Introduction to CNN

Convolutional Neural Network

Review: Convolution

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

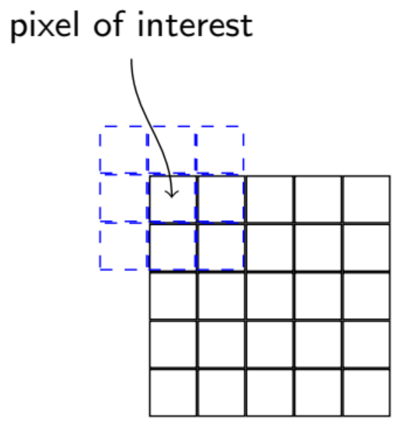
$$Y(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} W(u,v)X(i-u,j-v)$$

Review: Convolution

- More generally, given a $m_1 \times m_2$ filter K, we can write it as:

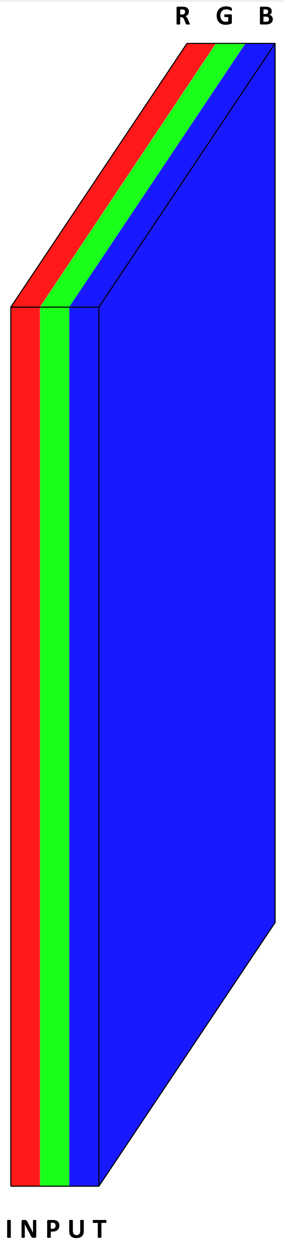
$$S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a,j-b} K_{\frac{m}{2}+a,\frac{n}{2}+b}$$

- This allows kernel to be centered on pixel of interest

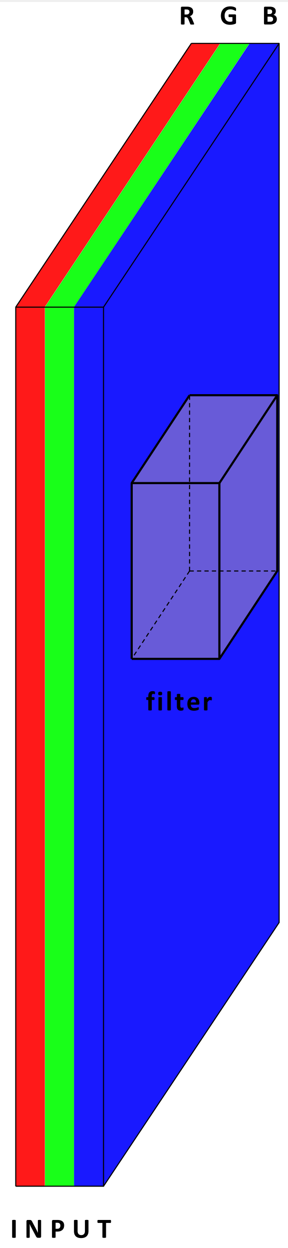


Pause and ponder

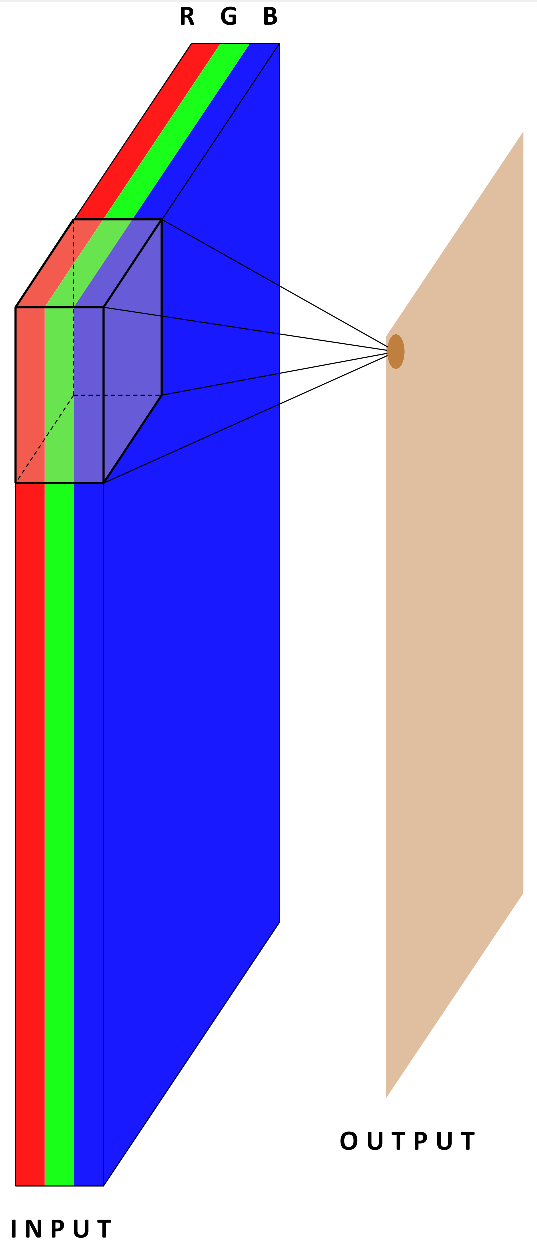
- In the 1D case, we slide a one dimensional filter over a one dimensional input
- In the 2D case, we slide a two dimensional filter over a two dimensional output
- What would happen in the 3D case, where images have RGB channels?



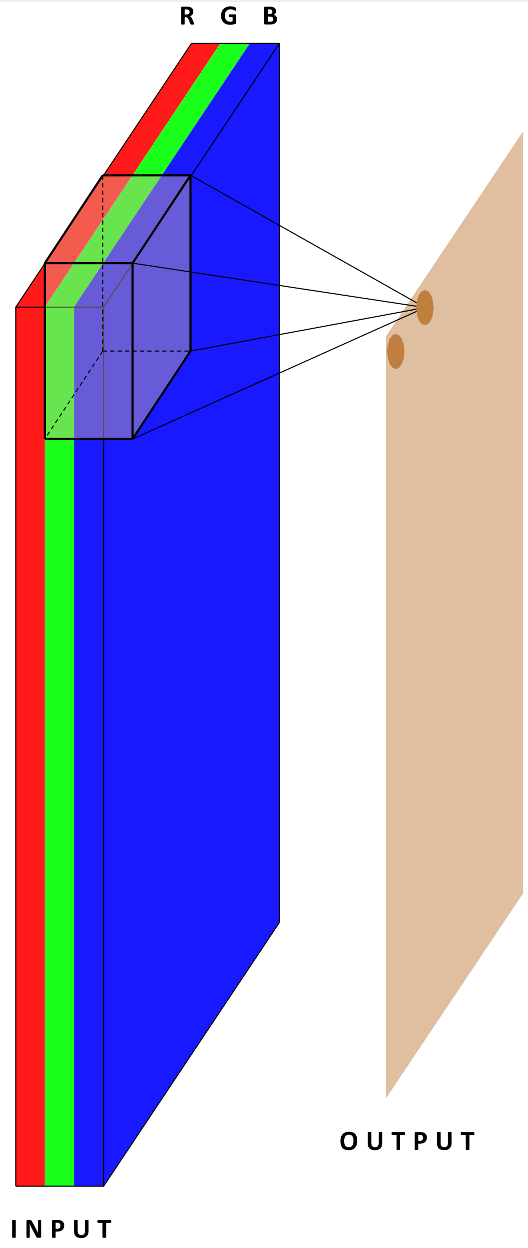
- What would a 3D filter look like?



