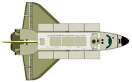
CS7015 (Deep Learning) : Lecture 11

Convolutional Neural Networks, LeNet, AlexNet, ZF-Net, VGGNet, GoogLeNet and ResNet

Mitesh M. Khapra

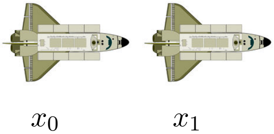
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Module 11.1 : The convolution operation

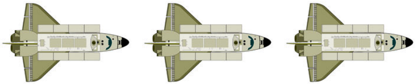


x_0

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals

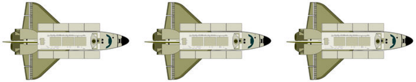


- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals



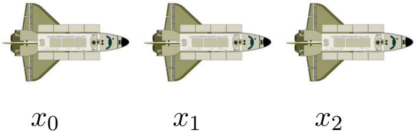
x_0 x_1 x_2

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals

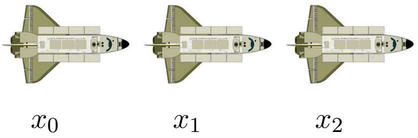


x_0 x_1 x_2

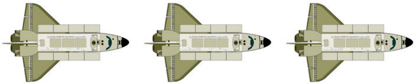
- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy



- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements



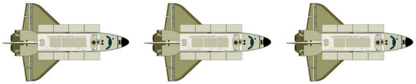
- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements
- More recent measurements are more important so we would like to take a weighted average



x_0 x_1 x_2

$$s_t = \sum_{a=0}^{\infty} x_{t-a} w_{-a} =$$

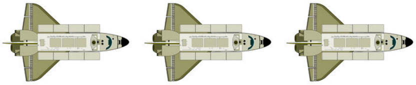
- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements
- More recent measurements are more important so we would like to take a weighted average



x_0 x_1 x_2

$$s_t = \sum_{a=0}^{\infty} x_{t-a} w_{-a} = (x * w)_t$$

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements
- More recent measurements are more important so we would like to take a weighted average



$$s_t = \sum_{a=0}^{\infty} x_{t-a} w_{-a} = (x * w)_t$$

input

convolution

filter

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements
- More recent measurements are more important so we would like to take a weighted average

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

- In practice, we would only sum over a small window

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0					
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5					
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80					

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0					
W		0.01	0.01	0.02	0.02	0.04	0.4	0.5				
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96				

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0						
W			0.01	0.01	0.02	0.02	0.04	0.4	0.5				
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70	
S							1.80	1.96	2.11				

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96	2.11	2.16		

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t

					w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0	
W					0.01	0.01	0.02	0.02	0.04	0.4	0.5	
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96	2.11	2.16	2.28	

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t

						w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W						0.01	0.01	0.02	0.02	0.04	0.4	0.5
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96	2.11	2.16	2.28	2.42

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96	2.11	2.16	2.28	2.42

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t
- Here the input (and the kernel) is one dimensional

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96	2.11	2.16	2.28	2.42

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t
- Here the input (and the kernel) is one dimensional
- Can we use a convolutional operation on a 2D input also?

- We can think of images as 2D inputs

- We can think of images as 2D inputs
- We would now like to use a 2D filter ($m \times n$)

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i-a,j-b} K_{a,b}$$

- We can think of images as 2D inputs
- We would now like to use a 2D filter $(m \times n)$
- First let us see what the 2D formula looks like

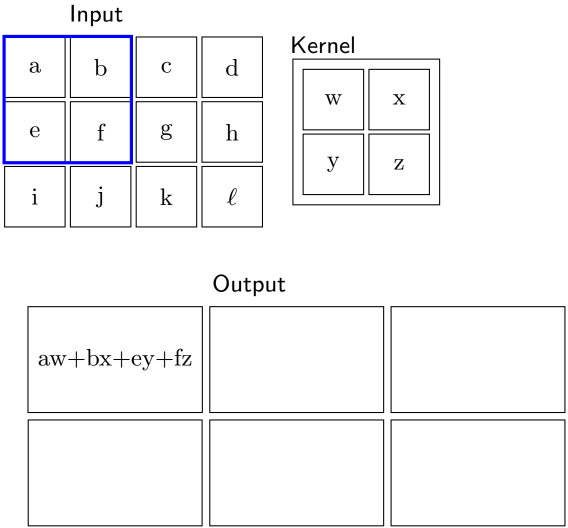
- We can think of images as 2D inputs
- We would now like to use a 2D filter $(m \times n)$
- First let us see what the 2D formula looks like
- This formula looks at all the preceding neighbours $(i-a, j-b)$

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i-a,j-b} K_{a,b}$$

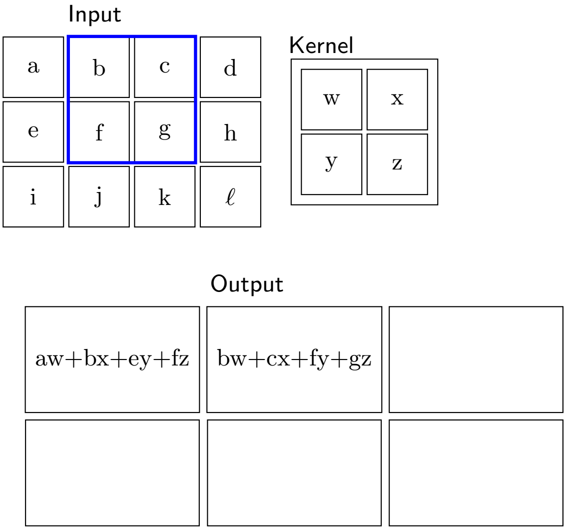
$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a,j+b} K_{a,b}$$

- We can think of images as 2D inputs
- We would now like to use a 2D filter $(m \times n)$
- First let us see what the 2D formula looks like
- This formula looks at all the preceding neighbours $(i - a, j - b)$
- In practice, we use the following formula which looks at the succeeding neighbours

- Let us apply this idea to a toy example and see the results



- Let us apply this idea to a toy example and see the results



- Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

Kernel

w	x
y	z

- Let us apply this idea to a toy example and see the results

Output

aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz

Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

Kernel

w	x
y	z

Output

aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz
ew+fx+iy+jz		

- Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

Kernel

w	x
y	z

Output

aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz
ew+fx+iy+jz	fw+gx+jy+kz	

- Let us apply this idea to a toy example and see the results

Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

Kernel

w	x
y	z

Output

aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz
ew+fx+iy+jz	fw+gx+jy+kz	gw+hx+ky+ℓz

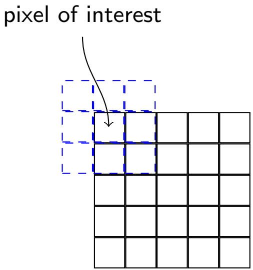
- Let us apply this idea to a toy example and see the results

- For the rest of the discussion we will use the following formula for convolution

$$S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a,j-b} K_{\frac{m}{2}+a, \frac{n}{2}+b}$$

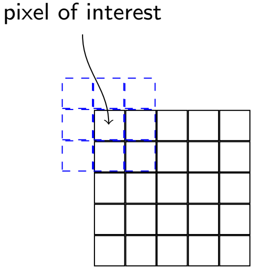
- For the rest of the discussion we will use the following formula for convolution

$$S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a,j-b} K_{\frac{m}{2}+a, \frac{n}{2}+b}$$



- For the rest of the discussion we will use the following formula for convolution
- In other words we will assume that the kernel is centered on the pixel of interest

$$S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a,j-b} K_{\frac{m}{2}+a, \frac{n}{2}+b}$$



- For the rest of the discussion we will use the following formula for convolution
- In other words we will assume that the kernel is centered on the pixel of interest
- So we will be looking at both preceeding and succeeding neighbors

Let us see some examples of 2D convolutions applied to images

$$* \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} =$$

$$* \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} =$$

blurs the image

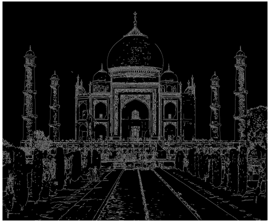
$$* \begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix} =$$

$$* \quad \begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix} \quad =$$

sharpens the image

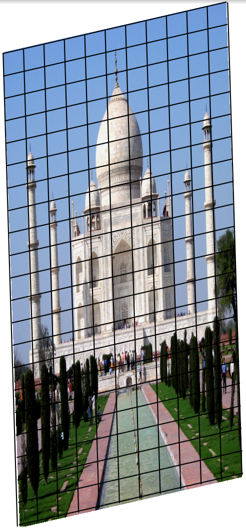
$$* \begin{array}{ccc} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{array} =$$

$$* \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix} =$$

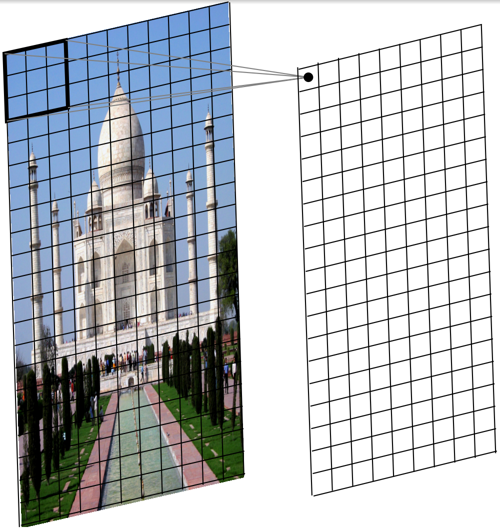


detects the edges

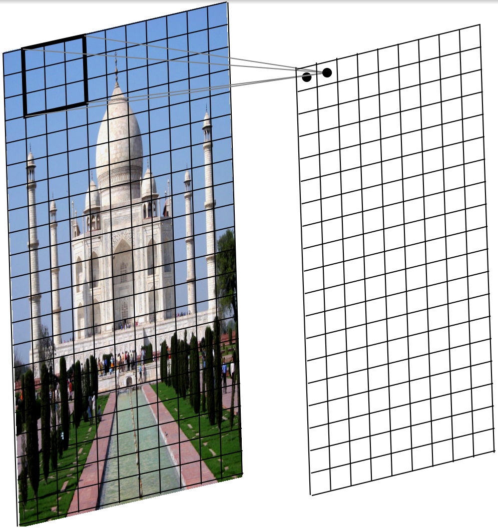
We will now see a working example of 2D convolution.



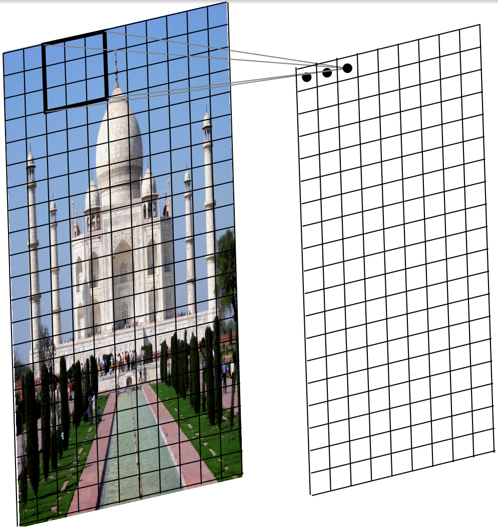
- We just slide the kernel over the input image



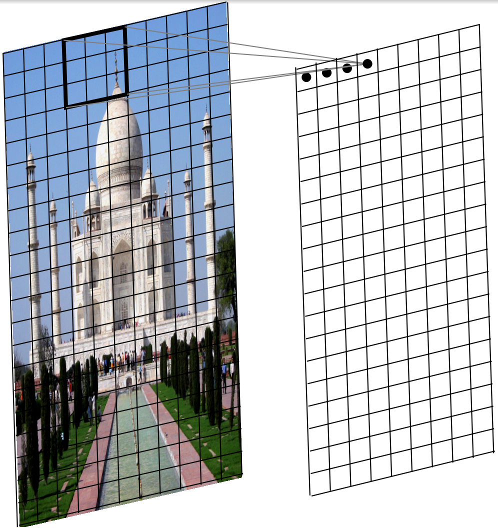
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



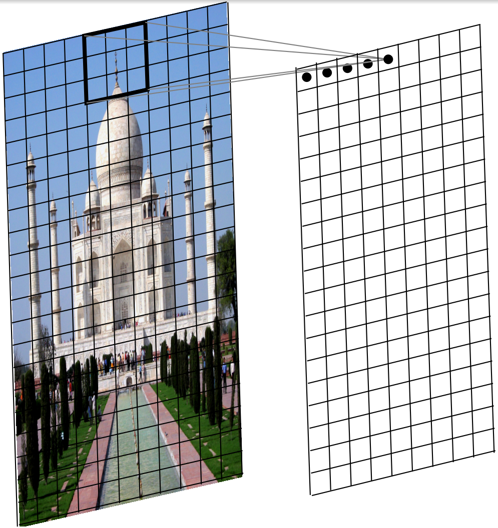
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



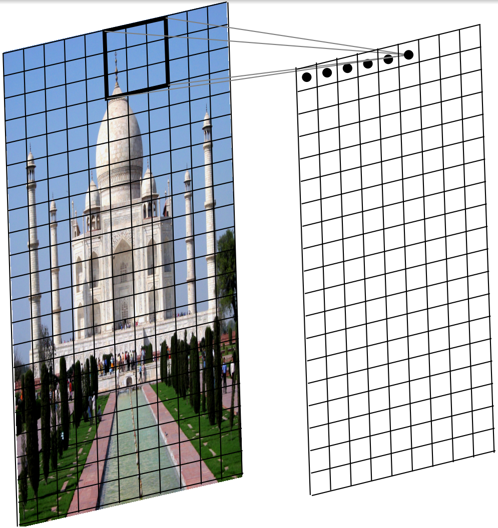
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



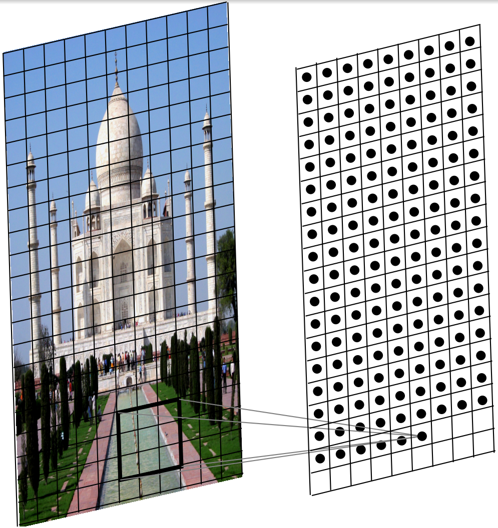
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



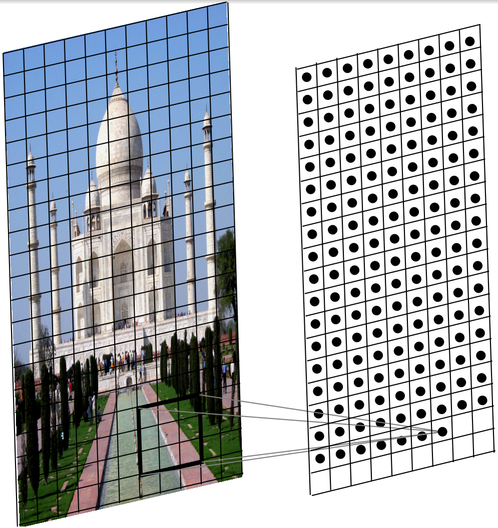
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



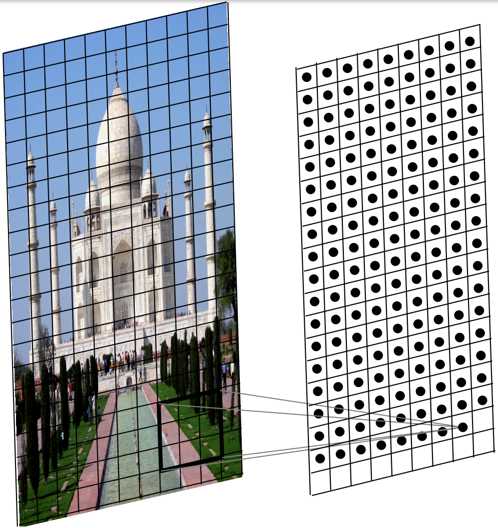
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



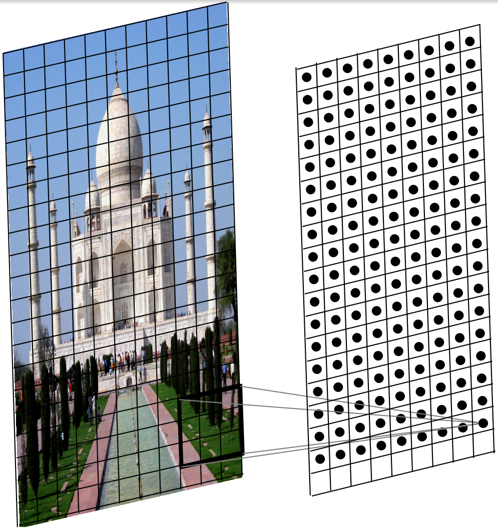
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



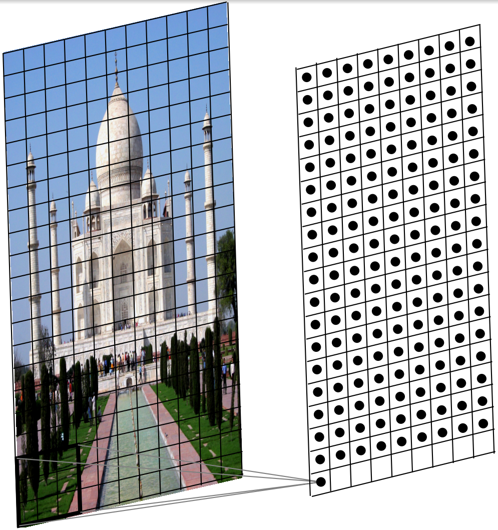
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



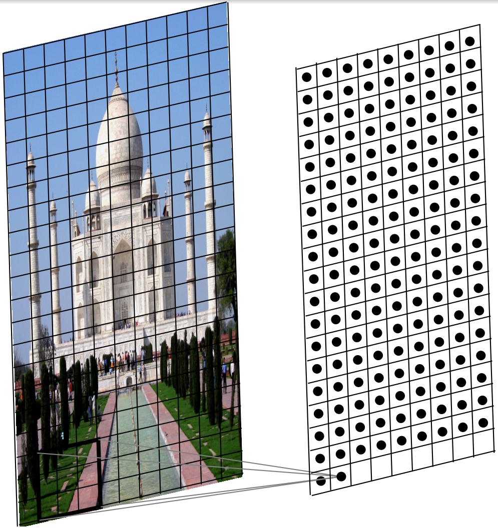
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



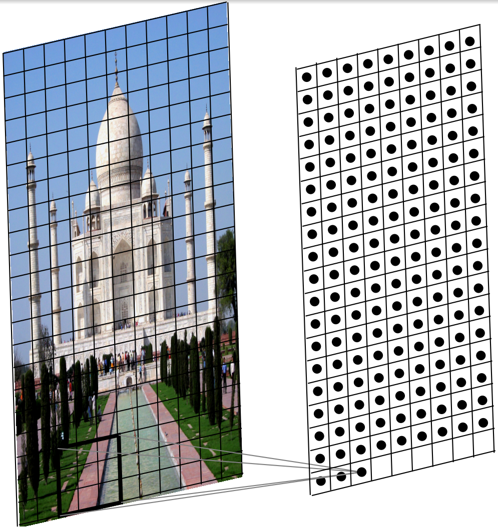
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



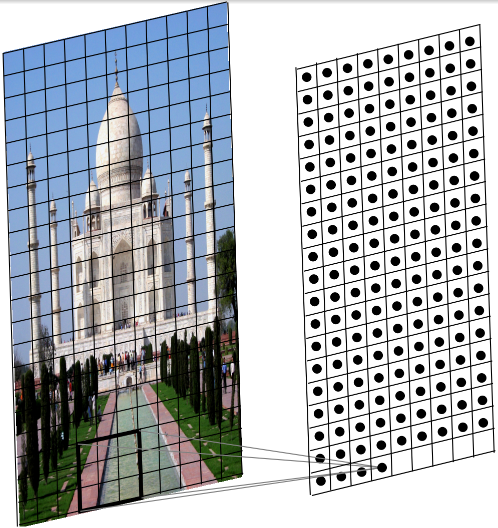
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



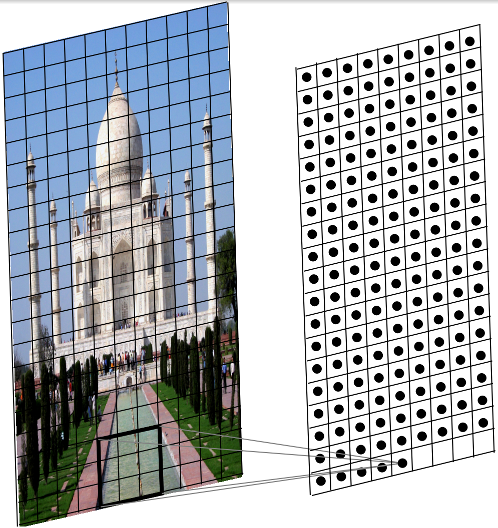
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



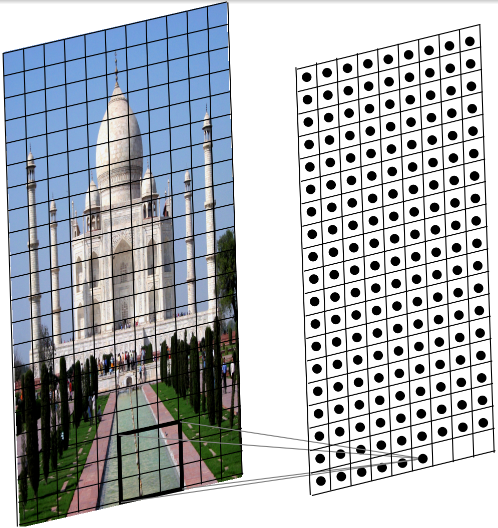
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



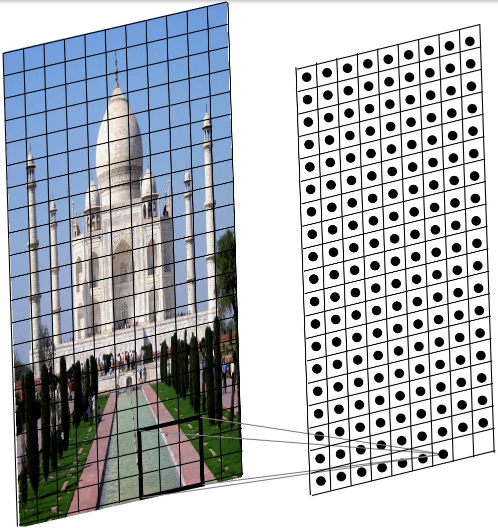
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



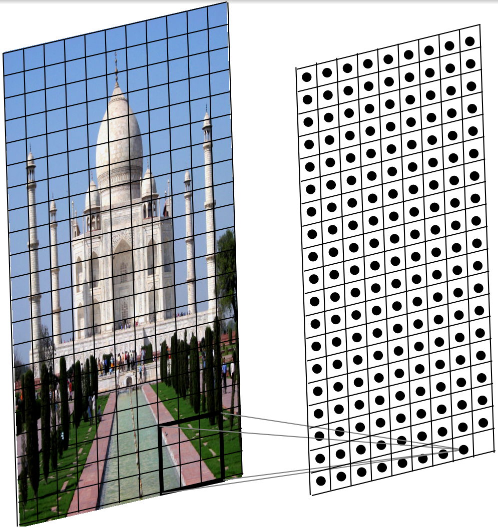
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



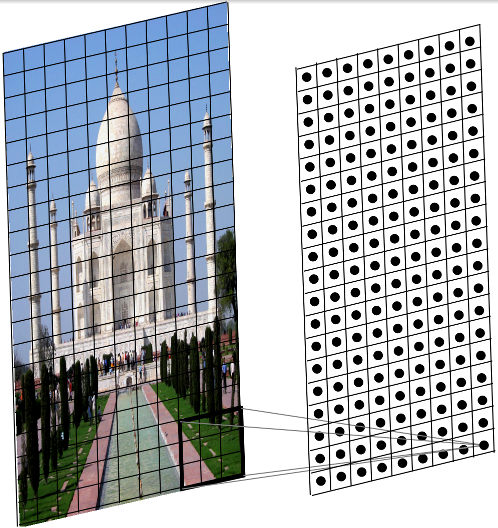
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



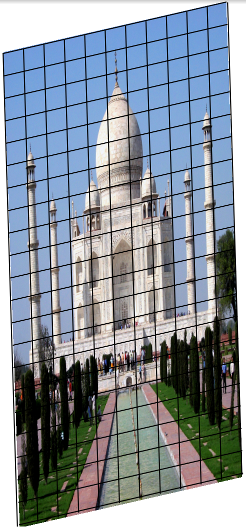
- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

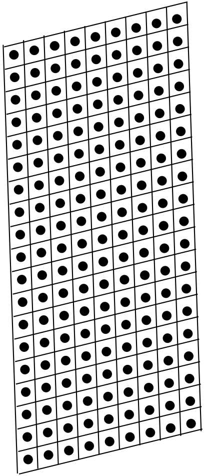


- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output



- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output
- The resulting output is called a feature map.





- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output
- The resulting output is called a feature map.
- We can use multiple filters to get multiple feature maps.

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input

A	B	C	B	A	B	C

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input

A	B	C	B	A	B	C

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input

A	B	C	B	A	B	C

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input

A	B	C	$\boxed{B}$	$\boxed{A}$	$\boxed{B}$	C

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input

A	B	C	B	A	B	C

Question

- 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 output

Question

- 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 output

a	b	c	d
e	f	g	h
i	j	k	l

Question

- 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 output

a	b	c	d
e	f	g	h
i	j	k	l

Question

- 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 output

a	b	c	d
e	f	g	h
i	j	k	l

Question

- 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 output

a	b	c	d
e	f	g	h
i	j	k	l

Question

- 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 output

a	b	c	d
e	f	g	h
i	j	k	l

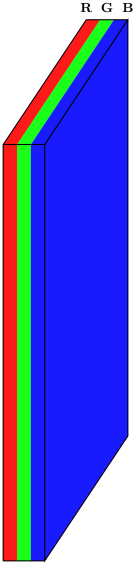
Question

- 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 output

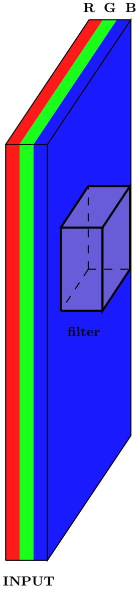
a	b	c	d
e	f	g	h
i	j	k	l

Question

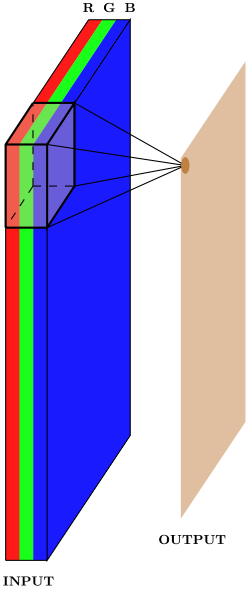
- In the 1D case, we slide a one dimensional filter over a one dimensional input
- In the 2D case, we slide a two dimensional filter over a two dimensional output
- What would happen in the 3D case?



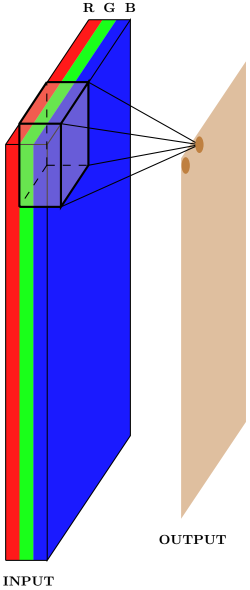
- What would a 3D filter look like?



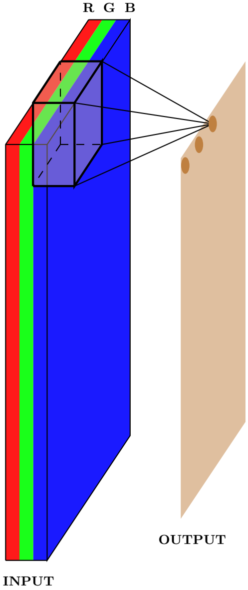
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume



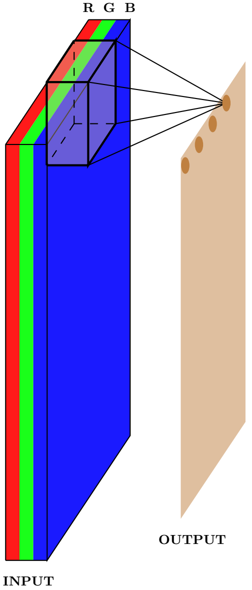
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



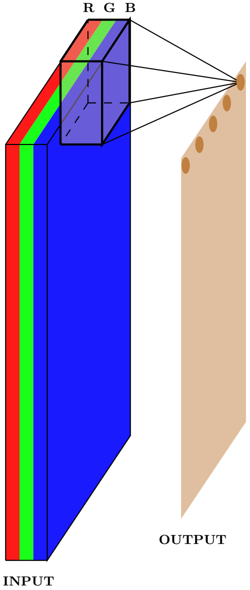
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



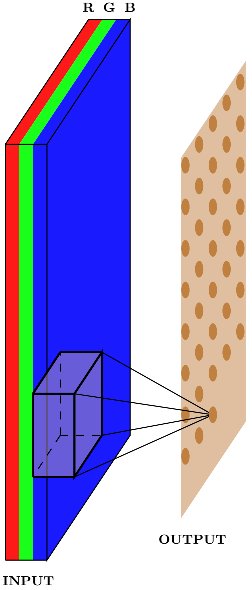
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



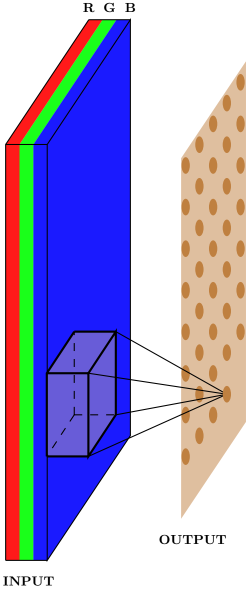
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



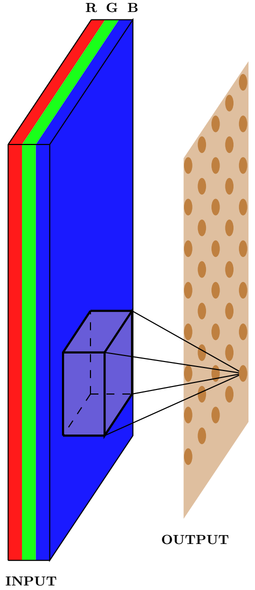
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



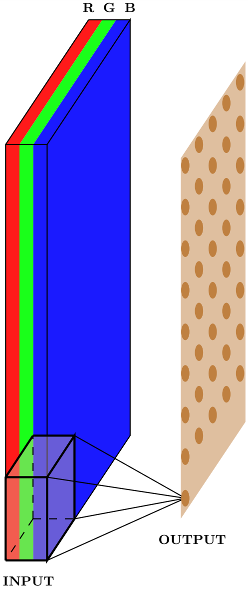
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



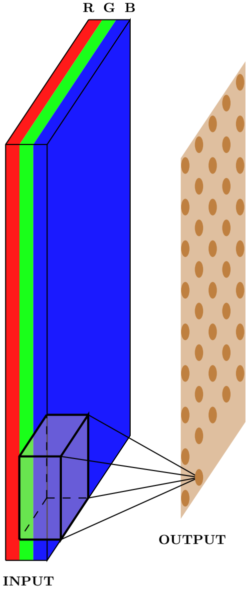
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



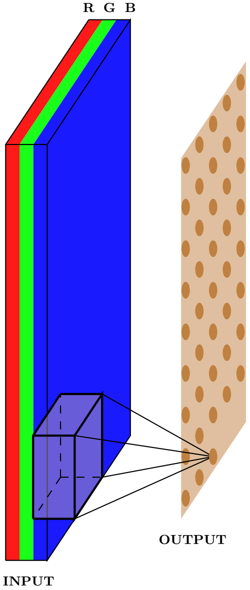
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



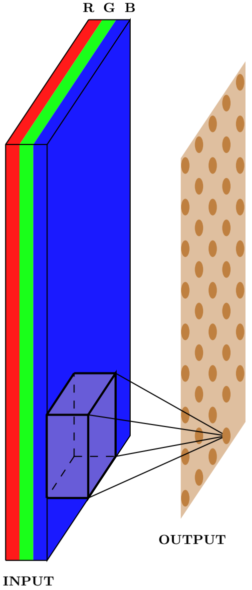
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



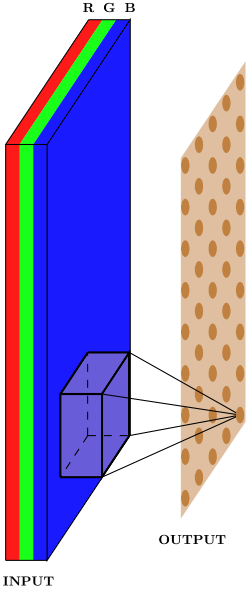
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



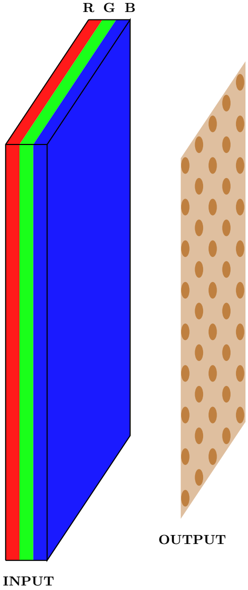
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



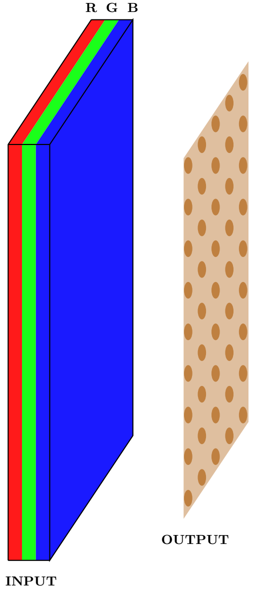
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



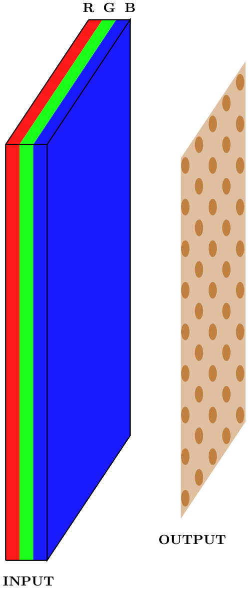
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation
- Note that in this lecture we will assume that the filter always extends to the depth of the image



- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation
- Note that in this lecture we will 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)



- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation
- Note that in this lecture we will 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)



- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation
- Note that in this lecture we will 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)
- Once again we can apply multiple filters to get multiple feature maps

Module 11.2 : Relation between input size, output size and filter size

- So far we have not said anything explicit about the dimensions of the

- So far we have not said anything explicit about the dimensions of the
  1. inputs

- So far we have not said anything explicit about the dimensions of the
  1. inputs
  2. filters

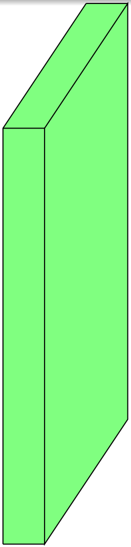
- So far we have not said anything explicit about the dimensions of the
  1. inputs
  2. filters
  3. outputs

- So far we have not said anything explicit about the dimensions of the
  1. inputs
  2. filters
  3. outputs

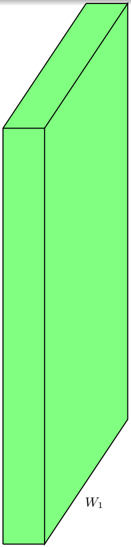
  and the relations between them

- So far we have not said anything explicit about the dimensions of the
  1. inputs
  2. filters
  3. outputs

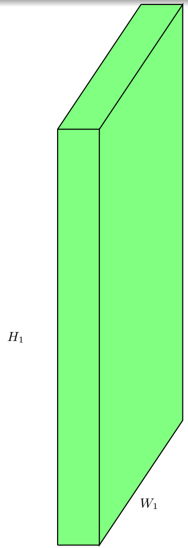
  and the relations between them
- We will see how they are related but before that we will define a few quantities



