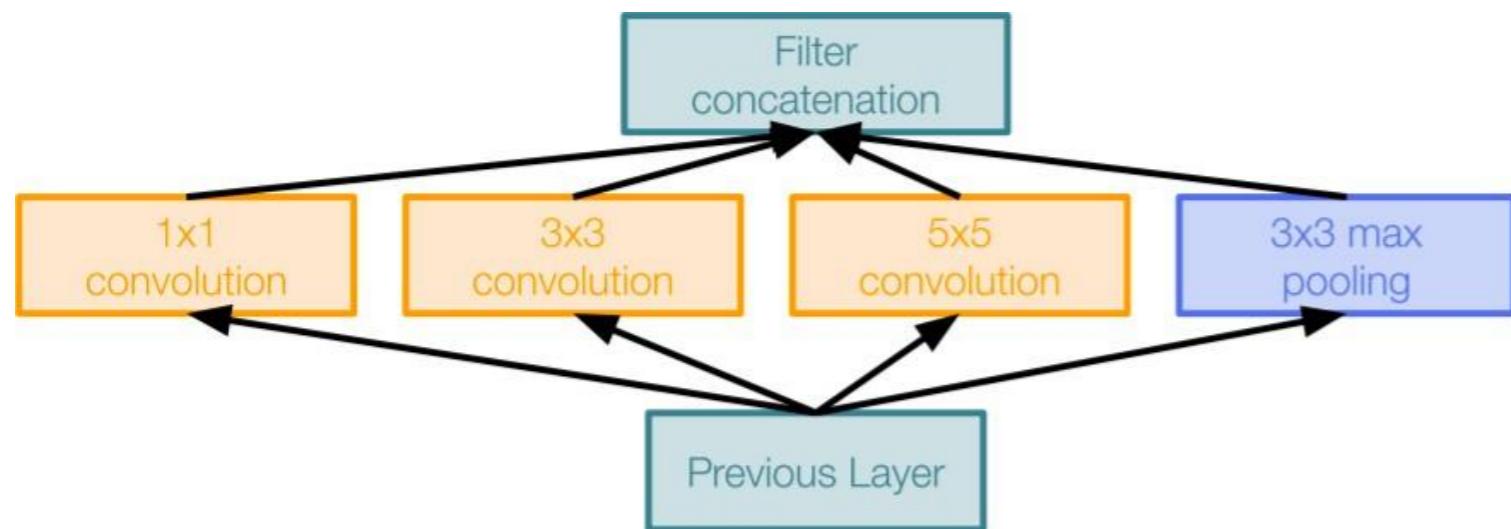
GoogleNet

- **Inception module:**
Local unit with parallel branches
- Local structure repeated many times throughout the network



Credit: Justin Johnson, Univ of Michigan

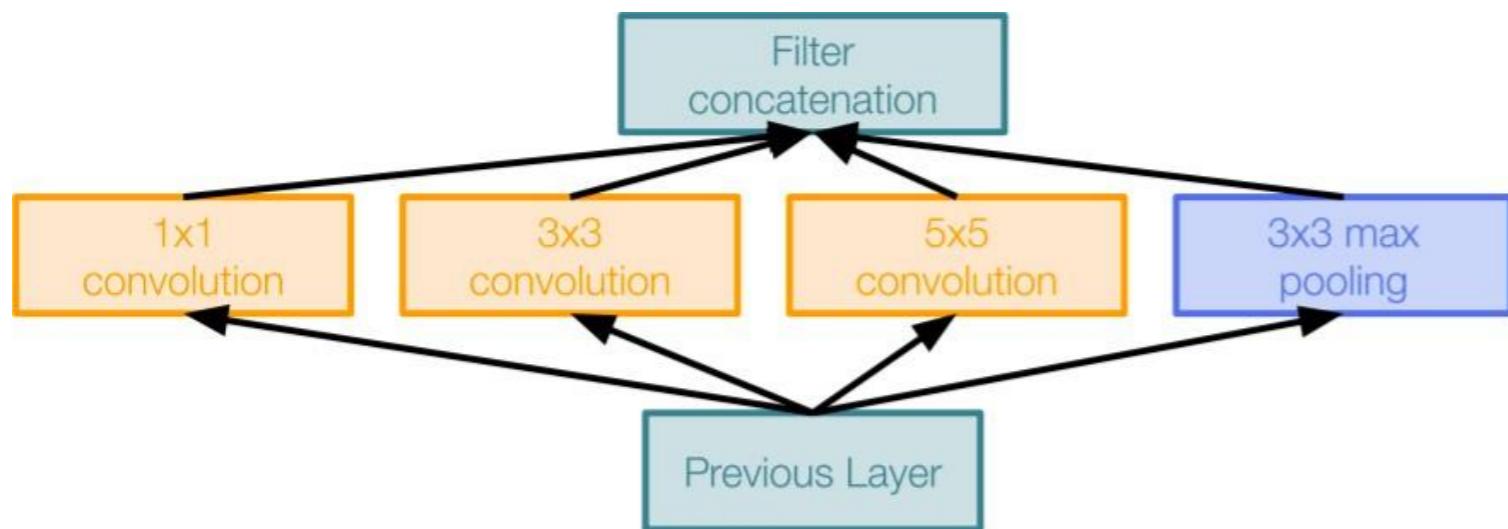
GoogleNet



Naive Inception module

- Apply parallel filter operations on the input from previous layer:
 - Multiple receptive field sizes for convolution (1×1 , 3×3 , 5×5)
 - Pooling operation (3×3 max pooling)
- Concatenate all filter outputs together depth-wise

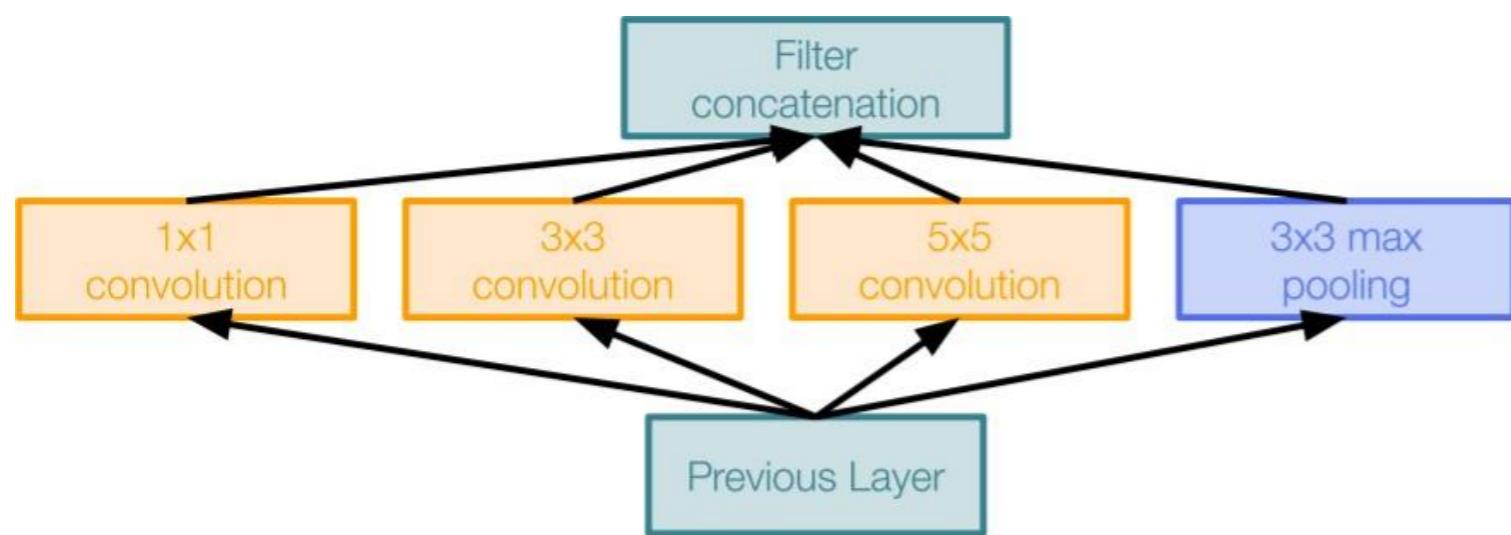
GoogleNet



Naive Inception module

- Apply parallel filter operations on the input from previous layer:
 - Multiple receptive field sizes for convolution (1×1 , 3×3 , 5×5)
 - Pooling operation (3×3 max pooling)
- Concatenate all filter outputs together depth-wise
- **What's the problem with this?**

GoogleNet



Naive Inception module

- Apply parallel filter operations on the input from previous layer:
 - Multiple receptive field sizes for convolution (1×1 , 3×3 , 5×5)
 - Pooling operation (3×3 max pooling)
- Concatenate all filter outputs together depth-wise
- **What's the problem with this?**
Computationally very expensive

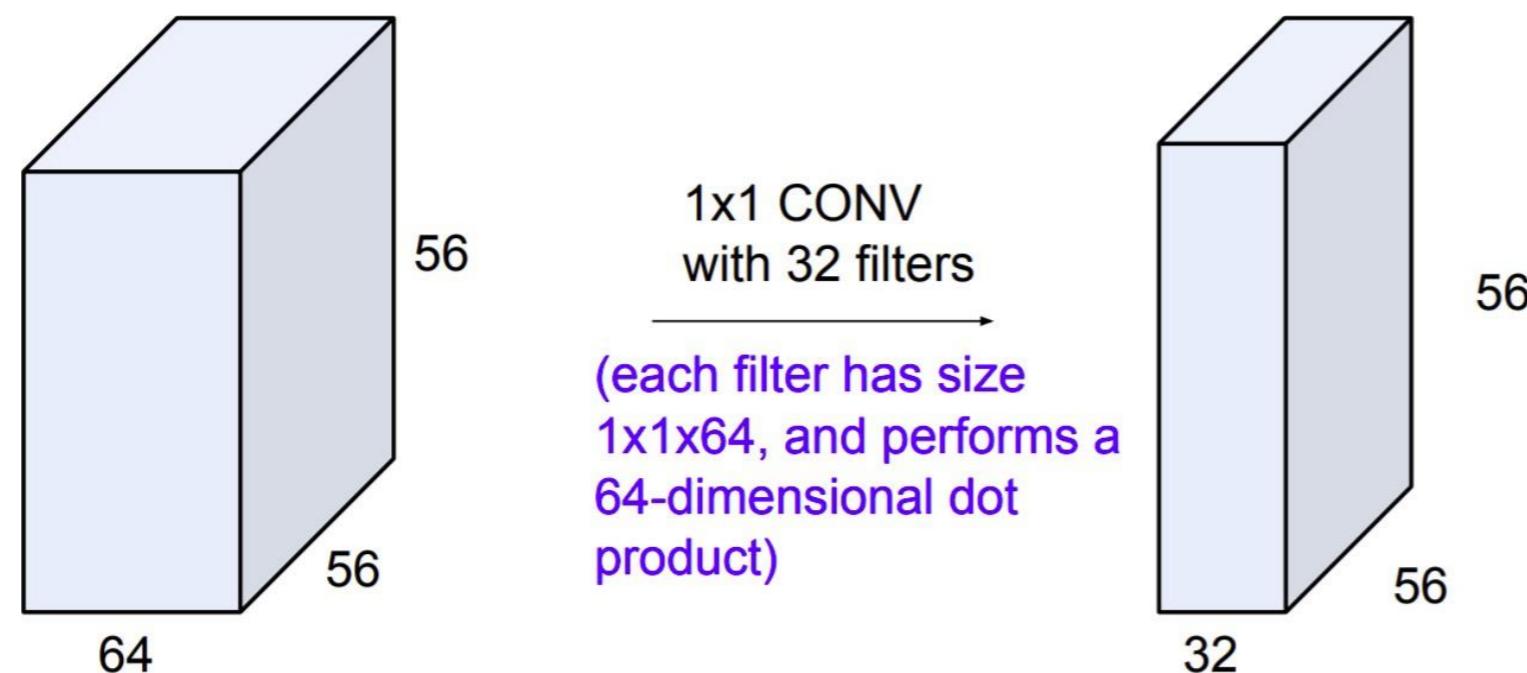
Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

GoogleNet

Solution: Use 1×1 “Bottleneck” layers to reduce channel dimension before expensive conv layers