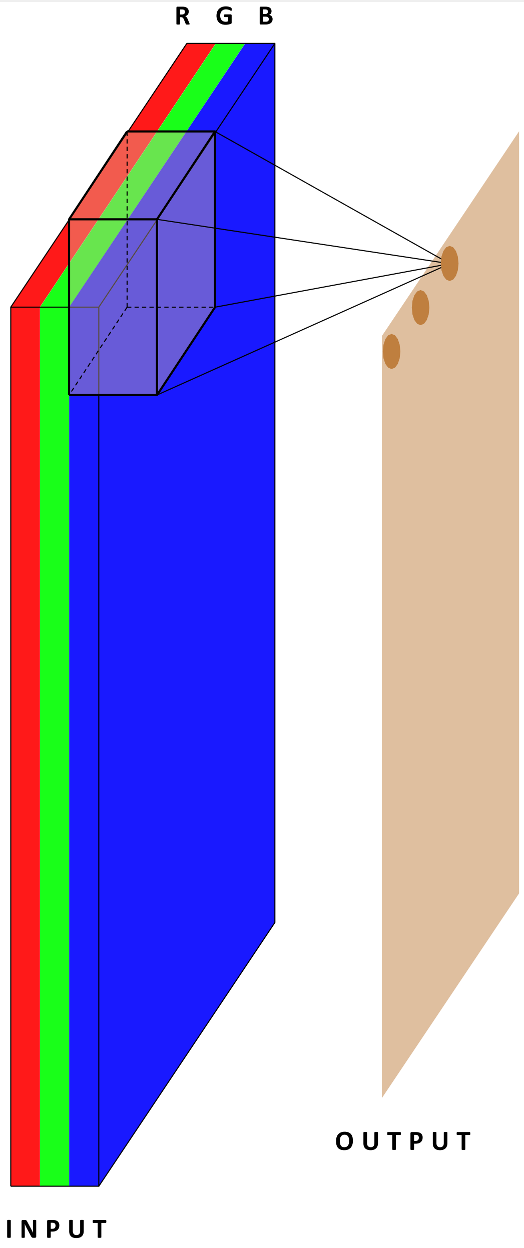
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume



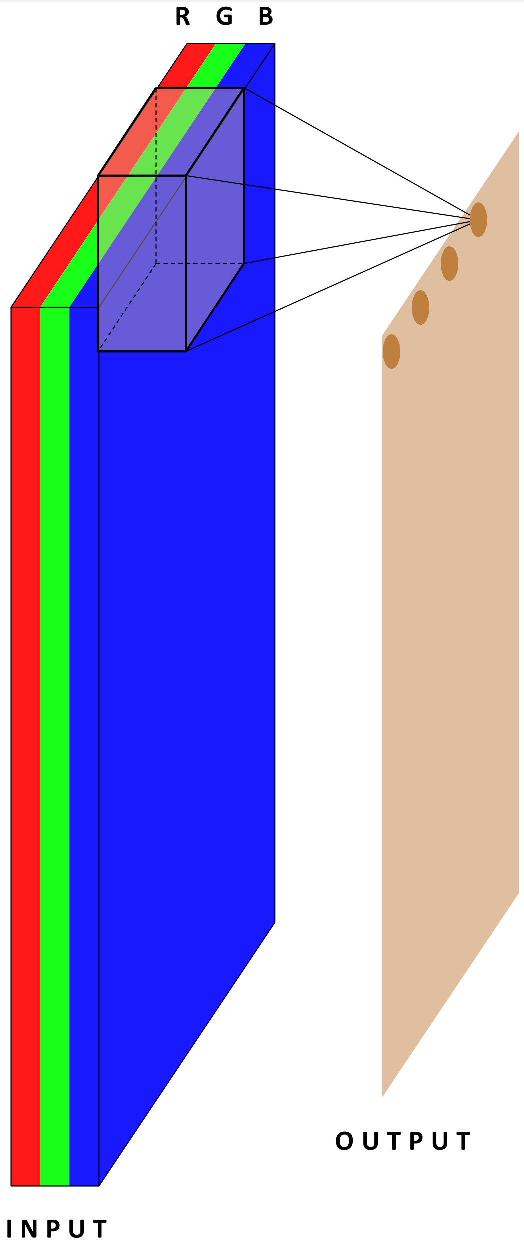
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



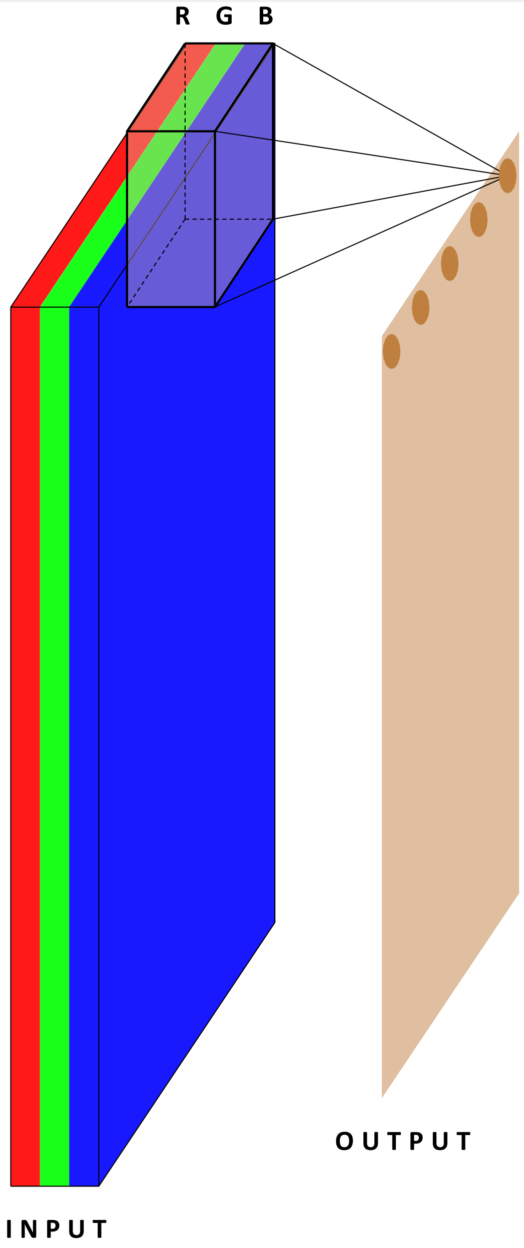
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