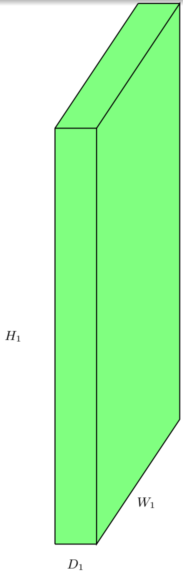
- We first define the following quantit-ies



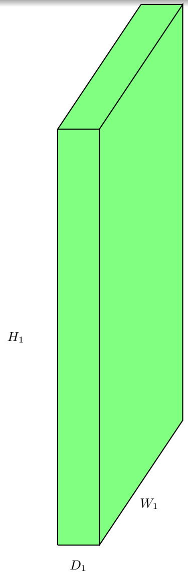
- We first define the following quantities
- Width (W_1),



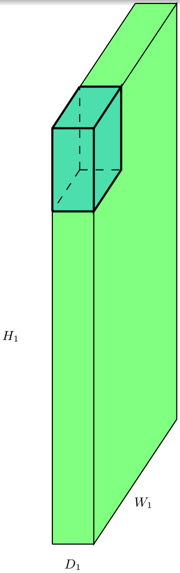
- We first define the following quantities
- Width (W_1), Height (H_1)



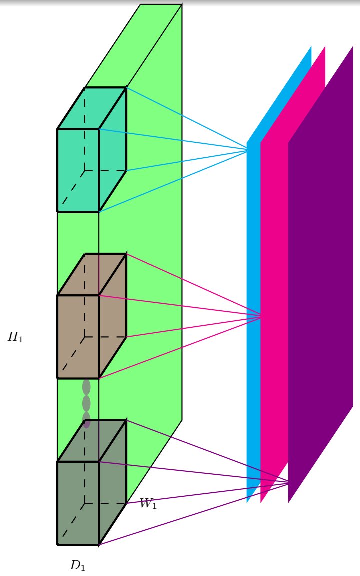
- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input



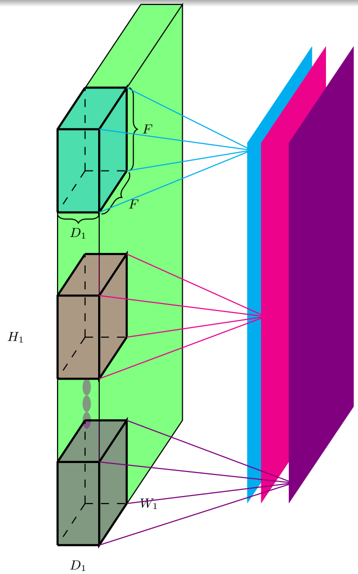
- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)



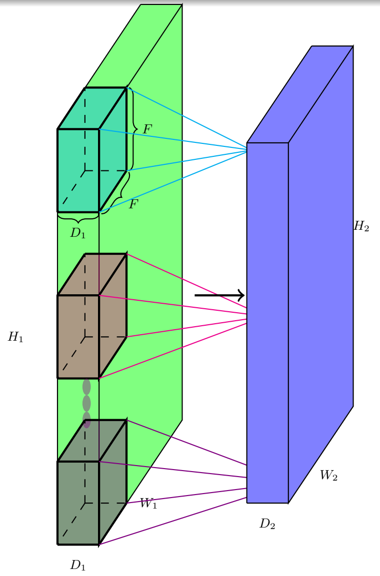
- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)



- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)
- The number of filters K

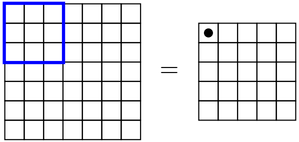


- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)
- The number of filters K
- The spatial extent (F) of each filter (the depth of each filter is same as the depth of each input)

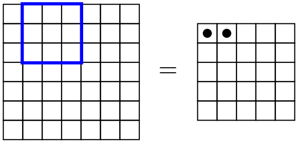


- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)
- The number of filters K
- The spatial extent (F) of each filter (the depth of each filter is same as the depth of each input)
- The output is $W_2 \times H_2 \times D_2$ (we will soon see a formula for computing W_2, H_2 and D_2)

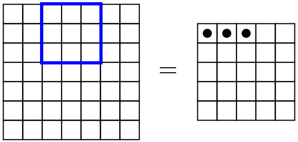
- Let us compute the dimension (W_2, H_2) of the output



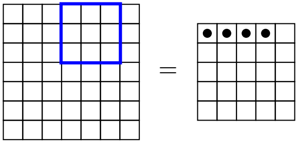
- Let us compute the dimension (W_2, H_2) of the output



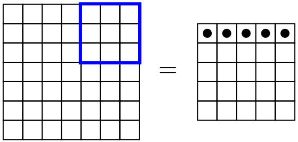
- Let us compute the dimension (W_2, H_2) of the output



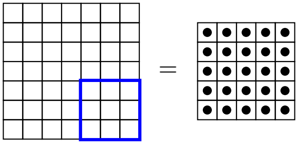
- Let us compute the dimension (W_2, H_2) of the output



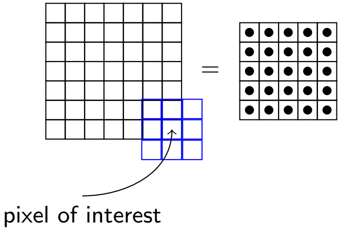
- Let us compute the dimension (W_2, H_2) of the output



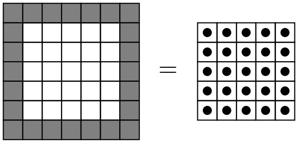
- Let us compute the dimension (W_2, H_2) of the output



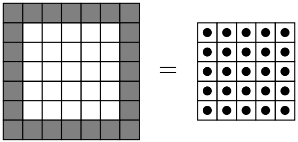
- Let us compute the dimension (W_2, H_2) of the output



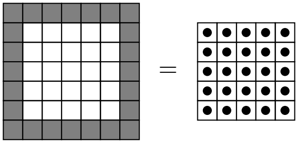
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary



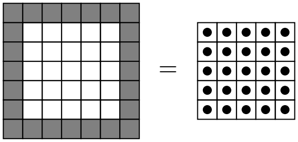
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary



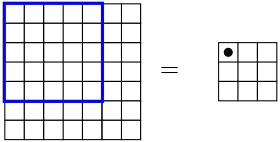
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)



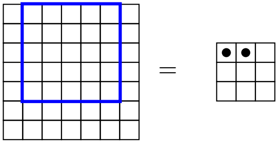
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input



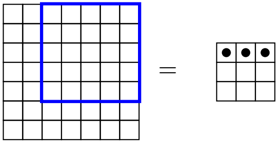
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input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



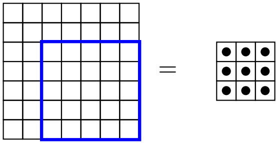
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input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



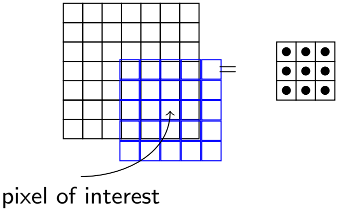
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input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



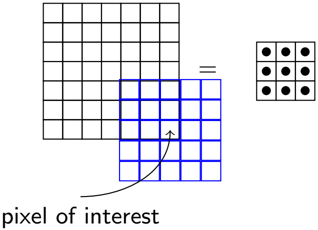
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input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



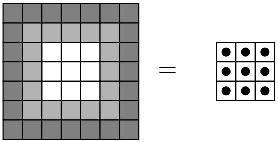
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input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



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



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



$$\textit{In general, } W_2 = W_1 - F + 1$$
$$H_2 = H_1 - F + 1$$

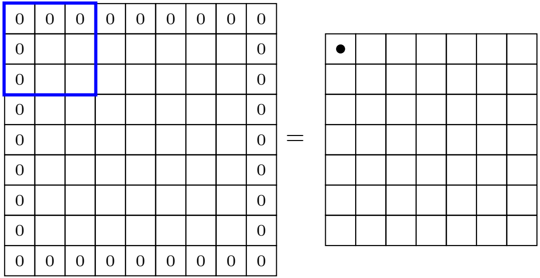
We will refine this formula further

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

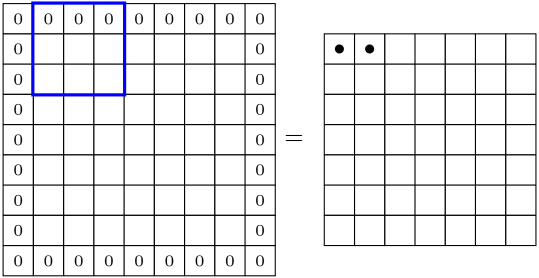
- What if we want the output to be of same size as the input?

- What if we want the output to be of same size as the input?
- We can use something known as padding

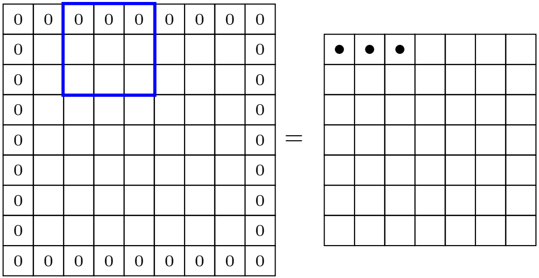
- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners



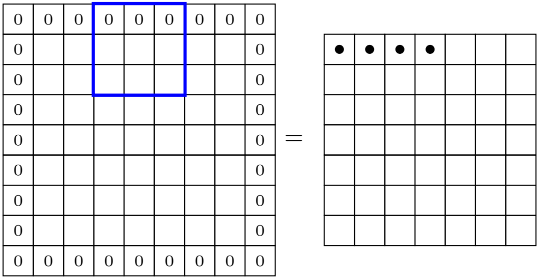
- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad $P = 1$ with a 3×3 kernel



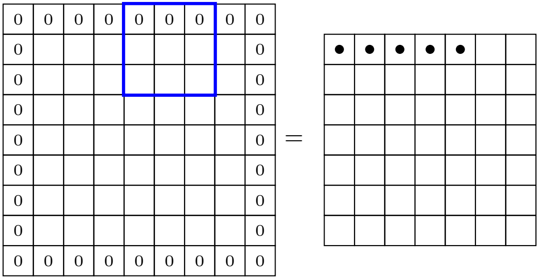
- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad P = 1 with a 3×3 kernel
- This means we will add one row and one column of 0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



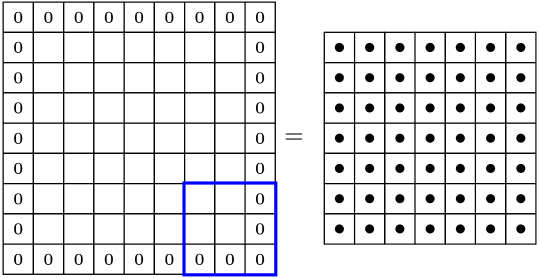
- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad P = 1 with a 3×3 kernel
- This means we will add one row and one column of 0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



- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the 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



- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the 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



- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad P = 1 with a 3 $\times$ 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,

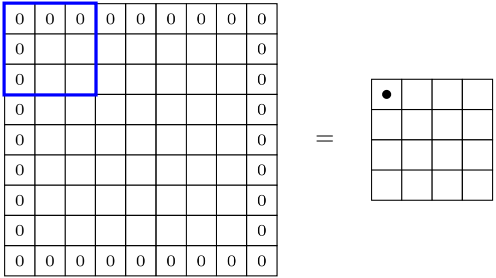
$$W_2 = W_1 - F + 2P + 1$$

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

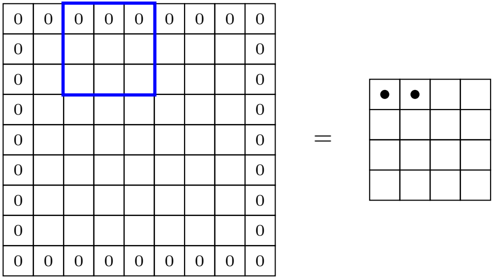
We will refine this formula further

- What does the stride S do?

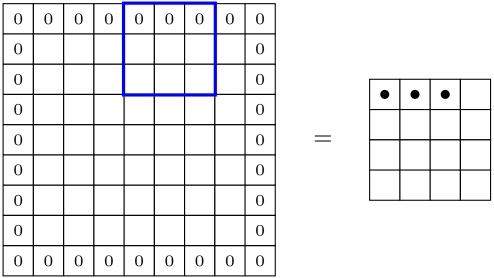
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)



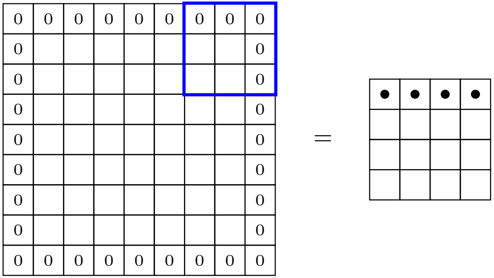
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



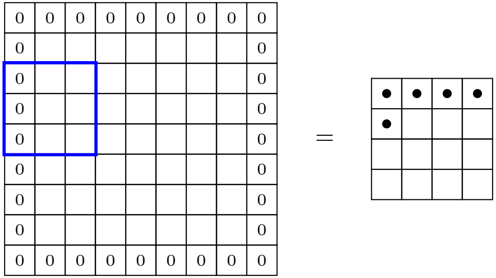
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



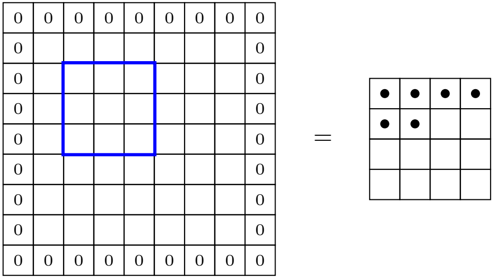
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



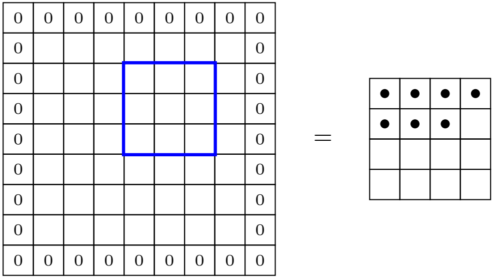
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



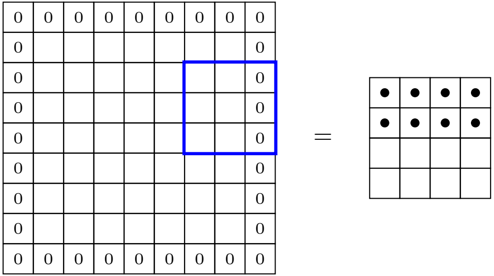
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



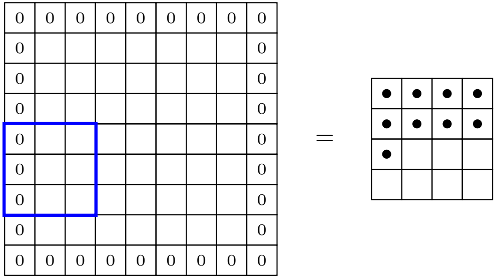
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



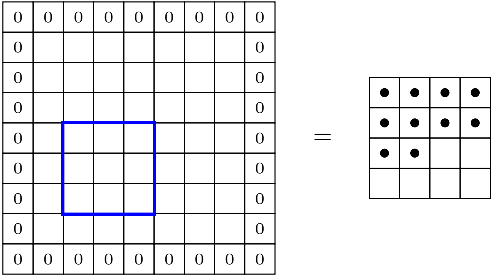
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



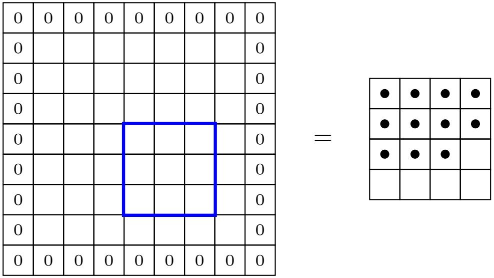
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



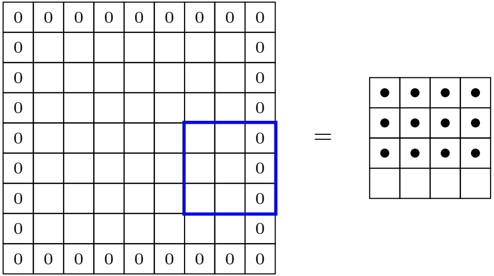
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



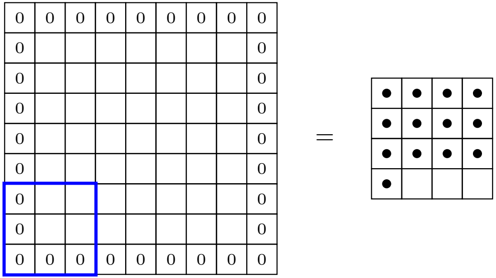
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



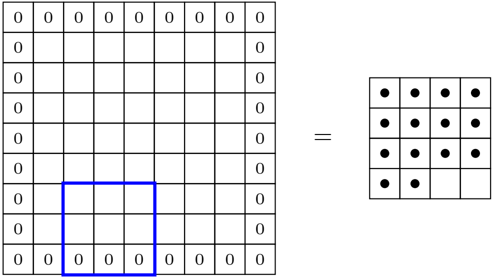
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



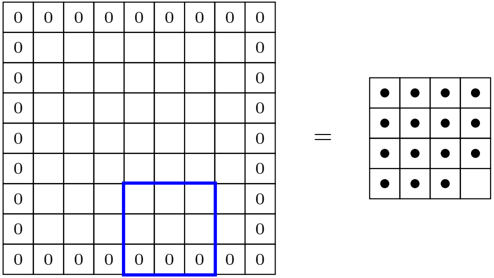
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

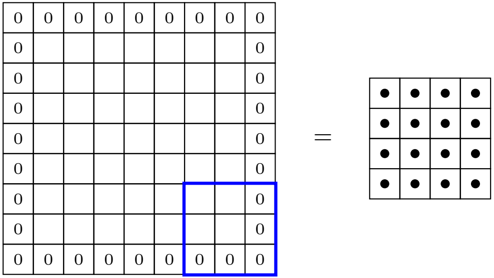


- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

So what should our final formula look like,

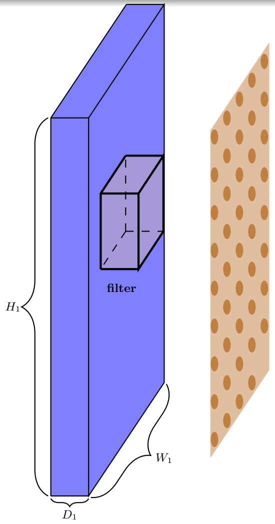


- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

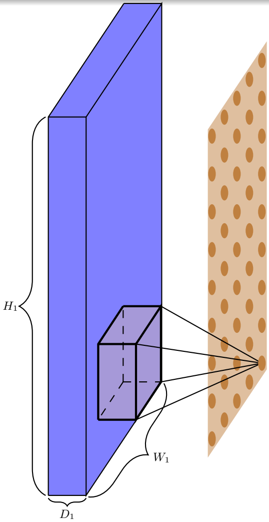
So what should our final formula look like,

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

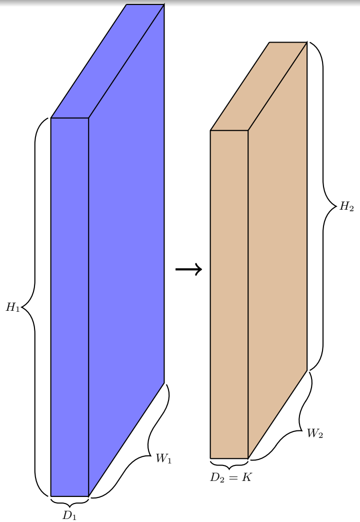
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$



- Finally, coming to the depth of the output.



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

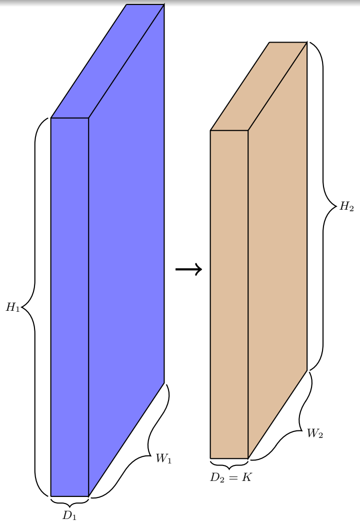


$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

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

$$D_2 = K$$

- Finally, coming to the depth of the output.
- Each filter gives us one 2D output.
- K filters will give us K such 2D outputs

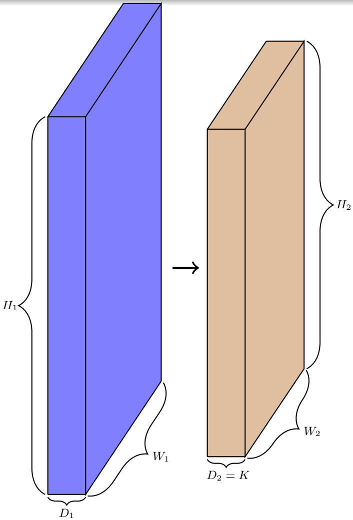


$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

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

$$D_2 = K$$

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



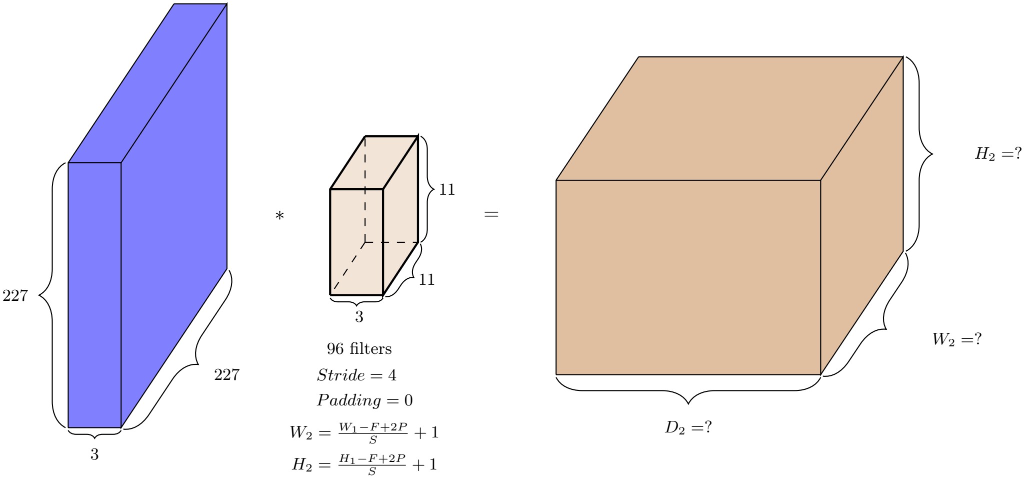
$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

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

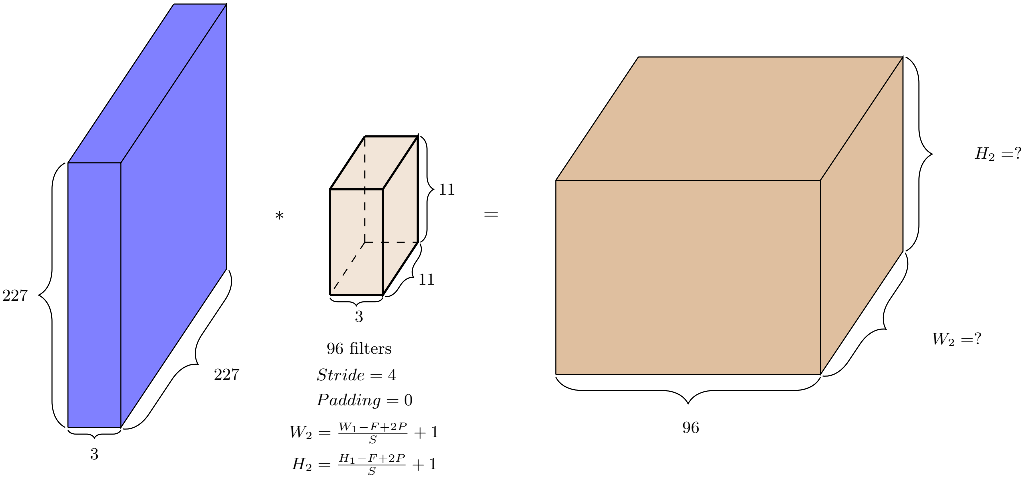
$$D_2 = K$$

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

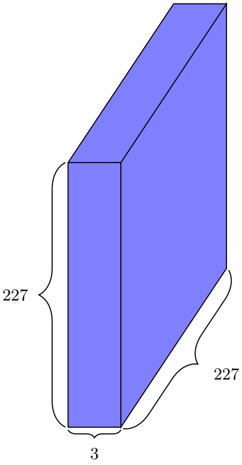
Let us do a few exercises



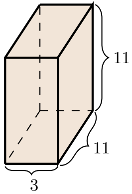
Let us do a few exercises



Let us do a few exercises



$*$



96 filters

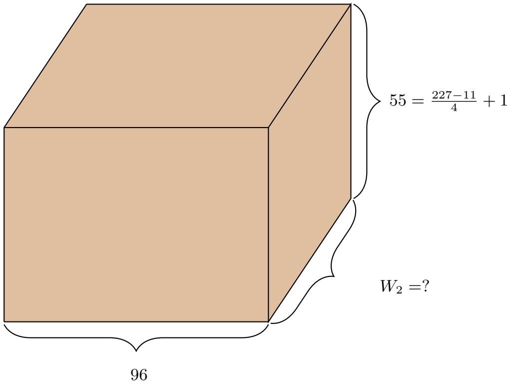
$Stride = 4$

$Padding = 0$

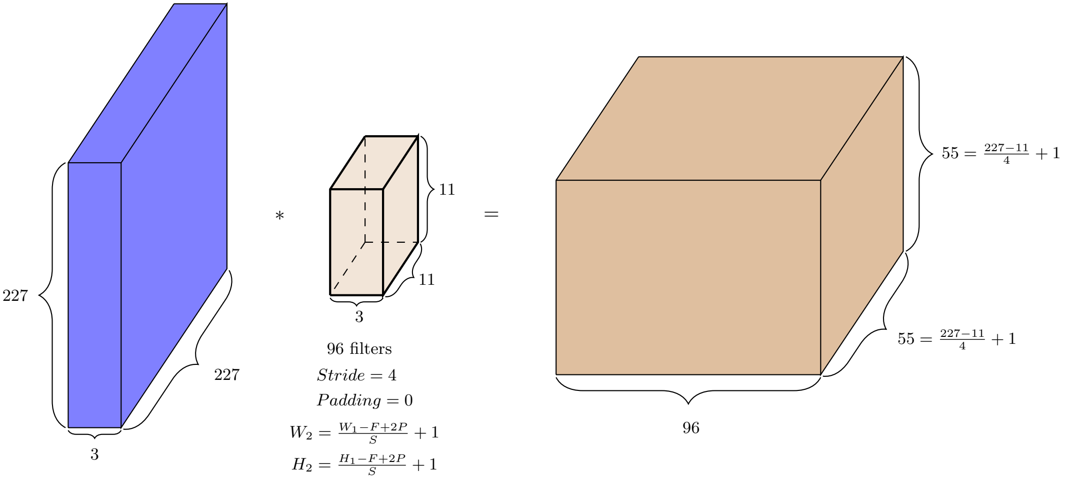
$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

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

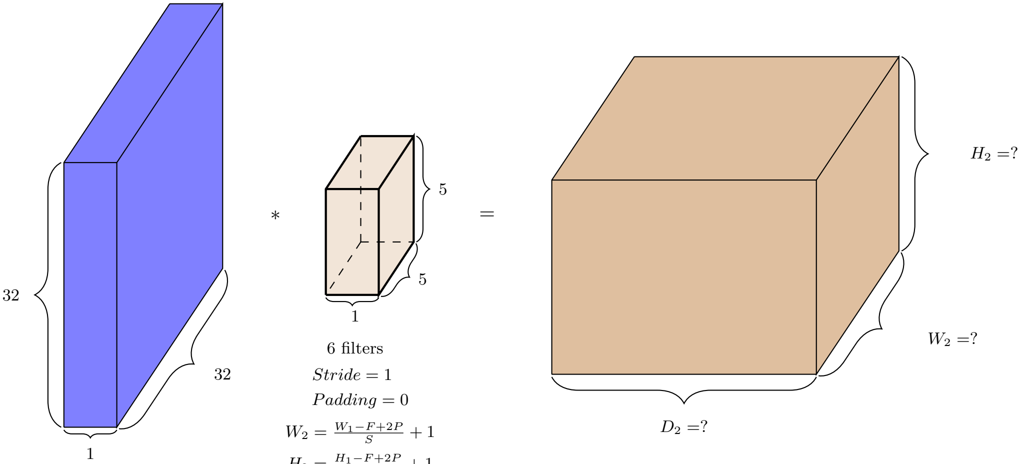
$=$



Let us do a few exercises



Let us do a few exercises



6 filters

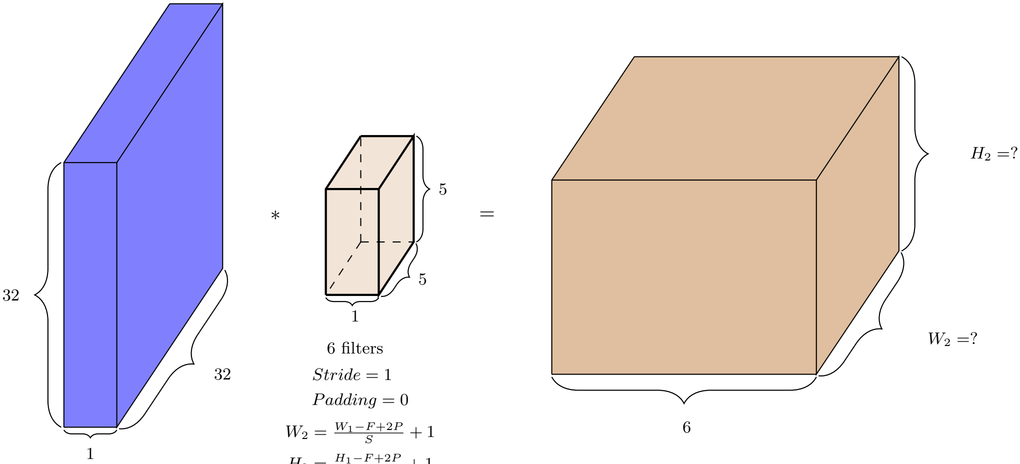
$Stride = 1$

$Padding = 0$

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

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

Let us do a few exercises



6 filters

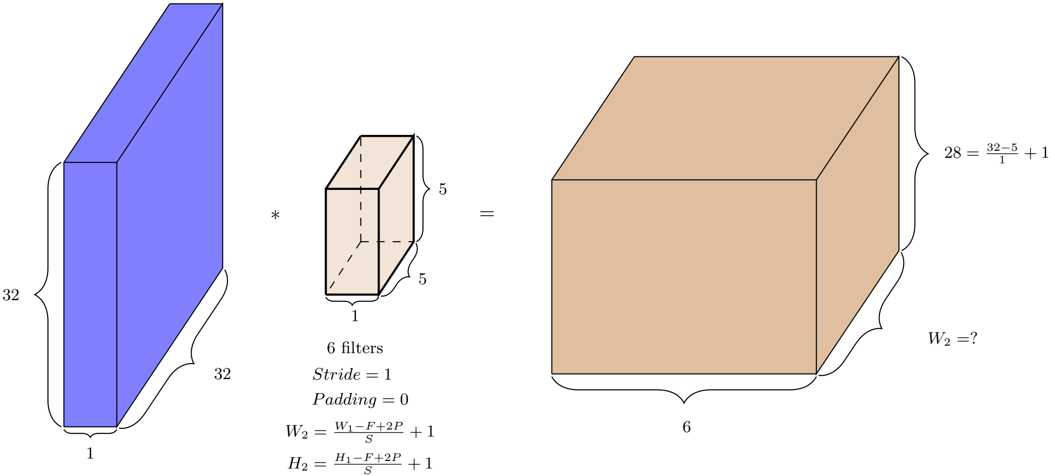
$Stride = 1$

$Padding = 0$

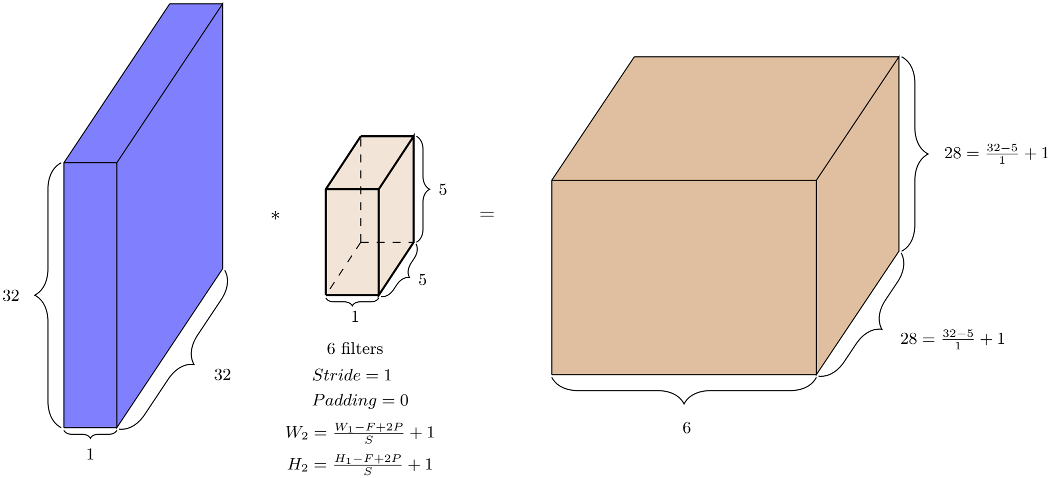
$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

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

Let us do a few exercises



Let us do a few exercises



Module 11.3 : Convolutional Neural Networks

Putting things into perspective

- What is the connection between this operation (convolution) and neural networks?