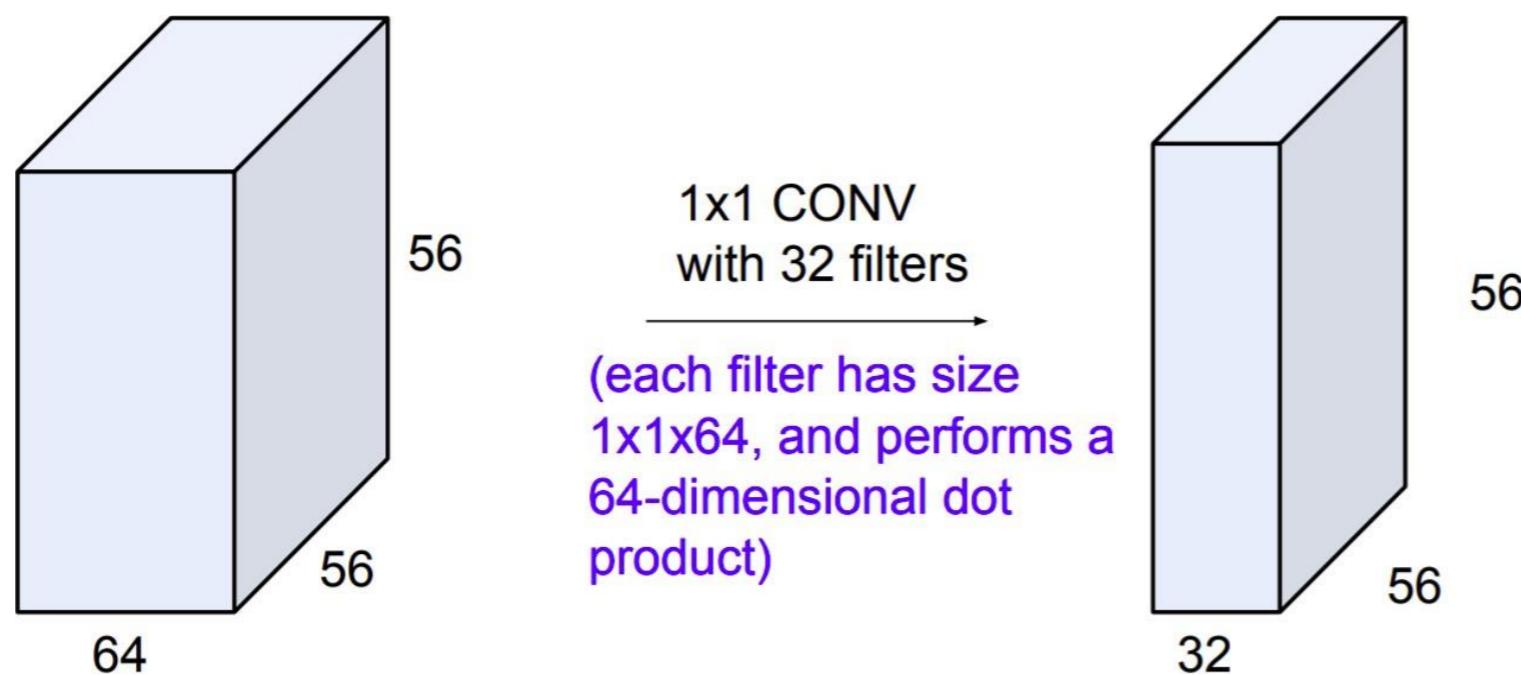
GoogleNet

Solution: Use 1×1 “Bottleneck” layers to reduce channel dimension before expensive conv layers



GoogleNet

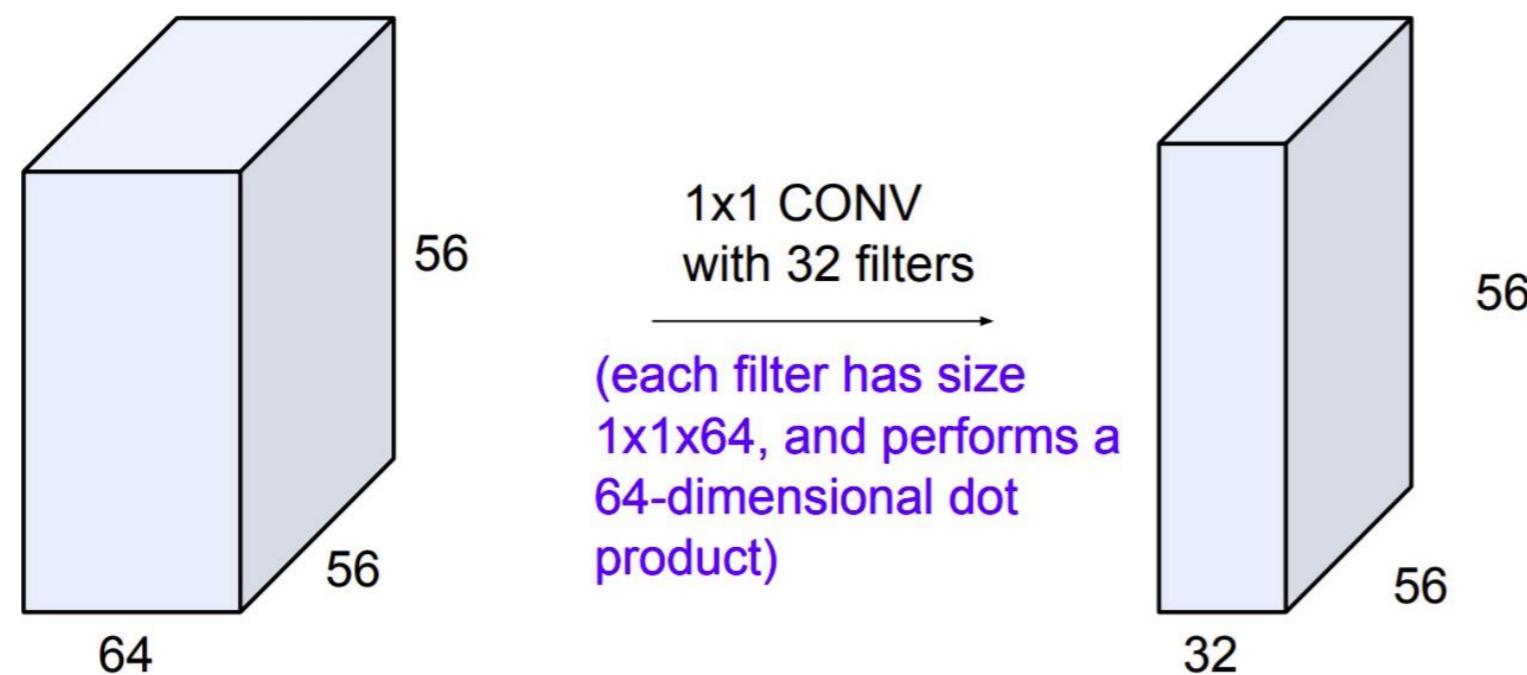
Solution: Use 1×1 “Bottleneck” layers to reduce channel dimension before expensive conv layers



- Preserves spatial dimensions, reduces depth!

GoogleNet

Solution: Use 1×1 “Bottleneck” layers to reduce channel dimension before expensive conv layers

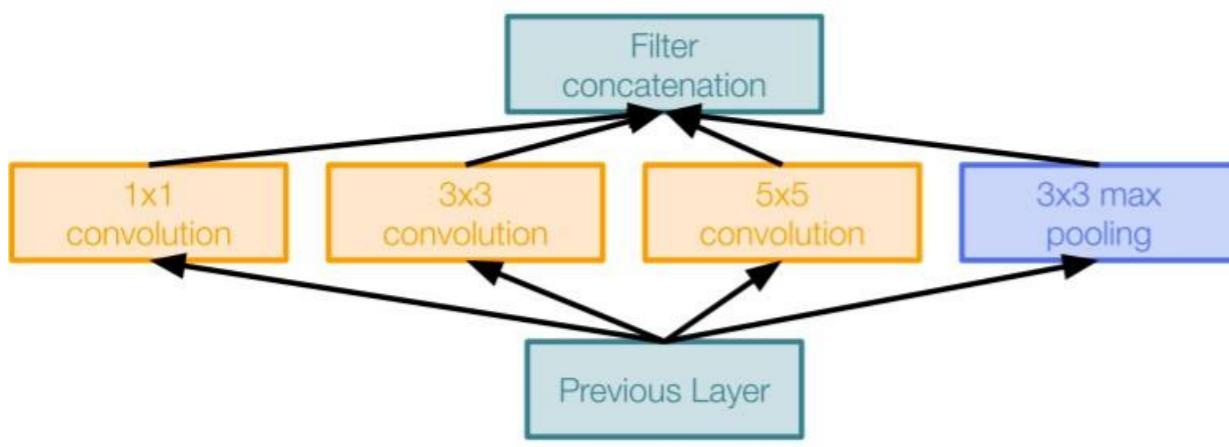


- Preserves spatial dimensions, reduces depth!
- Projects depth to lower dimension (combination of feature maps)

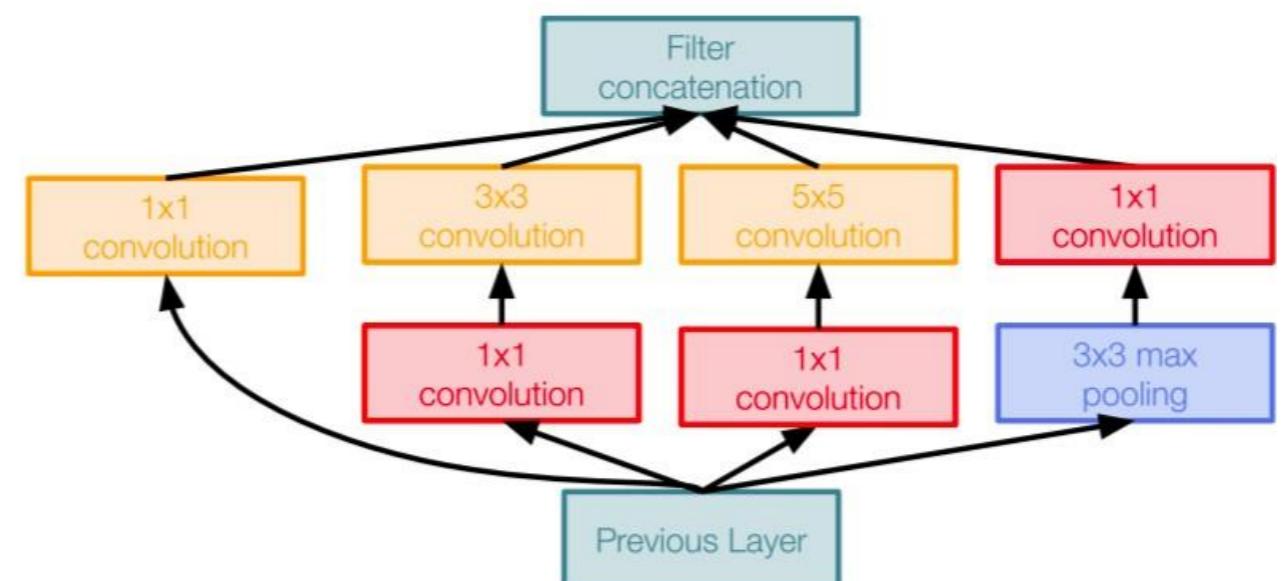
Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