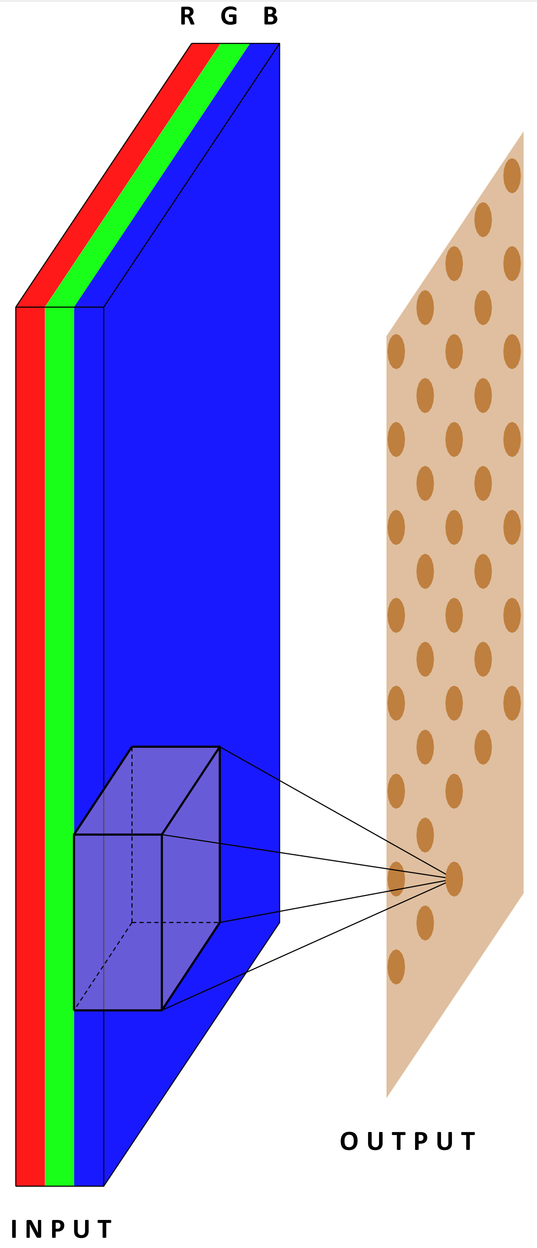
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



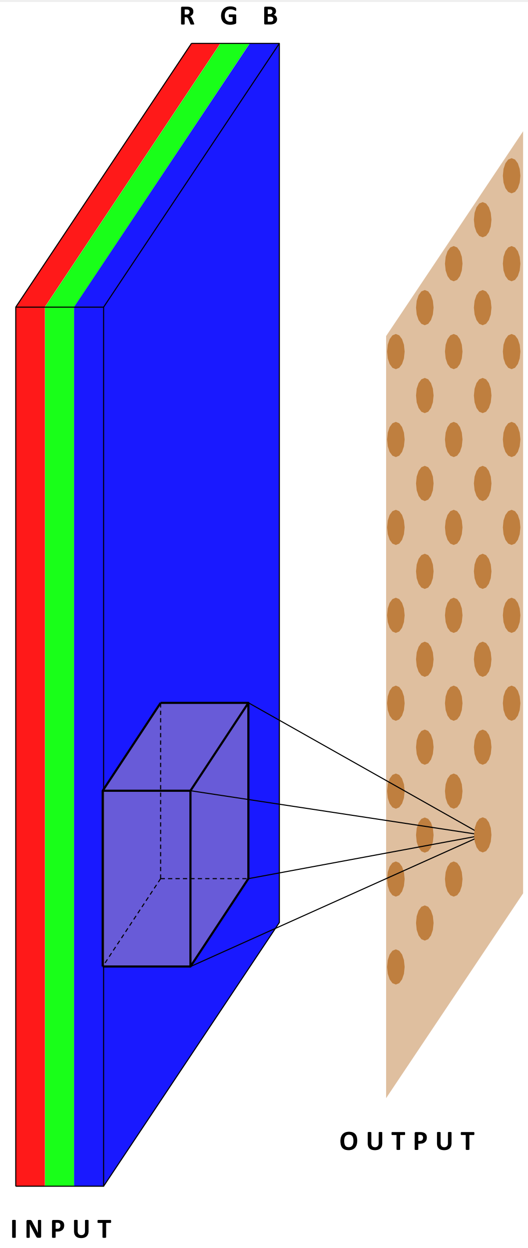
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



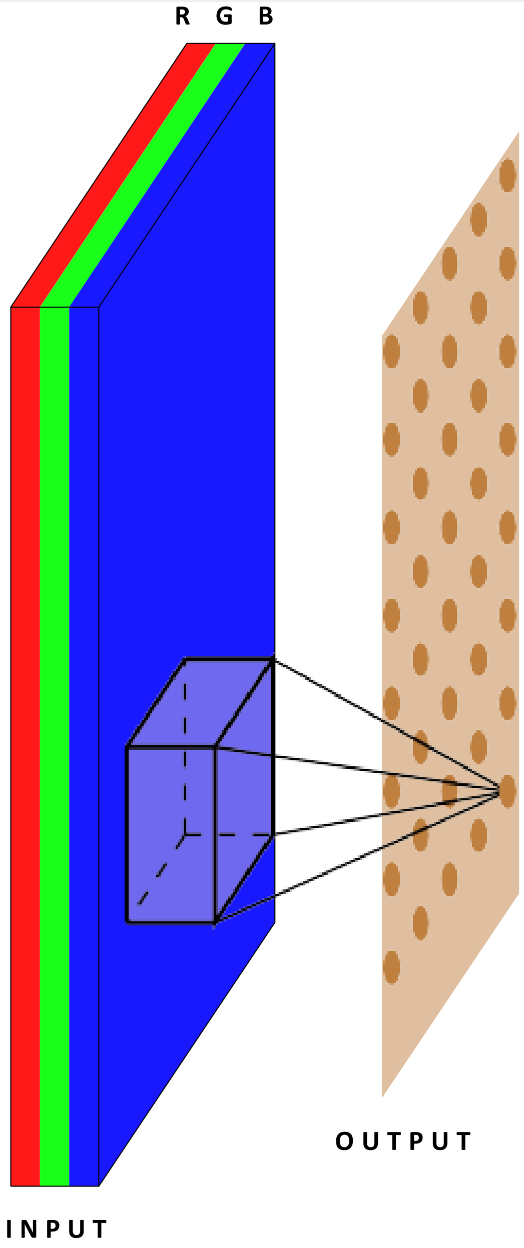
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



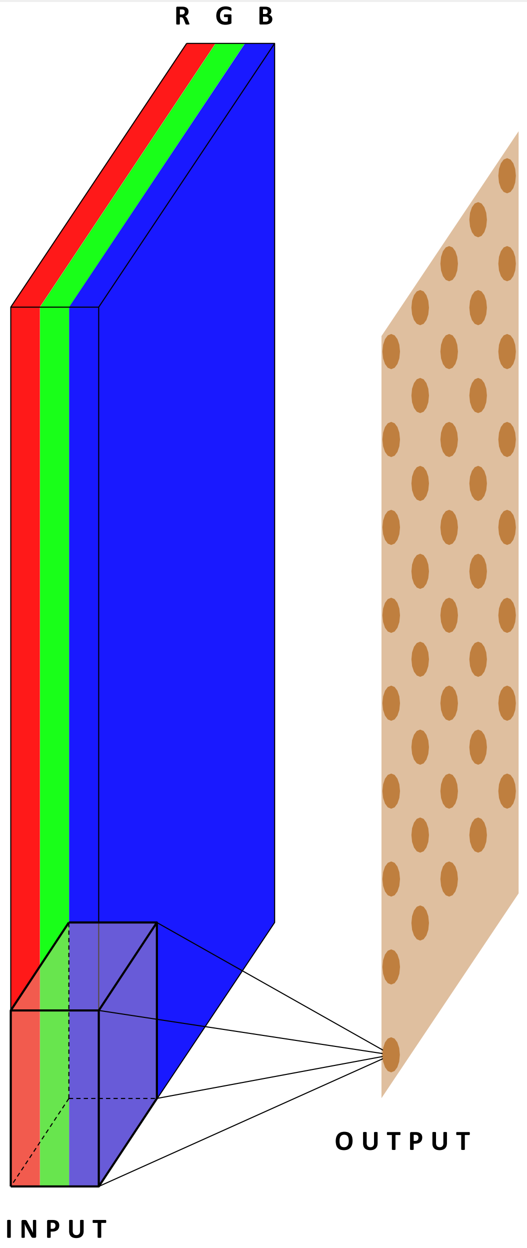
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