Putting things into perspective

- What is the connection between this operation (convolution) and neural networks?
- We will try to understand this by considering the task of "image classification"

Features

$\xrightarrow{Raw\ pixels}$

Features

 $\xrightarrow{\textit{Raw pixels}}$  $\longrightarrow$ car, bus, monument, flower

Features

Raw pixels $\longrightarrow$

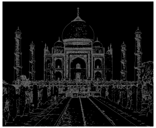
$\longrightarrow$ car, bus, monument, flower

Features

Raw pixels ⟶

⟶ car, bus, monument, flower

Edge Detector ⟶

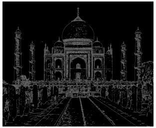


Features

Raw pixels $\longrightarrow$

$\longrightarrow$ car, bus, monument, flower

Edge Detector $\longrightarrow$



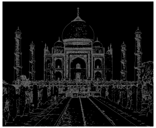
$\longrightarrow$ car, bus, monument, flower

Features

Raw pixels ⟶

⟶ car, bus, monument, flower

Edge Detector ⟶



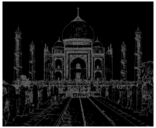
⟶ car, bus, monument, flower

Features

Raw pixels

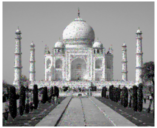
car, bus, monument, flower

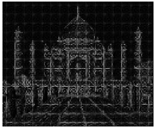
Edge Detector



car, bus, monument, flower

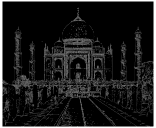
SIFT/HOG

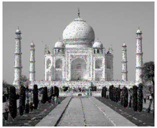
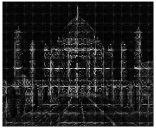




Features

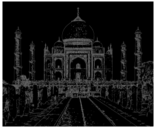
 $\xrightarrow{\textit{Raw pixels}}$  $\longrightarrow$ car, bus, monument, flower

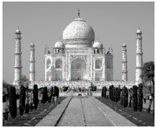
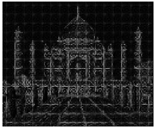
 $\xrightarrow{\textit{Edge Detector}}$  $\longrightarrow$ car, bus, monument, flower

 $\xrightarrow{\textit{SIFT/HOG}}$   $\longrightarrow$ car, bus, monument, flower

Features

 $\xrightarrow{\textit{Raw pixels}}$  $\longrightarrow$ car, bus, monument, flower

 $\xrightarrow{\textit{Edge Detector}}$  $\longrightarrow$ car, bus, monument, flower

 $\xrightarrow{\textit{SIFT/HOG}}$   $\longrightarrow$ car, bus, monument, flower

static feature extraction (no learning) | learning weights of classifier

Input	Features	Classifier

car, bus, monument, flower

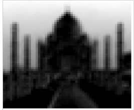
0	0	0	0	0
0	1	1	1	0
0	1	-8	1	0
0	1	1	1	0
0	0	0	0	0

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

Input | Features | Classifier

→ car, bus, monument, flower

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -8 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$



→ car, bus, monument, flower

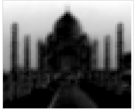
$$\begin{matrix} -1.21358689e{-}03 & 3.23652686e{-}03 & \cdots & \cdots & -2.06615720e{-}02 \\ -1.52757822e{-}03 & 2.36130832e{-}03 & \cdots & \cdots & -1.19824838e{-}02 \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ -8.25322699e{-}04 & -5.14897937e{-}03 & \cdots & \cdots & -9.90395527e{-}03 \end{matrix}$$

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

Input | Features | Classifier

 $\longrightarrow$  $\longrightarrow$ car, bus, monument, flower

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -8 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

 $\longrightarrow$  $\longrightarrow$ car, bus, monument, flower

$$\begin{matrix} \text{-1.21358689e-03} & \text{3.23652686e-03} & \cdots & \cdots & \text{-2.06615720e-02} \\ \text{-1.52757822e-03} & \text{2.36130832e-03} & \cdots & \cdots & \text{-1.19824838e-02} \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ \text{-8.25322699e-04} & \text{-5.14897937e-03} & \cdots & \cdots & \text{-9.90395527e-03} \end{matrix} \longleftarrow \text{Learn these weights}$$

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

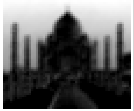
Input Features Classifier

$\longrightarrow$

$\longrightarrow$ car, bus, monument, flower

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -8 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

$\longrightarrow$



$\longrightarrow$ car, bus, monument, flower

$$\begin{matrix} -1.21358689e\text{-}03 & 3.23652686e\text{-}03 & \cdots & \cdots & -2.06615720e\text{-}02 \\ -1.52757822e\text{-}03 & 2.36130832e\text{-}03 & \cdots & \cdots & -1.19824838e\text{-}02 \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ -8.25322699e\text{-}04 & -5.14897937e\text{-}03 & \cdots & \cdots & -9.90395527e\text{-}03 \end{matrix}$$

- **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?**

Input	Features	Classifier


car, bus, monument, flower

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -8 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$


car, bus, monument, flower

$$\begin{matrix} -0.02337041 & -0.03243878 & \cdots & \cdots & -0.04728875 \\ -0.05375158 & -0.05350766 \cdots & \cdots & -0.04323674 & \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ -0.00792501 & -0.00503319 & \cdots & \cdots & 0.00174674 \end{matrix}$$

- **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?**

Input Features Classifier

car, bus, monument, flower

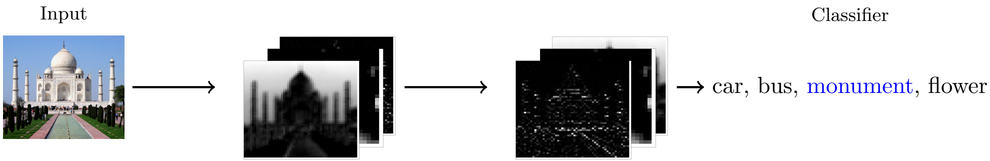
$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -8 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

car, bus, monument, flower

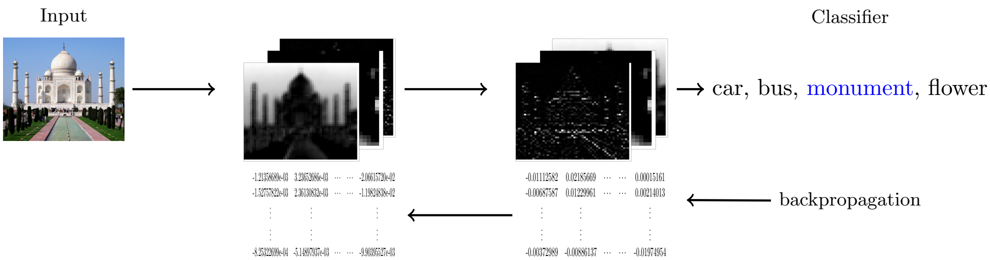
$$\begin{matrix} -0.01871333 & -0.01075948 & \cdots & \cdots & 0.04684572 \\ 0.00104325 & 0.01935937 & \cdots & \cdots & 0.01016542 \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ 0.03008777 & 0.00335217 & \cdots & \cdots & -0.02791128 \end{matrix}$$

- **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?**

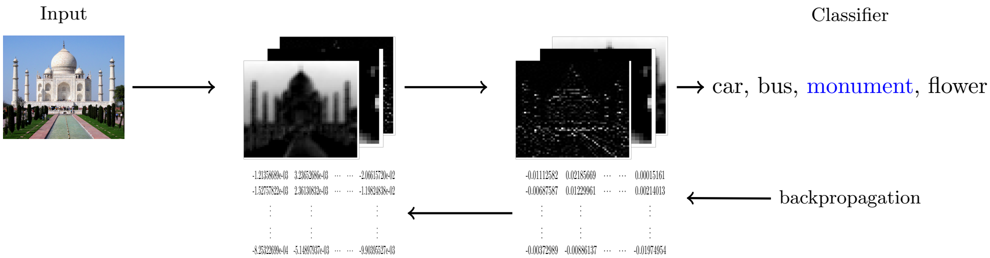
- **Can we learn multiple layers of meaningful kernels/filters in addition to learning the weights of the classifier?**



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



- **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 back propagation)



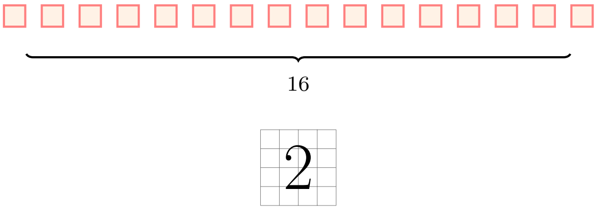
- **Can we learn multiple layers of meaningful kernels/filters in addition to learning the weights of the classifier?**
- Yes, we can !
- Simply by treating these kernels as parameters and learning them in addition to the weights of the classifier (using back propagation)
- Such a network is called a Convolutional Neural Network.

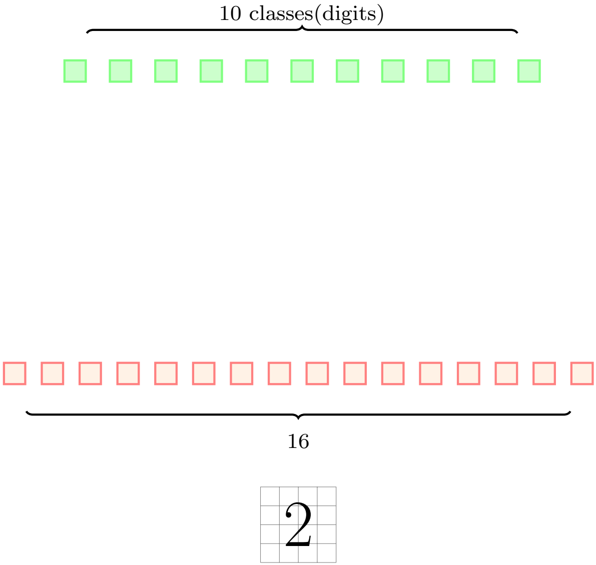
- Okay, I get it that the idea is to learn the kernel/filters by just treating them as parameters of the classification model

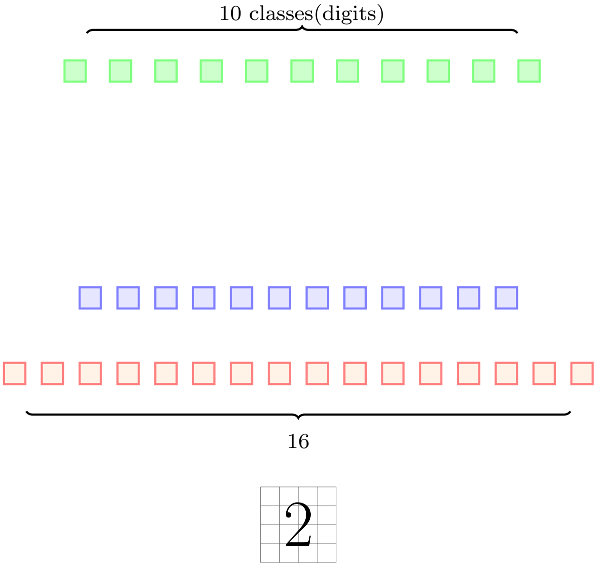
- Okay, I get it that the idea is to learn the kernel/filters by just treating them as parameters of the classification model
- But how is this different from a regular feedforward neural network

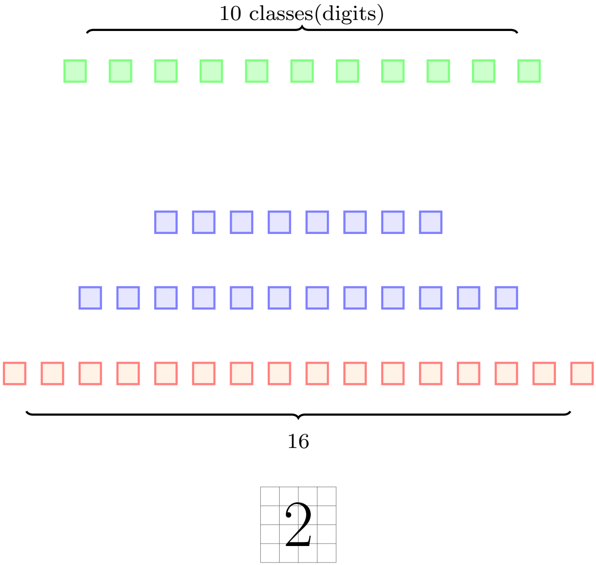
- Okay, I get it that the idea is to learn the kernel/filters by just treating them as parameters of the classification model
- But how is this different from a regular feedforward neural network
- Let us see

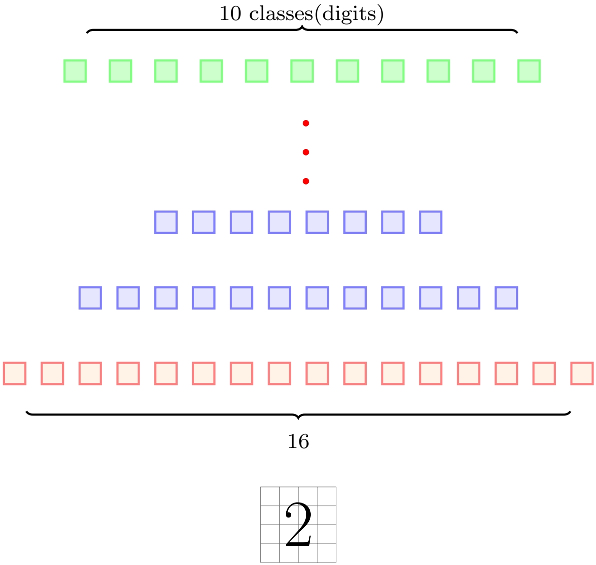

2

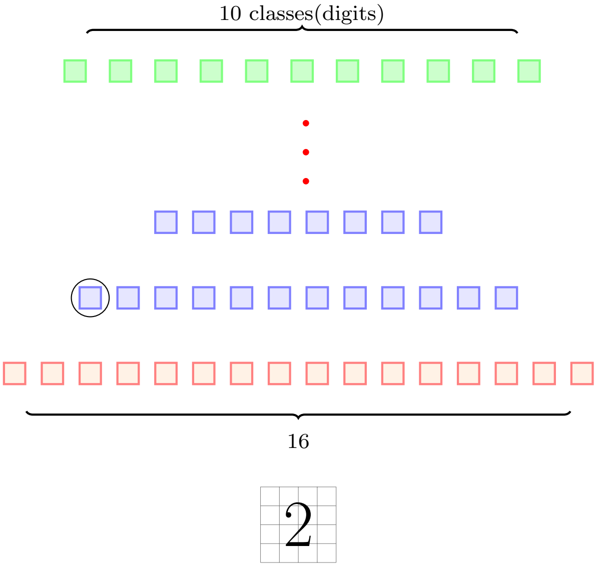

16
2

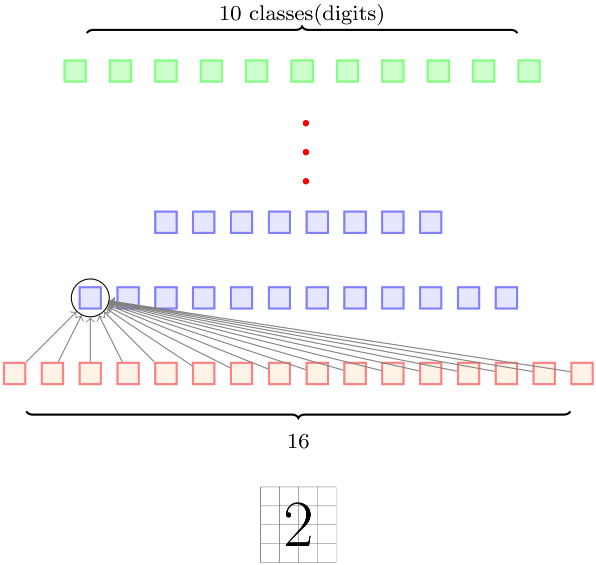

10 classes(digits)
16
2

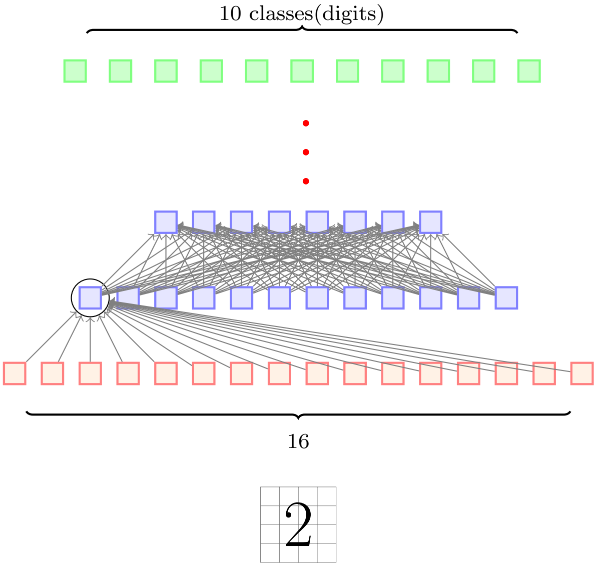

10 classes(digits)
16
2

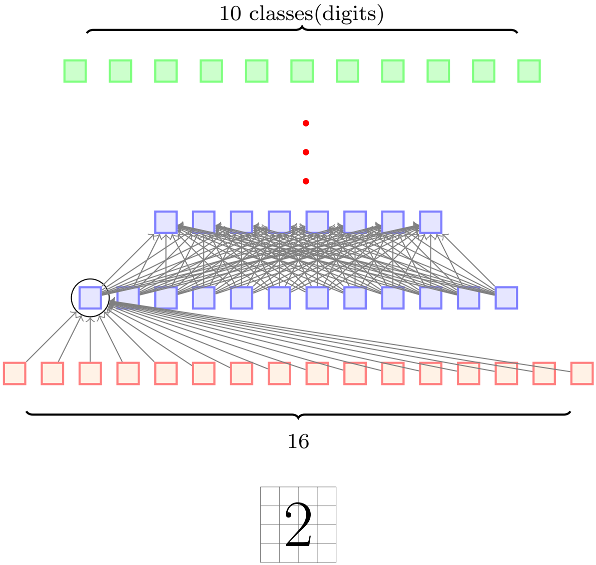

10 classes(digits)
16
2


10 classes(digits)
16
2

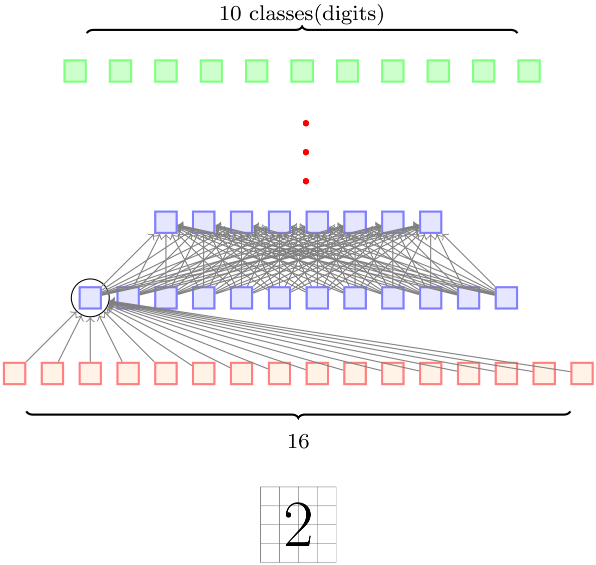

10 classes(digits)
16
2


10 classes(digits)
16
2

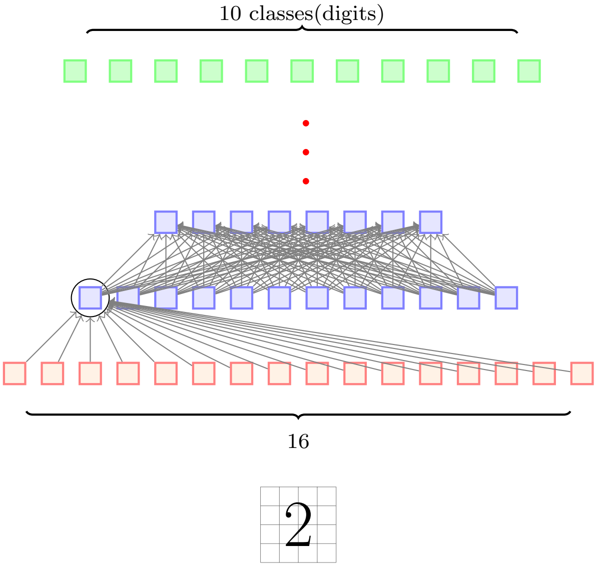

10 classes(digits)
16
2



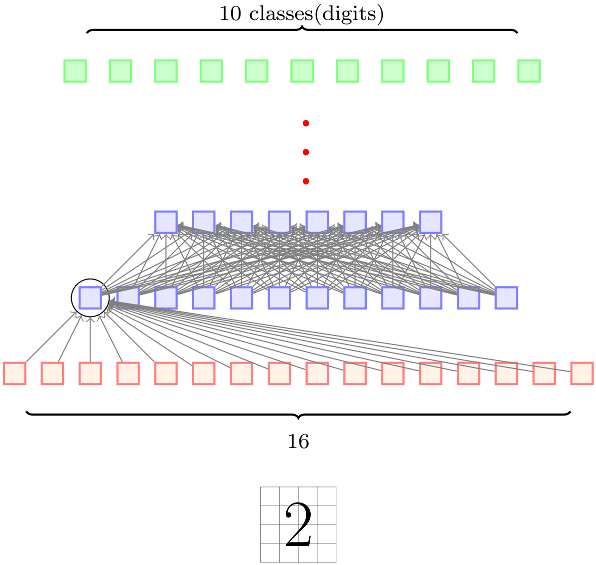
- This is what a regular feed-forward neural network will look like



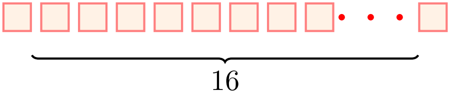
- This is what a regular feed-forward neural network will look like
- There are many dense connections here

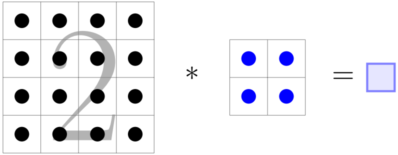


- This is what a regular feed-forward neural network will look like
- There are many dense connections here
- For example all the 16 input neurons are contributing to the computation of h_{11}

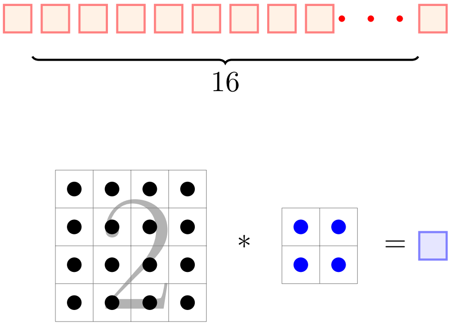


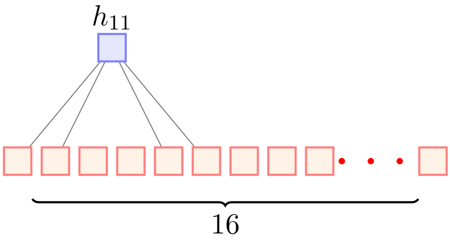
- This is what a regular feed-forward neural network will look like
- There are many dense connections here
- For example all the 16 input neurons are contributing to the computation of h_{11}
- Contrast this to what happens in the case of convolution

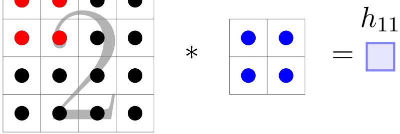

16



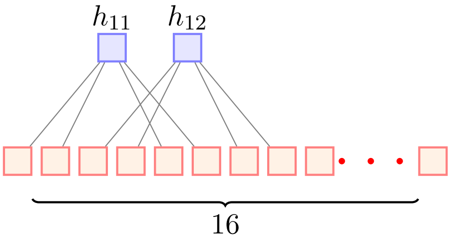
- Only a few local neurons participate in the computation of h_{11}

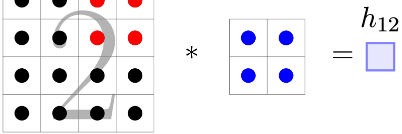




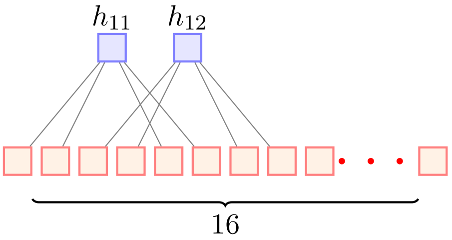


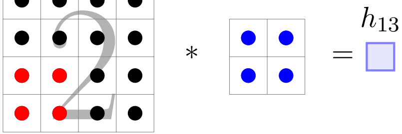
- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}



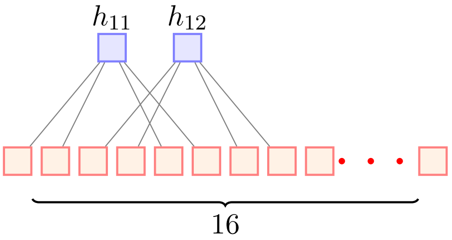


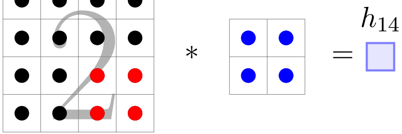
- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}



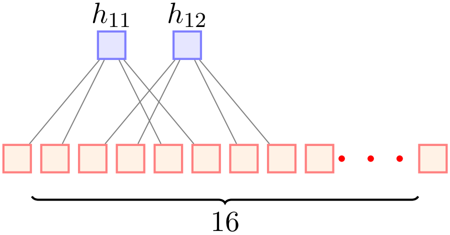


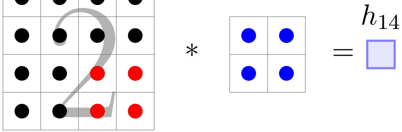
- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}



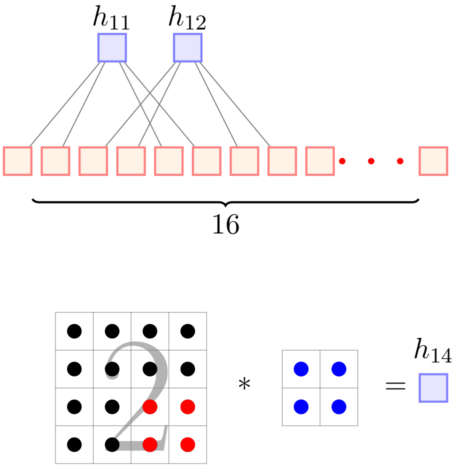


- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}

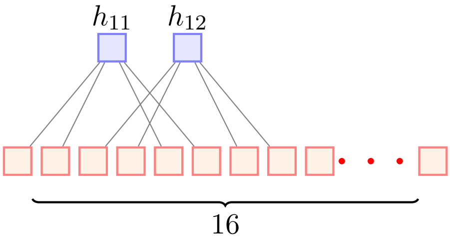


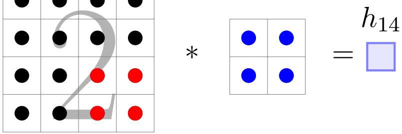


- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}
- The connections are much sparser



- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}
- The connections are much sparser
- We are taking advantage of the structure of the image(interactions between neighboring pixels are more interesting)





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

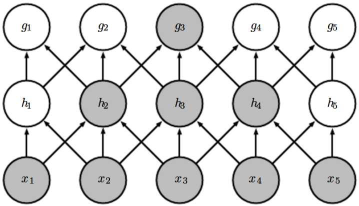
- But is sparse connectivity really good thing ?

---

* Goodfellow-et-al-2016

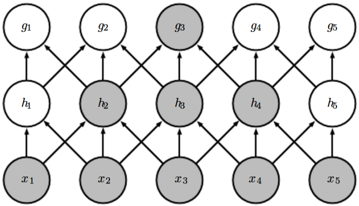
- But is sparse connectivity really good thing ?
- Aren't we losing information (by losing interactions between some input pixels)

* Goodfellow-et-al-2016



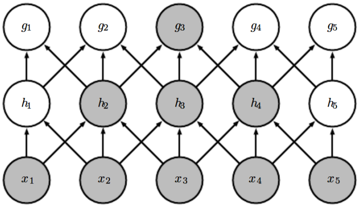
- But is sparse connectivity really good thing ?
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really

* Goodfellow-et-al-2016



- But is sparse connectivity really 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

* Goodfellow-et-al-2016



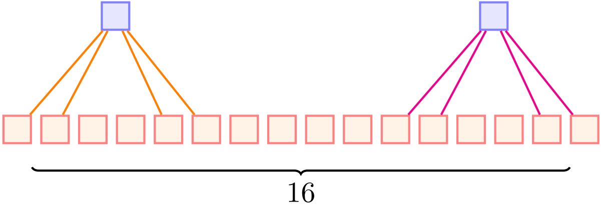
- But is sparse connectivity really good thing ?
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really
- The two highlighted neurons (x_1 & x_5)* do not interact in *layer* 1
- But they indirectly contribute to the computation of g_3 and hence interact indirectly

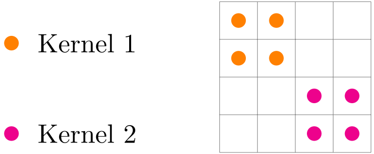
---

* Goodfellow-et-al-2016

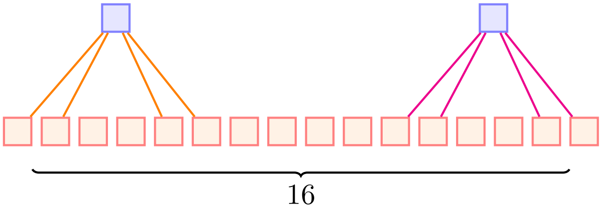
- Another characteristic of CNNs is **weight sharing**

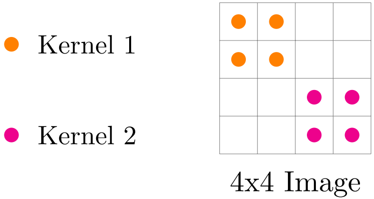
- Another characteristic of CNNs is **weight sharing**
- Consider the following network



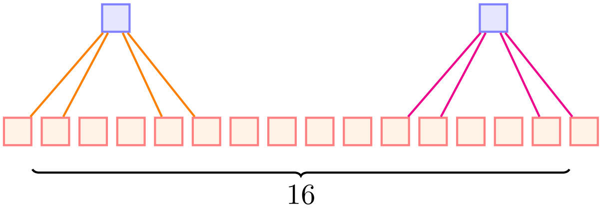


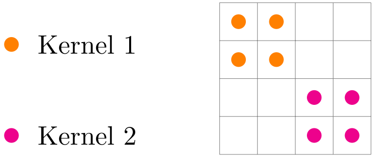
- Another characteristic of CNNs is **weight sharing**
- Consider the following network





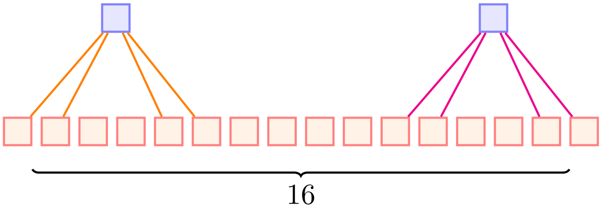
- Another characteristic of CNNs is **weight sharing**
- Consider the following network
- Do we want the kernel weights to be different for different portions of the image?

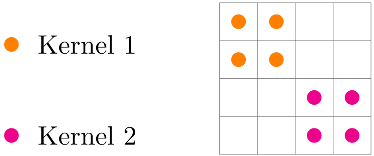




4x4 Image

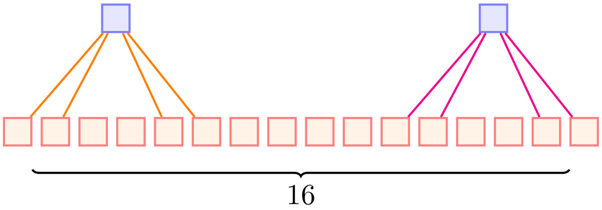
- Another characteristic of CNNs is **weight sharing**
- Consider the following network
- Do we want the kernel weights to be different for different portions of the image?
- Imagine that we are trying to learn a kernel that detects edges



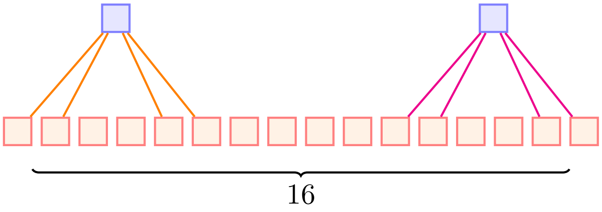


4x4 Image

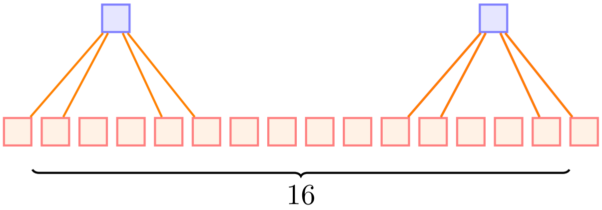
- Another characteristic of CNNs is **weight sharing**
- Consider the following network
- Do we want the kernel weights to be different for different portions of the image?
- Imagine that we are trying to learn a kernel that detects edges
- Shouldn't we be applying the same kernel at all the portions of the image?



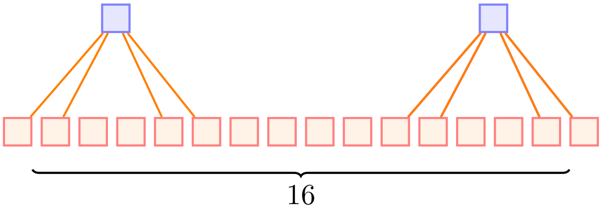
- In other words shouldn't the *orange* and *pink* kernels be the same



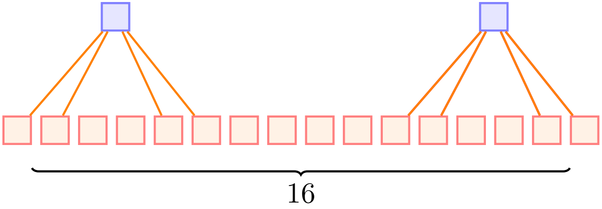
- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed



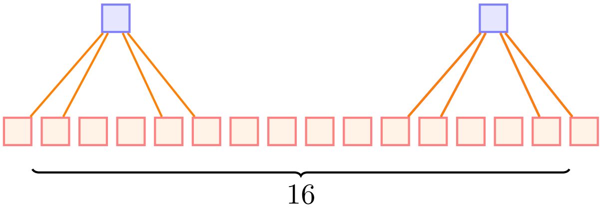
- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed



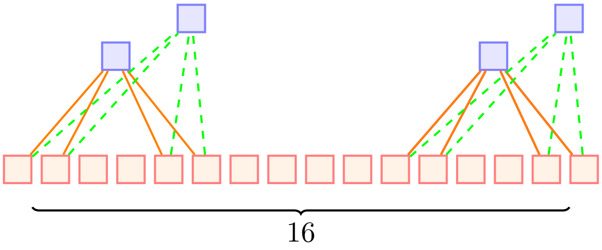
- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)



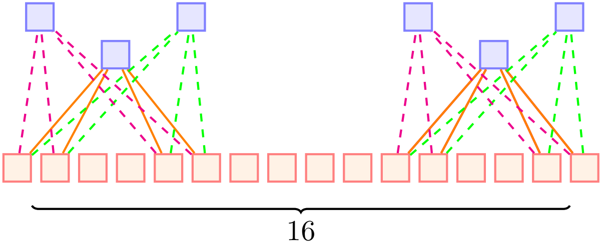
- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?