GoogleNet

1x1 conv “bottleneck”
layers



Naive Inception module

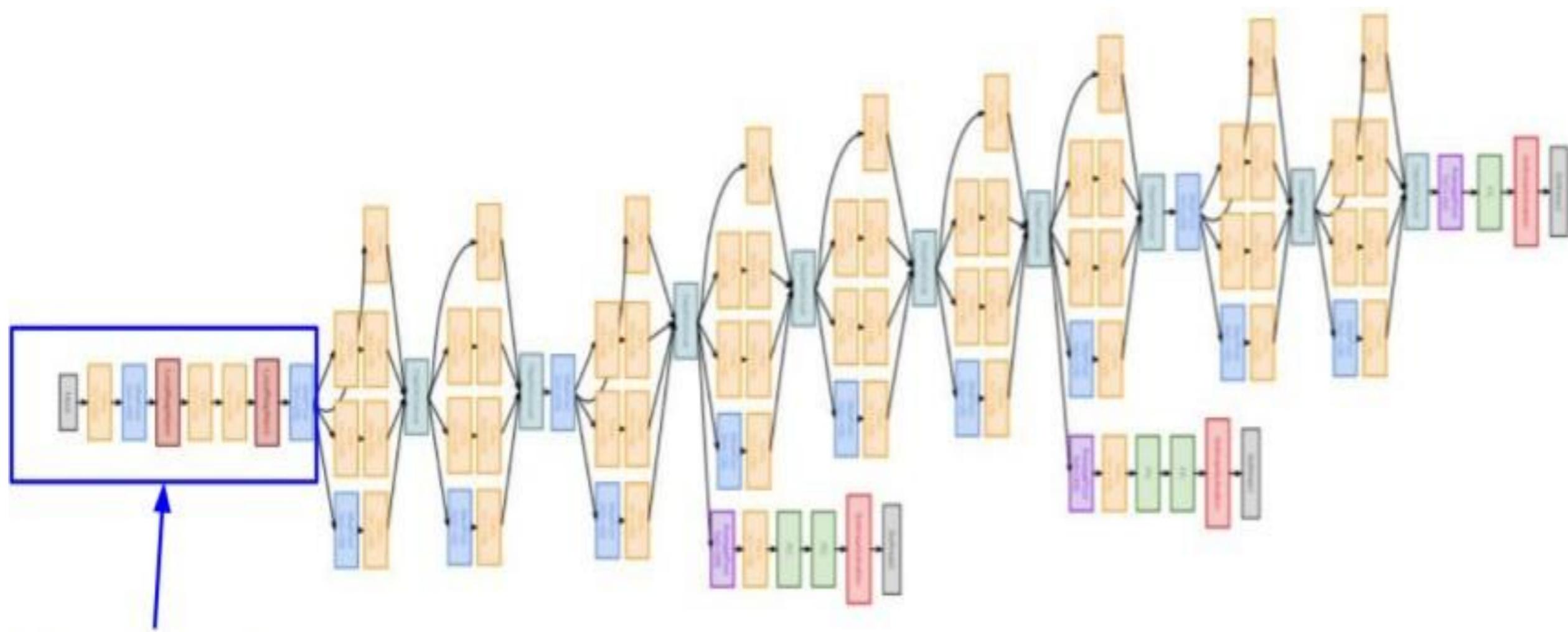


Inception module with dimension reduction

Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

GoogleNet

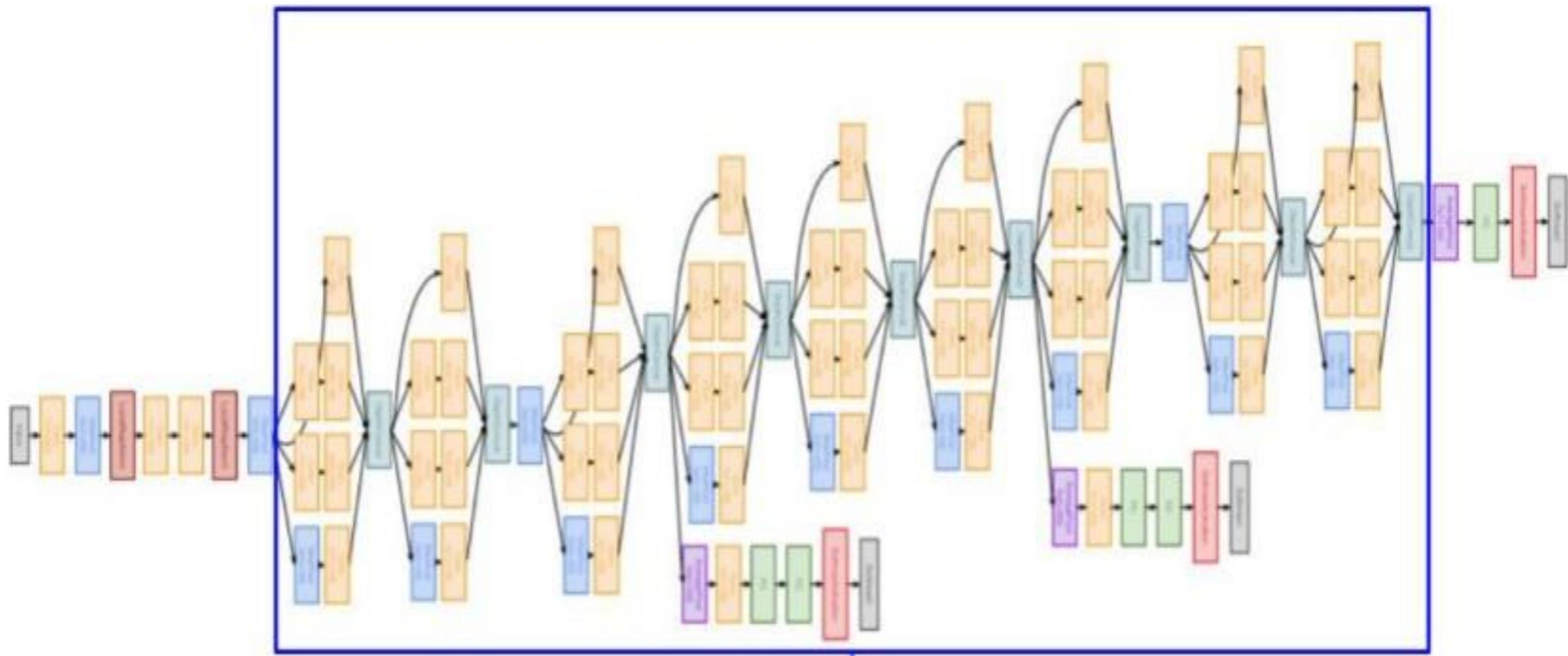
Full Architecture:



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

GoogleNet

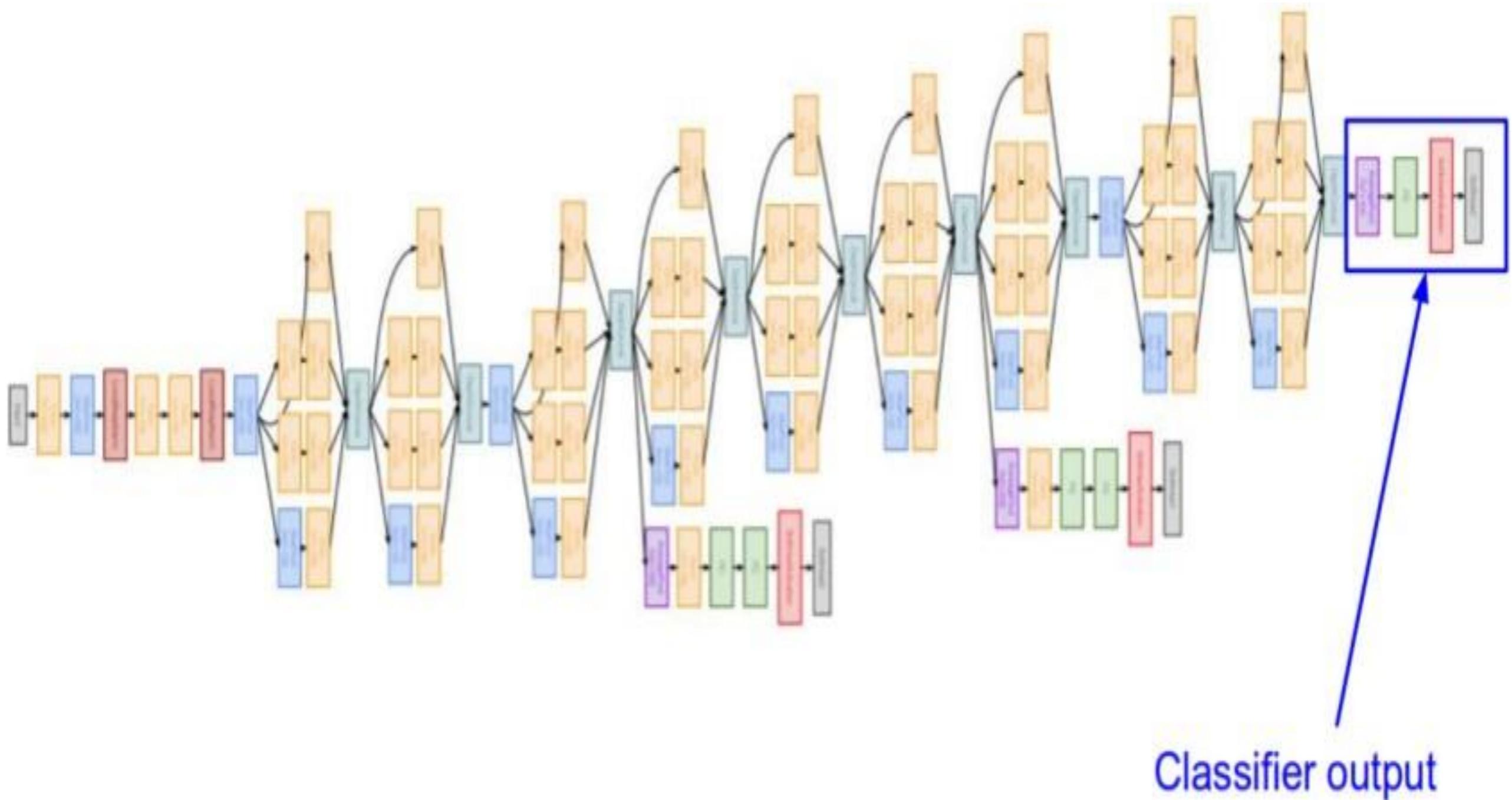
Full Architecture:



Stacked Inception
Modules

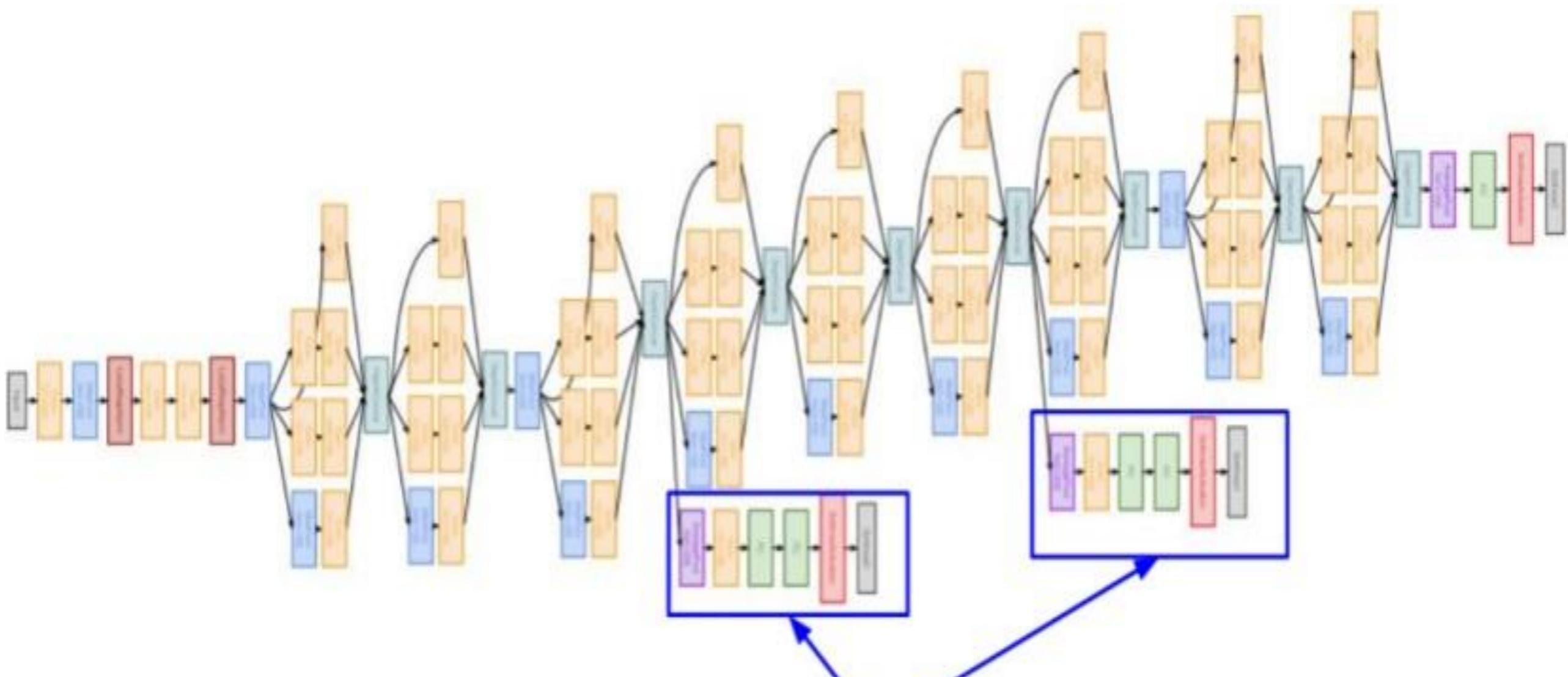
GoogleNet

Full Architecture:



GoogleNet

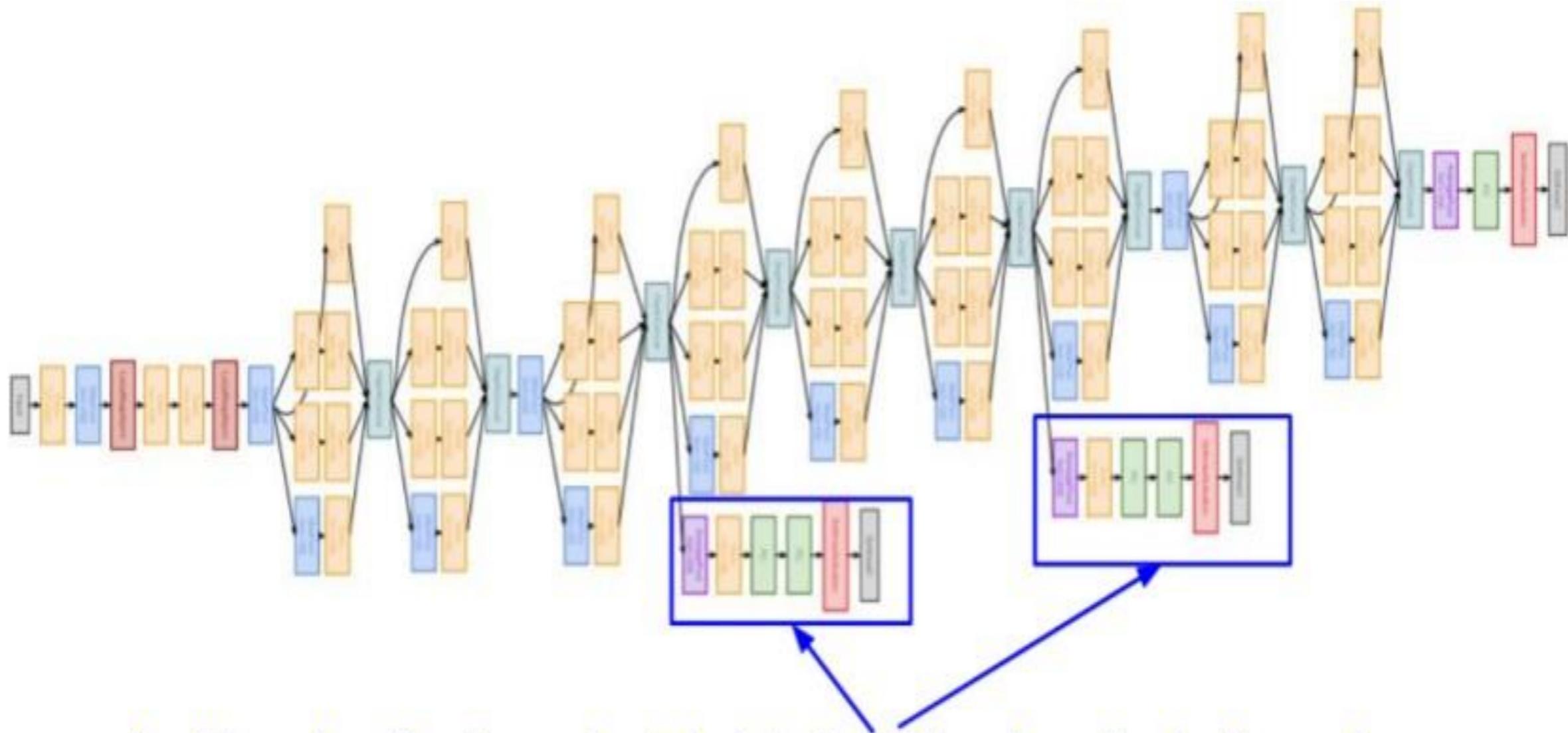
Full Architecture:



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

GoogleNet

Full Architecture:

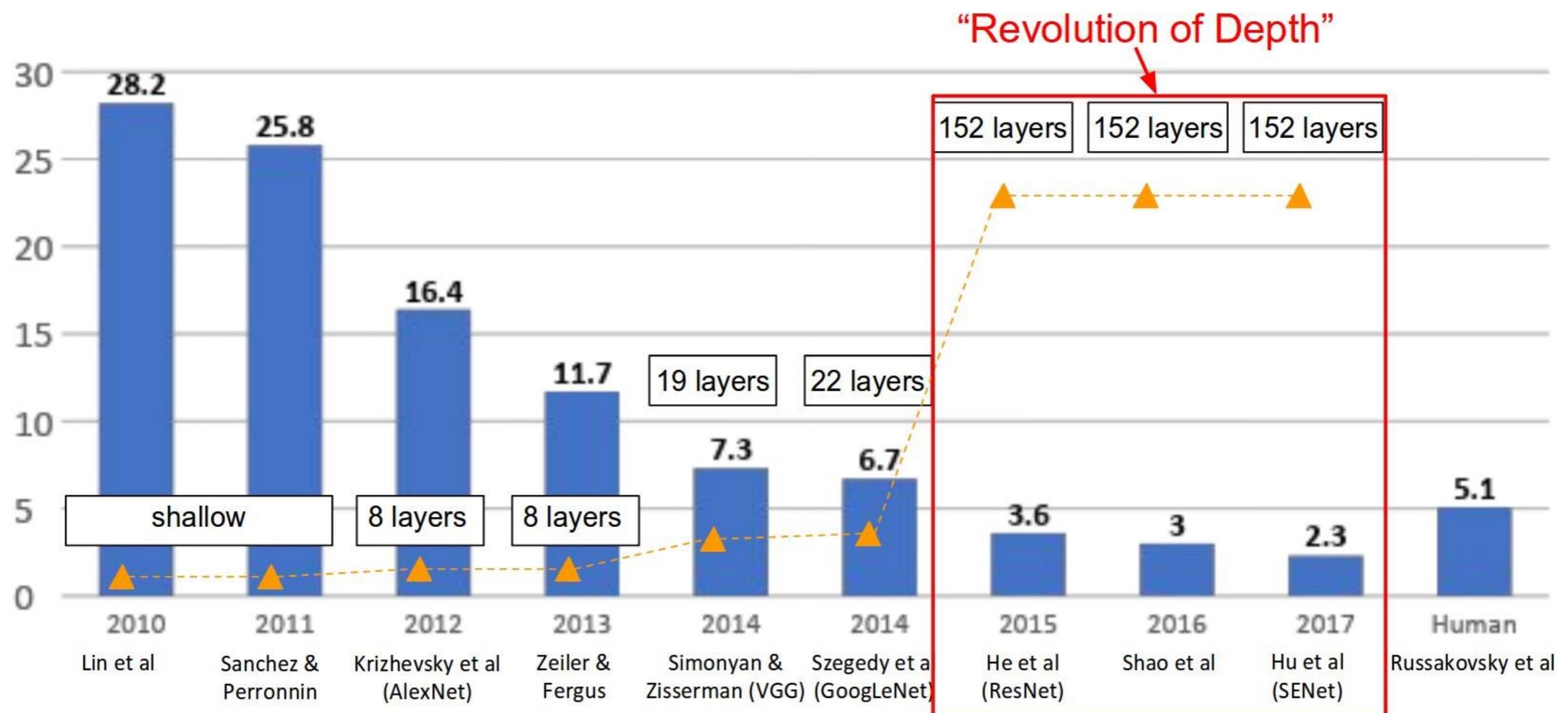


Auxiliary classification outputs to inject additional gradient at lower layers
(AvgPool-1x1Conv-FC-FC-Softmax)

22 total layers (parallel layers count as 1 layer. Auxiliary output layers not counted)

Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

Deeper the Merrier



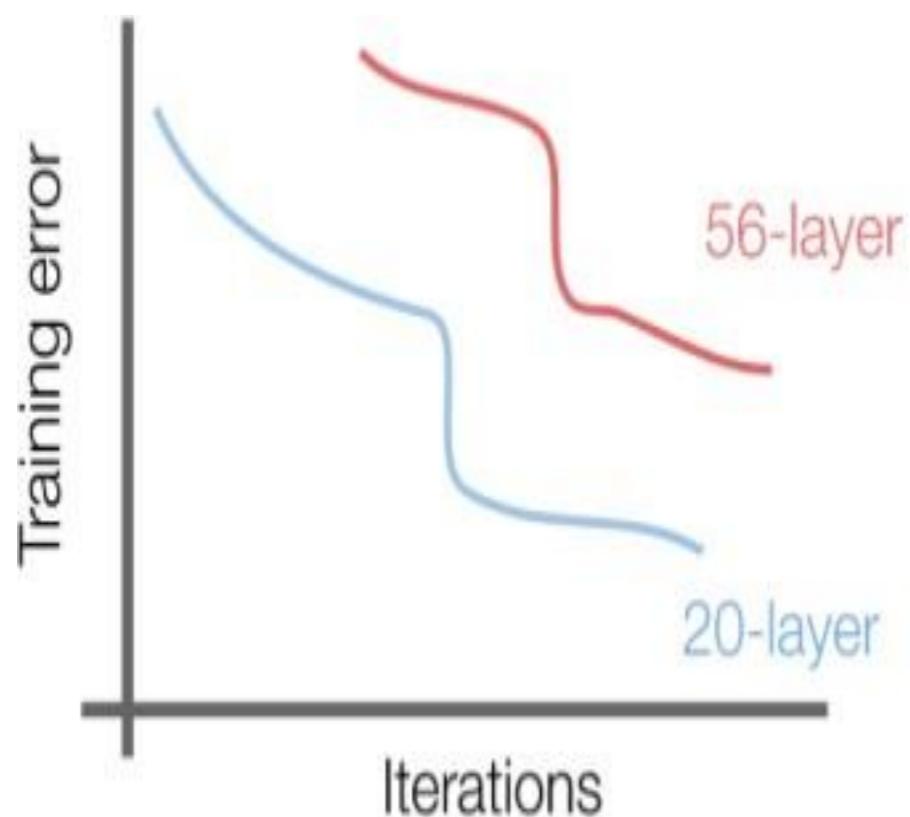
Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

How deep can we go?