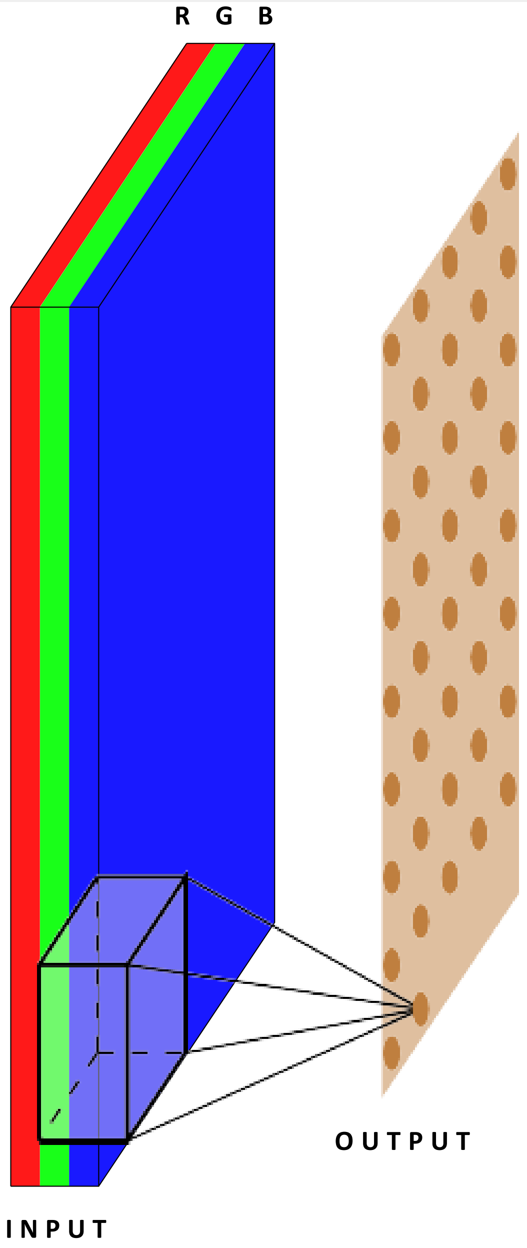
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



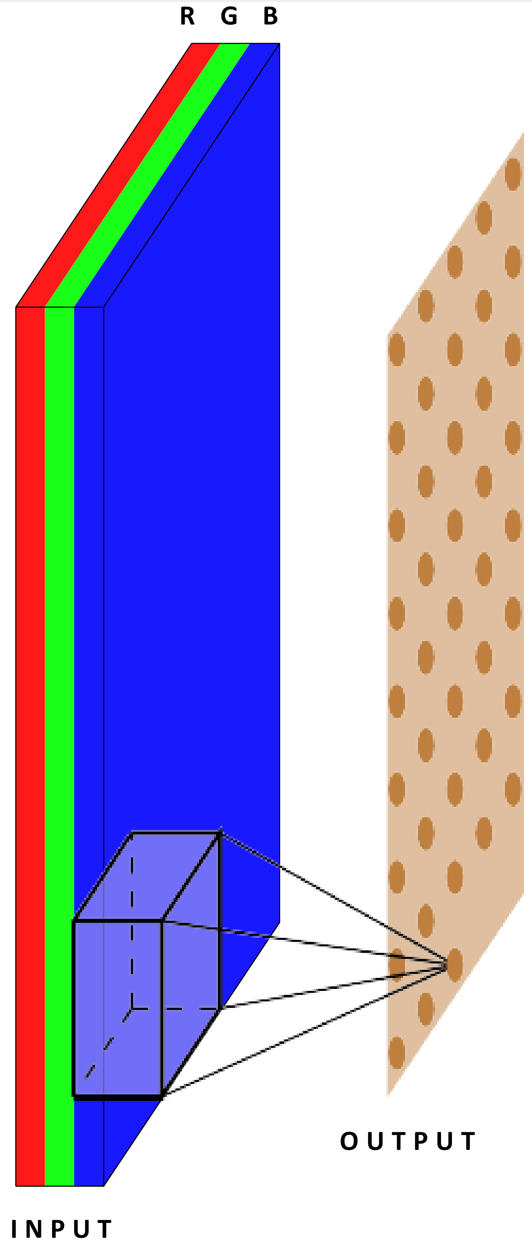
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



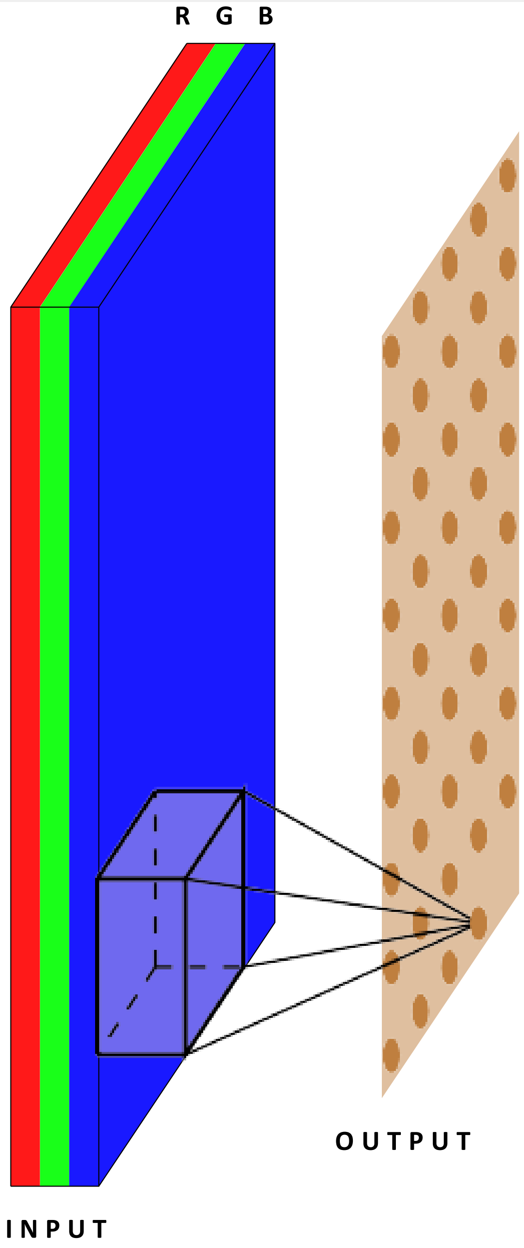
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



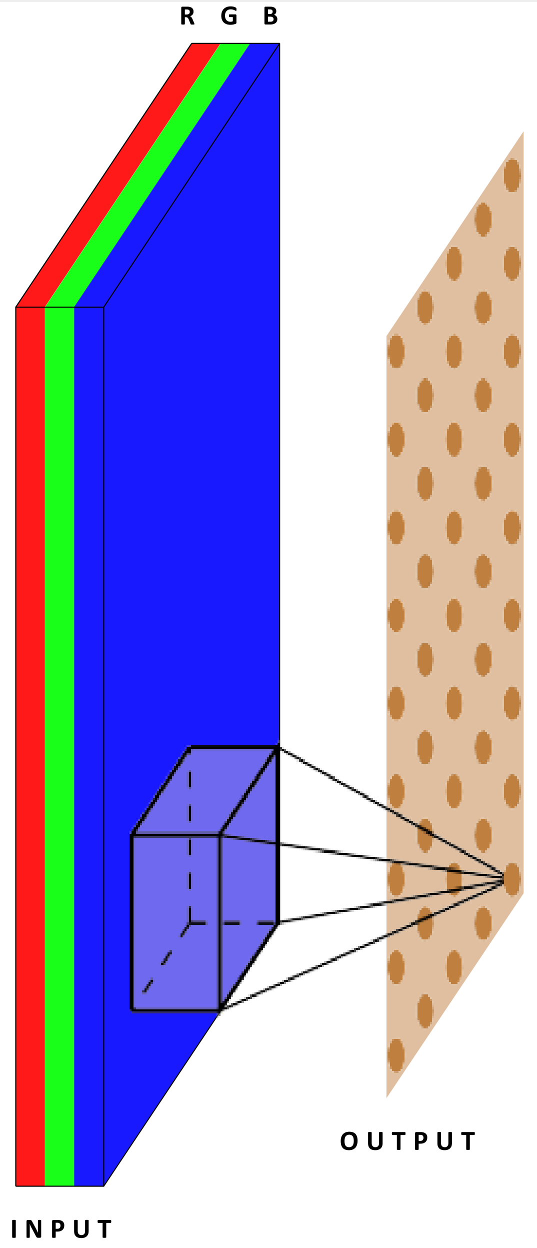
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