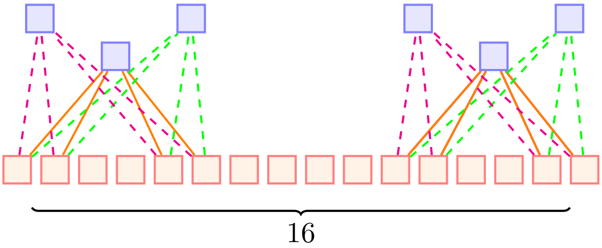
- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image



- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image



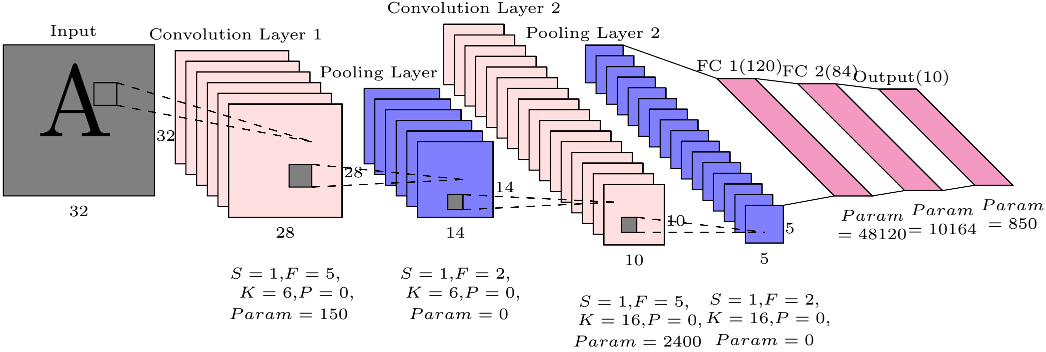
- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image

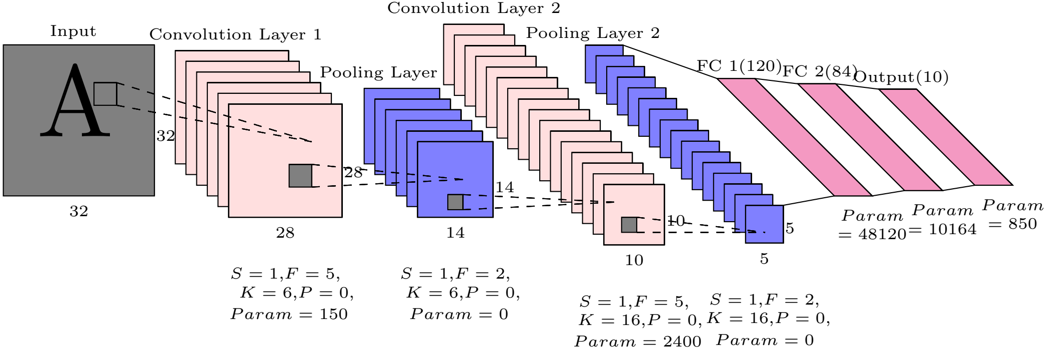


- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image
- This is called "weight sharing"

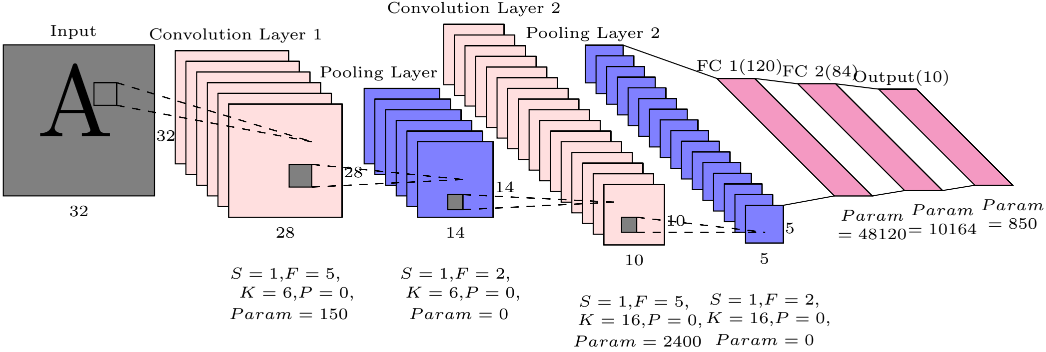
- So far, we have focused only on the convolution operation

- So far, we have focused only on the convolution operation
- Let us see what a full convolutional neural network looks like

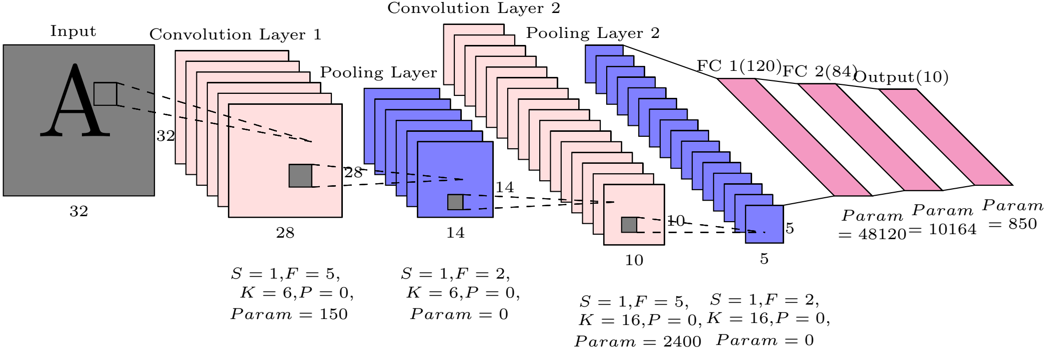

Input
Convolution Layer 1
Convolution Layer 2
Pooling Layer
Pooling Layer 2
FC 1(120)
FC 2(84)
Output(10)
32
32
28
28
14
14
10
10
5
5
$S = 1, F = 5,$
$K = 6, P = 0,$
$Param = 150$
$S = 1, F = 2,$
$K = 6, P = 0,$
$Param = 0$
$S = 1, F = 5,$
$K = 16, P = 0,$
$Param = 2400$
$S = 1, F = 2,$
$K = 16, P = 0,$
$Param = 0$
$Param = 48120$
$Param = 10164$
$Param = 850$



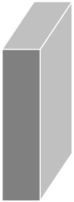
- It has alternate convolution and pooling layers



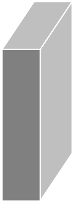
- It has alternate convolution and pooling layers
- What does a pooling layer do?



- It has alternate convolution and pooling layers
- What does a pooling layer do?
- Let us see

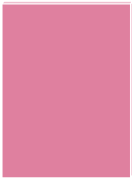


Input

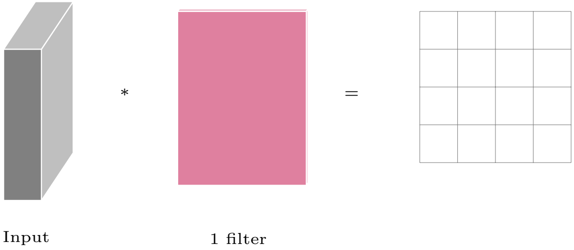


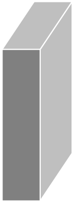
Input

$*$



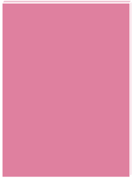
1 filter


*
=
Input
1 filter



Input

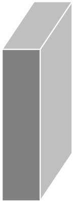
$*$



1 filter

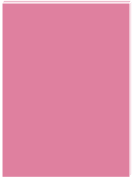
$=$

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



Input

$*$

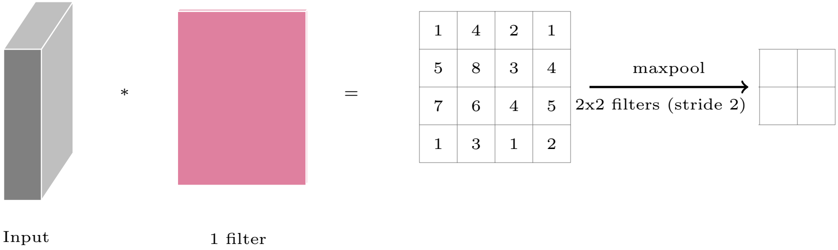


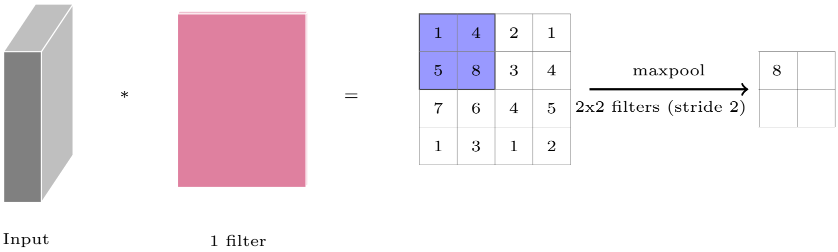
1 filter

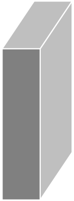
$=$

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

$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

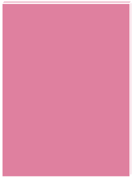

*
=
1 4 2 1
5 8 3 4
7 6 4 5
1 3 1 2
maxpool
2x2 filters (stride 2)
Input
1 filter


*
=
1 4 2 1
5 8 3 4
7 6 4 5
1 3 1 2
maxpool
2x2 filters (stride 2)
8
Input
1 filter



Input

*



1 filter

=

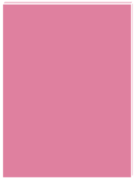
1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4

Input

*



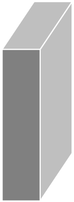
1 filter

=

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

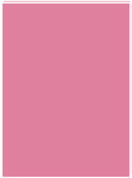
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	



Input

*



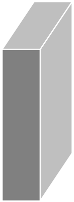
1 filter

=

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

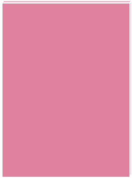
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5



Input

*



1 filter

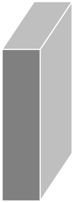
=

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

$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

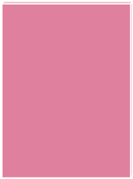
8	4
7	5

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



Input

$*$



1 filter

$=$

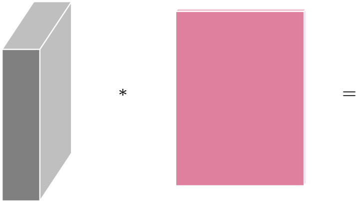
1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

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

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$



Input

1 filter

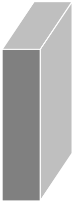
1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

maxpool →
2x2 filters (stride 2)

8	4
7	5

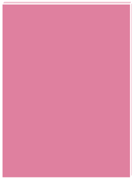
1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

maxpool →
2x2 filters (stride 1)



Input

$*$



1 filter

$=$

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

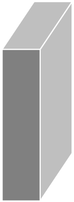
maxpool →
2x2 filters (stride 2)

8	4
7	5

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

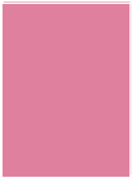
maxpool →
2x2 filters (stride 1)

8		



Input

*



1 filter

=

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

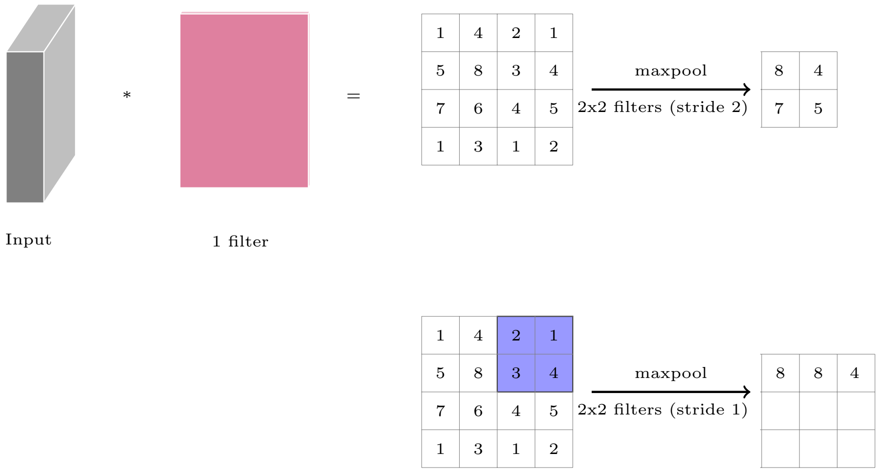
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

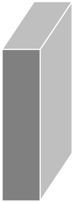
8	4
7	5

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

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

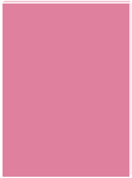
8	8	


Input
*
1 filter
=
1 4 2 1
5 8 3 4
7 6 4 5
1 3 1 2
maxpool
2x2 filters (stride 2)
8 4
7 5
1 4 2 1
5 8 3 4
7 6 4 5
1 3 1 2
maxpool
2x2 filters (stride 1)
8 8 4



Input

*



1 filter

=

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

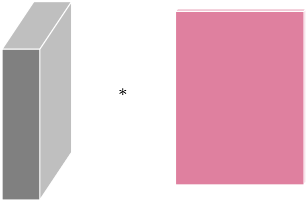
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

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

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

8	8	4
8		



Input * 1 filter =

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

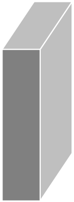
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

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

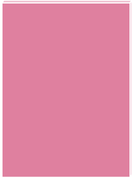
$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

8	8	4
8	8	



Input

$*$



1 filter

$=$

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

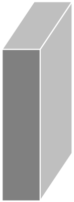
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

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

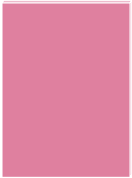
$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

8	8	4
8	8	5



Input

*



1 filter

=

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

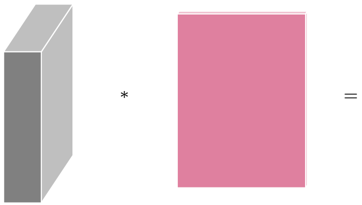
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

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

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

8	8	4
8	8	5
7		



Input

1 filter

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

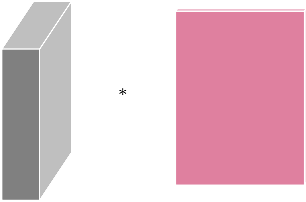
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

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

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

8	8	4
8	8	5
7	6	



Input   *   1 filter   =

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

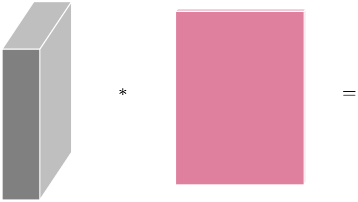
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

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

$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

8	8	4
8	8	5
7	6	5



Input

1 filter

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

$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

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

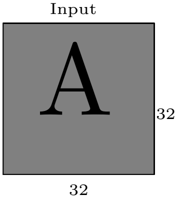
$\xrightarrow[\text{2x2 filters (stride 1)}]{\text{maxpool}}$

8	8	4
8	8	5
7	6	5

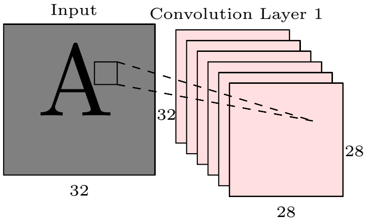
- Instead of max pooling we can also do average pooling

We will now see some case studies where convolution neural networks have been successful

LeNet-5 for handwritten character recognition

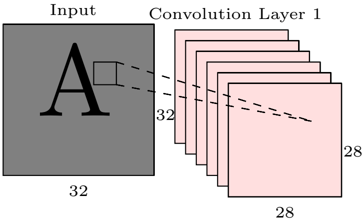


LeNet-5 for handwritten character recognition



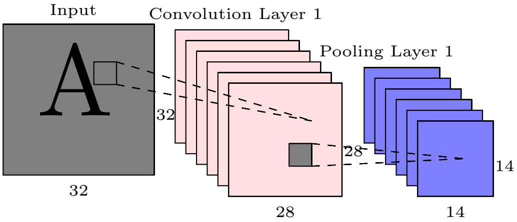
$S = 1, F = 5,$
$K = 6, P = 0,$
$Param = ?$

LeNet-5 for handwritten character recognition



$S = 1, F = 5,$
$K = 6, P = 0,$
$Param = 150$

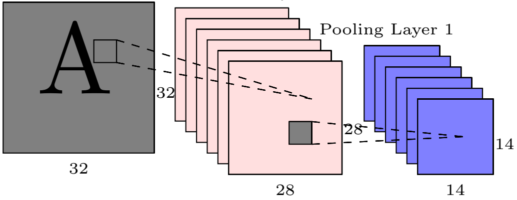
LeNet-5 for handwritten character recognition



$S = 1, F = 5,$
$K = 6, P = 0,$
$Param = 150$

$S = 1, F = 2,$
$K = 6, P = 0,$
$Param = ?$

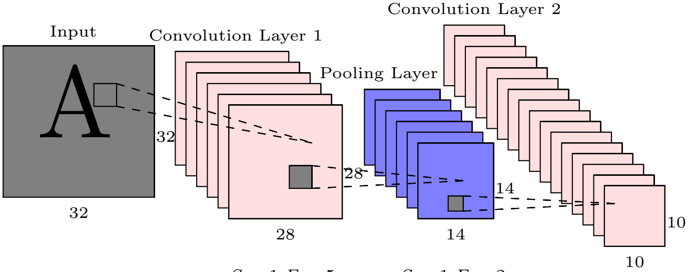
LeNet-5 for handwritten character recognition



$S = 1, F = 5,$
$K = 6, P = 0,$
$Param = 150$

$S = 1, F = 2,$
$K = 6, P = 0,$
$Param = 0$

LeNet-5 for handwritten character recognition

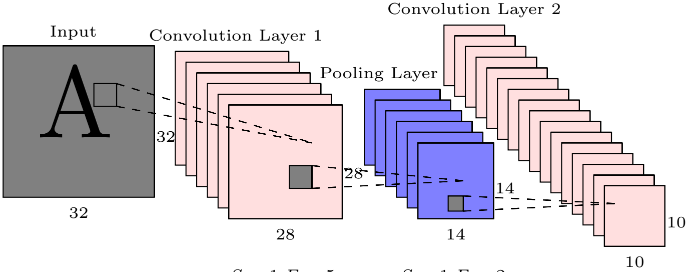


$S = 1, F = 5,$
$K = 6, P = 0,$
$Param = 150$

$S = 1, F = 2,$
$K = 6, P = 0,$
$Param = 0$

$S = 1, F = 5,$
$K = 16, P = 0,$
$Param = ?$

LeNet-5 for handwritten character recognition

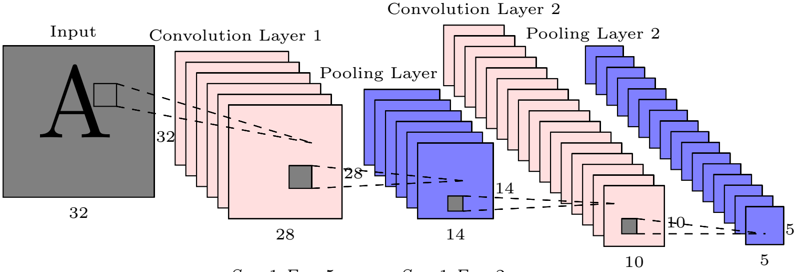


$S = 1, F = 5,$
$K = 6, P = 0,$
$Param = 150$

$S = 1, F = 2,$
$K = 6, P = 0,$
$Param = 0$

$S = 1, F = 5,$
$K = 16, P = 0,$
$Param = 2400$

LeNet-5 for handwritten character recognition



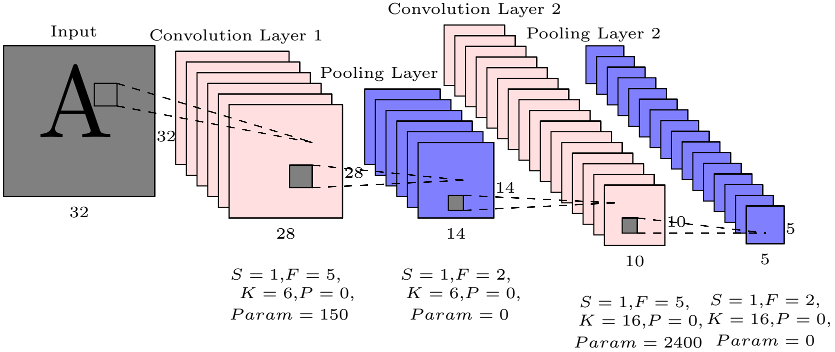
$S = 1, F = 5,$
$K = 6, P = 0,$
$Param = 150$

$S = 1, F = 2,$
$K = 6, P = 0,$
$Param = 0$

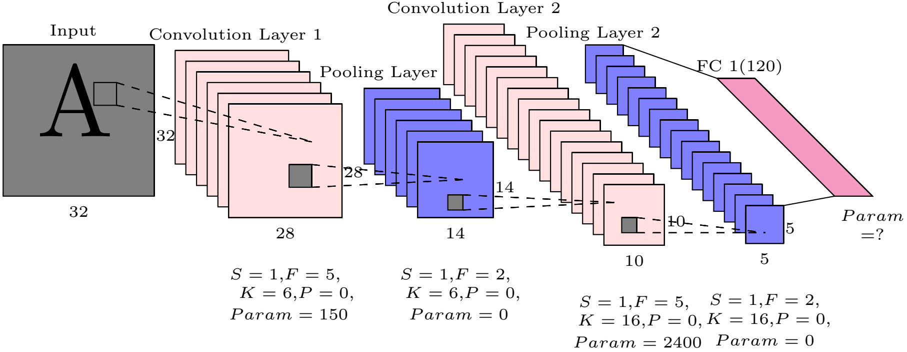
$S = 1, F = 5,$
$K = 16, P = 0,$
$Param = 2400$

$S = 1, F = 2,$
$K = 16, P = 0,$
$Param = ?$

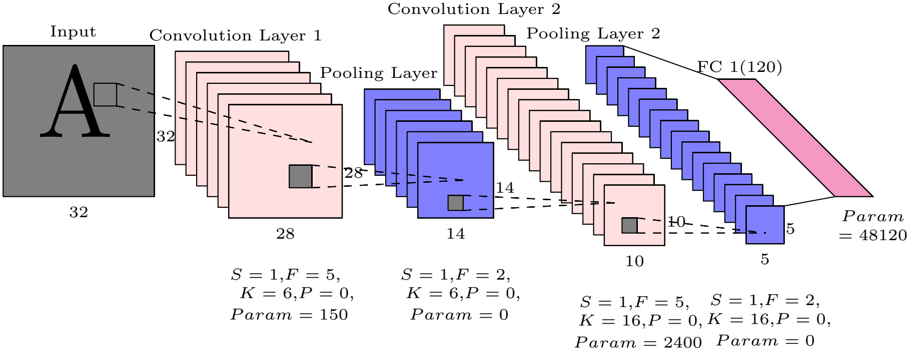
LeNet-5 for handwritten character recognition



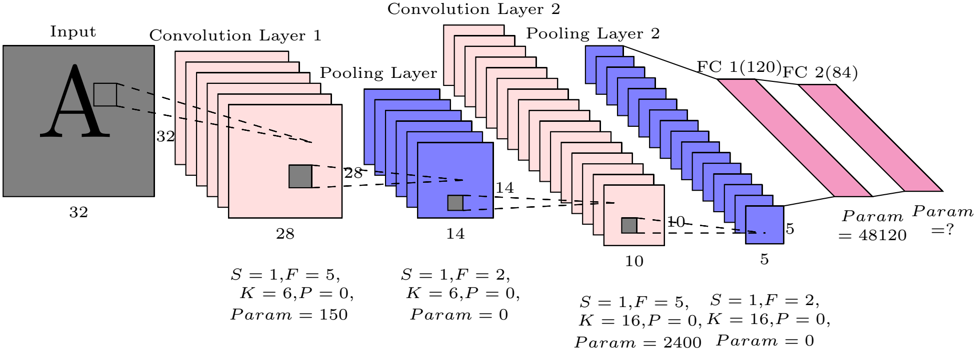
LeNet-5 for handwritten character recognition



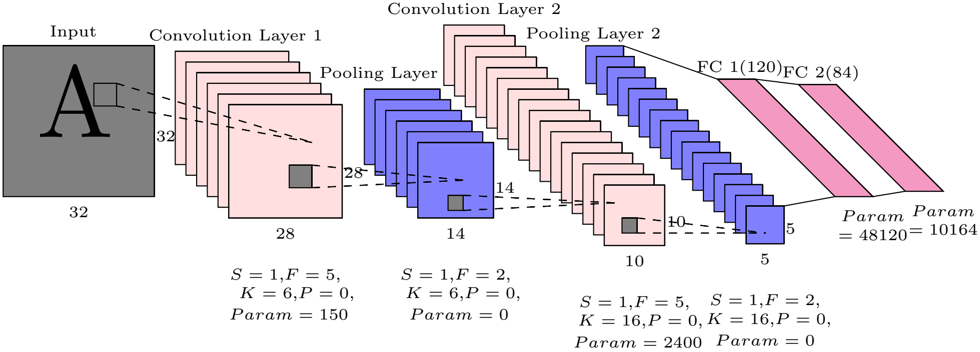
LeNet-5 for handwritten character recognition



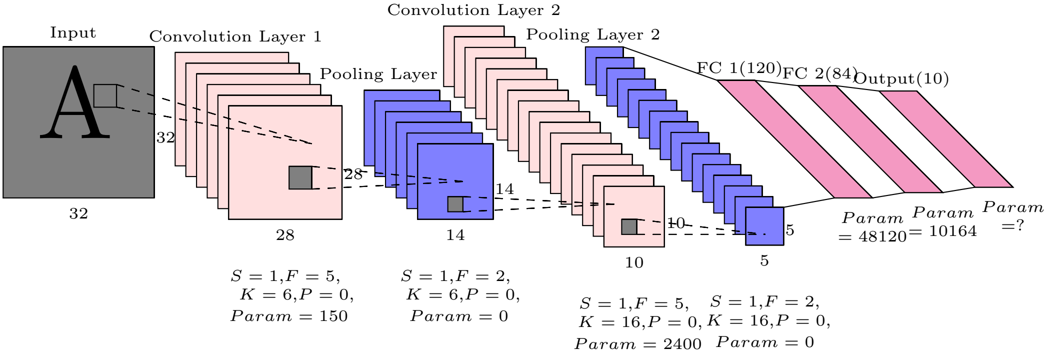
LeNet-5 for handwritten character recognition



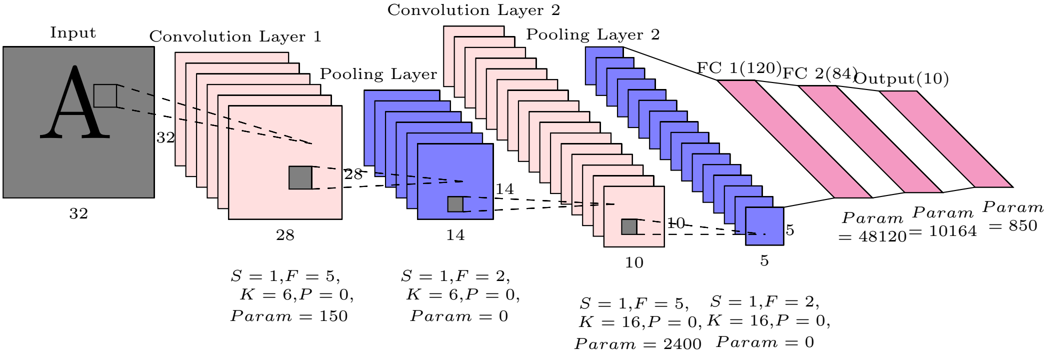
LeNet-5 for handwritten character recognition



LeNet-5 for handwritten character recognition



LeNet-5 for handwritten character recognition



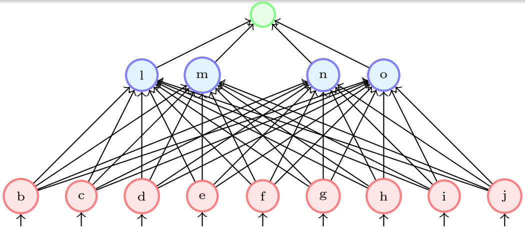
- How do we train a convolutional neural network ?

Input

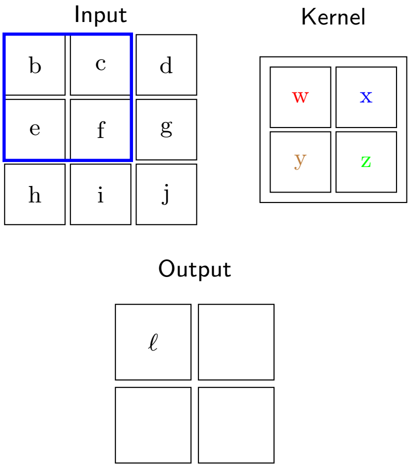
b	c	d
e	f	g
h	i	j

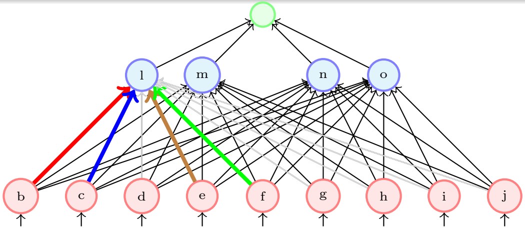
Kernel

w	x
y	z

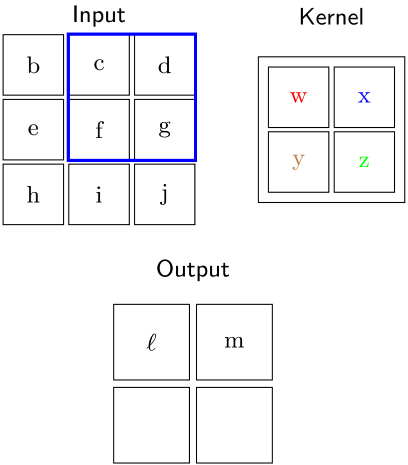


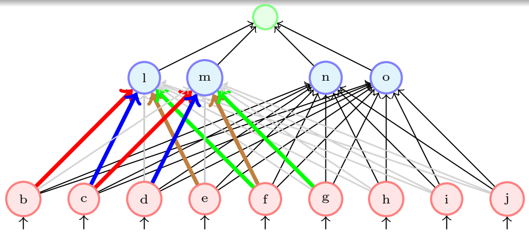
- A CNN can be implemented as a feedforward neural network



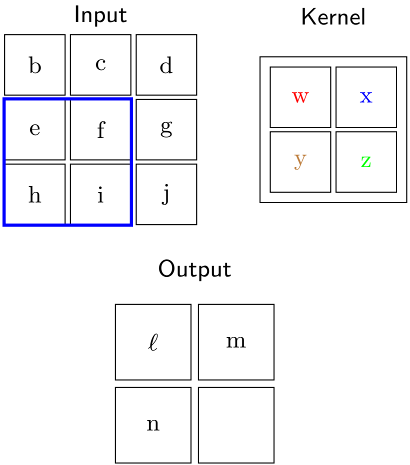


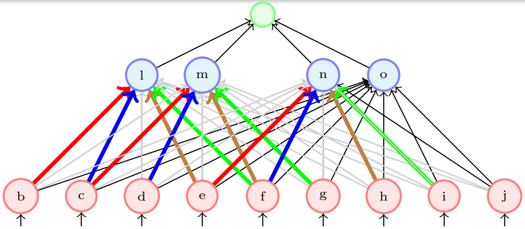
- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active



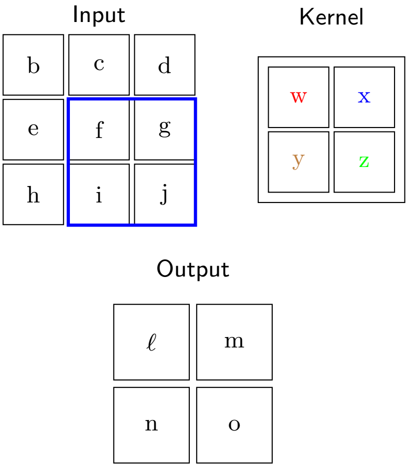


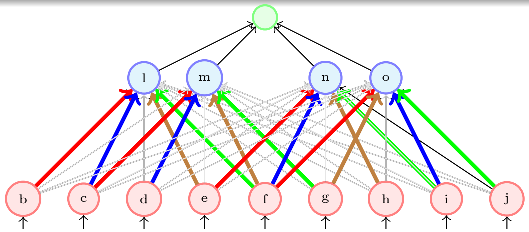
- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero





- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero





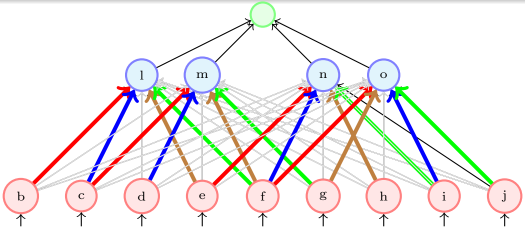
- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero

Input

b	c	d
e	f	g
h	i	j

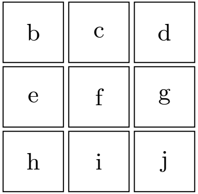
Kernel

w	x
y	z

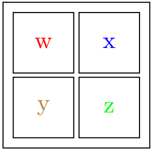


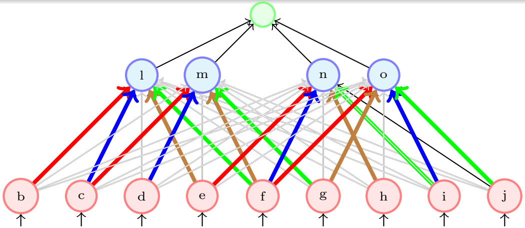
- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero

Input



Kernel





- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero

- We can thus train a convolution neural network using backpropagation by thinking of it as a feedforward neural network with sparse connections

Module 11.4 : CNNs (success stories on ImageNet)

ImageNet Success Stories(roadmap for rest of the talk)

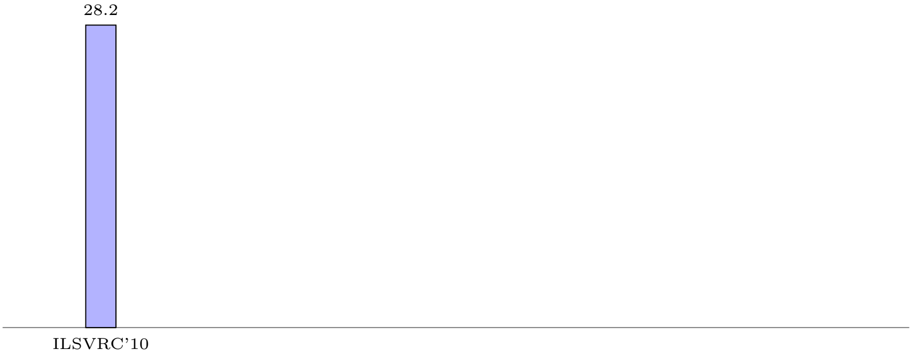
- AlexNet

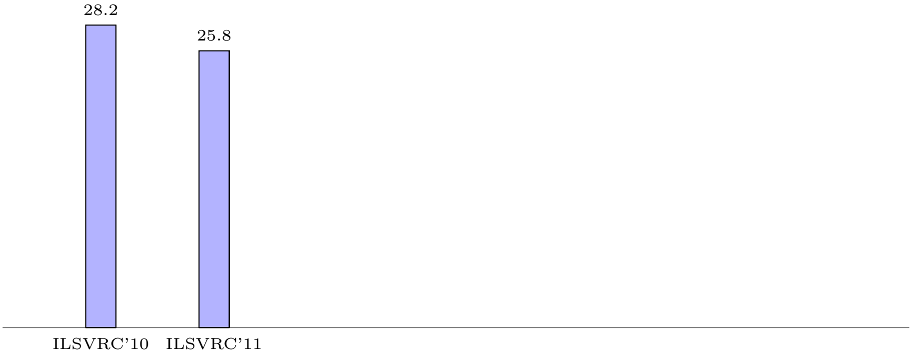
ImageNet Success Stories(roadmap for rest of the talk)

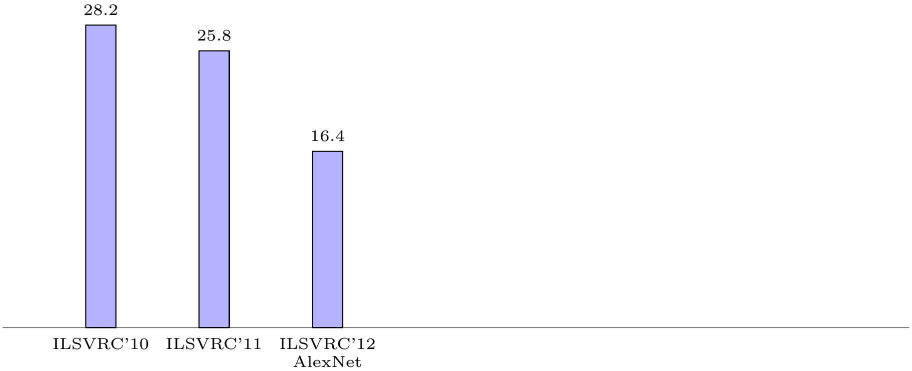
- AlexNet
- ZFNet

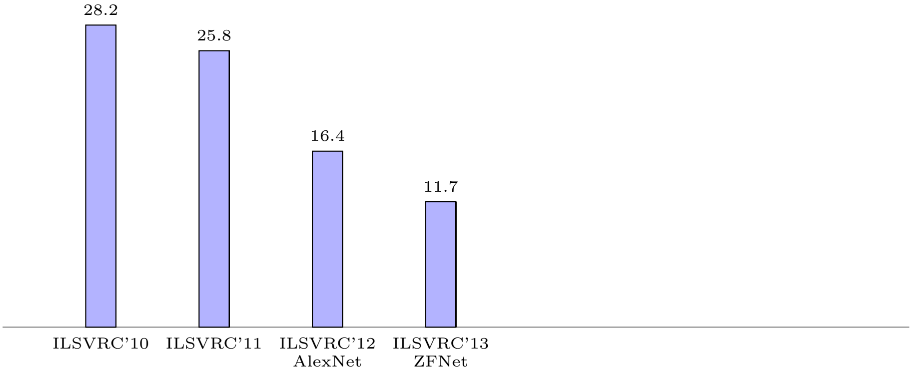
ImageNet Success Stories(roadmap for rest of the talk)