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

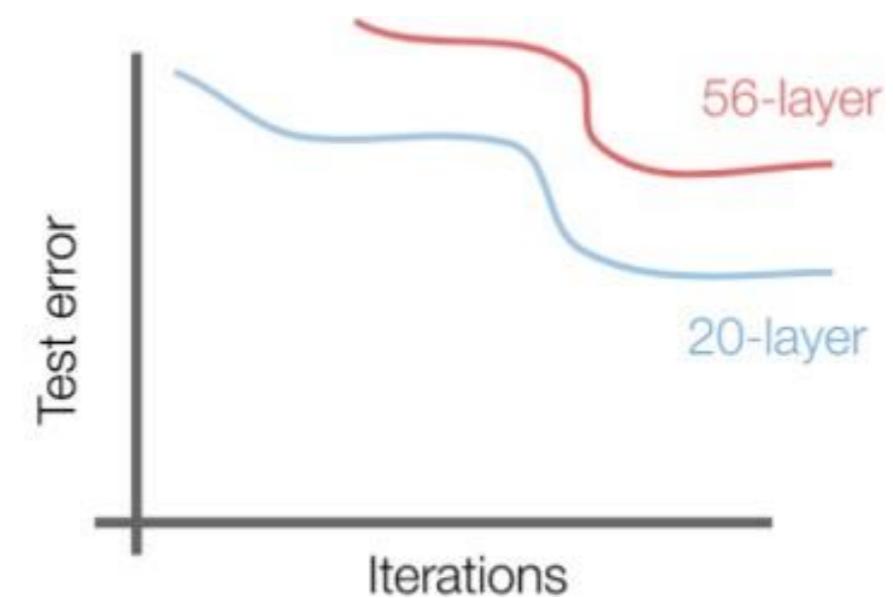
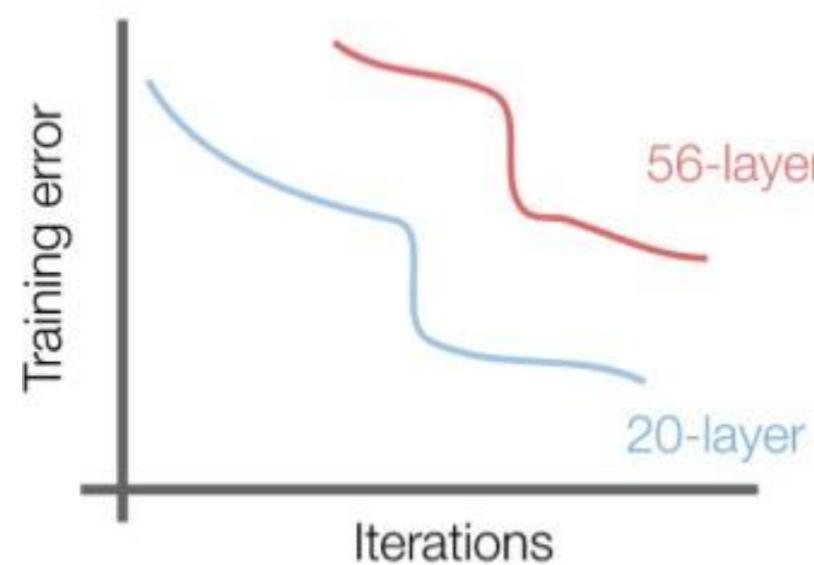
How deep can we go?

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



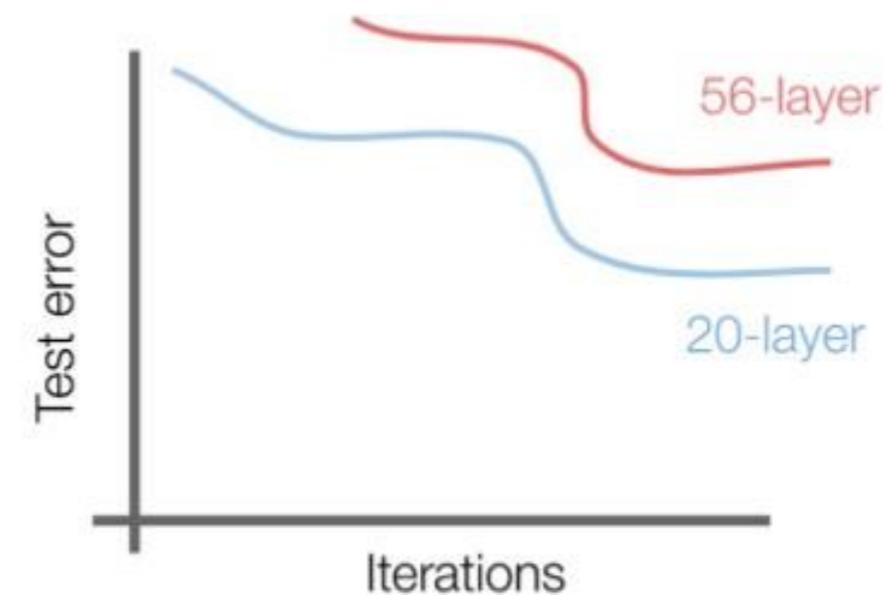
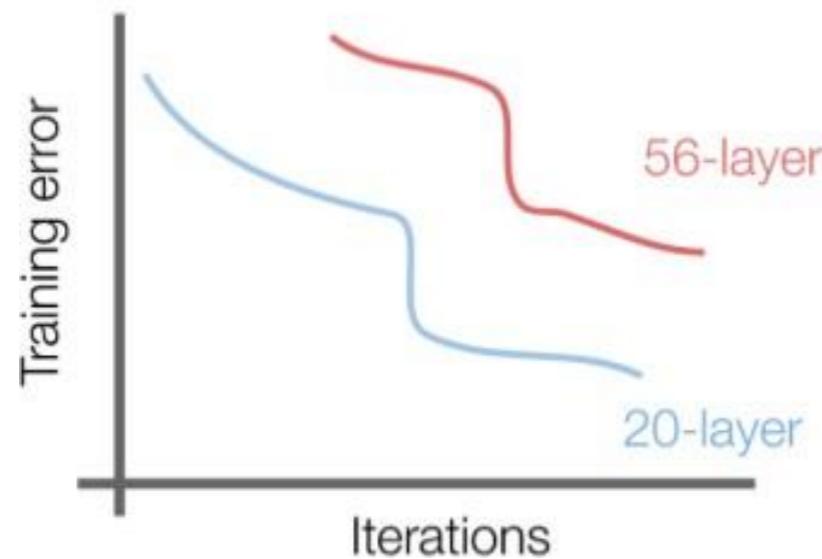
How deep can we go?

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



How deep can we go?

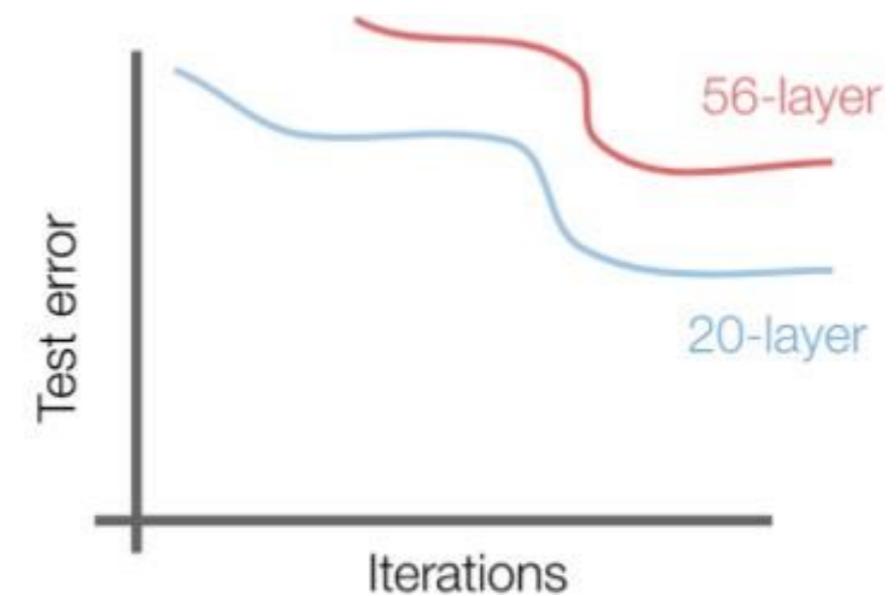
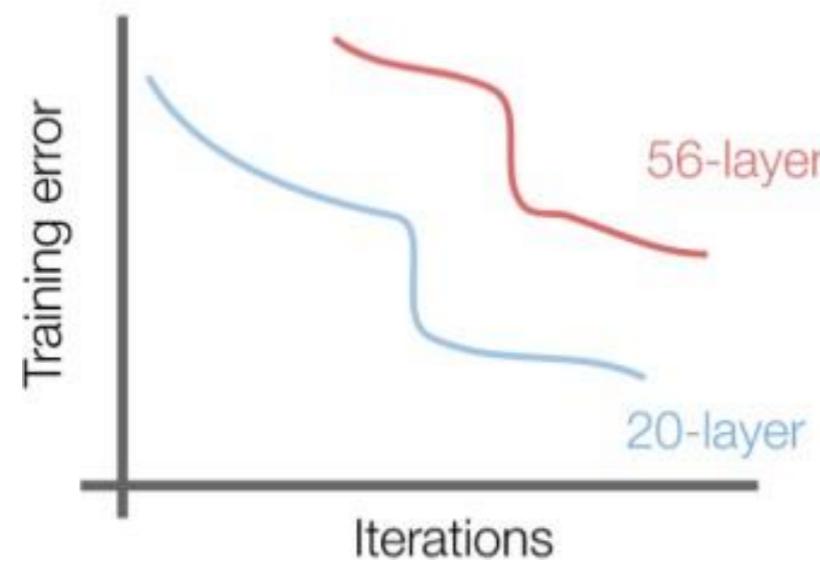
What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



Deeper model does worse than shallow model!

How deep can we go?

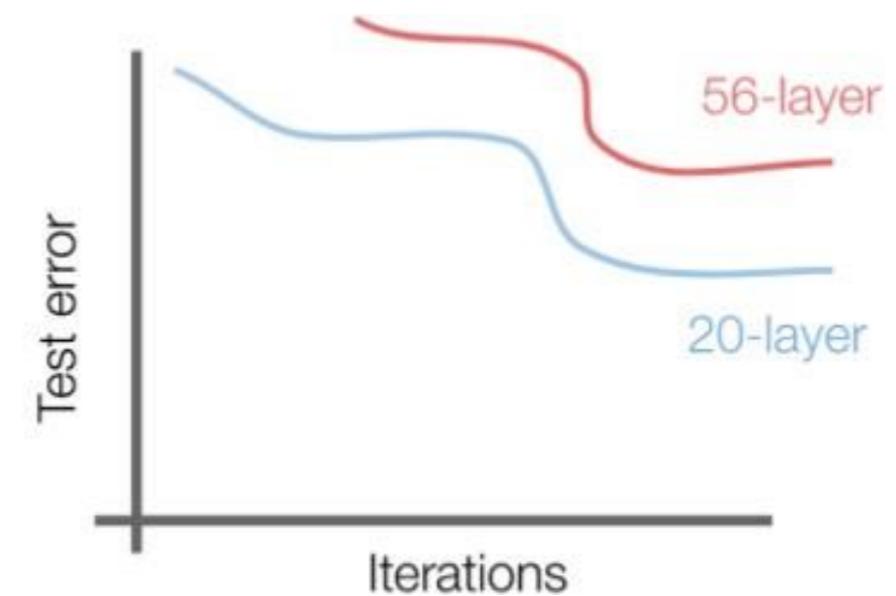
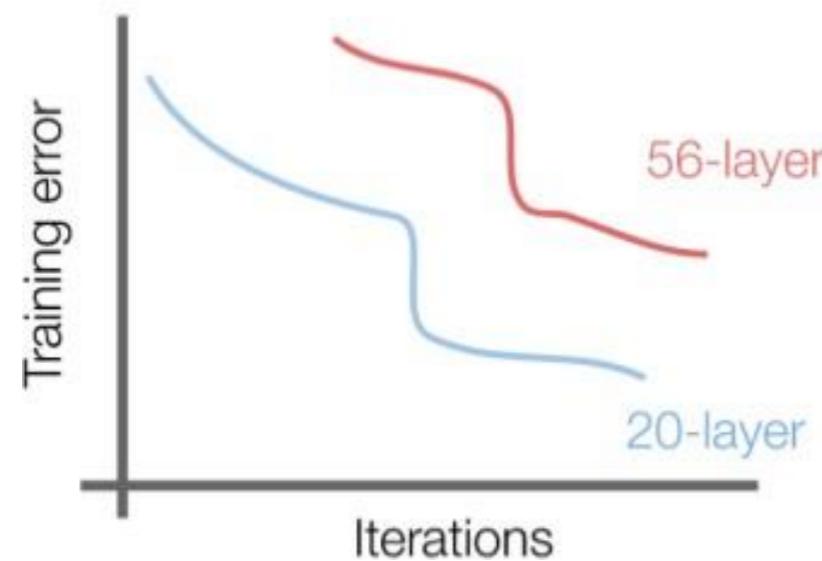
What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



Deeper model does worse than shallow model! **Why?**

How deep can we go?