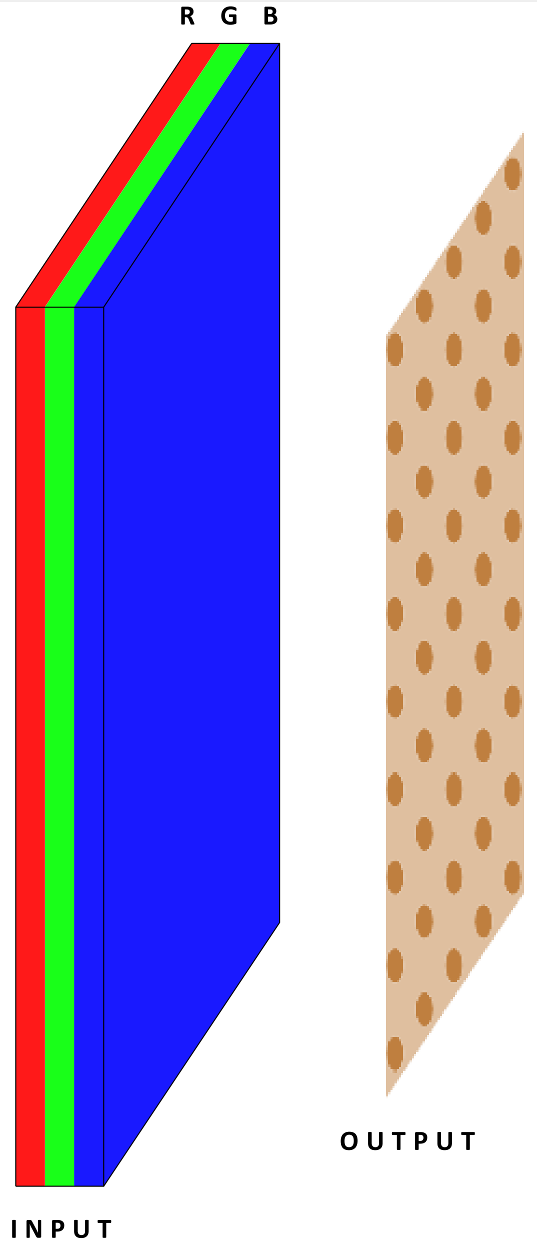
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



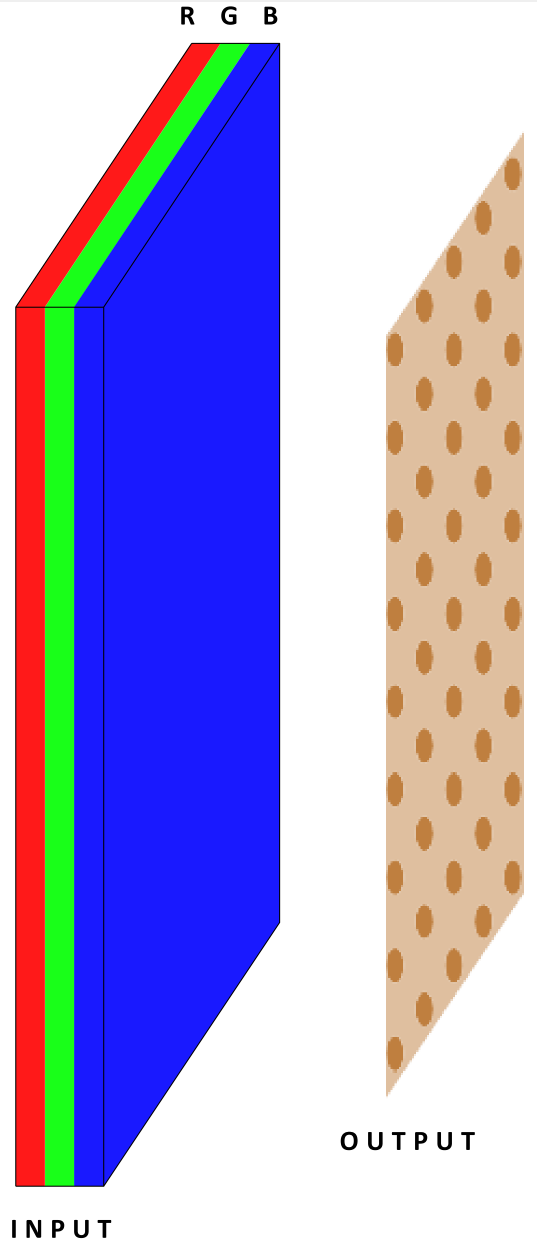
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation



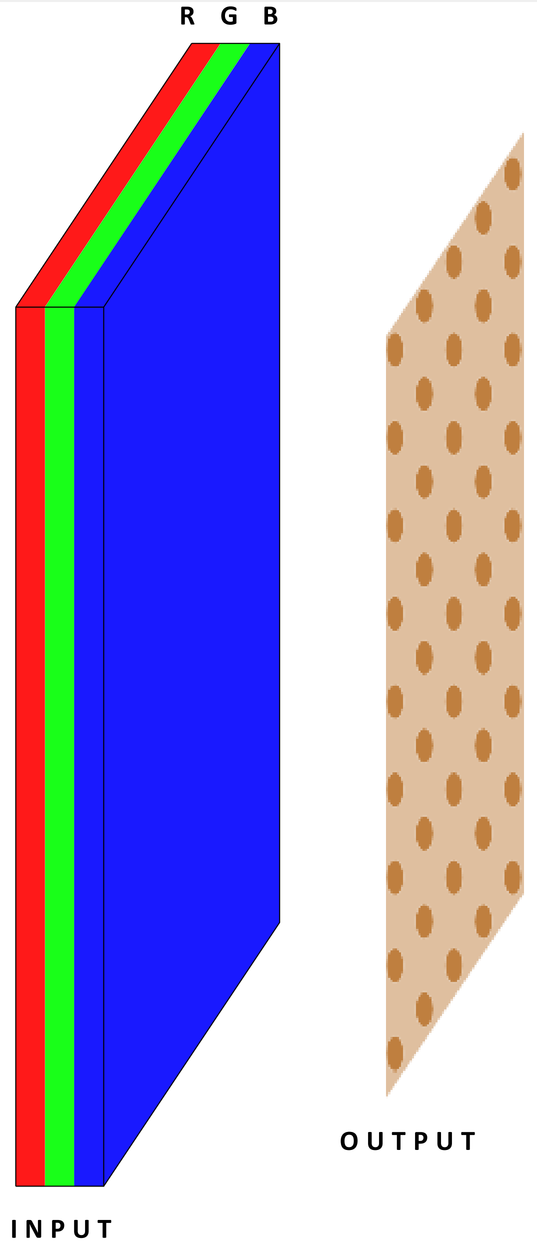
- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation
- Note that in this lecture we will assume that the filter always extends to the depth of the image