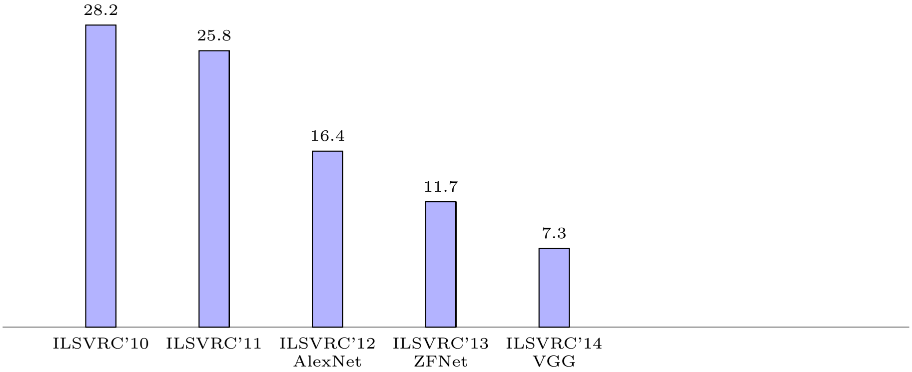
- AlexNet
- ZFNet
- VGGNet

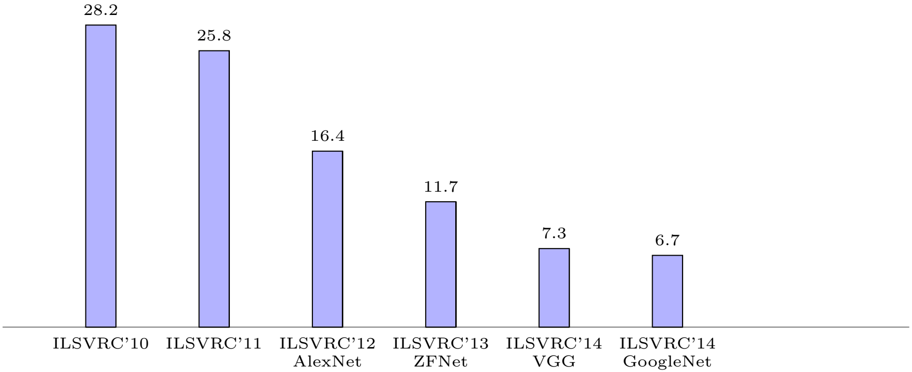

28.2
ILSVRC'10

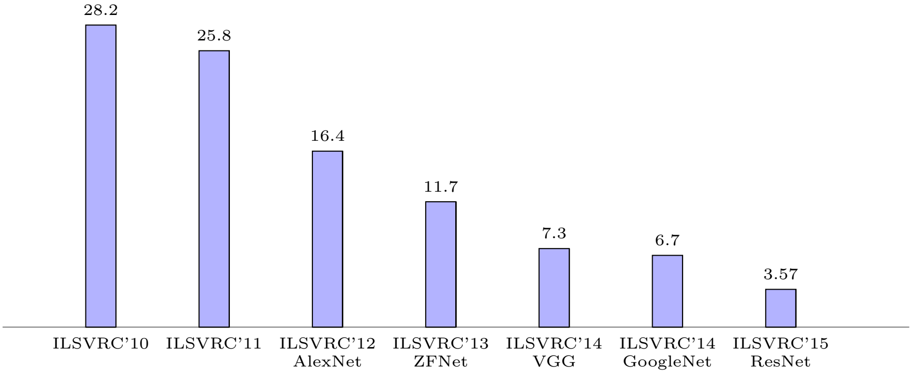

28.2
25.8
ILSVRC'10
ILSVRC'11

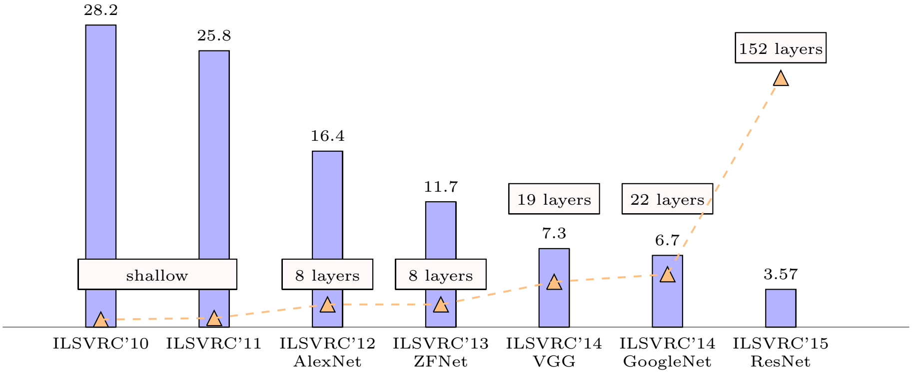

28.2
25.8
16.4
ILSVRC'10
ILSVRC'11
ILSVRC'12
AlexNet

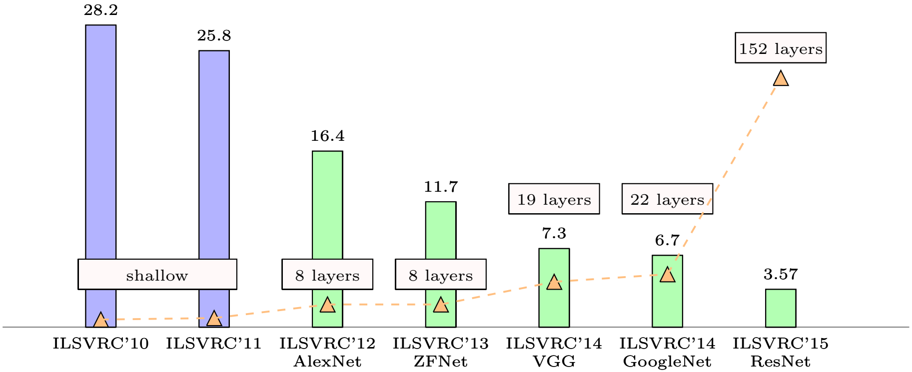

28.2
25.8
16.4
11.7
ILSVRC'10
ILSVRC'11
ILSVRC'12
AlexNet
ILSVRC'13
ZFNet


28.2
25.8
16.4
11.7
7.3
ILSVRC'10
ILSVRC'11
ILSVRC'12
AlexNet
ILSVRC'13
ZFNet
ILSVRC'14
VGG


28.2
25.8
16.4
11.7
7.3
6.7
ILSVRC'10
ILSVRC'11
ILSVRC'12
AlexNet
ILSVRC'13
ZFNet
ILSVRC'14
VGG
ILSVRC'14
GoogleNet


28.2
25.8
16.4
11.7
7.3
6.7
3.57
ILSVRC'10
ILSVRC'11
ILSVRC'12 AlexNet
ILSVRC'13 ZFNet
ILSVRC'14 VGG
ILSVRC'14 GoogleNet
ILSVRC'15 ResNet


28.2
25.8
16.4
11.7
7.3
6.7
3.57
shallow
8 layers
8 layers
19 layers
22 layers
152 layers
ILSVRC'10
ILSVRC'11
ILSVRC'12 AlexNet
ILSVRC'13 ZFNet
ILSVRC'14 VGG
ILSVRC'14 GoogleNet
ILSVRC'15 ResNet


28.2
25.8
16.4
11.7
7.3
6.7
3.57
152 layers
19 layers
22 layers
shallow
8 layers
8 layers
ILSVRC'10
ILSVRC'11
ILSVRC'12
AlexNet
ILSVRC'13
ZFNet
ILSVRC'14
VGG
ILSVRC'14
GoogleNet
ILSVRC'15
ResNet

ImageNet Success Stories(roadmap for rest of the talk)

- AlexNet
- ZFNet
- VGGNet


227
227
3
Input

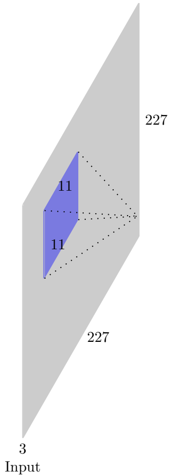
Input: $227 \times 227 \times 3$

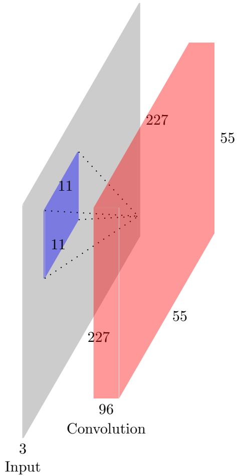
Conv1: $K = 96, F = 11$

$S = 4, P = 0$

Output: $W_2 = ?,\ H_2 = ?$

Parameters: ?





Input: $227 \times 227 \times 3$

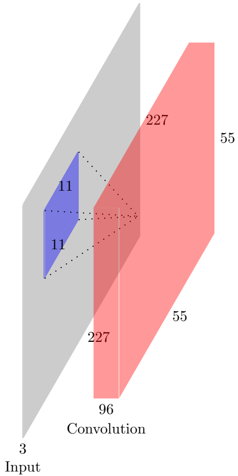
Conv1: $K = 96, F = 11$

$S = 4, P = 0$

Output: $W_2 = 55,\ H_2 = 55$

Parameters: ?

Input: $227 \times 227 \times 3$
Conv1: $K = 96, F = 11$
$S = 4, P = 0$
Output: $W_2 = 55,\ H_2 = 55$
Parameters: $(11 \times 11 \times 3) \times 96 = 34K$

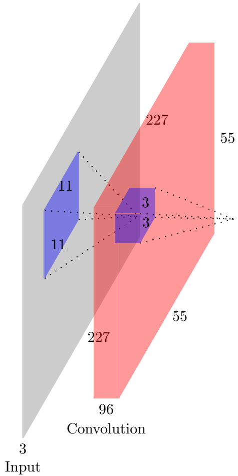


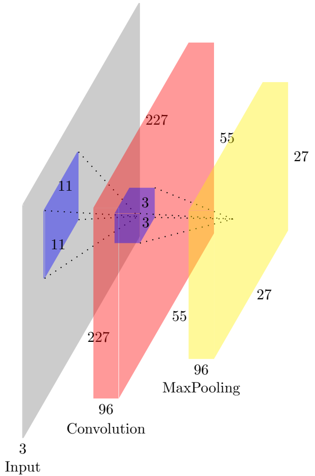
Max Pool Input: $55 \times 55 \times 96$

$F = 3, S = 2$

Output: $W_2 = ?,\ H_2 = ?$

Parameters: ?





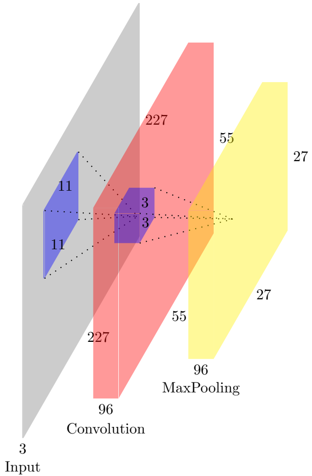
Max Pool Input: $55 \times 55 \times 96$
$F = 3, S = 2$
Output: $W_2 = 27,\ H_2 = 27$
Parameters: ?

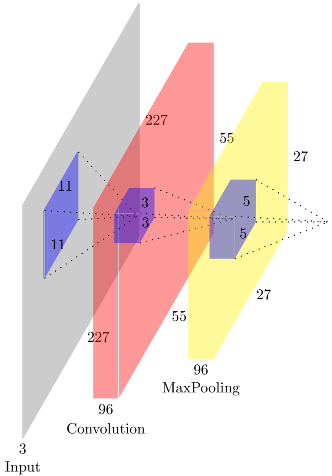
Max Pool Input: $55 \times 55 \times 96$

$F = 3, S = 2$

Output: $W_2 = 27,\ H_2 = 27$

Parameters: 0





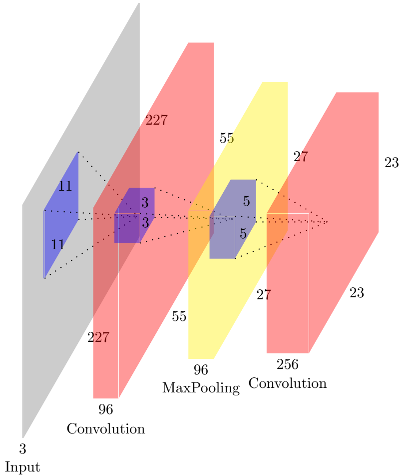
Input: $27 \times 27 \times 96$

Conv1: $K = 256, F = 5$

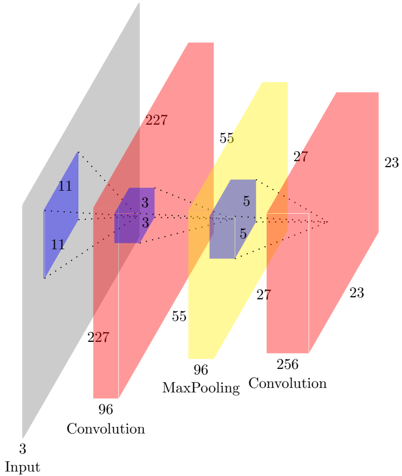
$S = 1, P = 0$

Output: $W_2 = ?,\ H_2 = ?$

Parameters: ?



Input: $27 \times 27 \times 96$
Conv1: $K = 256, F = 5$
$S = 1, P = 0$
Output: $W_2 = 23,\ H_2 = 23$
Parameters: ?



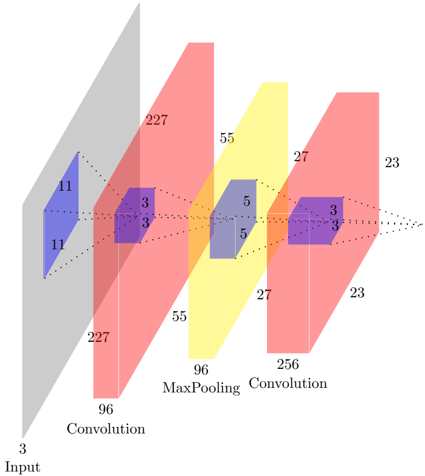
Input: $27 \times 27 \times 96$
Conv1: $K = 256, F = 5$
$S = 1, P = 0$
Output: $W_2 = 23, \; H_2 = 23$
Parameters: $(5 \times 5 \times 96) \times 256 = 0.6M$

Max Pool Input: $23 \times 23 \times 256$

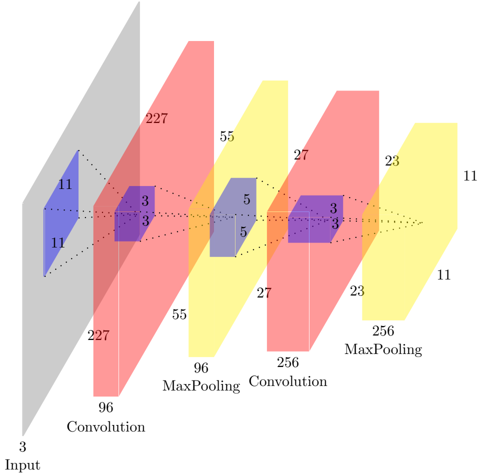
$F = 3, S = 2$

Output: $W_2 = ?,\ H_2 = ?$

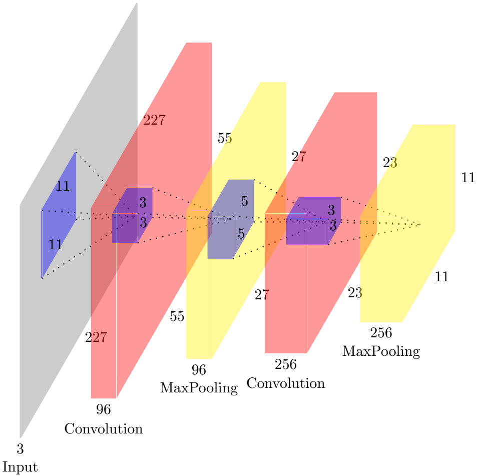
Parameters: ?

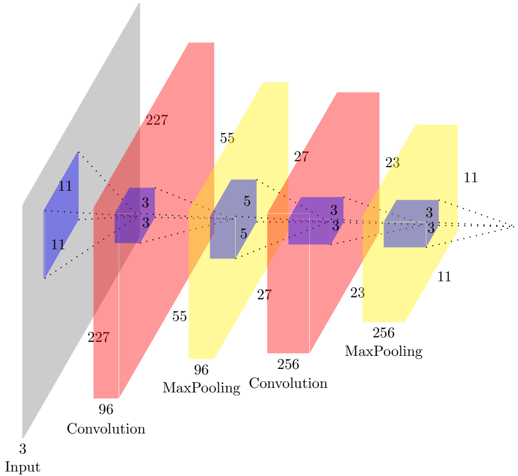


Max Pool Input: $23 \times 23 \times 256$
$F = 3, S = 2$
Output: $W_2 = 11, \; H_2 = 11$
Parameters: ?

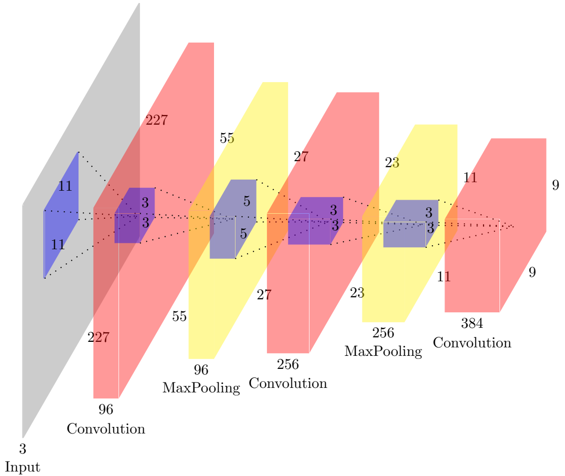


Max Pool Input: $23 \times 23 \times 256$
$F = 3, S = 2$
Output: $W_2 = 11,\ H_2 = 11$
Parameters: 0





Input: $11 \times 11 \times 256$
Conv1: $K = 384, F = 3$
$S = 1, P = 0$
Output: $W_2 = ?,\ H_2 = ?$
Parameters: ?



Input: $11 \times 11 \times 256$

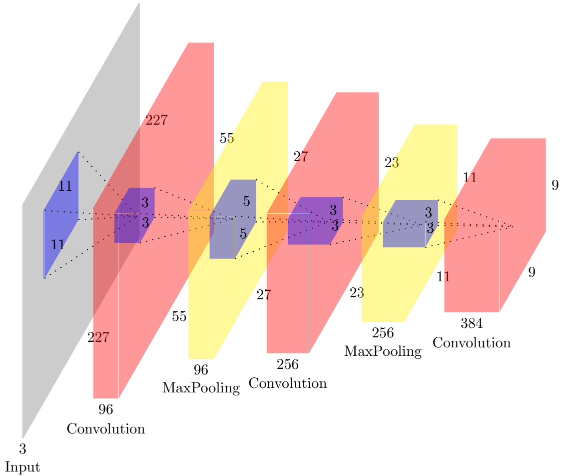
Conv1: $K = 384, F = 3$

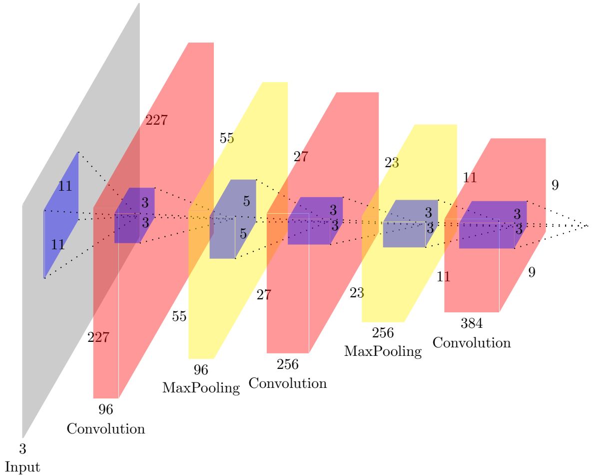
$S = 1, P = 0$

Output: $W_2 = 9,\ H_2 = 9$

Parameters: ?

Input: $11 \times 11 \times 256$
Conv1: $K = 384, F = 3$
$S = 1, P = 0$
Output: $W_2 = 9,\ H_2 = 9$
Parameters: $(3 \times 3 \times 256) \times 384 = 0.8M$





Input: $9 \times 9 \times 384$

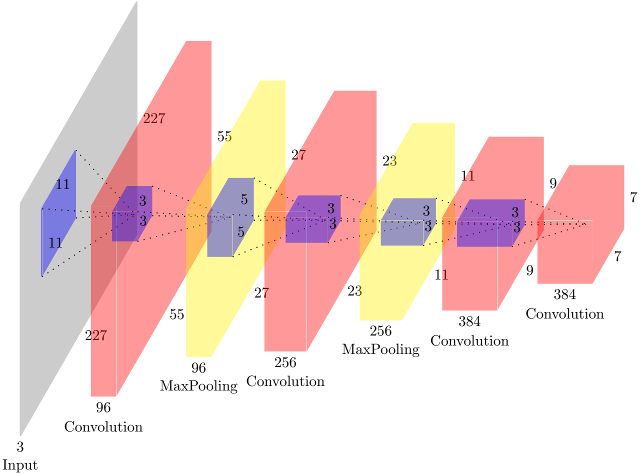
Conv1: $K = 384, F = 3$

$S = 1, P = 0$

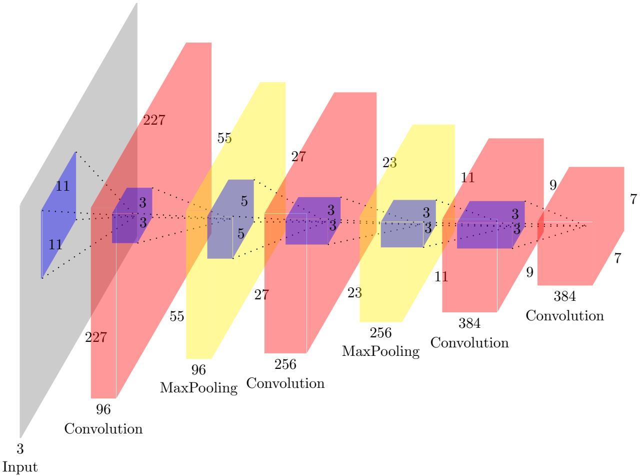
Output: $W_2 =?,\ H_2 =?$

Parameters: ?

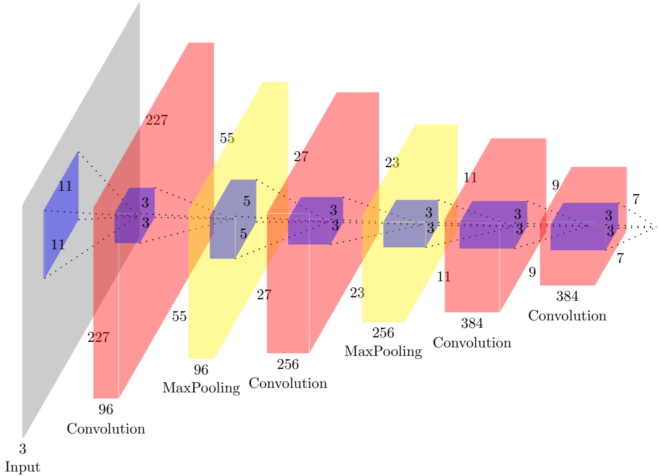
Input: $9 \times 9 \times 384$
Conv1: $K = 384, F = 3$
$S = 1, P = 0$
Output: $W_2 = 7,\ H_2 = 7$
Parameters: ?



Input: $9 \times 9 \times 384$
Conv1: $K = 384, F = 3$
$S = 1, P = 0$
Output: $W_2 = 7,\ H_2 = 7$
Parameters: $(3 \times 3 \times 384) \times 384 = 1.327M$



Input: $7 \times 7 \times 384$
Conv1: $K = 256, F = 3$
$S = 1, P = 0$
Output: $W_2 = ?,\ H_2 = ?$
Parameters: ?



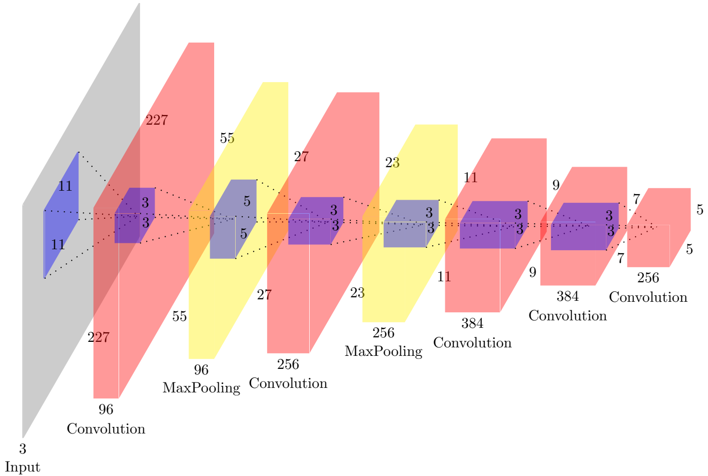
Input: $7 \times 7 \times 384$

Conv1: $K = 256, F = 3$

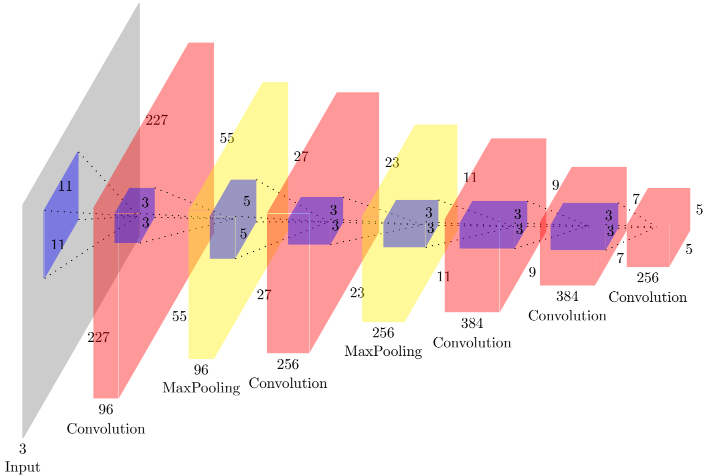
$S = 1, P = 0$

Output: $W_2 = 5, \ H_2 = 5$

Parameters: ?



Input: $7 \times 7 \times 384$
Conv1: $K = 256, F = 3$
$S = 1, P = 0$
Output: $W_2 = 5,\ H_2 = 5$
Parameters: $(3 \times 3 \times 384) \times 256 = 0.8M$

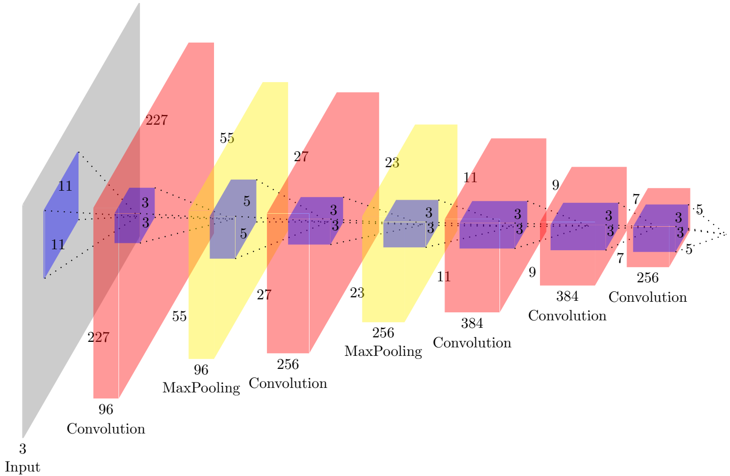


Max Pool Input: $5 \times 5 \times 256$

$F = 3, S = 2$

Output: $W_2 = ?,\ H_2 = ?$

Parameters: ?

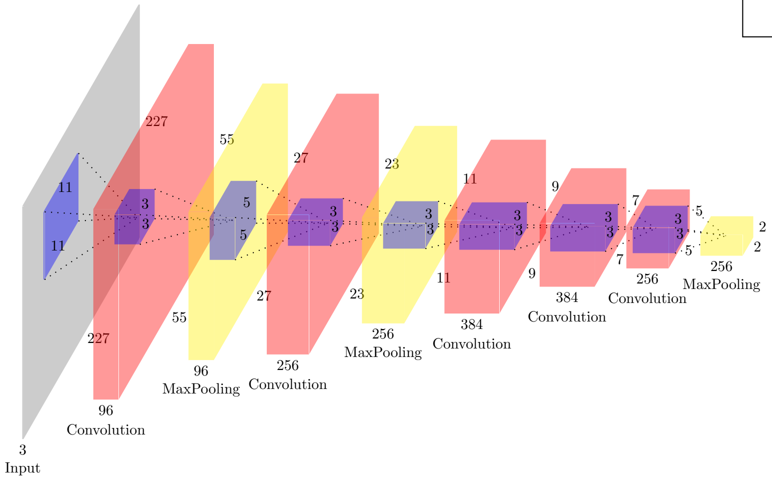


Max Pool Input: $5 \times 5 \times 256$

$F = 3, S = 2$

Output: $W_2 = 2, \ H_2 = 2$

Parameters: ?

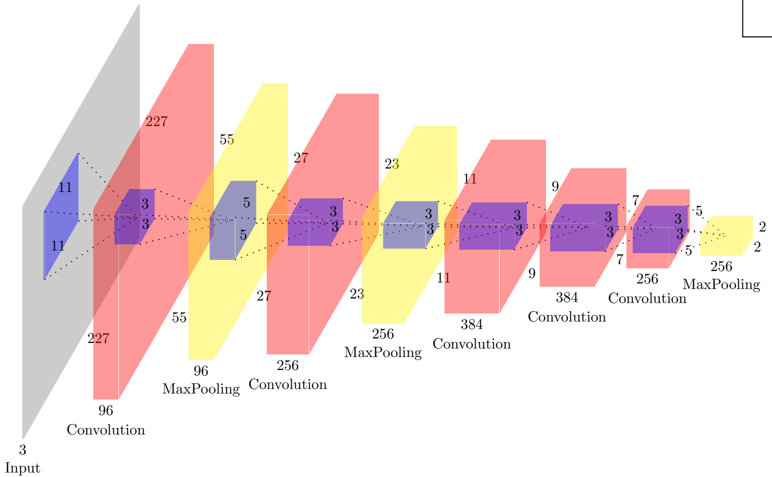


Max Pool Input: $5 \times 5 \times 256$

$F = 3, S = 2$

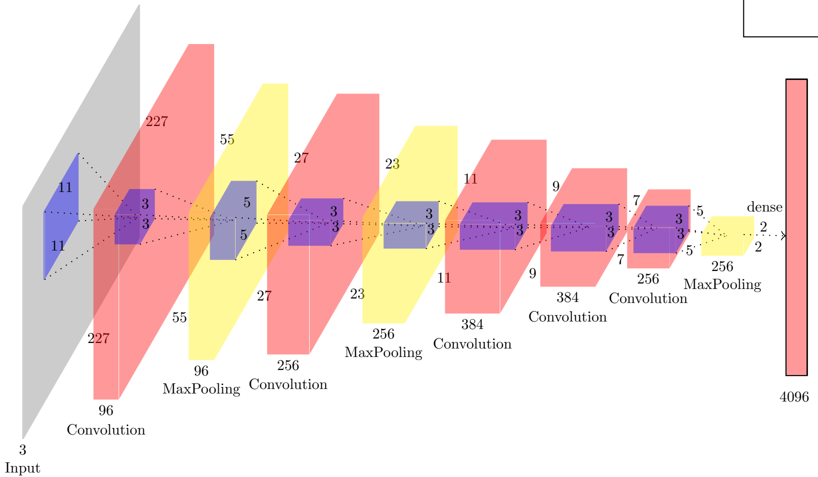
Output: $W_2 = 2,\ H_2 = 2$

Parameters: 0

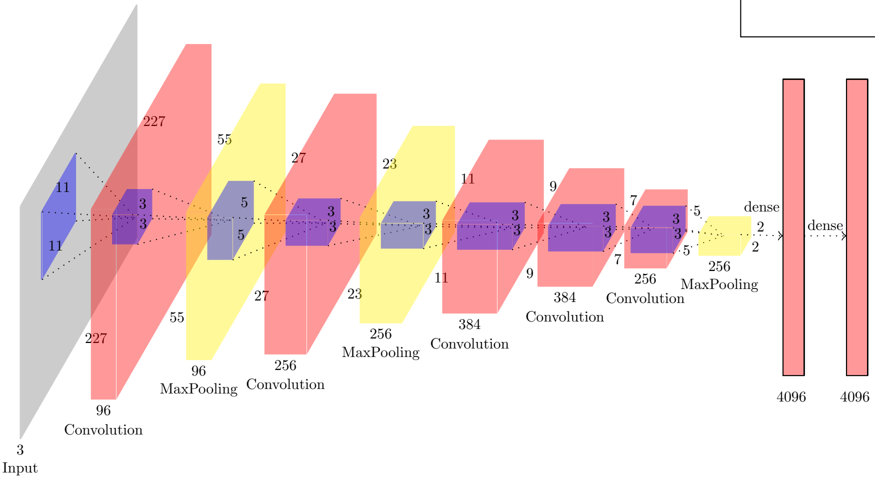


FC1

Parameters: $(2 \times 2 \times 256) \times 4096 = 4M$

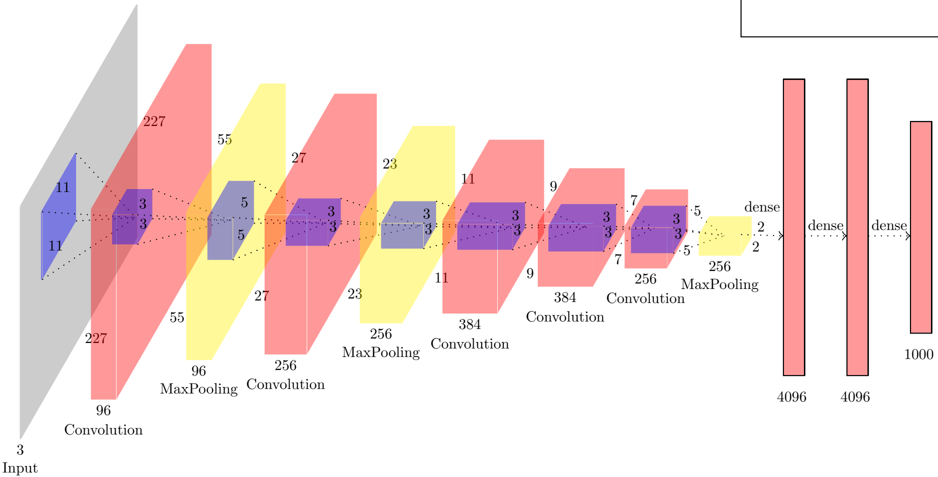


FC1
Parameters: $4096 \times 4096 = 16M$

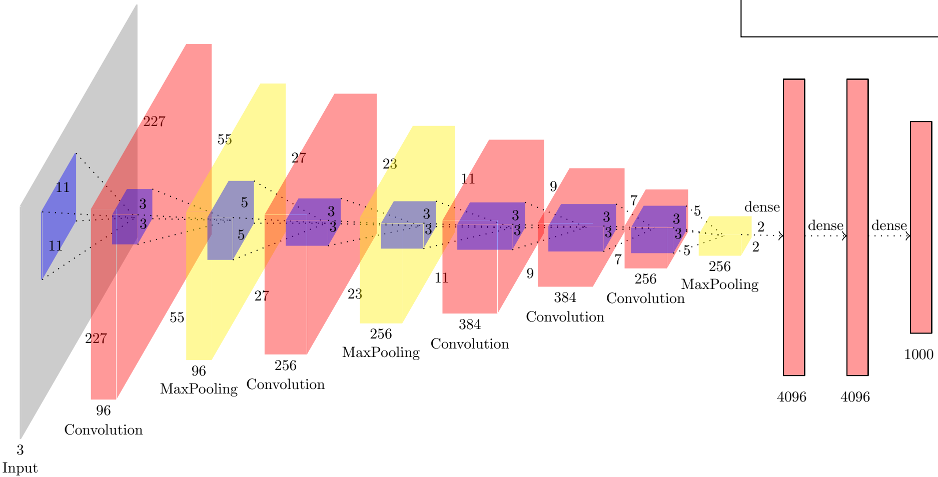


FC1

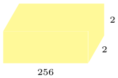
Parameters: $4096 \times 1000 = 4M$



Total Parameters: $27.55M$

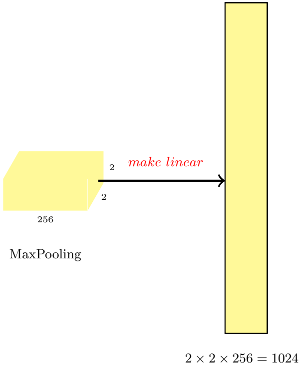


- Let us look at the connections in the fully connected layers in more detail

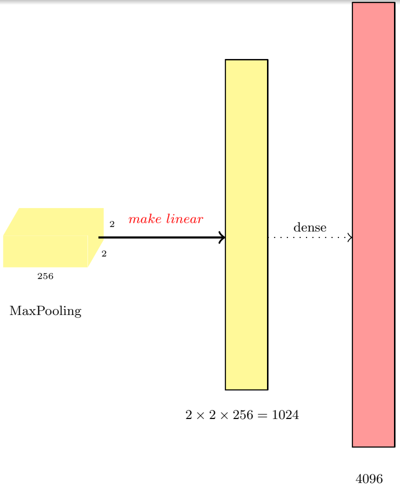


MaxPooling

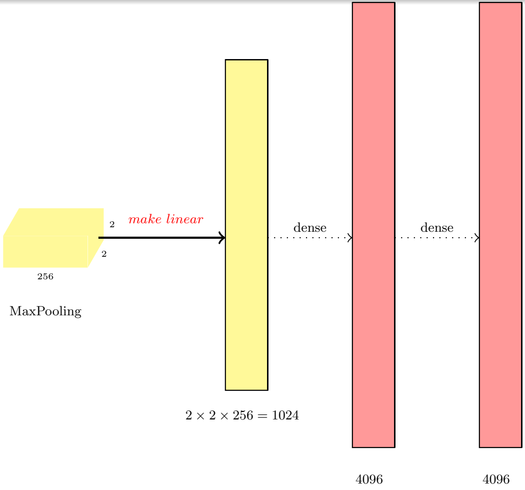
- Let us look at the connections in the fully connected layers in more detail
- We will first stretch out the last conv or maxpool layer to make it a 1d vector