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

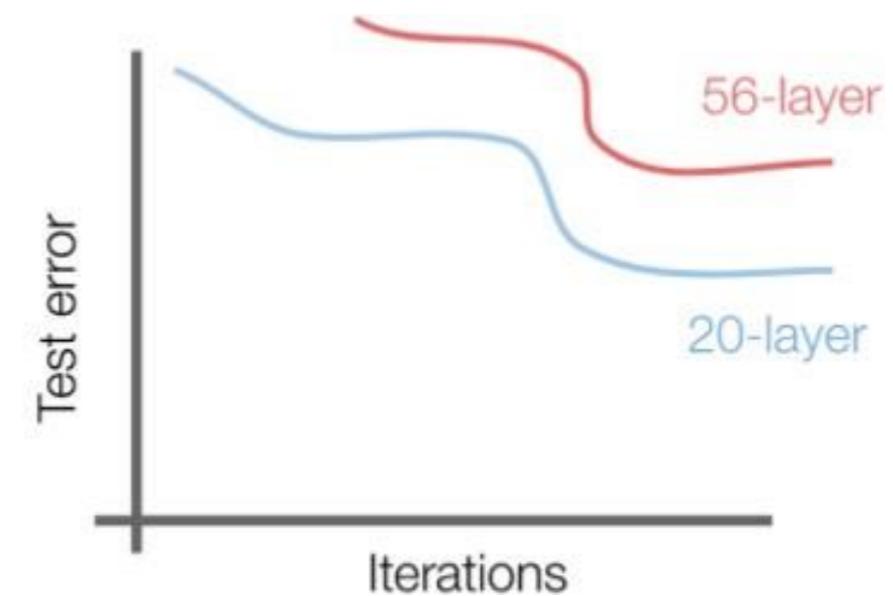
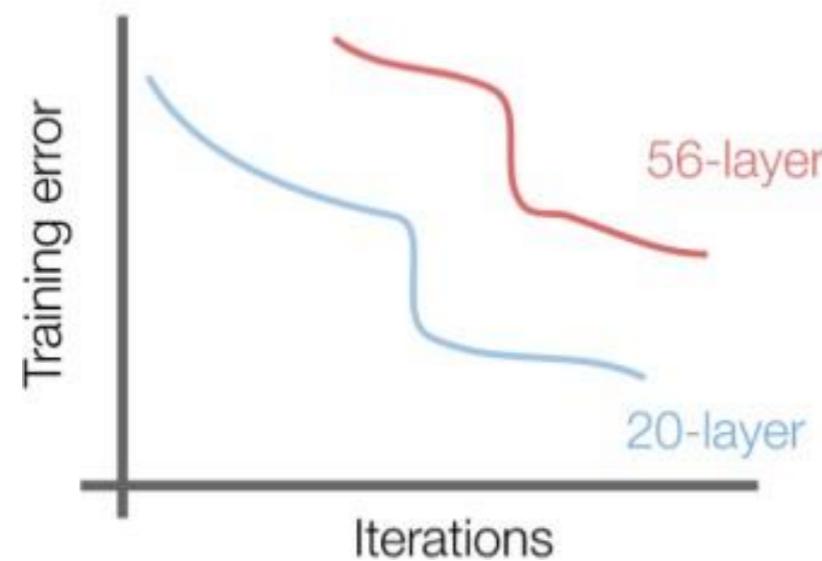


Deeper model does worse than shallow model! Why?

The initial guess is that the deep model is **overfitting** since it is much bigger than the shallow model

How deep can we go?

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



Deeper model does worse than shallow model! Why?

The deep model is actually **underfitting** since it also performs worse than the shallow model on the training set

Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford; Justin Johnson, Univ of Michigan

How deep can we go? Vanishing/Exploding Gradient

Vanishing gradients: Deeper the network, gradients vanish quickly, thereby slowing the rate of change in initial layers

Exploding gradients: Happen when the individual layer gradients are much higher than 1, for instance - can be overcome by **gradient clipping**

ResNet

The deeper model should be able to perform at least as well as the shallower model; **how?**

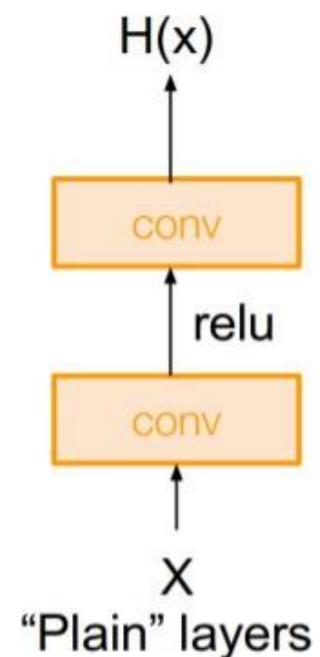
ResNet

The deeper model should be able to perform at least as well as the shallower model;**how?**

Solution: Change the network with identity connections between layers:

The deeper model should be able to perform at least as well as the shallower model;**how?**

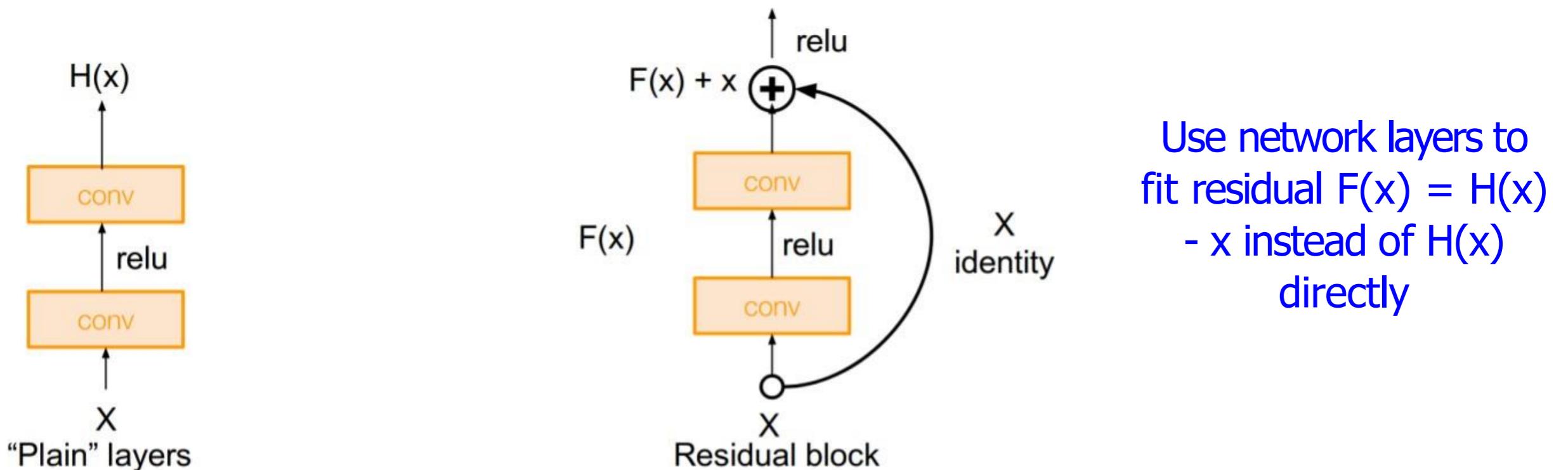
Solution: Change the network with identity connections between layers:
ResNet



ResNet

The deeper model should be able to perform at least as well as the shallower model; **how?**

Solution: Change the network with identity connections between layers:

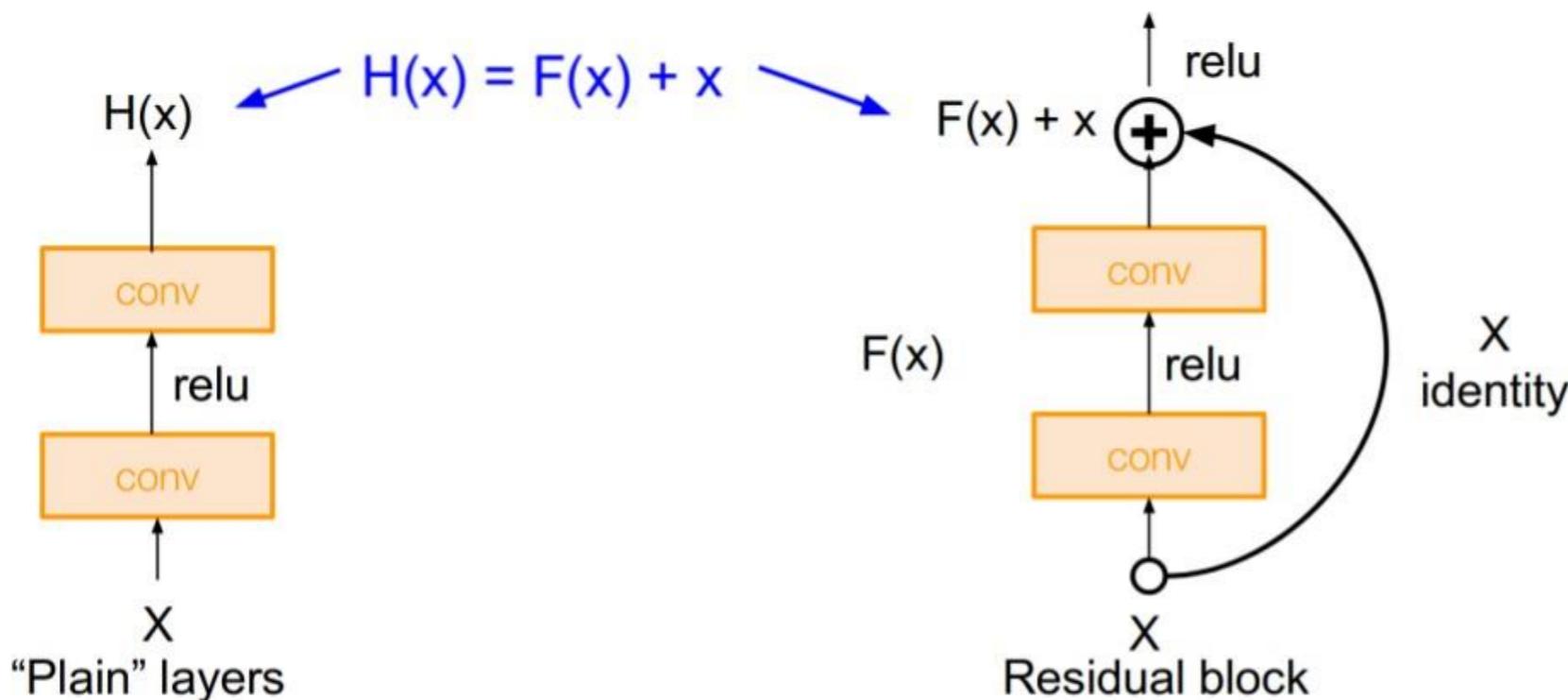


Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

ResNet

The deeper model should be able to perform at least as well as the shallower model;**how?**

Solution: Change the network with identity connections between layers:

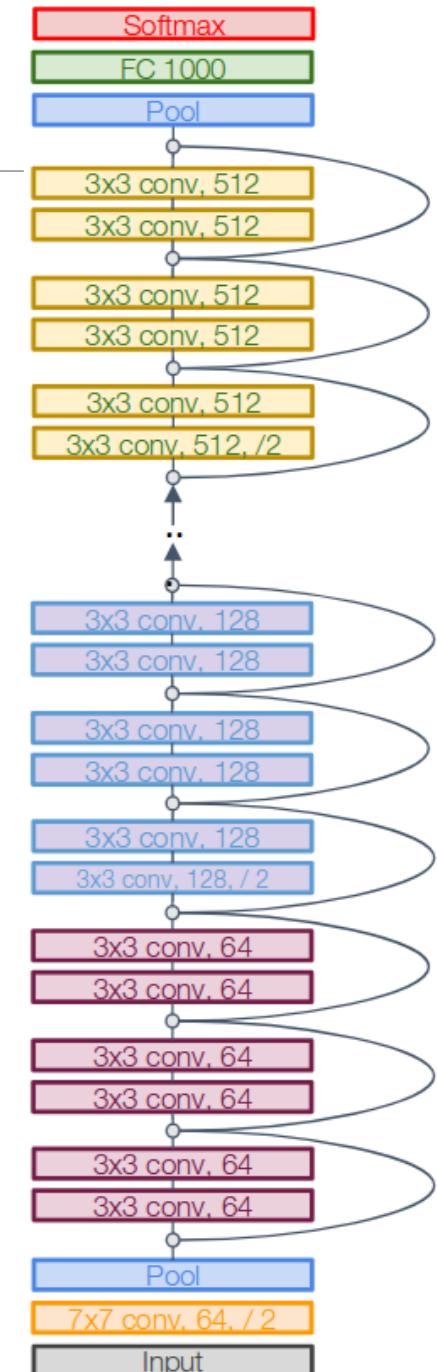


Use network layers to fit residual $F(x) = H(x) - x$ instead of $H(x)$ directly

Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

ResNet

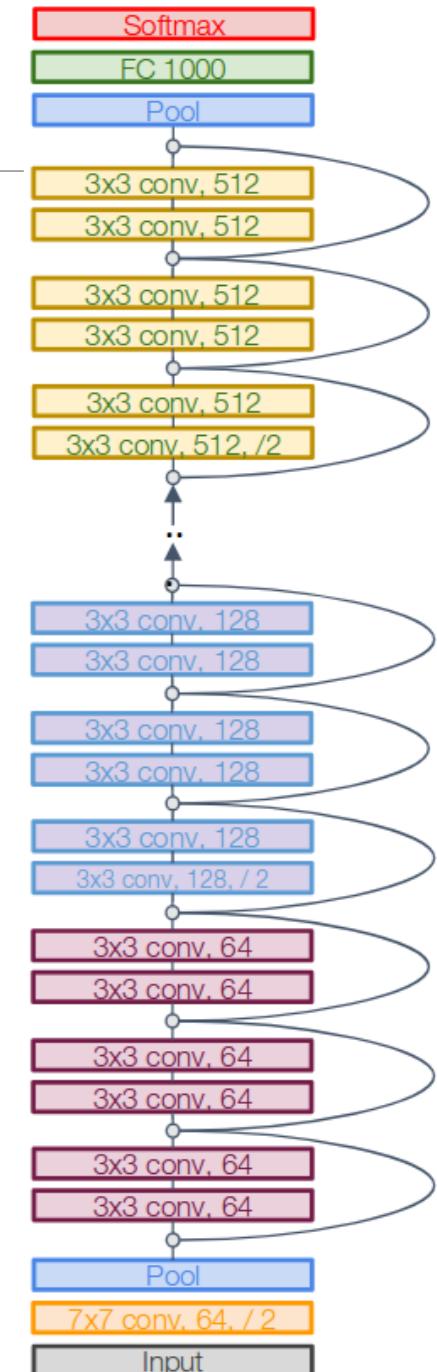
- A residual network is a stack of many residual blocks
 - Each residual block has two 3×3 conv layers



Credit: Fei-Fei Li, Justin Johnson and S Yeung, CS231n course, Stanford, Spring 2019

ResNet

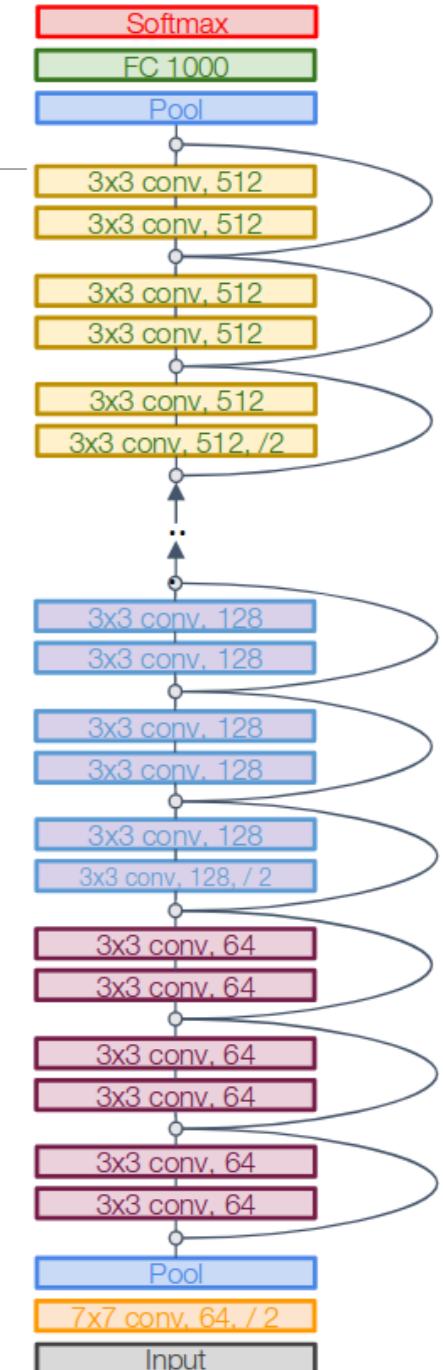
- A residual network is a stack of many residual blocks
- Each residual block has two 3×3 conv layers
- Periodically, double number of filters and downsample spatially using stride 2 (/2 in each dimension)



Credit: Fei-Fei Li, Justin Johnson and S Yeung, [CS231n course](#), Stanford, Spring 2019

ResNet

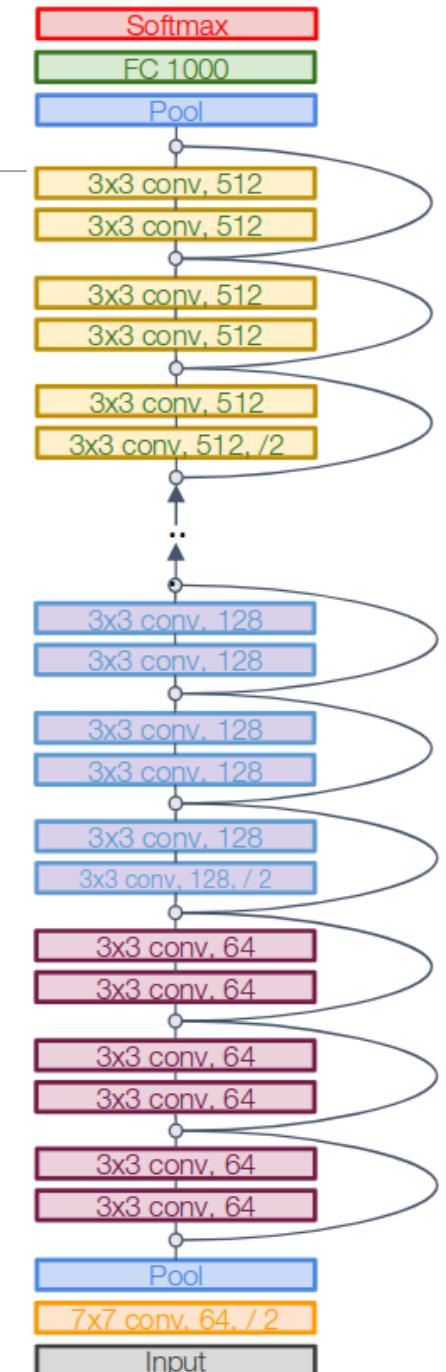
- A residual network is a stack of many residual blocks
- Each residual block has two 3×3 conv layers
- Periodically, double number of filters and downsample spatially using stride 2 (/2 in each dimension)
- Use **global average pooling** and a single linear layer at the end (FC 1000 to output classes)



Credit: Fei-Fei Li, Justin Johnson and S Yeung, [CS231n course](#), Stanford, Spring 2019

ResNet

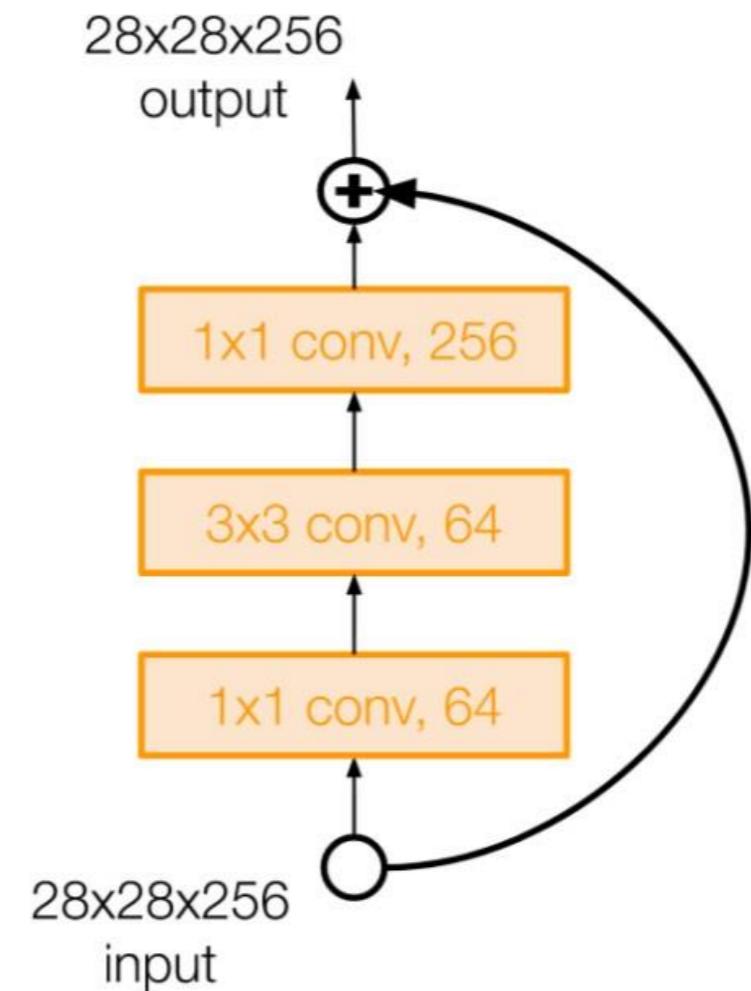
- A residual network is a stack of many residual blocks
- Each residual block has two 3×3 conv layers
- Periodically, double number of filters and downsample spatially using stride 2 (/2 in each dimension)
- Use **global average pooling** and a single linear layer at the end (FC 1000 to output classes)
- Total depths of 34, 50, 101, or 152 layers for ImageNet dataset



Credit: Fei-Fei Li, Justin Johnson and S Yeung, [CS231n course](#), Stanford, Spring 2019

ResNet

For deeper networks
(ResNet-50+), use
“bottleneck” layer to
improve efficiency (similar
to GoogLeNet)



Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

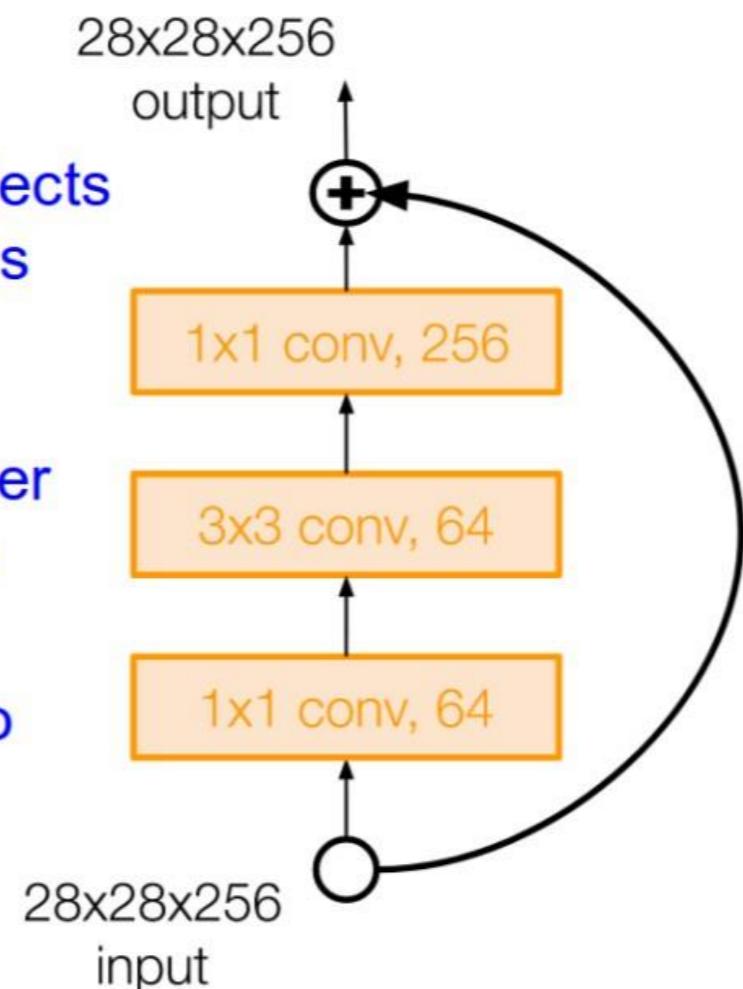
ResNet

For deeper networks (ResNet-50+), use “bottleneck” layer to improve efficiency (similar to GoogLeNet)