- 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)

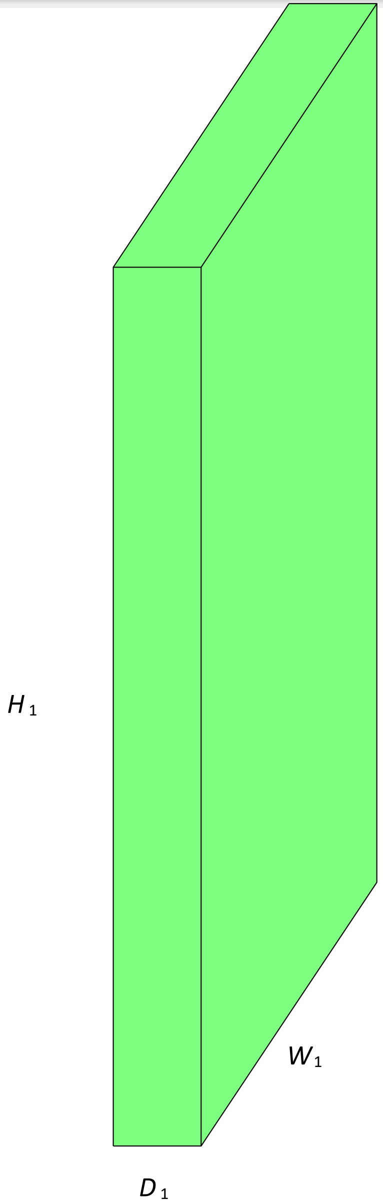


- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation
- Note that in this lecture we will assume that the filter always extends to the depth of the image
- In effect, we are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

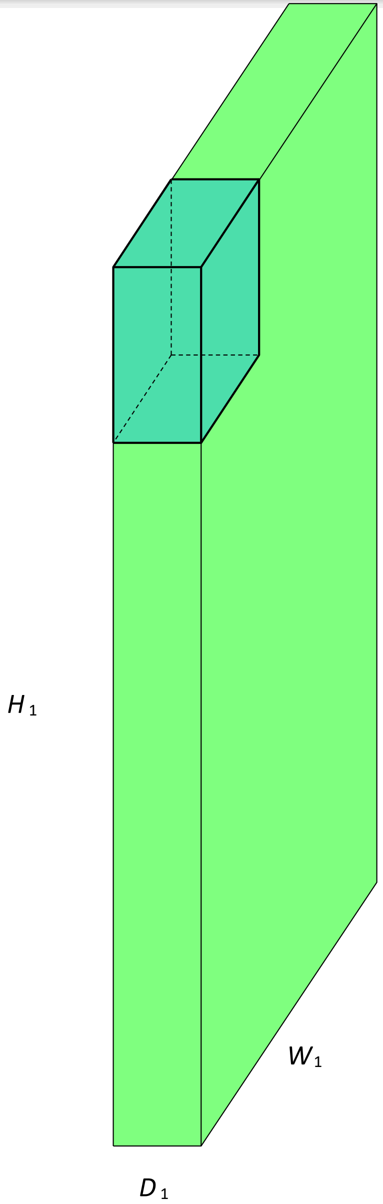


- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation
- Note that in this lecture we will assume that the filter always extends to the depth of the image
- In effect, we are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

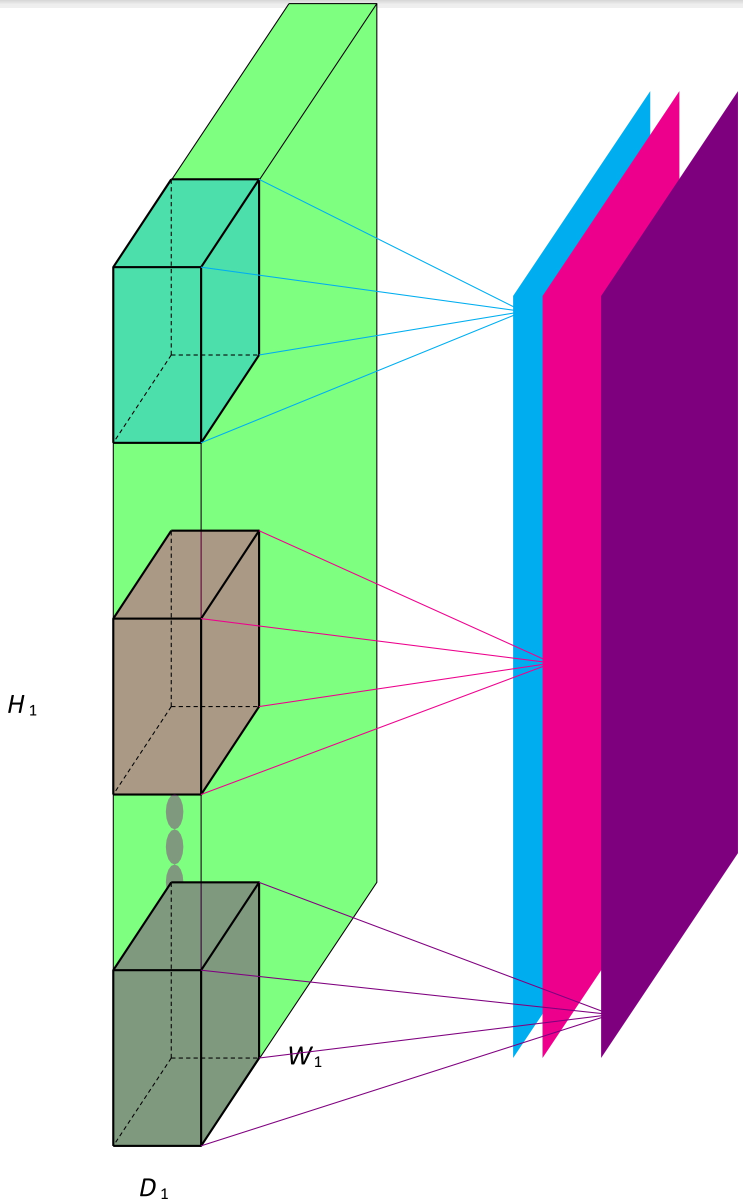
Relation between input size, output size and filter size