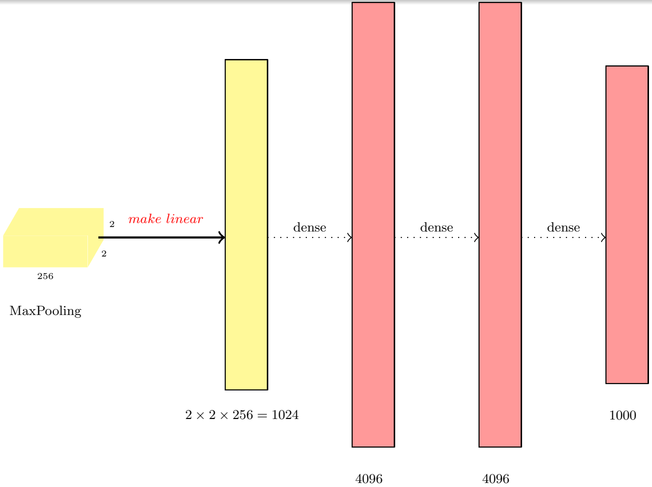
- Let us look at the connections in the fully connected layers in more detail
- We will first stretch out the last conv or maxpool layer to make it a 1d vector
- This 1d vector is then densely connected to other layers just as in a regular feedforward neural network



- Let us look at the connections in the fully connected layers in more detail
- We will first stretch out the last conv or maxpool layer to make it a 1d vector
- This 1d vector is then densely connected to other layers just as in a regular feedforward neural network

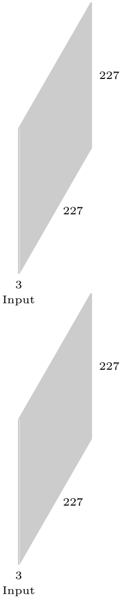


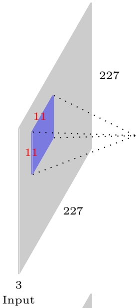
- Let us look at the connections in the fully connected layers in more detail
- We will first stretch out the last conv or maxpool layer to make it a 1d vector
- This 1d vector is then densely connected to other layers just as in a regular feedforward neural network

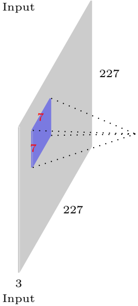


ImageNet Success Stories(roadmap for rest of the talk)

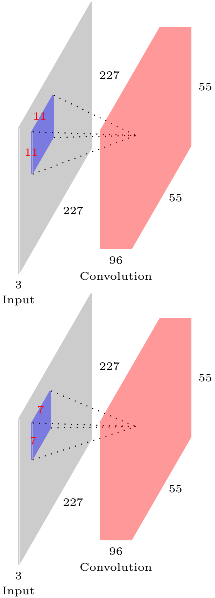
- AlexNet
- ZFNet
- VGGNet


227
227
3
Input
227
227
3
Input

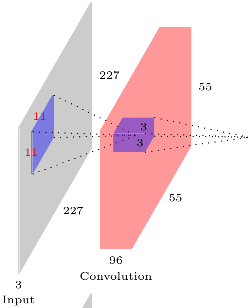


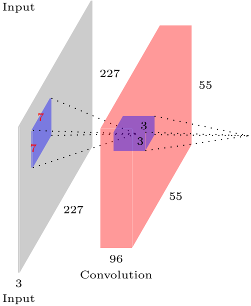


Layer1: $F = 11 \rightarrow 7$
Difference in Parameters
$((11^2 - 7^2) \times 3) \times 96 = 20.7K$

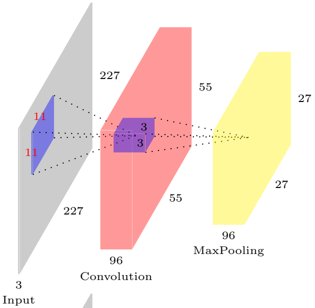


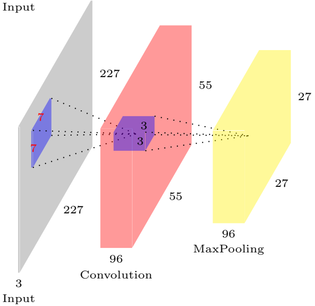
Layer1: $F = 11 \rightarrow 7$
Difference in Parameters
$((11^2 - 7^2) \times 3) \times 96 = 20.7K$



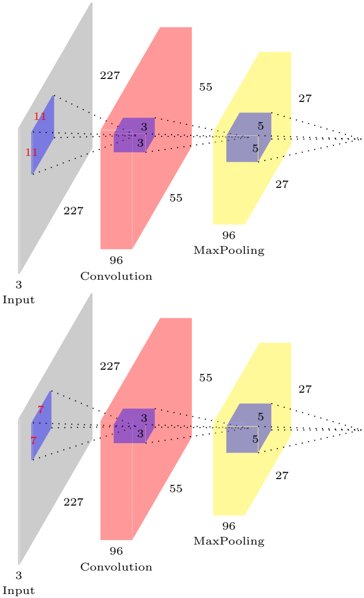


Layer2: No difference

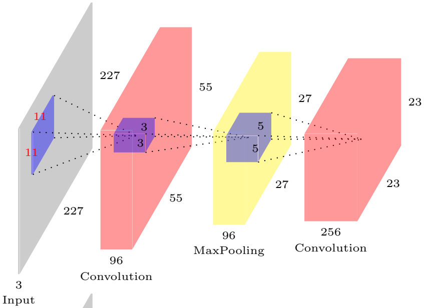


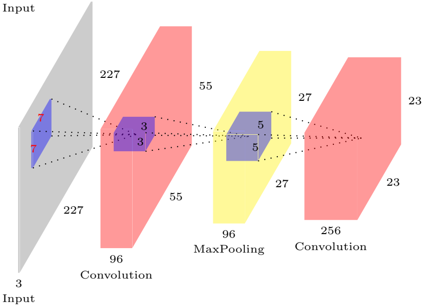


Layer2: No difference

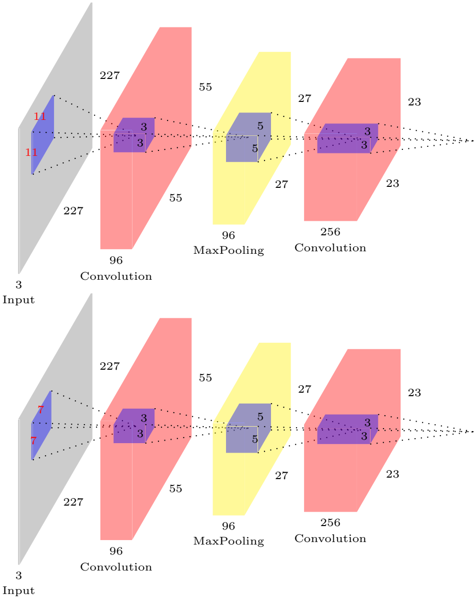


Layer3: No difference

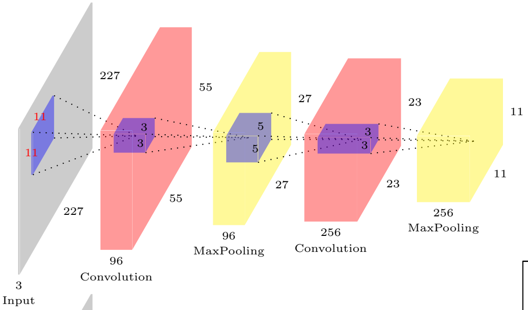




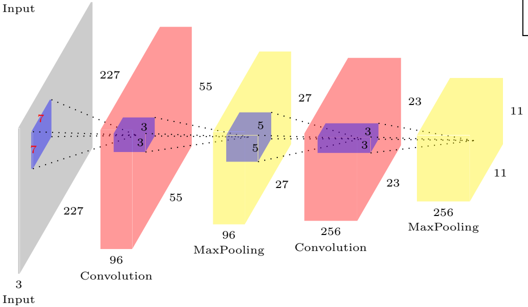
Layer3: No difference

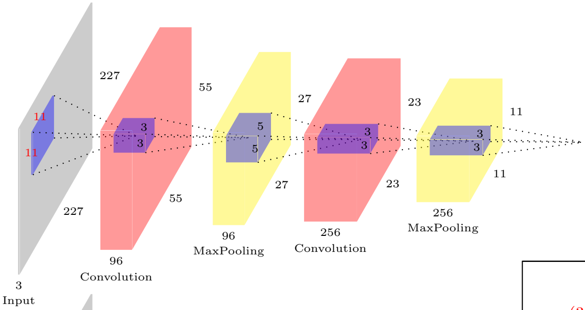


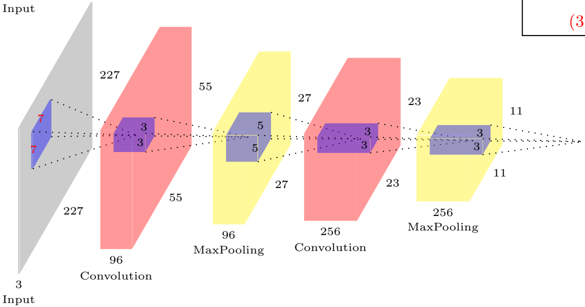
Layer4: No difference



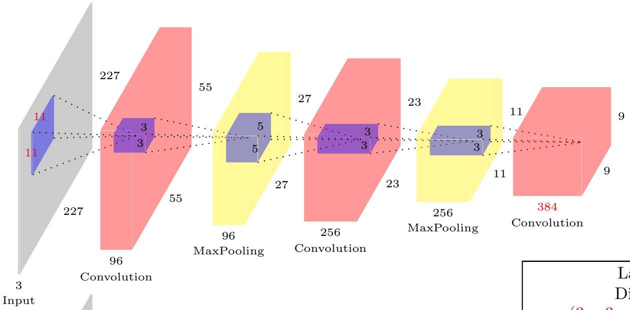
Layer4: No difference



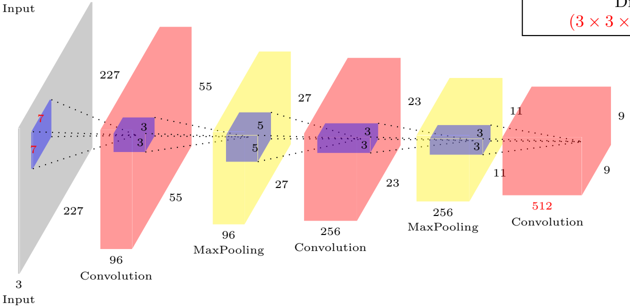


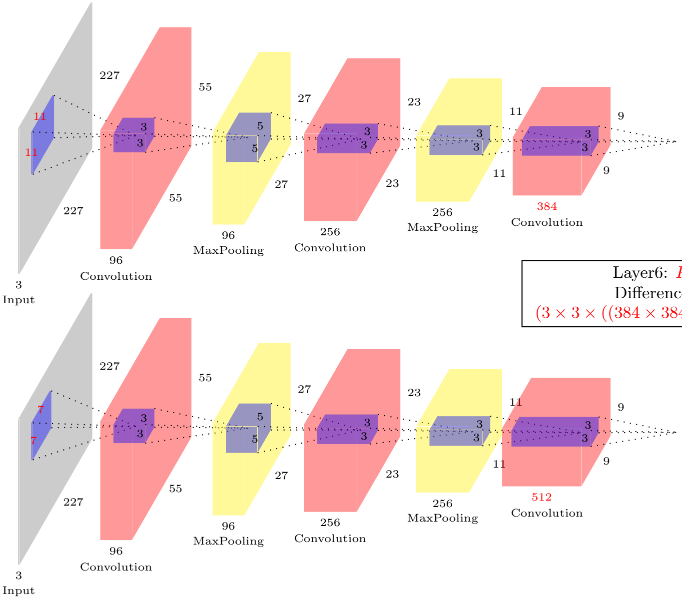


Layer5: $K = 384 \rightarrow 512$
Difference in Parameters
$(3 \times 3 \times 256) \times (512 - 384) = 0.29M$



Layer5: $K = 384 \rightarrow 512$
Difference in Parameters
$(3 \times 3 \times 256) \times (512 - 384) = 0.29M$

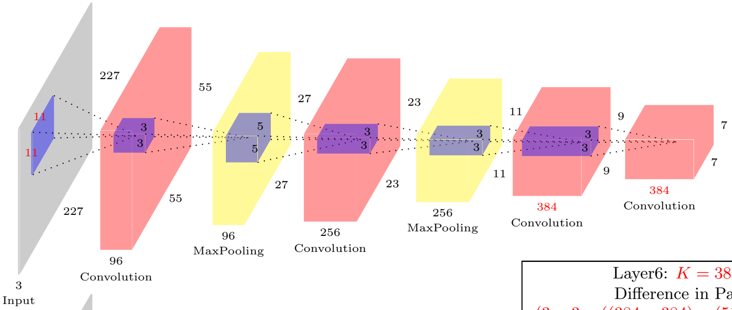




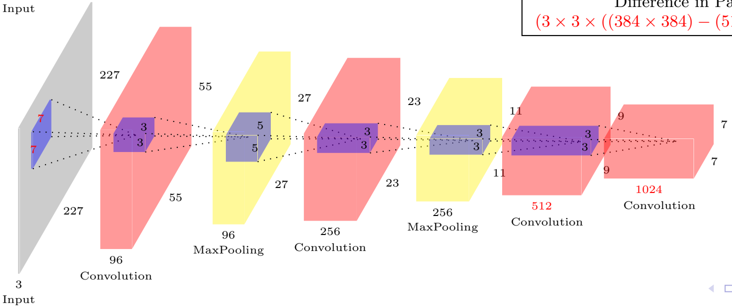
Layer6: $K = 384 \rightarrow 1024$

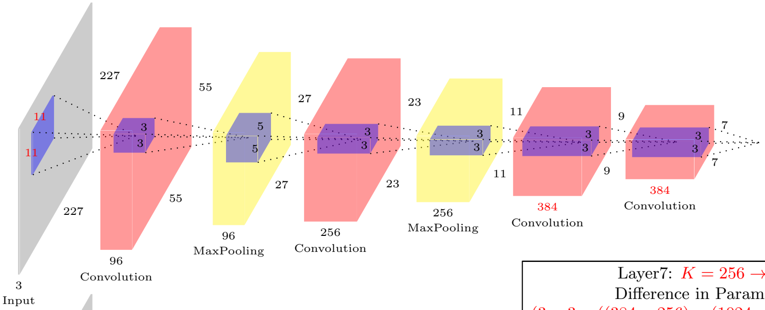
Difference in Parameters

$(3 \times 3 \times ((384 \times 384) - (512 \times 1024)) = 0.8M$



Layer6: $K = 384 \rightarrow 1024$
Difference in Parameters
$(3 \times 3 \times ((384 \times 384) - (512 \times 1024)) = 0.8M$

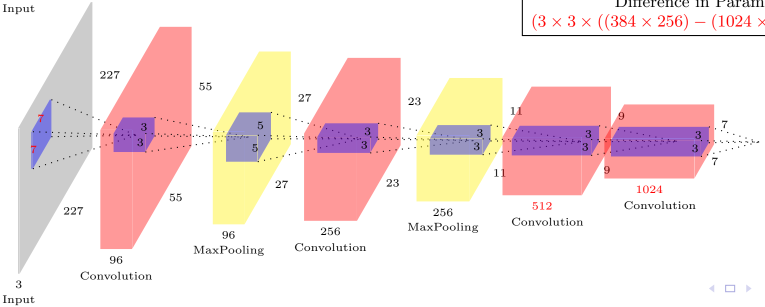


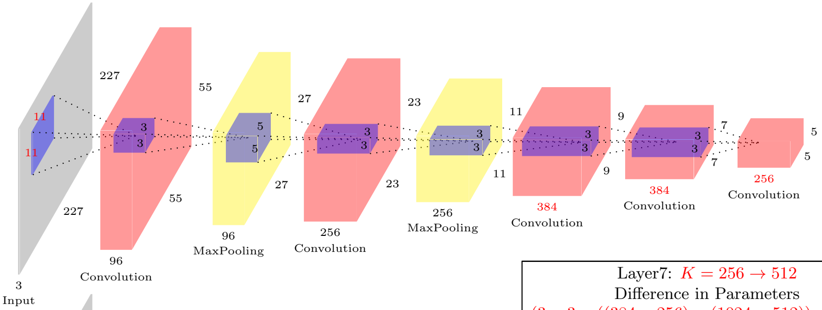


Layer7: $K = 256 \rightarrow 512$

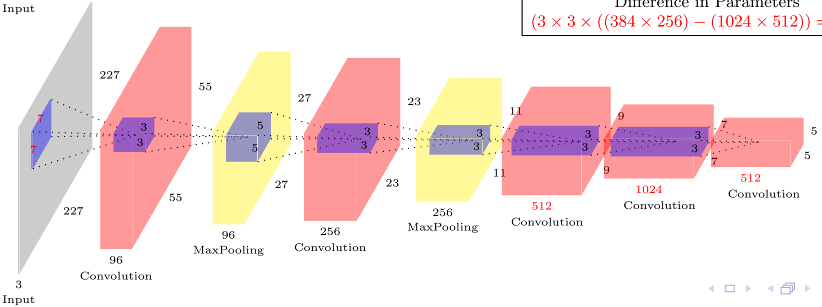
Difference in Parameters

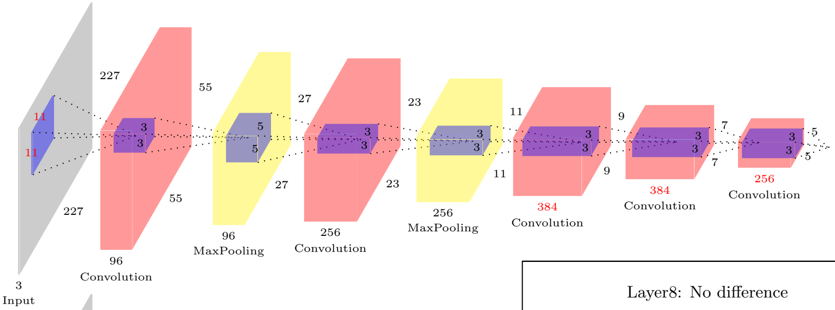
$(3 \times 3 \times ((384 \times 256) - (1024 \times 512)) = 0.36M$



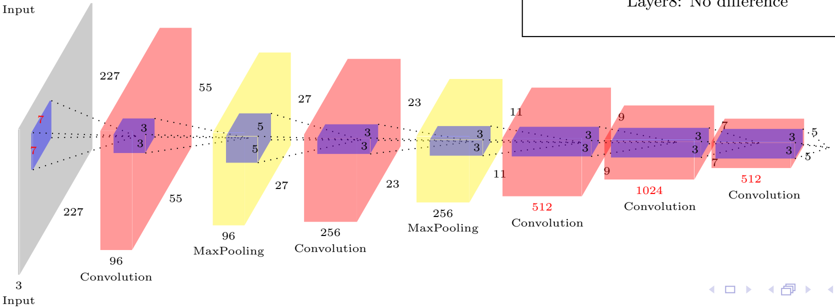


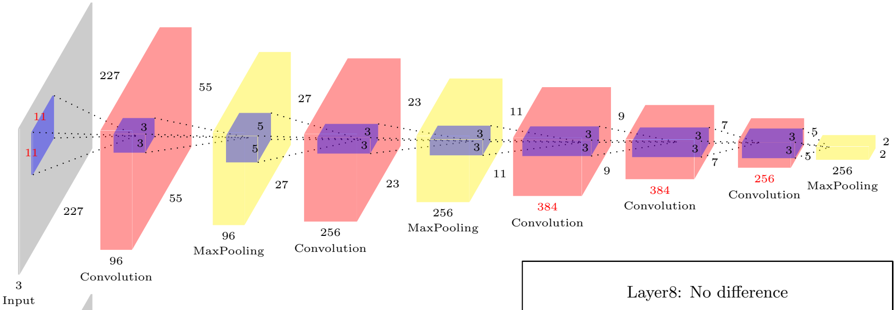
Layer7: $K = 256 \rightarrow 512$
Difference in Parameters
$(3 \times 3 \times ((384 \times 256) - (1024 \times 512)) = 0.36M$



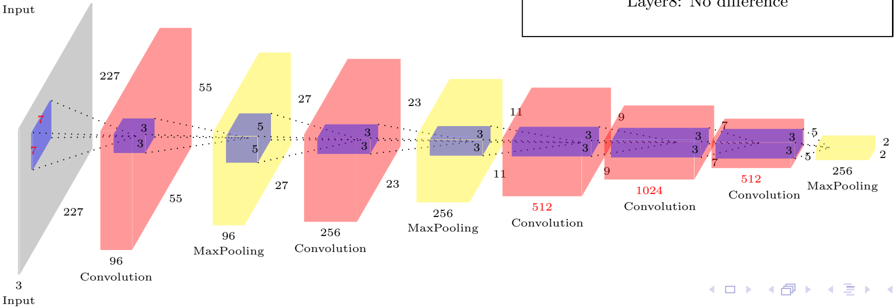


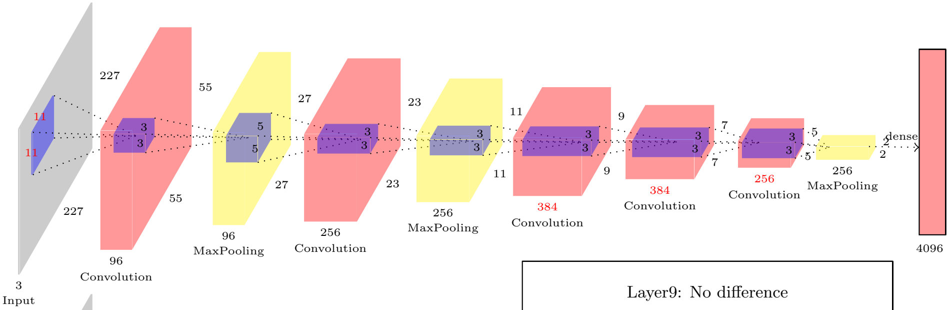
Layer8: No difference



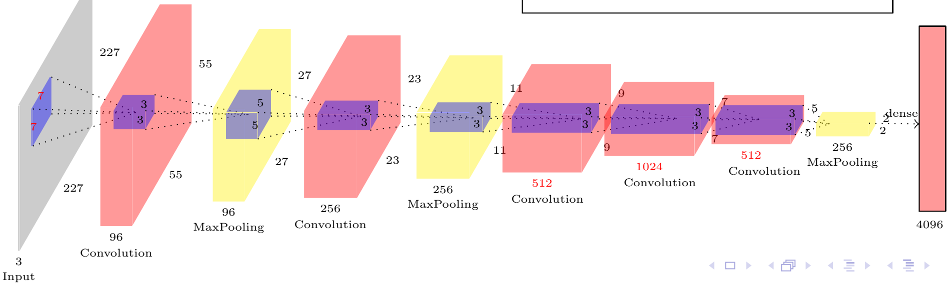


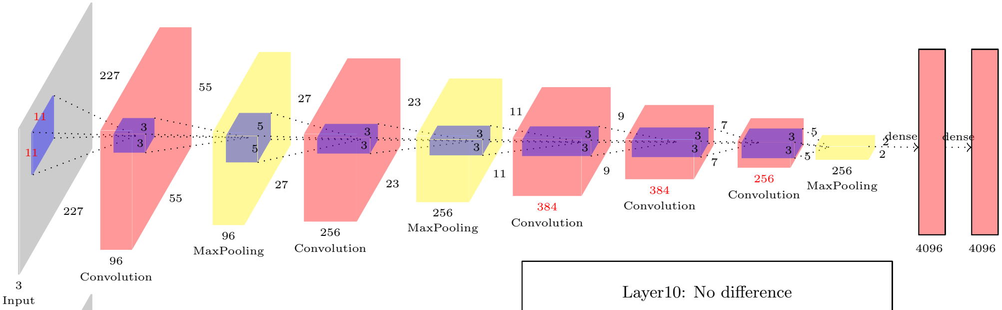
Layer8: No difference



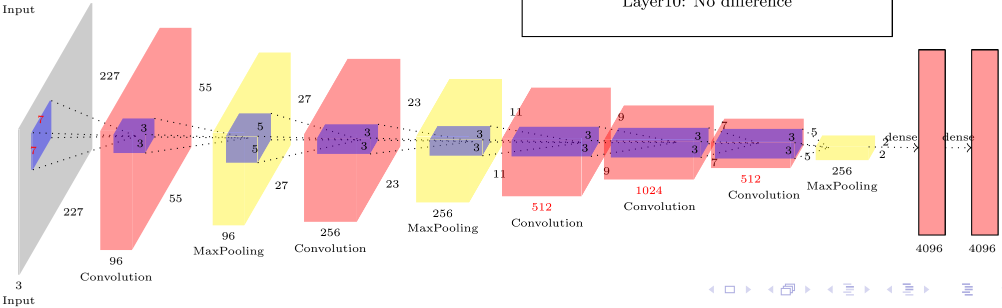


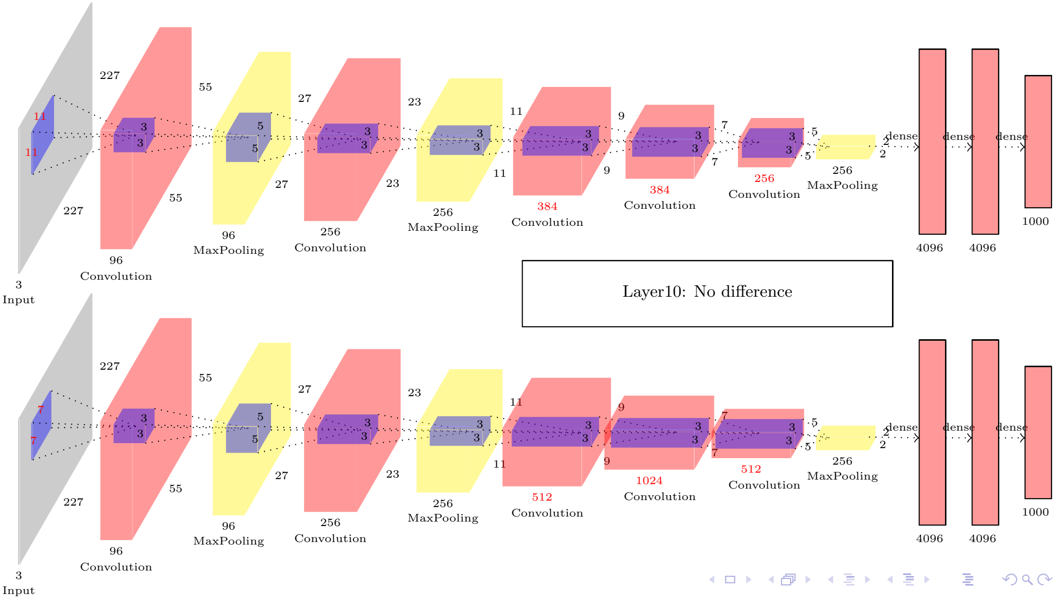
Layer9: No difference



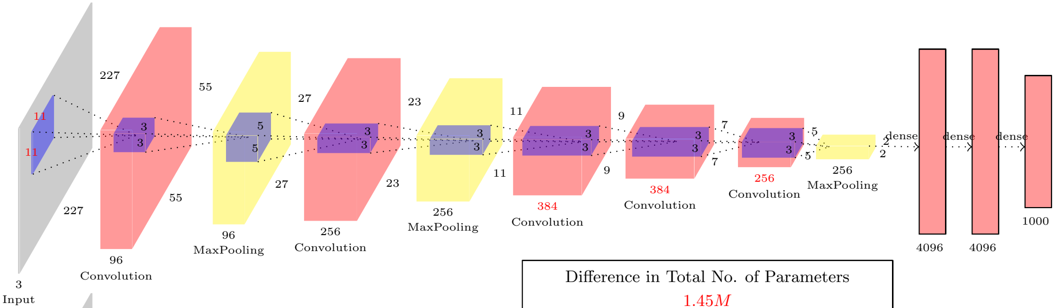


Layer10: No difference



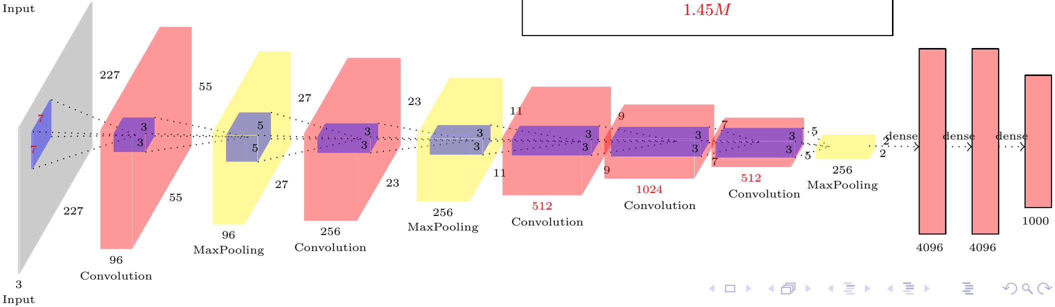


Layer10: No difference



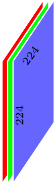
Difference in Total No. of Parameters

$1.45M$

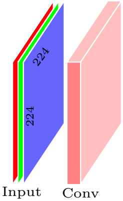


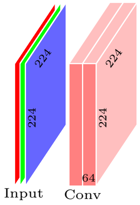
ImageNet Success Stories(roadmap for rest of the talk)

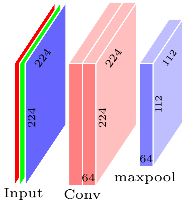
- AlexNet
- ZFNet
- VGGNet

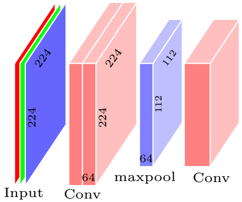


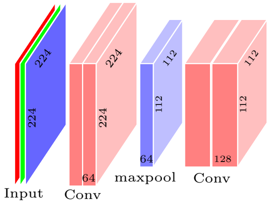
Input

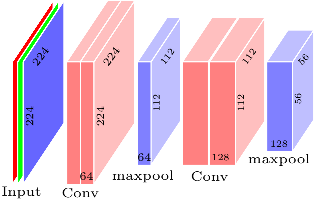

224
224
Input
Conv

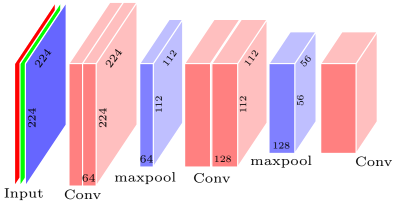

224
224
224
224
64
Input
Conv

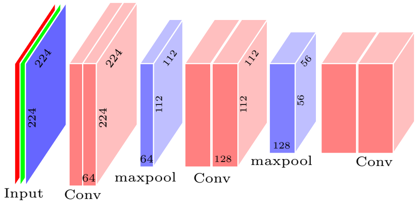

224
224
224
224
64
112
112
64
maxpool
Input
Conv

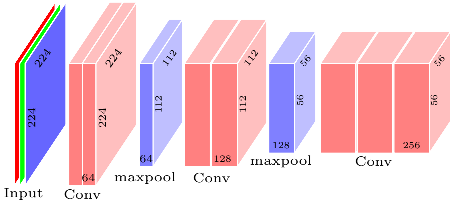

224
224
Input
224
224
64
Conv
112
112
64
maxpool
Conv

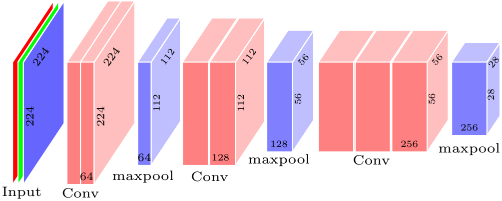

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv


224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool

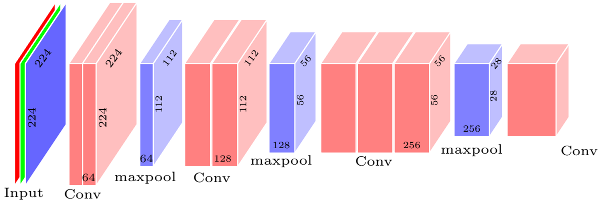

224
224
224
224
64
112
112
64
112
112
128
56
56
128
Input
Conv
maxpool
Conv
maxpool
Conv

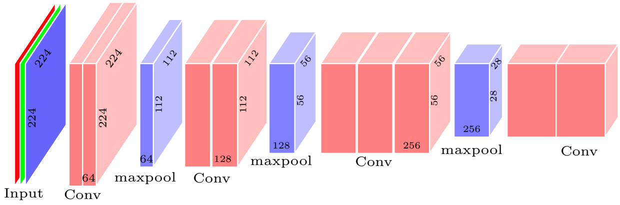

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
Conv

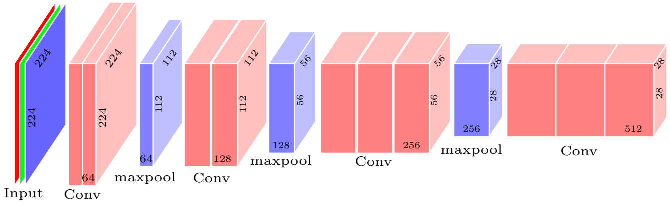

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv

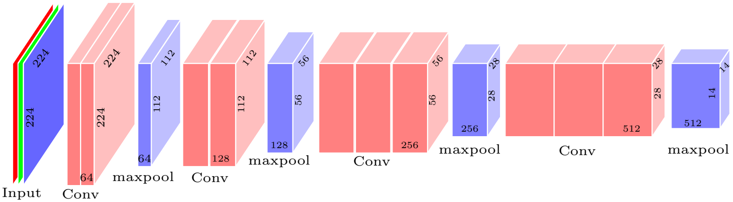


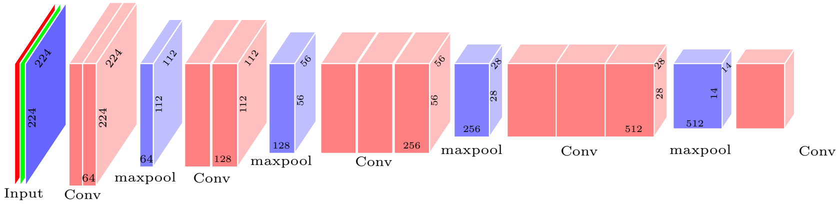
224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool

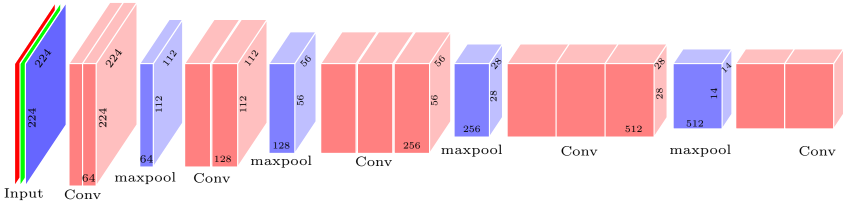

224
224
224
224
64
112
112
64
112
112
128
56
56
128
56
56
256
28
28
256
Input
Conv
maxpool
Conv
maxpool
Conv
maxpool
Conv