1x1 conv, 256 filters projects back to 256 feature maps (28x28x256)

3x3 conv operates over only 64 feature maps

1x1 conv, 64 filters to project to 28x28x64



Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

ResNet

- Able to train very deep networks
- Deeper networks performs better than shallow networks now (as expected); residual blocks help avoid vanishing gradient
- 1st place in all ILSVRC and COCO 2015 competitions
- We will discuss detection, localization, segmentation and the COCO dataset a bit later

ResNet

- Able to train very deep networks
- Deeper networks performs better than shallow networks now (as expected); residual blocks help avoid vanishing gradient
- 1st place in all ILSVRC and COCO 2015 competitions
- We will discuss detection, localization, segmentation and the COCO dataset a bit later

ResNet

- Able to train very deep networks
- Deeper networks performs better than shallow networks now (as expected); residual blocks help avoid vanishing gradient
- 1st place in all ILSVRC and COCO 2015 competitions
- We will discuss detection, localization, segmentation and the COCO dataset a bit later

ResNet

- Able to train very deep networks
- Deeper networks performs better than shallow networks now (as expected); residual blocks help avoid vanishing gradient
- 1st place in all ILSVRC and COCO 2015 competitions
- We will discuss detection, localization, segmentation and the COCO dataset a bit later

ResNet

- Able to train very deep networks
- Deeper networks performs better than shallow networks now (as expected); residual blocks help avoid vanishing gradient
- 1st place in all ILSVRC and COCO 2015 competitions
- We will discuss detection, localization, segmentation and the COCO dataset a bit later

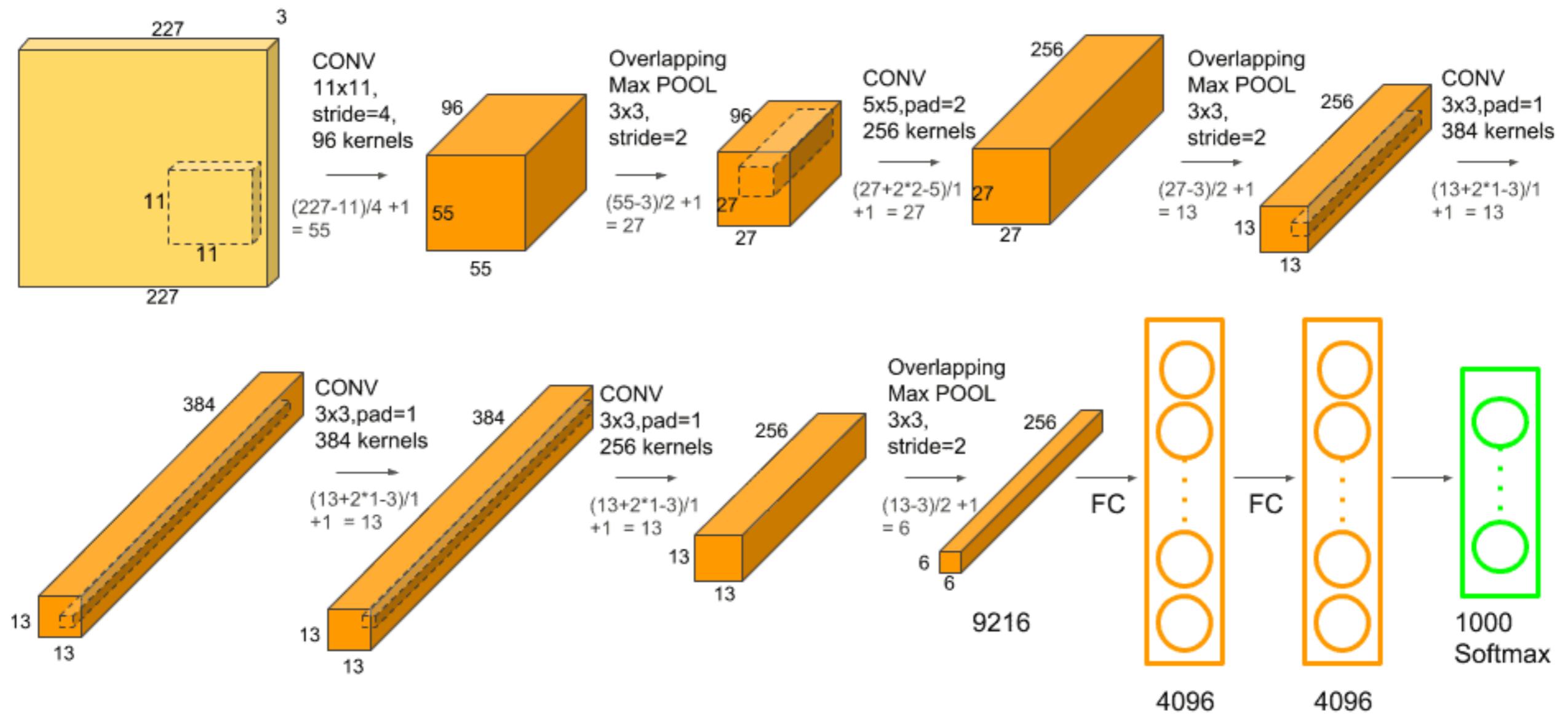
ResNet @ ILSVRC & COCO 2015 Competitions

1st place in all five major challenges

- [ImageNet Classification](#): "Ultra-deep" **152-layer** nets
- [ImageNet Detection](#): **16%** better than the 2nd best
- [ImageNet Localization](#): **27%** better than the 2nd best
- [COCO Detection](#): **11%** better than the 2nd best
- [COCO Segmentation](#): **12%** better than the 2nd best

Credit: Fei-Fei Li, Justin Johnson and Serena Yeung, CS231n course, Stanford, Spring 2019

Example of ConvNet-AlexNet



Parameter calculation

((width of the filter * height of the filter *
number of filters in the previous
layer)+1)*number of filters)

Alexnet

Number of Parameters of a Conv Layer

Let's define,

W_c = Number of weights of the Conv Layer.

B_c = Number of biases of the Conv Layer.

P_c = Number of parameters of the Conv Layer.

K = Size (width) of kernels used in the Conv Layer.

N = Number of kernels.

C = Number of channels of the input image.

$$W_c = K^2 \times C \times N$$

$$B_c = N$$

$$P_c = W_c + B_c$$

Alexnet

Number of Parameters of a Fully Connected (FC) Layer-

Case 1: Number of Parameters of a Fully Connected (FC) Layer connected to a Conv Layer

Let's define,

W_{cf} = Number of weights of a FC Layer which is connected to a Conv Layer.

B_{cf} = Number of biases of a FC Layer which is connected to a Conv Layer.

O = Size (width) of the output image of the previous Conv Layer.

N = Number of kernels in the previous Conv Layer.

F = Number of neurons in the FC Layer.

$$W_{cf} = O^2 \times N \times F$$

$$B_{cf} = F$$

$$P_{cf} = W_{cf} + B_{cf}$$

Alexnet

Case 2: Number of Parameters of a Fully Connected (FC) Layer connected to a FC Layer

Let's define,

W_{ff} = Number of weights of a FC Layer which is connected to an FC Layer.

B_{ff} = Number of biases of a FC Layer which is connected to an FC Layer.

P_{ff} = Number of parameters of a FC Layer which is connected to an FC Layer.

F = Number of neurons in the FC Layer.

$F-1$ = Number of neurons in the previous FC Layer.

$$W_{ff} = F_{-1} \times F$$

$$B_{ff} = F$$

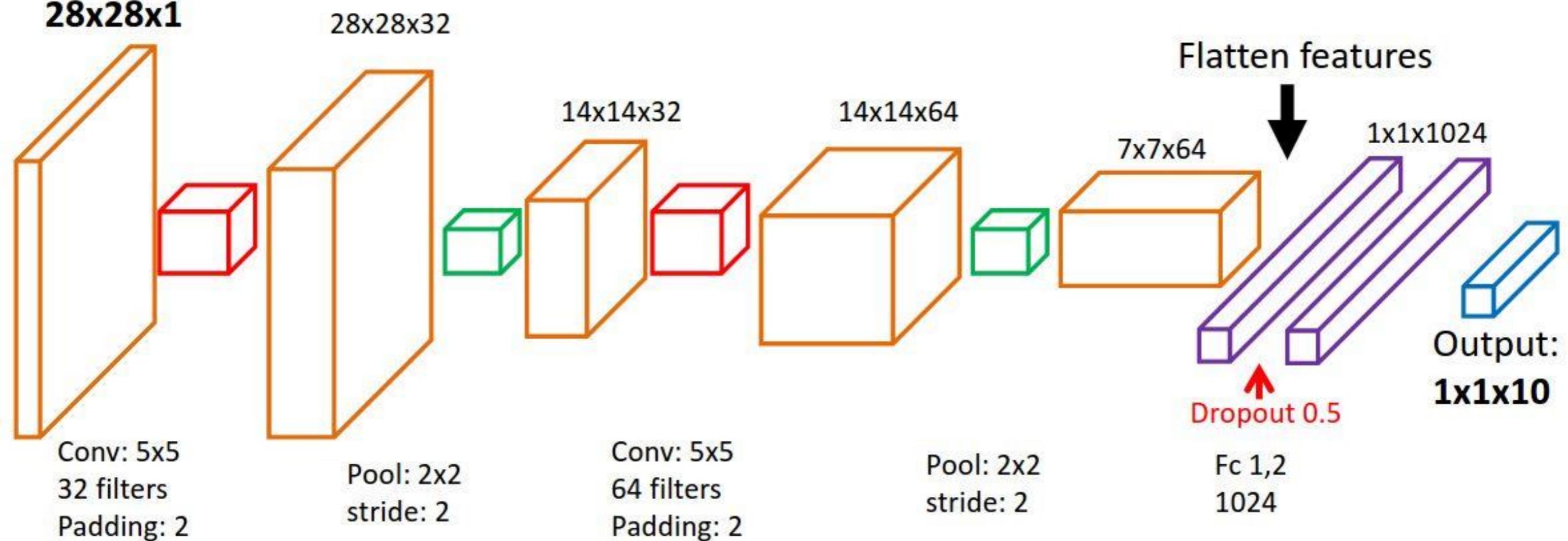
$$P_{ff} = W_{ff} + B_{ff}$$

Layer Name	Tensor Size	Weights	Biases	Parameters
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	884,736	384	885,120
Conv-4	13x13x384	1,327,104	384	1,327,488
Conv-5	13x13x256	884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096x1	37,748,736	4,096	37,752,832
FC-2	4096x1	16,777,216	4,096	16,781,312
FC-3	1000x1	4,096,000	1,000	4,097,000
Output	1000x1	0	0	0
Total				62,378,344

LeNet-5 for MNIST

Input:

28x28x1



References

-  David Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60 (Nov. 2004), pp. 91–110.
-  Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* 1 (2005), 886–893 vol. 1.
-  Svetlana Lazebnik, Cordelia Schmid, and J. Ponce. "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories". In: vol. 2. Feb. 2006, pp. 2169 –2178.
-  Kaiming He et al. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". In: *Lecture Notes in Computer Science* (2014), 346–361.
-  Dingjun Yu et al. "Mixed Pooling for Convolutional Neural Networks". In: Oct. 2014, pp. 364–375.
-  Oren Rippel, Jasper Snoek, and Ryan P. Adams. "Spectral Representations for Convolutional Neural Networks". In: *NIPS*. 2015.
-  Nal Kalchbrenner et al. "Neural Machine Translation in Linear Time". In: *ArXiv* abs/1610.10099 (2016).

References

-  Yann LeCun et al. "Gradient-based learning applied to document recognition". In: 1998.
-  Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *NIPS*. 2012.
-  Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2015).
-  Christian Szegedy et al. "Going deeper with convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 1–9.
-  Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.
-  Johnson, Justin, EECS 498-007 / 598-005 - Deep Learning for Computer Vision (Fall 2019). URL: <https://web.eecs.umich.edu/~justincj/teaching/eecs498/> (visited on 06/29/2020).
-  Li, Fei-Fei; Johnson, Justin; Serena, Yeung; CS 231n - Convolutional Neural Networks for Visual Recognition (Spring 2019). URL: <http://cs231n.stanford.edu/2019/> (visited on 06/29/2020).