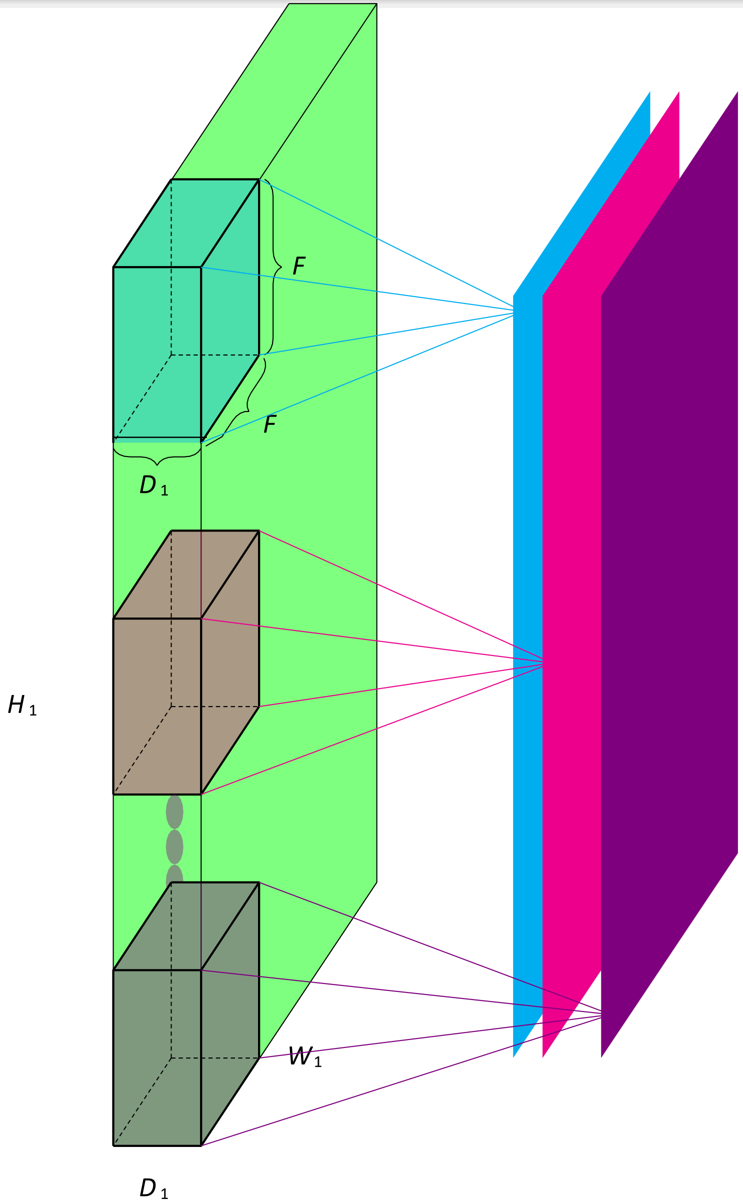
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)



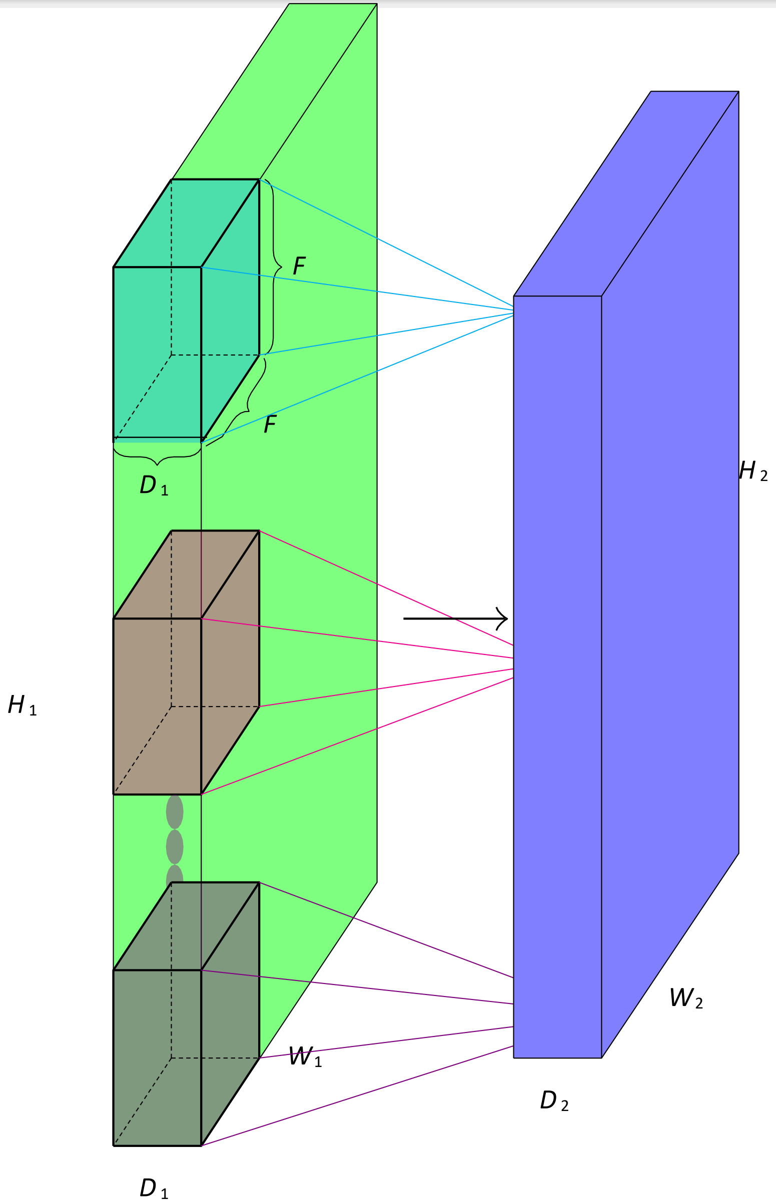
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)



- 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

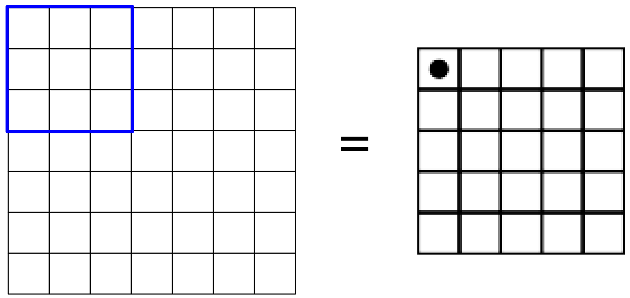


- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)
- The number of filters K
- The spatial extent (F) of each filter (the depth of each filter is same as the depth of each input)

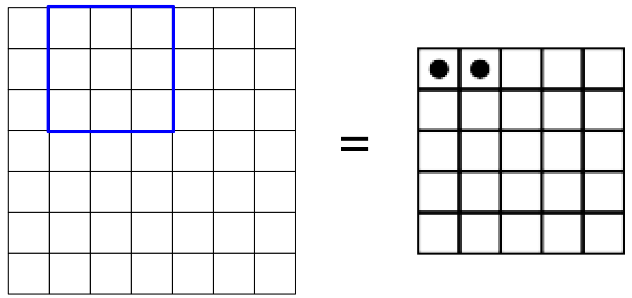


- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)
- The number of filters K
- The spatial extent (F) of each filter (the depth of each filter is same as the depth of each input)
- The output is $W_2 \times H_2 \times D_2$ (we will soon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

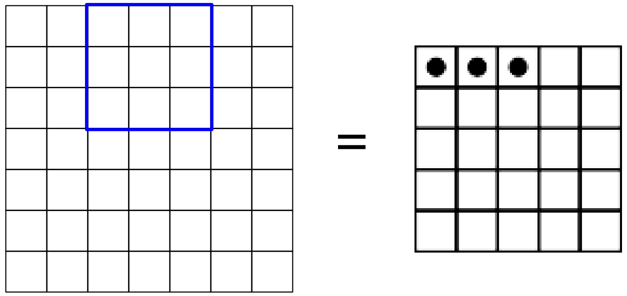
- Let us compute the dimension (W_2, H_2) of the output



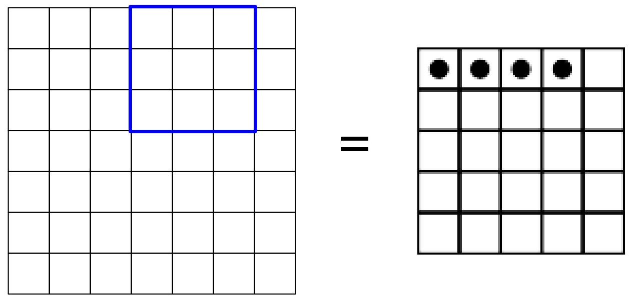
- Let us compute the dimension (W_2, H_2) of the output