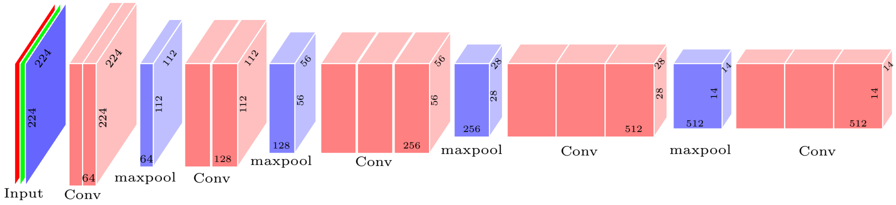

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool
Conv


224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool
28
28
512
Conv

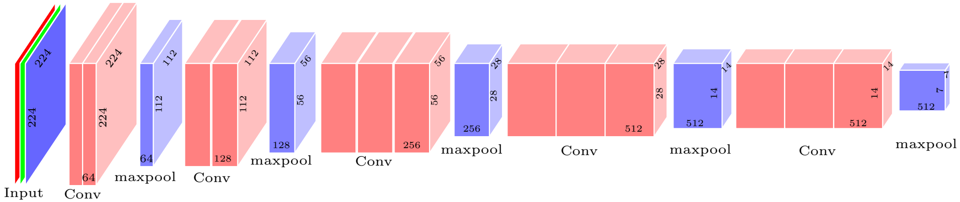

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool
28
28
512
Conv
14
14
512
maxpool


224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool
28
28
512
Conv
14
14
512
maxpool
Conv

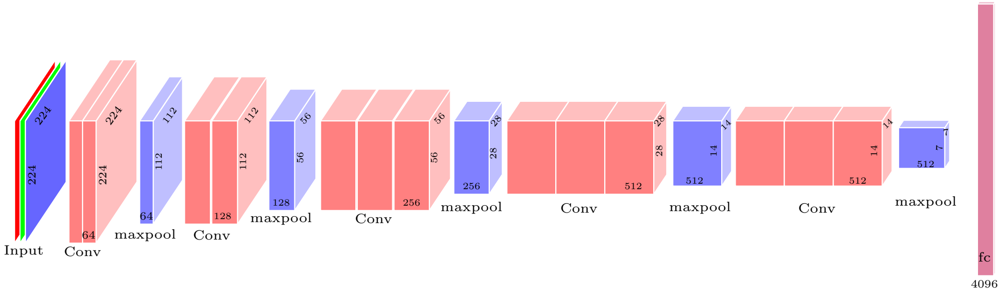

224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool
28
28
512
Conv
14
14
512
maxpool
Conv

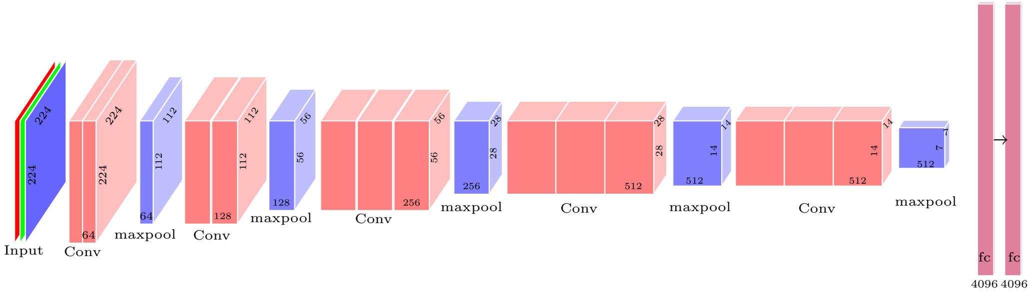


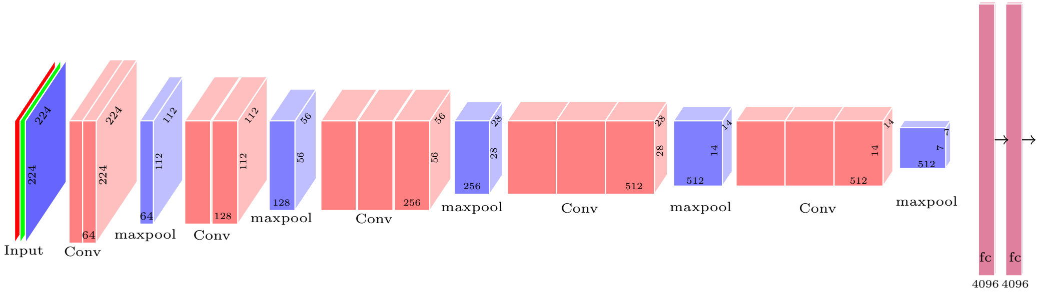
224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool
28
28
512
Conv
14
14
512
maxpool
14
14
512
Conv

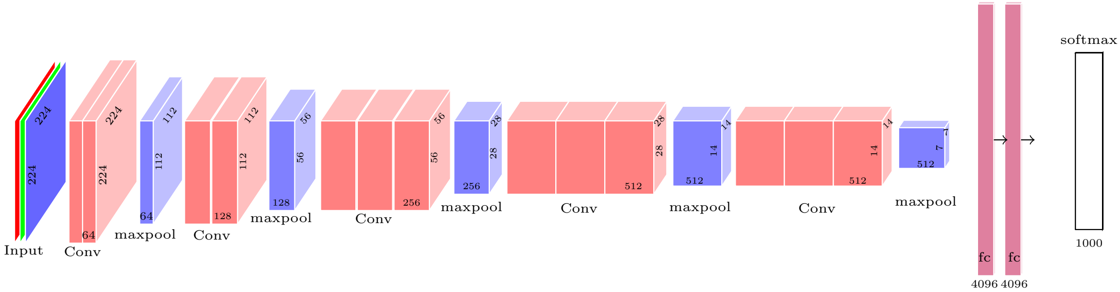


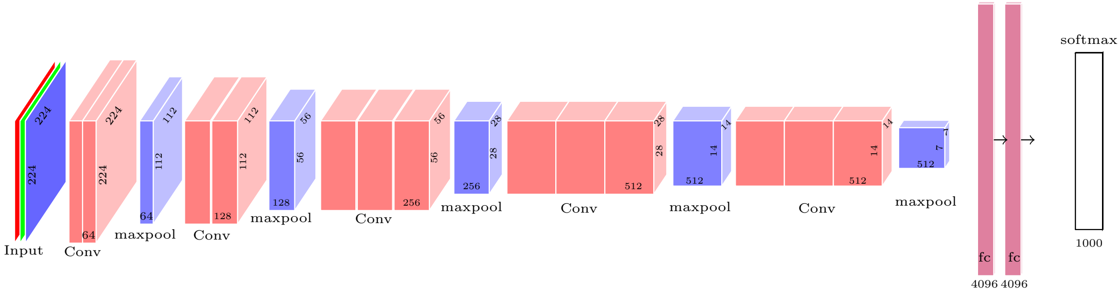
224
224
Input
224
224
64
Conv
112
112
64
maxpool
112
112
128
Conv
56
56
128
maxpool
56
56
256
Conv
28
28
256
maxpool
28
28
512
Conv
14
14
512
maxpool
14
14
512
Conv
7
7
512
maxpool


Input
224
224
Conv
64
224
224
maxpool
64
112
112
Conv
128
112
112
maxpool
128
56
56
Conv
256
56
56
maxpool
256
28
28
Conv
512
28
28
maxpool
512
14
14
Conv
512
14
14
maxpool
512
7
7
fc
4096

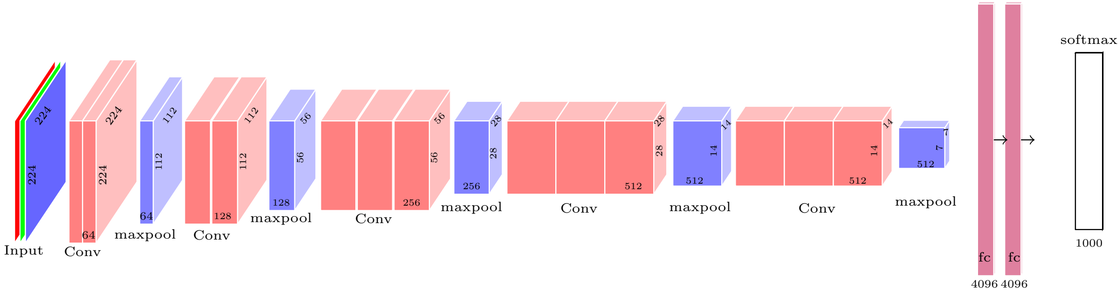

Input
224
224
Conv
64
224
224
maxpool
64
112
112
Conv
128
112
112
maxpool
128
56
56
Conv
256
56
56
maxpool
256
28
28
Conv
512
28
28
maxpool
512
14
14
Conv
512
14
14
maxpool
512
7
7
fc
fc
4096 4096


Input
224
224
Conv
64
224
224
maxpool
64
112
112
Conv
128
112
112
maxpool
128
56
56
Conv
256
56
56
maxpool
256
28
28
Conv
512
28
28
maxpool
512
14
14
Conv
512
14
14
maxpool
512
7
7
fc
fc
4096 4096

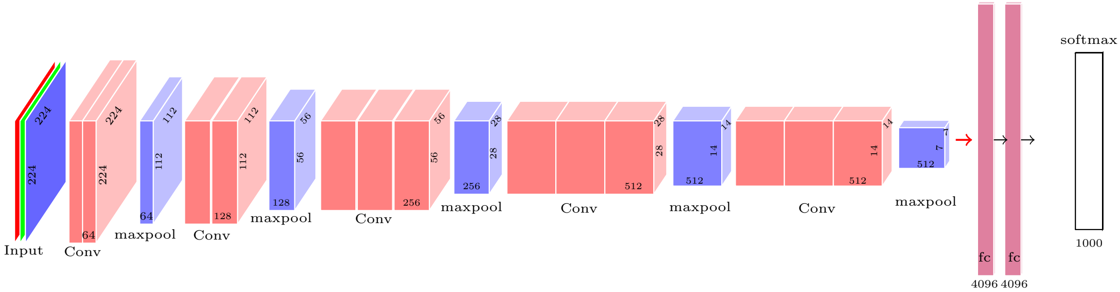

softmax
Input
224
224
Conv
64
224
224
maxpool
64
112
112
Conv
128
112
112
maxpool
128
56
56
Conv
256
56
56
maxpool
256
28
28
Conv
512
28
28
maxpool
512
14
14
Conv
512
14
14
maxpool
512
7
7
fc
fc
4096 4096
1000



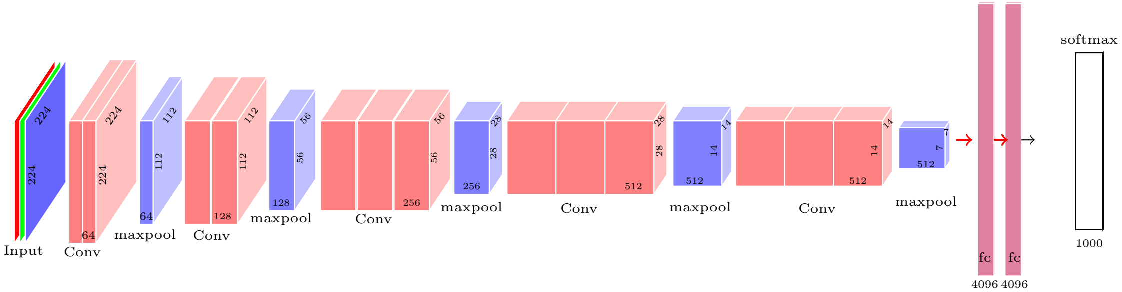
- Kernel size is 3×3 throughout



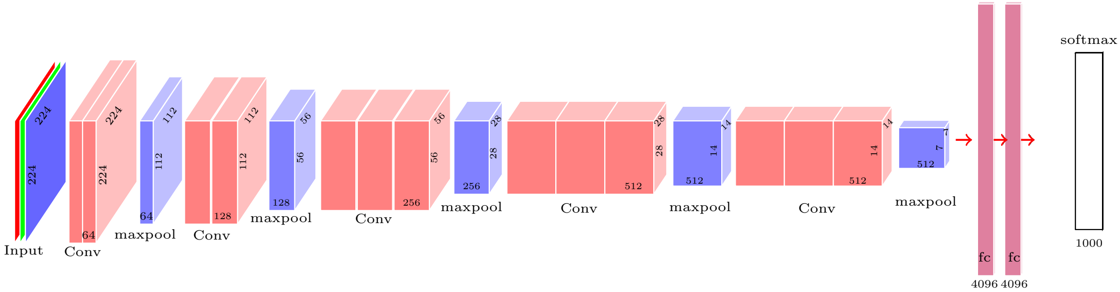
- Kernel size is 3×3 throughout
- Total parameters in non FC layers = $\sim 16M$



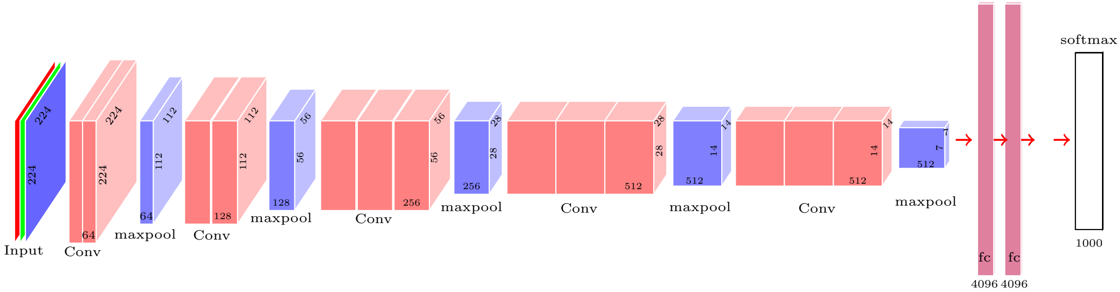
- Kernel size is 3×3 throughout
- Total parameters in non FC layers = $\sim 16M$
- Total Parameters in FC layers =



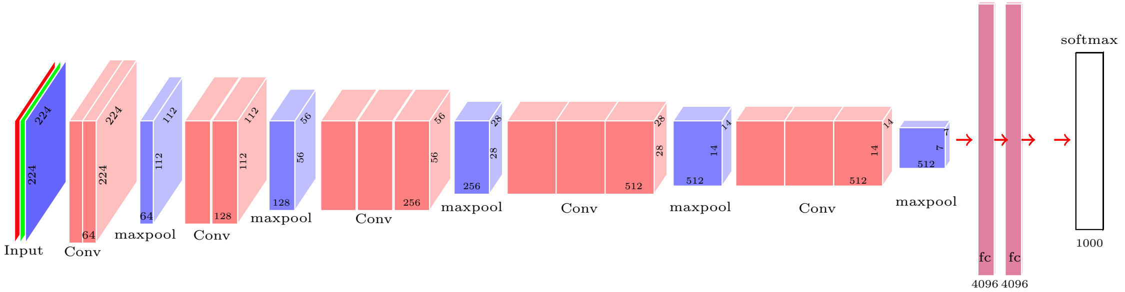
- Kernel size is 3×3 throughout
- Total parameters in non FC layers $= \sim 16M$
- Total Parameters in FC layers $=$ $(512 \times 7 \times 7 \times 4096)$



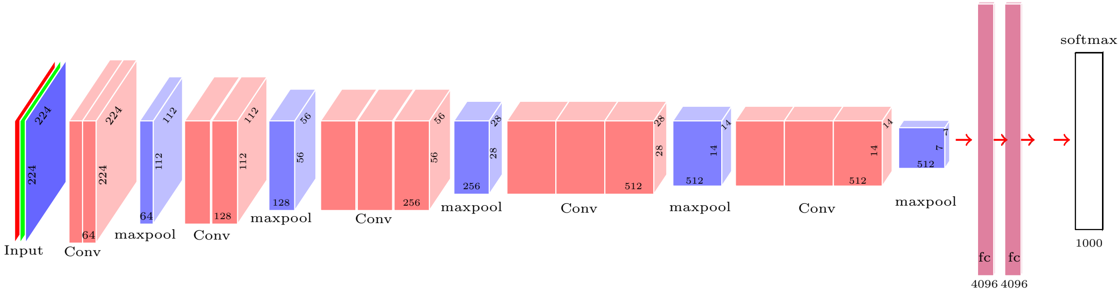
- Kernel size is 3×3 throughout
- Total parameters in non FC layers $= \sim 16M$
- Total Parameters in FC layers $= \ (512 \times 7 \times 7 \times 4096) + (4096 \times 4096)$



- Kernel size is 3×3 throughout
- Total parameters in non FC layers = $\sim 16M$
- Total Parameters in FC layers = $(512 \times 7 \times 7 \times 4096) + (4096 \times 4096) + (4096 \times 1024)$

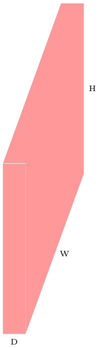


- Kernel size is 3×3 throughout
- Total parameters in non FC layers $= \sim 16M$
- Total Parameters in FC layers $=$ $(512 \times 7 \times 7 \times 4096) + (4096 \times 4096)$ $+ (4096 \times 1024) = \sim 122M$

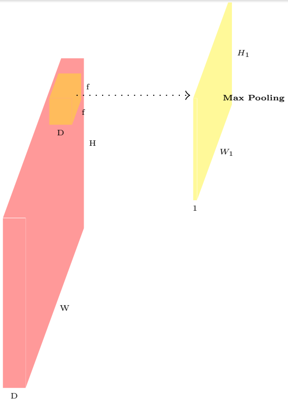


- Kernel size is 3×3 throughout
- Total parameters in non FC layers = $\sim 16M$
- Total Parameters in FC layers = $(512 \times 7 \times 7 \times 4096) + (4096 \times 4096) + (4096 \times 1024) = \sim 122M$
- Most parameters are in the first FC layer ($\sim$ 102M)

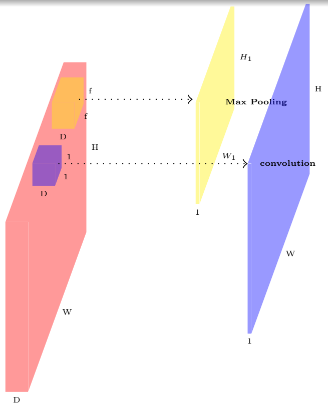
Module 11.5 : Image Classification continued (GoogLeNet and ResNet)



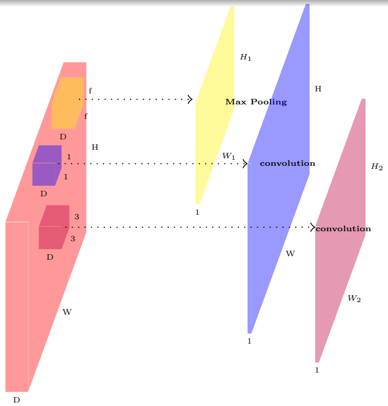
- Consider the output at a certain layer of a convolutional neural network



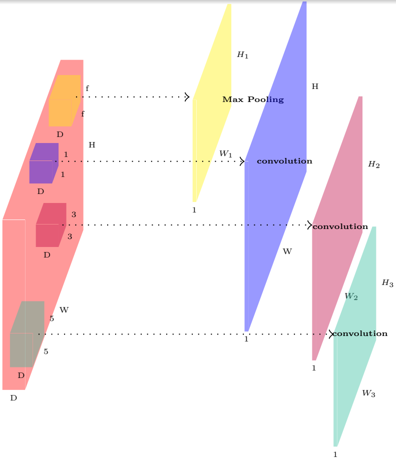
- Consider the output at a certain layer of a convolutional neural network
- After this layer we could apply a max-pooling layer



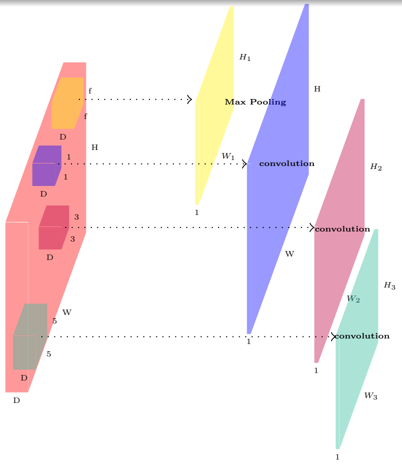
- Consider the output at a certain layer of a convolutional neural network
- After this layer we could apply a max-pooling layer
- Or a 1×1 convolution



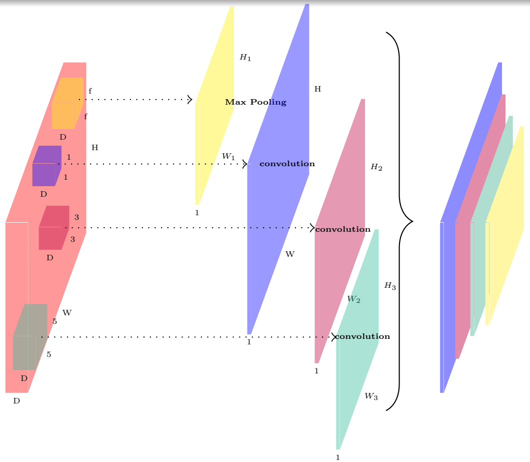
- Consider the output at a certain layer of a convolutional neural network
- After this layer we could apply a max-pooling layer
- Or a 1×1 convolution
- Or a 3×3 convolution



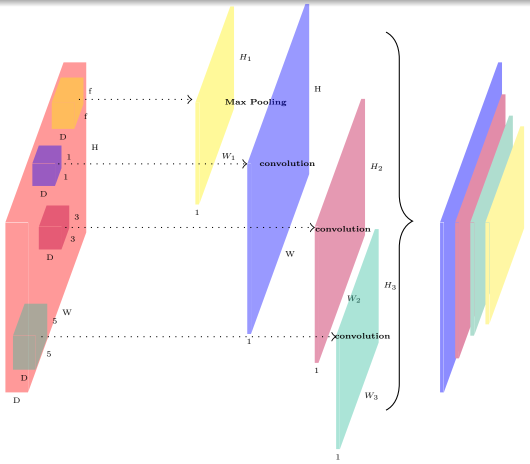
- Consider the output at a certain layer of a convolutional neural network
- After this layer we could apply a max-pooling layer
- Or a 1×1 convolution
- Or a 3×3 convolution
- Or a 5×5 convolution



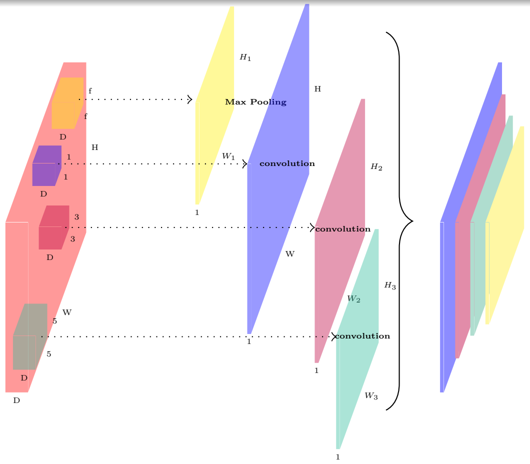
- Consider the output at a certain layer of a convolutional neural network
- After this layer we could apply a max-pooling layer
- Or a 1×1 convolution
- Or a 3×3 convolution
- Or a 5×5 convolution
- **Question:** Why choose between these options (convolution, maxpooling, filter sizes)?



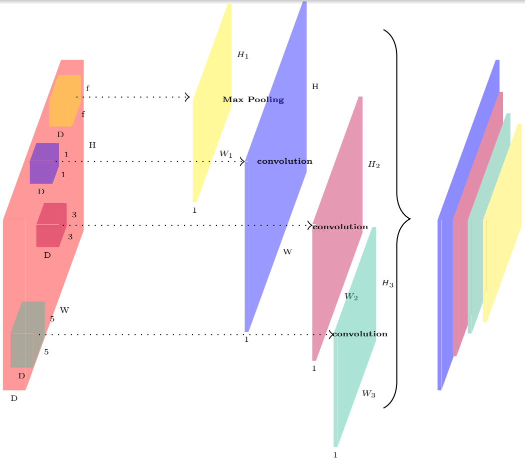
- Consider the output at a certain layer of a convolutional neural network
- After this layer we could apply a max-pooling layer
- Or a 1×1 convolution
- Or a 3×3 convolution
- Or a 5×5 convolution
- **Question:** Why choose between these options (convolution, maxpooling, filter sizes)?
- **Idea:** Why not apply all of them at the same time and then concatenate the feature maps?



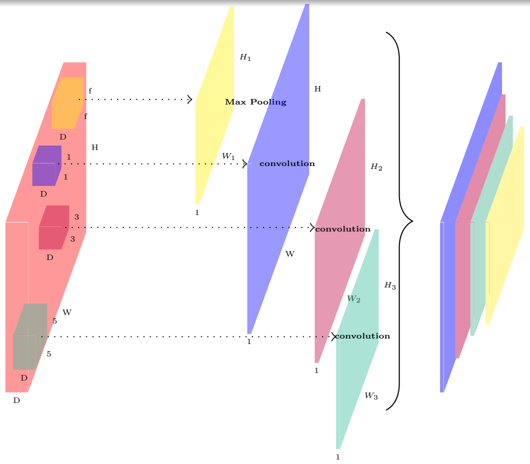
- Well this naive idea could result in a large number of computations



- Well this naive idea could result in a large number of computations
- If $P = 0$ & $S = 1$ then convolving a $W \times H \times D$ input with a $F \times F \times D$ filter results in a $(W - F + 1)(H - F + 1)$ sized output

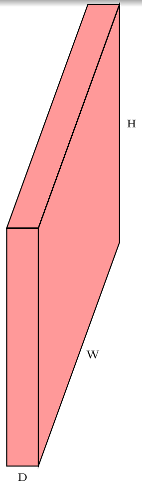


- Well this naive idea could result in a large number of computations
- If $P = 0$ & $S = 1$ then convolving a $W \times H \times D$ input with a $F \times F \times D$ filter results in a $(W - F + 1)(H - F + 1)$ sized output
- Each element of the output requires $O(F \times F \times D)$ computations

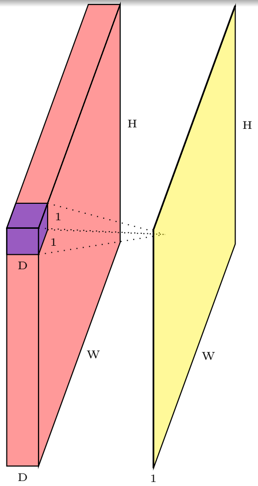


- Well this naive idea could result in a large number of computations
- If $P = 0$ & $S = 1$ then convolving a $W \times H \times D$ input with a $F \times F \times D$ filter results in a $(W - F + 1)(H - F + 1)$ sized output
- Each element of the output requires $O(F \times F \times D)$ computations
- Can we reduce the number of computations?

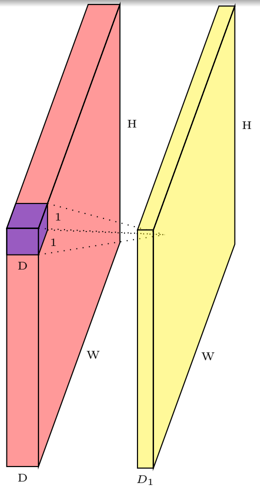
- Yes, by using 1×1 convolutions



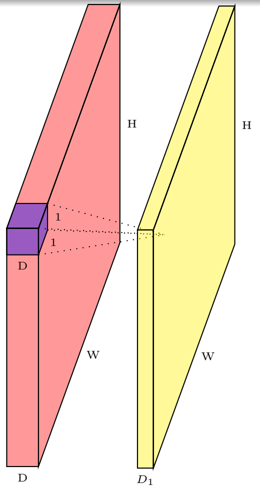
- Yes, by using 1×1 convolutions
- Huh?? What does a 1×1 convolution do ?



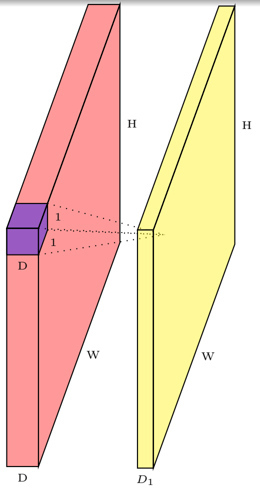
- Yes, by using 1×1 convolutions
- Huh?? What does a 1×1 convolution do ?
- It aggregates along the depth



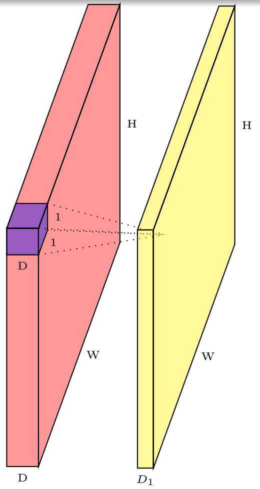
- Yes, by using 1×1 convolutions
- Huh?? What does a 1×1 convolution do ?
- It aggregates along the depth
- So convolving a $D \times W \times H$ input with D_1 1×1 ($D_1 < D$) filters will result in a $D_1 \times W \times H$ output ($S = 1, P = 0$)



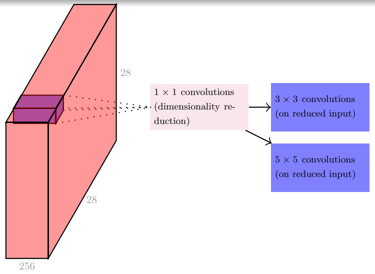
- Yes, by using 1×1 convolutions
- Huh?? What does a 1×1 convolution do ?
- It aggregates along the depth
- So convolving a $D \times W \times H$ input with D_1 1×1 ($D_1 < D$) filters will result in a $D_1 \times W \times H$ output ($S = 1, P = 0$)
- If $D_1 < D$ then this effectively reduces the dimension of the input and hence the computations



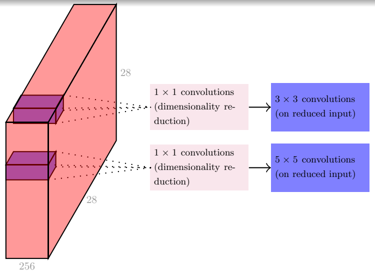
- Yes, by using 1×1 convolutions
- Huh?? What does a 1×1 convolution do ?
- It aggregates along the depth
- So convolving a $D \times W \times H$ input with D_1 1×1 ($D_1 < D$) filters will result in a $D_1 \times W \times H$ output ($S = 1, P = 0$)
- If $D_1 < D$ then this effectively reduces the dimension of the input and hence the computations
- Specifically instead of O($F \times F \times D$) we will need O($F \times F \times D_1$) computations



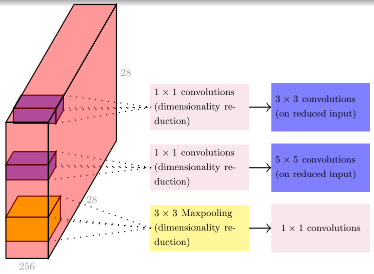
- Yes, by using 1×1 convolutions
- Huh?? What does a 1×1 convolution do ?
- It aggregates along the depth
- So convolving a $D \times W \times H$ input with D_1 1×1 $(D_1 < D)$ filters will result in a $D_1 \times W \times H$ output $(S = 1, P = 0)$
- If $D_1 < D$ then this effectively reduces the dimension of the input and hence the computations
- Specifically instead of $\mathrm{O}(F \times F \times D)$ we will need $\mathrm{O}(F \times F \times D_1)$ computations
- We could then apply subsequent 3×3, 5×5 filter on this reduced output



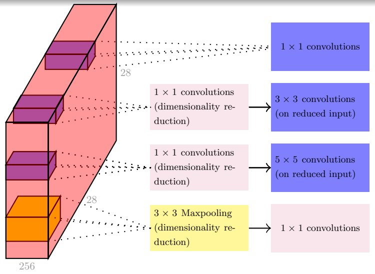
- But we might want to use different dimensionality reductions before the 3×3 and 5×5 filters



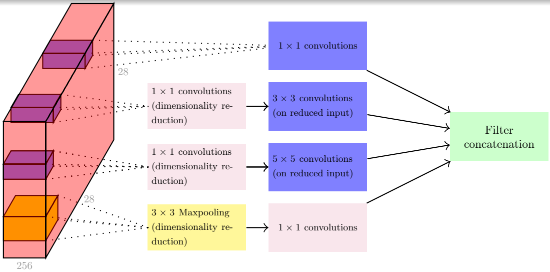
- But we might want to use different dimensionality reductions before the 3×3 and 5×5 filters
- So we can use D_1 and D_2 1×1 filters before the 3×3 and 5×5 filters respectively



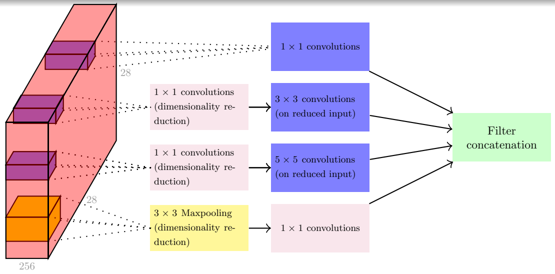
- But we might want to use different dimensionality reductions before the 3×3 and 5×5 filters
- So we can use D_1 and D_2 1×1 filters before the 3×3 and 5×5 filters respectively
- We can then add the maxpooling layer followed by dimensionality reduction



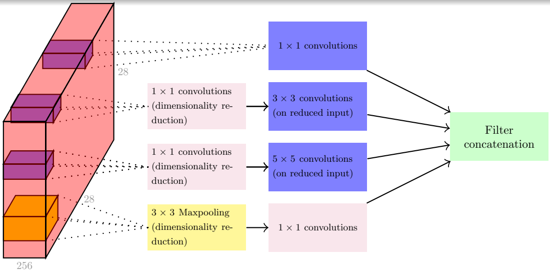
- But we might want to use different dimensionality reductions before the 3×3 and 5×5 filters
- So we can use D_1 and D_2 1×1 filters before the 3×3 and 5×5 filters respectively
- We can then add the maxpooling layer followed by dimensionality reduction
- And a new set of 1×1 convolutions



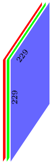
- But we might want to use different dimensionality reductions before the 3×3 and 5×5 filters
- So we can use D_1 and D_2 1×1 filters before the 3×3 and 5×5 filters respectively
- We can then add the maxpooling layer followed by dimensionality reduction
- And a new set of 1×1 convolutions
- And finally we concatenate all these layers



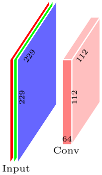
- But we might want to use different dimensionality reductions before the 3×3 and 5×5 filters
- So we can use D_1 and D_2 1×1 filters before the 3×3 and 5×5 filters respectively
- We can then add the maxpooling layer followed by dimensionality reduction
- And a new set of 1×1 convolutions
- And finally we concatenate all these layers
- This is called the **Inception module**

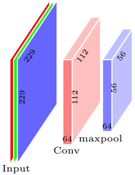


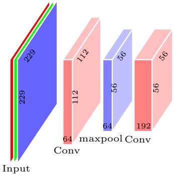
- But we might want to use different dimensionality reductions before the 3×3 and 5×5 filters
- So we can use D_1 and D_2 1×1 filters before the 3×3 and 5×5 filters respectively
- We can then add the maxpooling layer followed by dimensionality reduction
- And a new set of 1×1 convolutions
- And finally we concatenate all these layers
- This is called the **Inception module**
- We will now see **GoogLeNet** which contains many such inception modules

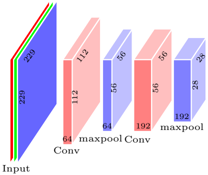


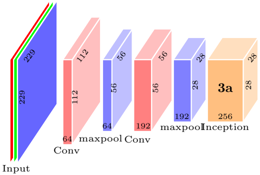
Input

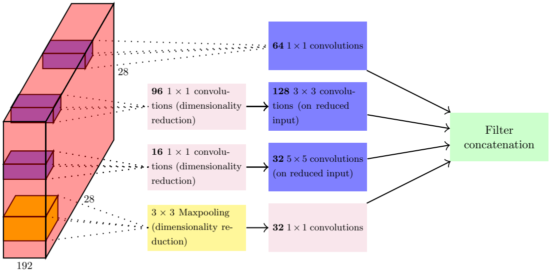

229
229
Input
112
112
64
Conv

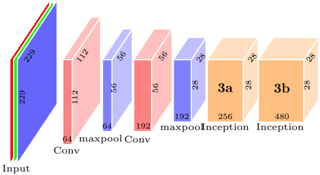

229
229
112
112
64
56
56
64
maxpool
Conv
Input

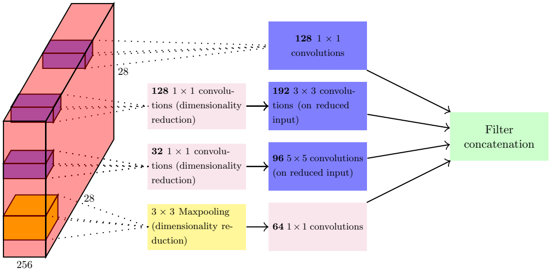

229
229
112
112
64
Conv
56
56
64
maxpool
56
56
192
Conv
Input

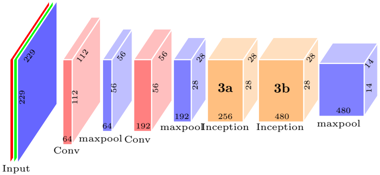

229
229
Input
112
112
64
Conv
56
56
64
maxpool
56
56
192
Conv
28
28
192
maxpool


229
229
Input
112
112
64
Conv
56
56
64
maxpool
56
56
192
Conv
28
28
192
maxpool
3a
28
28
256
Inception

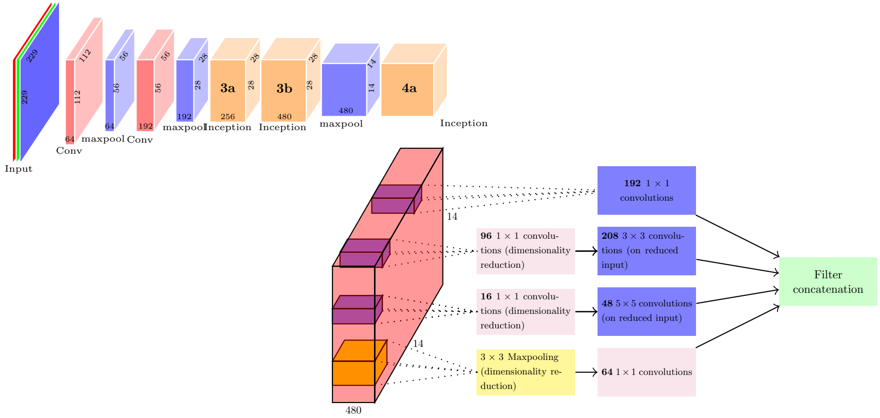

28
28
192
64 1 × 1 convolutions
96 1 × 1 convolutions (dimensionality reduction)
128 3 × 3 convolutions (on reduced input)
16 1 × 1 convolutions (dimensionality reduction)
32 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
32 1 × 1 convolutions
Filter concatenation


229
229
112
112
64
Conv
56
56
64
maxpool
56
56
192
Conv
28
28
192
maxpool
3a
28
28
256
Inception
3b
28
28
480
Inception
Input

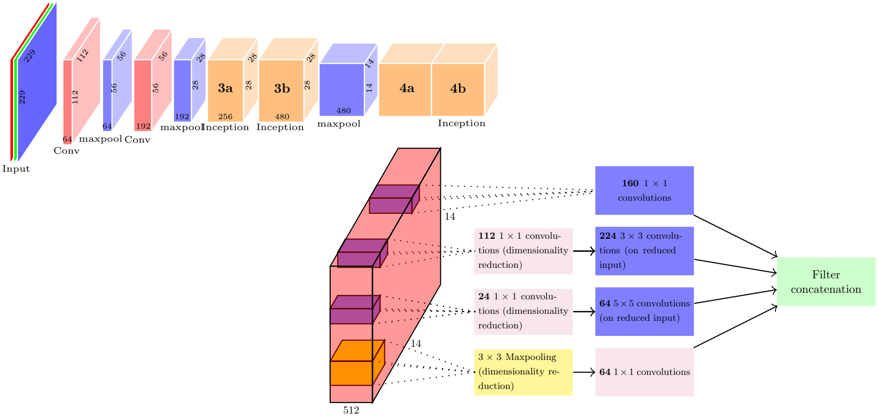

28
28
256
128 1 × 1 convolutions
128 1 × 1 convolutions (dimensionality reduction)
192 3 × 3 convolutions (on reduced input)
32 1 × 1 convolutions (dimensionality reduction)
96 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
64 1 × 1 convolutions
Filter concatenation



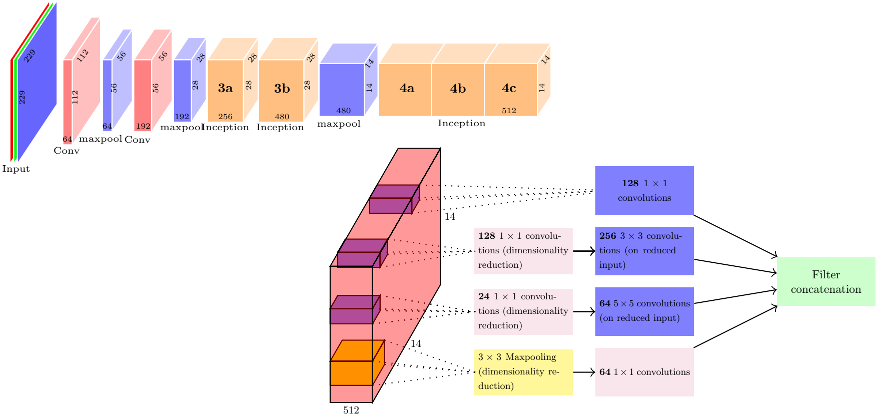
229
229
Input
112
112
64
Conv
56
56
64
maxpool
56
56
192
Conv
28
28
192
maxpool
3a
28
28
256
Inception
3b
28
28
480
Inception
14
14
480
maxpool

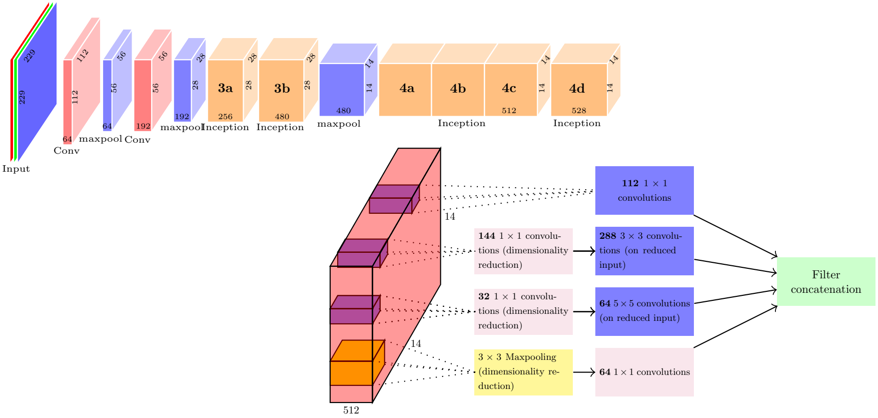


229
229
Input
112
112
64
Conv
56
56
64
maxpool
56
56
192
Conv
28
28
192
maxpool
3a
28
28
256
Inception
3b
28
28
480
Inception
14
14
480
maxpool
4a
Inception
14
14
480
192 1 × 1 convolutions
96 1 × 1 convolutions (dimensionality reduction)
208 3 × 3 convolutions (on reduced input)
16 1 × 1 convolutions (dimensionality reduction)
48 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
64 1 × 1 convolutions
Filter concatenation

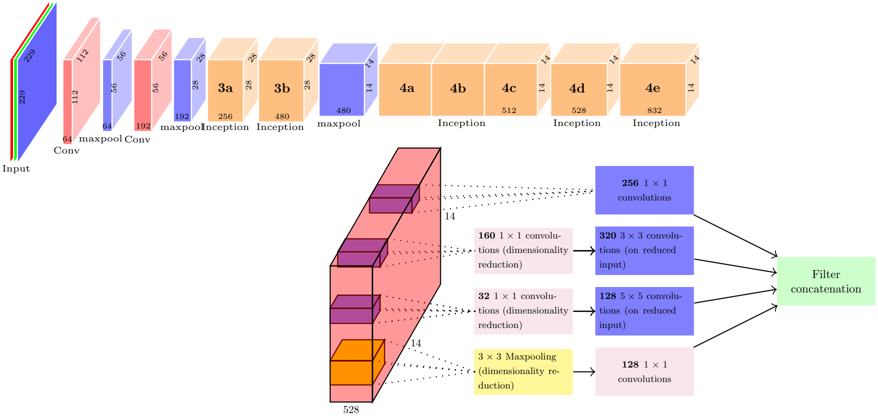


229
229
Input
112
112
64
Conv
56
56
64
maxpool
56
56
192
Conv
28
28
192
maxpool
3a
28
28
256
Inception
3b
28
28
480
Inception
14
14
480
maxpool
4a
4b
Inception
14
14
512
160 1 × 1 convolutions
112 1 × 1 convolutions (dimensionality reduction)
224 3 × 3 convolutions (on reduced input)
24 1 × 1 convolutions (dimensionality reduction)
64 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
64 1 × 1 convolutions
Filter concatenation

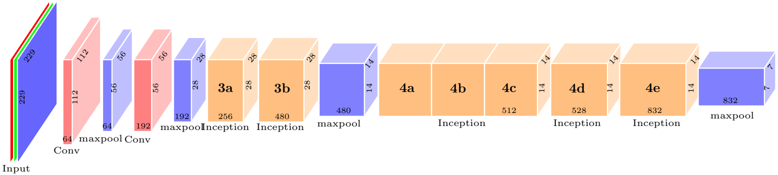

229
229
Input
112
112
64
Conv
56
56
64
maxpool
56
56
192
Conv
28
28
192
maxpool
3a
28
28
256
Inception
3b
28
28
480
Inception
14
14
480
maxpool
4a
4b
4c
14
14
512
Inception
14
14
512
128 1 × 1 convolutions
128 1 × 1 convolutions (dimensionality reduction)
256 3 × 3 convolutions (on reduced input)
24 1 × 1 convolutions (dimensionality reduction)
64 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
64 1 × 1 convolutions
Filter concatenation

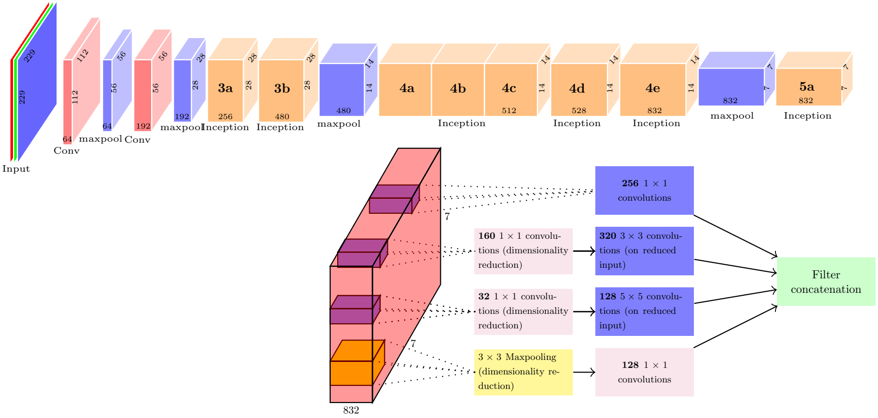


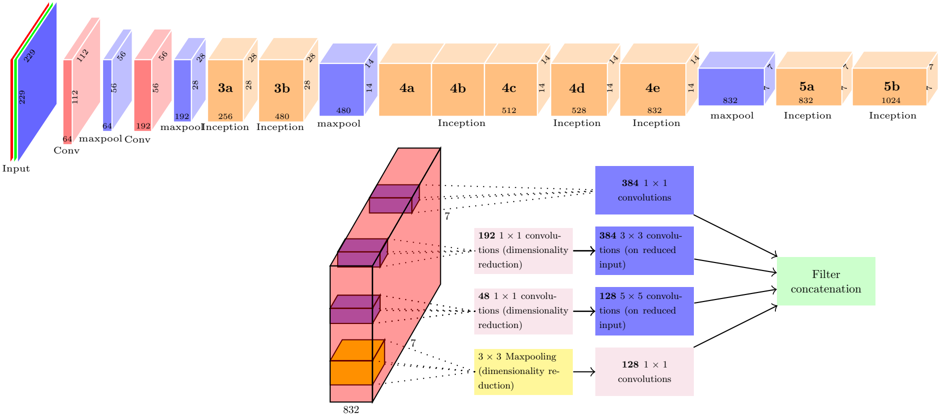
229
229
Input
112
112
64
Conv
56
56
64
maxpool
56
56
192
Conv
28
28
192
maxpool
3a
28
28
256
Inception
3b
28
28
480
Inception
14
14
480
maxpool
4a
4b
4c
14
14
512
Inception
4d
14
14
528
Inception
14
14
512
112 1 × 1 convolutions
144 1 × 1 convolutions (dimensionality reduction)
288 3 × 3 convolutions (on reduced input)
32 1 × 1 convolutions (dimensionality reduction)
64 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
64 1 × 1 convolutions
Filter concatenation

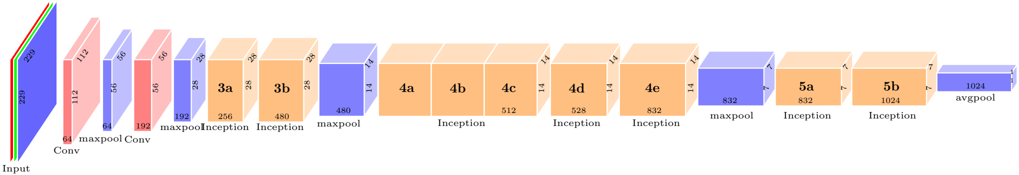


229
229
Input
112
112
64
Conv
56
56
64
maxpool
56
56
192
Conv
28
28
192
maxpool
3a
28
28
256
Inception
3b
28
28
480
Inception
14
14
480
maxpool
4a
4b
4c
14
14
512
Inception
4d
14
14
528
Inception
4e
14
14
832
Inception
14
14
528
256 1 × 1 convolutions
160 1 × 1 convolutions (dimensionality reduction)
320 3 × 3 convolutions (on reduced input)
32 1 × 1 convolutions (dimensionality reduction)
128 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
128 1 × 1 convolutions
Filter concatenation

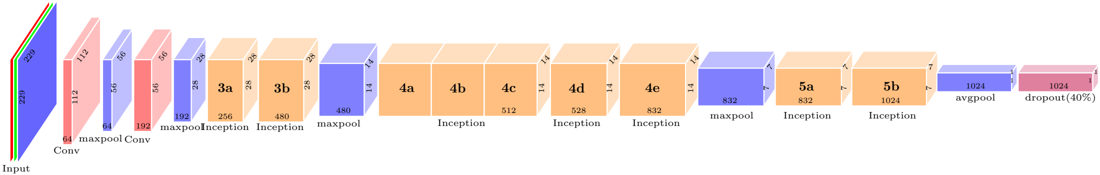

229
229
Input
112
112
64
Conv
56
56
64
maxpool
56
56
192
Conv
28
28
192
maxpool
3a
28
28
256
Inception
3b
28
28
480
Inception
14
14
480
maxpool
4a
4b
4c
14
14
512
Inception
4d
14
14
528
Inception
4e
14
14
832
Inception
7
7
832
maxpool

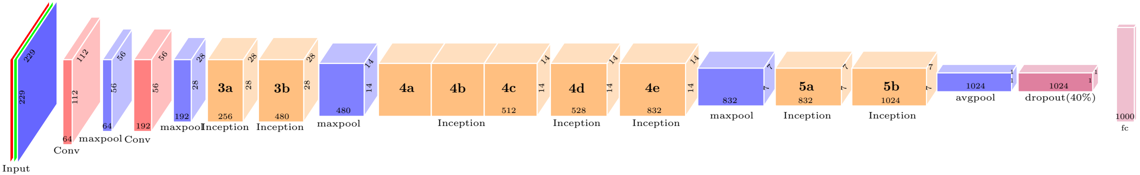

Input
229
229
Conv
112
112
64
maxpool
56
56
64
Conv
56
56
192
maxpool
28
28
192
Inception
3a
28
28
256
Inception
3b
28
28
480
maxpool
14
14
480
4a
4b
4c
14
14
512
Inception
4d
14
14
528
Inception
4e
14
14
832
Inception
maxpool
7
7
832
5a
7
7
832
Inception
7
7
832
256 1 × 1 convolutions
160 1 × 1 convolutions (dimensionality reduction)
320 3 × 3 convolutions (on reduced input)
32 1 × 1 convolutions (dimensionality reduction)
128 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
128 1 × 1 convolutions
Filter concatenation

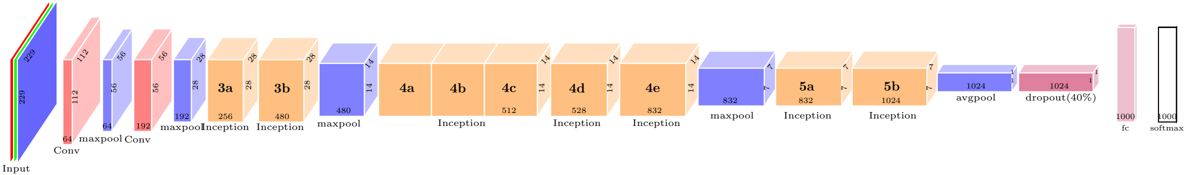

Input
229
229
112
112
64
Conv
56
56
64
maxpool
56
56
192
Conv
28
28
192
maxpool
3a
28
28
256
Inception
3b
28
28
480
Inception
14
14
480
maxpool
4a
4b
4c
14
14
512
Inception
4d
14
14
528
Inception
4e
14
14
832
Inception
7
7
832
maxpool
5a
7
7
832
Inception
5b
7
7
1024
Inception
7
7
832
384 1 × 1 convolutions
192 1 × 1 convolutions (dimensionality reduction)
384 3 × 3 convolutions (on reduced input)
48 1 × 1 convolutions (dimensionality reduction)
128 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
128 1 × 1 convolutions
Filter concatenation



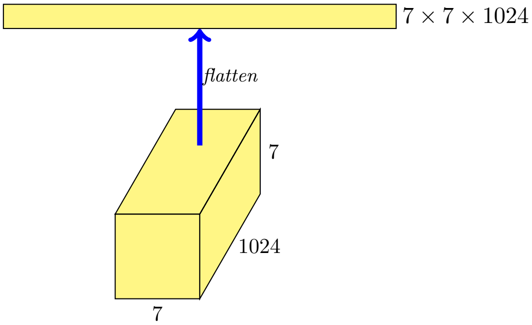
229
229
Input
112
112
64
Conv
56
56
64
maxpool
56
56
192
Conv
28
28
192
maxpool
3a
28
28
256
Inception
3b
28
28
480
Inception
14
14
480
maxpool
4a
4b
4c
14
14
512
Inception
4d
14
14
528
Inception
4e
14
14
832
Inception
7
7
832
maxpool
5a
7
7
832
Inception
5b
7
7
1024
Inception
1
1
1024
avgpool


Input
229
229
Conv
64
112
112
maxpool
64
56
56
Conv
192
56
56
maxpool
192
28
28
Inception
3a
256
28
28
Inception
3b
480
28
28
maxpool
480
14
14
Inception
4a
4b
4c
512
14
14
Inception
4d
528
14
14
Inception
4e
832
14
14
maxpool
832
7
7
Inception
5a
832
7
7
Inception
5b
1024
7
7
avgpool
1024
1
1
dropout(40%)
1024
1
1

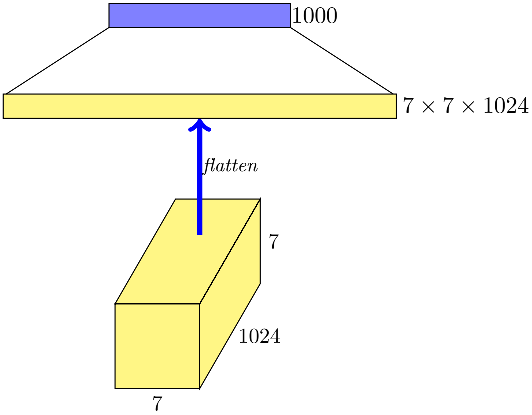

229
229
Input
112
112
64
Conv
56
56
64
maxpool
56
56
192
Conv
28
28
192
maxpool
3a
28
28
256
Inception
3b
28
28
480
Inception
14
14
480
maxpool
4a
4b
4c
14
14
512
Inception
4d
14
14
528
Inception
4e
14
14
832
Inception
7
7
832
maxpool
5a
7
7
832
Inception
5b
7
7
1024
Inception
1
1
1024
avgpool
1
1
1024
dropout(40%)
1000
fc


Input
229
229
Conv
112
112
64
maxpool
56
56
64
Conv
56
56
192
maxpool
28
28
192
Inception
3a
28
28
256
Inception
3b
28
28
480
maxpool
14
14
480
4a
4b
4c
14
14
512
Inception
4d
14
14
528
Inception
4e
14
14
832
Inception
maxpool
7
7
832
5a
7
7
832
Inception
5b
7
7
1024
Inception
1024
1
1
avgpool
1024
1
1
dropout(40%)
1000
fc
1000
softmax

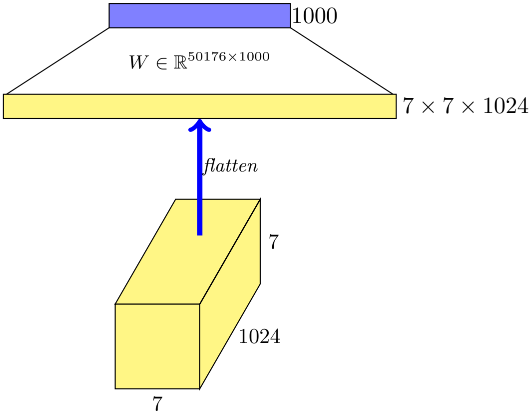
- **Important Trick:** Got rid of the fully connected layer



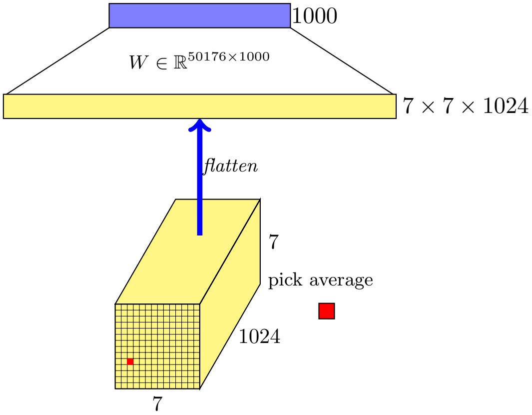
- **Important Trick:** Got rid of the fully connected layer
- Notice that output of the last layer is $7 \times 7 \times 1024$ dimensional



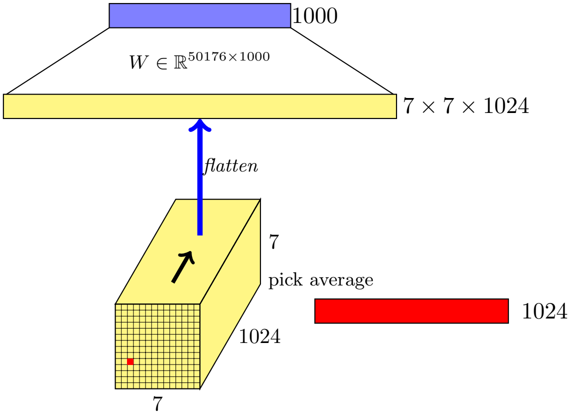
- **Important Trick:** Got rid of the fully connected layer
- Notice that output of the last layer is $7 \times 7 \times 1024$ dimensional
- What if we were to add a fully connected layer with 1000 nodes (for 1000 classes) on top of this



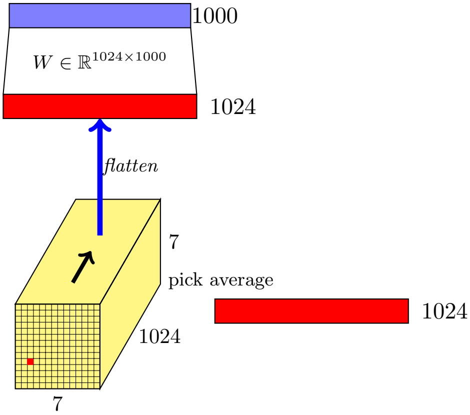
- **Important Trick:** Got rid of the fully connected layer
- Notice that output of the last layer is $7 \times 7 \times 1024$ dimensional
- What if we were to add a fully connected layer with 1000 nodes (for 1000 classes) on top of this
- We would have $7 \times 7 \times 1024 \times 1000 = 49M$ *parameters*



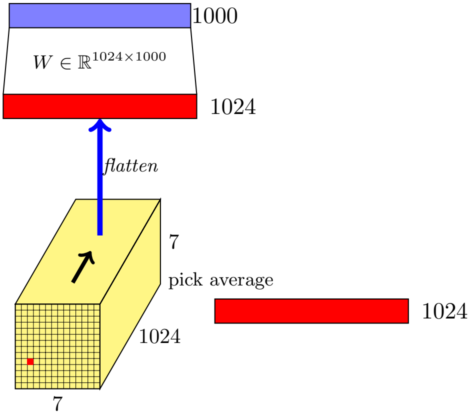
- **Important Trick:** Got rid of the fully connected layer
- Notice that output of the last layer is $7 \times 7 \times 1024$ dimensional
- What if we were to add a fully connected layer with 1000 nodes (for 1000 classes) on top of this
- We would have $7 \times 7 \times 1024 \times 1000 = 49M$ *parameters*
- Instead they use an average pooling of size 7×7 on each of the 1024 feature maps



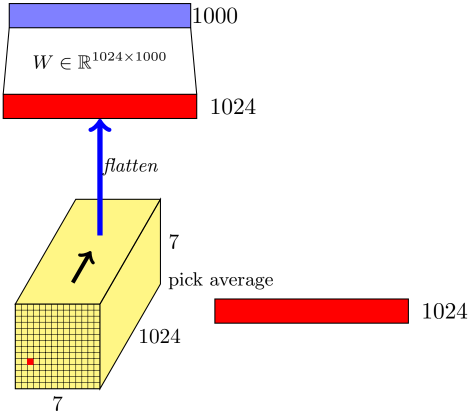
- **Important Trick:** Got rid of the fully connected layer
- Notice that output of the last layer is $7 \times 7 \times 1024$ dimensional
- What if we were to add a fully connected layer with 1000 nodes (for 1000 classes) on top of this
- We would have $7 \times 7 \times 1024 \times 1000 = 49M$ *parameters*
- Instead they use an average pooling of size 7×7 on each of the 1024 feature maps
- This results in a 1024 dimensional output



- **Important Trick:** Got rid of the fully connected layer
- Notice that output of the last layer is $7 \times 7 \times 1024$ dimensional
- What if we were to add a fully connected layer with 1000 nodes (for 1000 classes) on top of this
- We would have $7 \times 7 \times 1024 \times 1000 = 49M$ *parameters*
- Instead they use an average pooling of size 7×7 on each of the 1024 feature maps
- This results in a 1024 dimensional output
- Significantly reduces the number of parameters

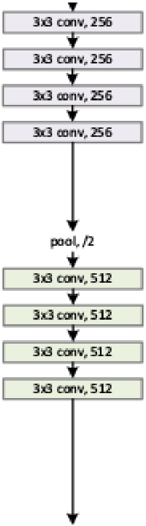


- $12\times$ less parameters than AlexNet

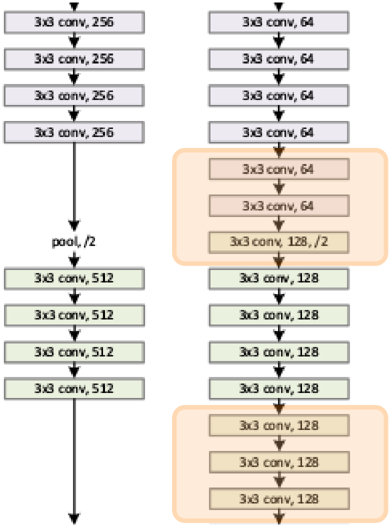


- $12\times$ less parameters than AlexNet
- $2\times$ more computations

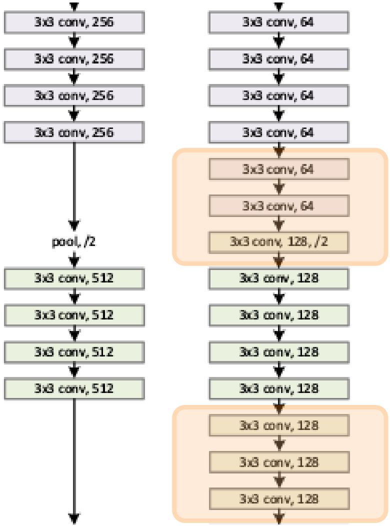
- GoogLeNet
- ResNet



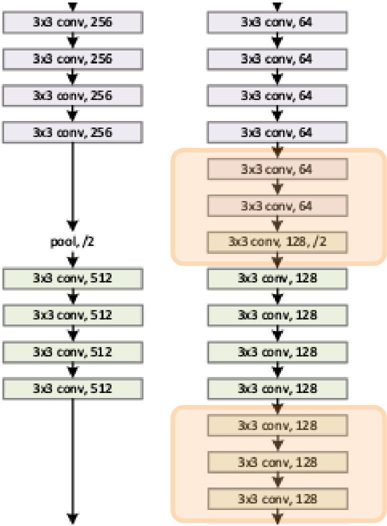
- Suppose we have been able to train a shallow neural network well



- Suppose we have been able to train a shallow neural network well
- Now suppose we construct a deeper network which has few more layers (in orange)

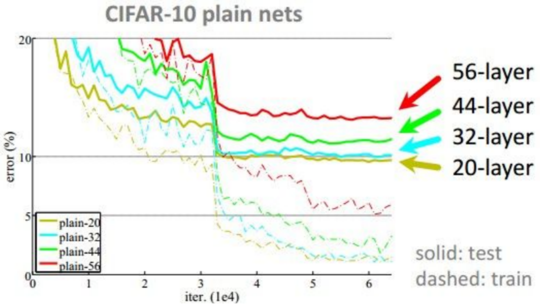


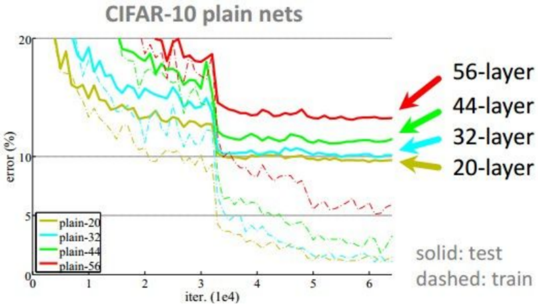
- Suppose we have been able to train a shallow neural network well
- Now suppose we construct a deeper network which has few more layers (in orange)
- Intuitively, if the shallow network works well then the deep network should also work well by simply learning to compute identity functions in the new layers



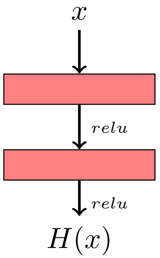
- Suppose we have been able to train a shallow neural network well
- Now suppose we construct a deeper network which has few more layers (in orange)
- Intuitively, if the shallow network works well then the deep network should also work well by simply learning to compute identity functions in the new layers
- Essentially, the solution space of a shallow neural network is a subset of the solution space of a deep neural network

- But in practice it is observed that this doesn't happen

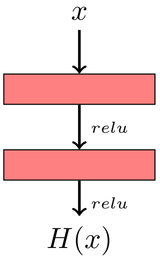




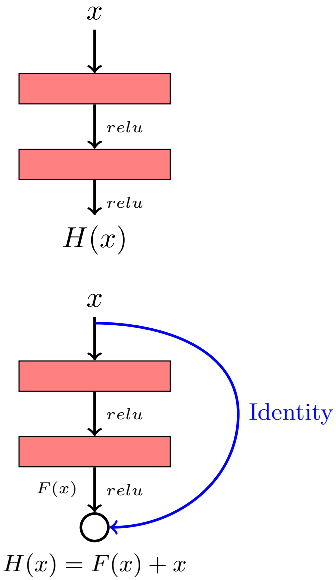
- But in practice it is observed that this doesn't happen
- Notice that the deep layers have a higher error rate on the test set



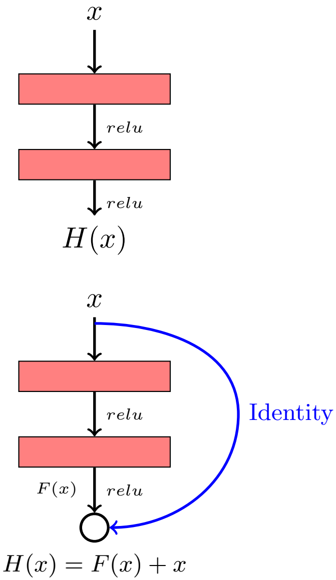
- Consider any two stacked layers in a CNN



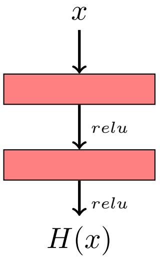
- Consider any two stacked layers in a CNN
- The two layers are essentially learning some function of the input

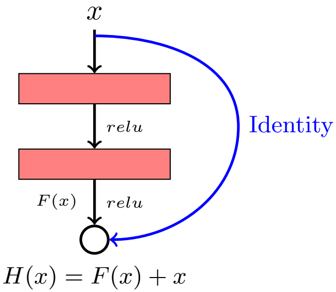


- Consider any two stacked layers in a CNN
- The two layers are essentially learning some function of the input
- What if we enable it to learn only a residual function of the input?

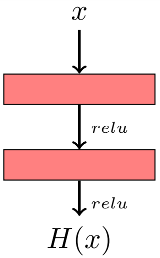


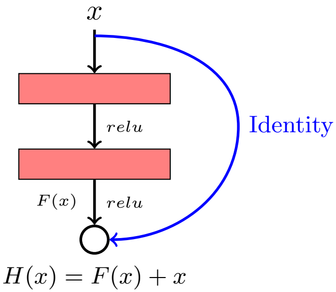
- Why would this help?



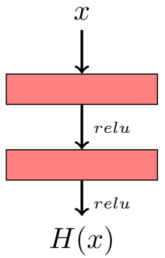


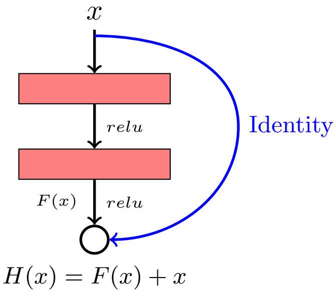
- Why would this help?
- Remember our argument that a deeper version of a shallow network would do just fine by learning identity transformations in the new layers





- Why would this help?
- Remember our argument that a deeper version of a shallow network would do just fine by learning identity transformations in the new layers
- This identity connection from the input allows a ResNet to retain a copy of the input





- Why would this help?
- Remember our argument that a deeper version of a shallow network would do just fine by learning identity transformations in the new layers
- This identity connection from the input allows a ResNet to retain a copy of the input
- Using this idea they were able to train really deep networks

ResNet, 152 layers

1^{st} place in all five main tracks

- **ImageNet Classification:** "Ultra-deep" 152-layer nets

ResNet, 152 layers

1^{st} place in all five main tracks

- **ImageNet Classification:** "Ultra-deep" 152-layer nets
- **ImageNet Detection:** 16% better than the 2nd best system

ResNet, 152 layers

1^{st} place in all five main tracks

- **ImageNet Classification:** "Ultra-deep" 152-layer nets
- **ImageNet Detection:** 16% better than the 2nd best system
- **ImageNet Localization:** 27% better than the 2nd best system

ResNet, 152 layers

1^{st} place in all five main tracks

- **ImageNet Classification:** "Ultra-deep" 152-layer nets
- **ImageNet Detection:** 16% better than the 2nd best system
- **ImageNet Localization:** 27% better than the 2nd best system
- **COCO Detection:** 11% better than the 2nd best system

ResNet, 152 layers

1^{st} place in all five main tracks

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

ResNet, 152 layers

Bag of tricks

- Batch Normalizaton after every CONV layer

ResNet, 152 layers

Bag of tricks

- Batch Normalizaton after every CONV layer
- Xavier/2 initialization from [He et al]

ResNet, 152 layers

Bag of tricks

- Batch Normalizaton after every CONV layer
- Xavier/2 initialization from [He et al]
- SGD + Momentum(0.9)

ResNet, 152 layers

Bag of tricks

- Batch Normalizaton after every CONV layer
- Xavier/2 initialization from [He et al]
- SGD + Momentum(0.9)
- Learning rate:0.1, divided by 10 when validation error plateaus

ResNet, 152 layers

Bag of tricks

- Batch Normalizaton after every CONV layer
- Xavier/2 initialization from [He et al]
- SGD + Momentum(0.9)
- Learning rate:0.1, divided by 10 when validation error plateaus
- Mini-batch size 256

ResNet, 152 layers

Bag of tricks

- Batch Normalizaton after every CONV layer
- Xavier/2 initialization from [He et al]
- SGD + Momentum(0.9)
- Learning rate:0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5

ResNet, 152 layers

Bag of tricks

- Batch Normalizaton after every CONV layer
- Xavier/2 initialization from [He et al]
- SGD + Momentum(0.9)
- Learning rate:0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used