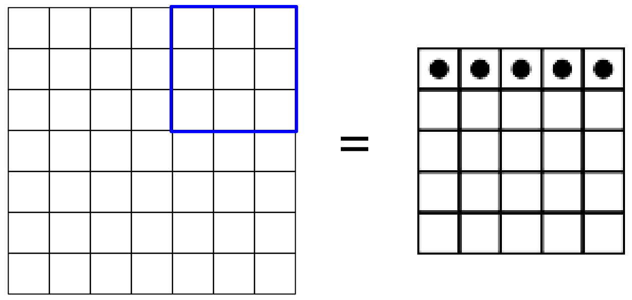
- Let us compute the dimension (W_2, H_2) of the output



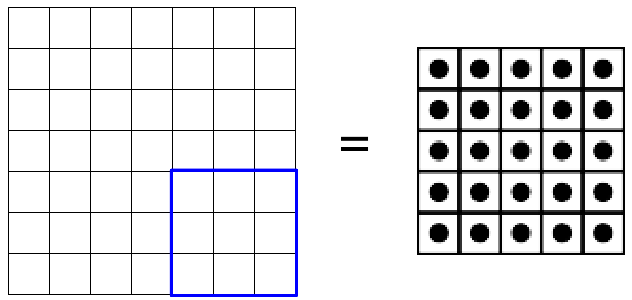
- Let us compute the dimension (W_2, H_2) of the output



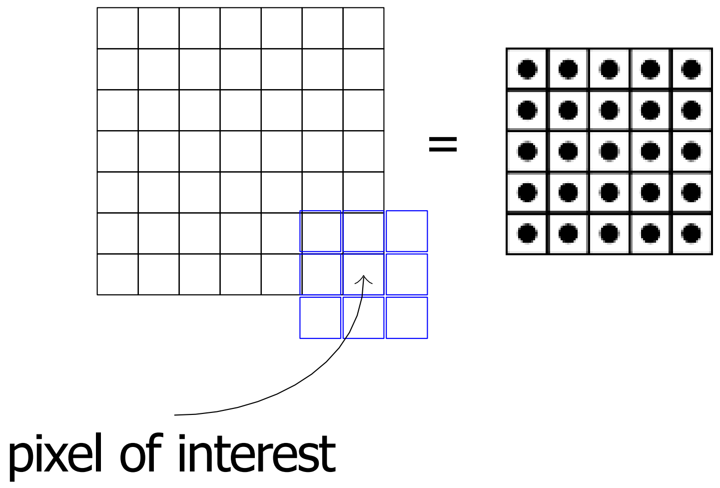
- Let us compute the dimension (W_2, H_2) of the output



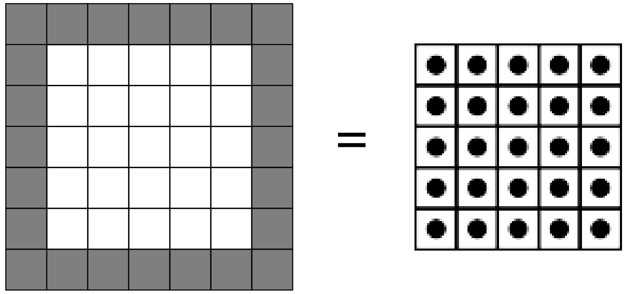
- Let us compute the dimension (W_2, H_2) of the output



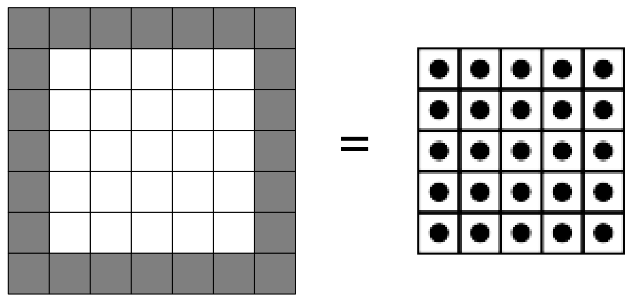
- Let us compute the dimension (W_2, H_2) of the output



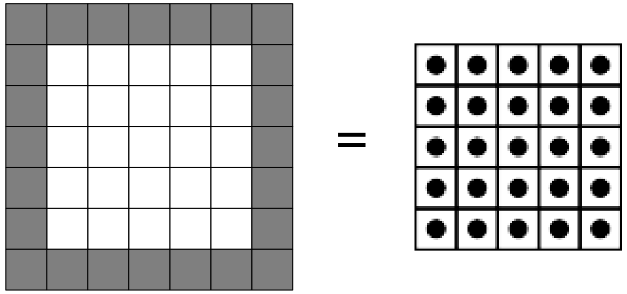
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary



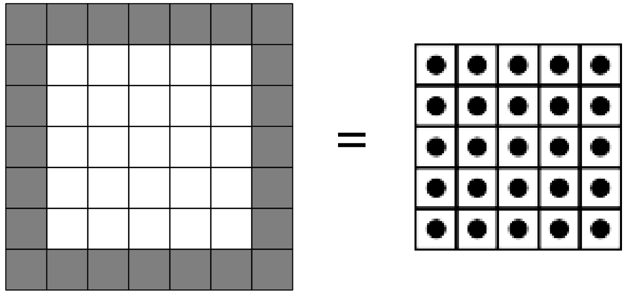
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary



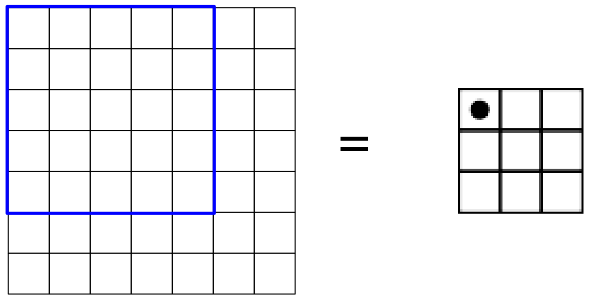
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)



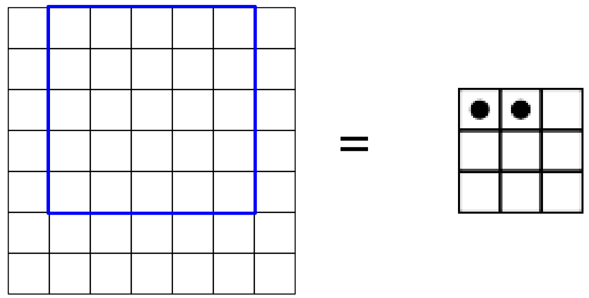
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input



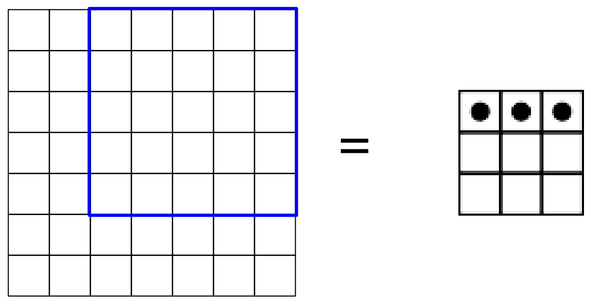
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input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



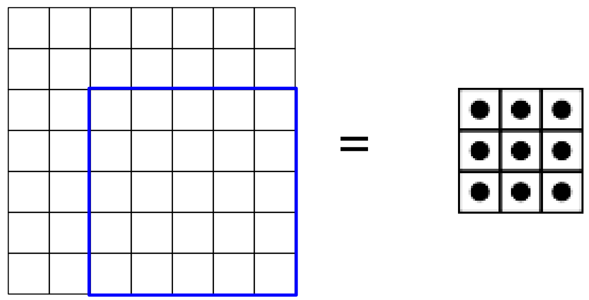
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input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



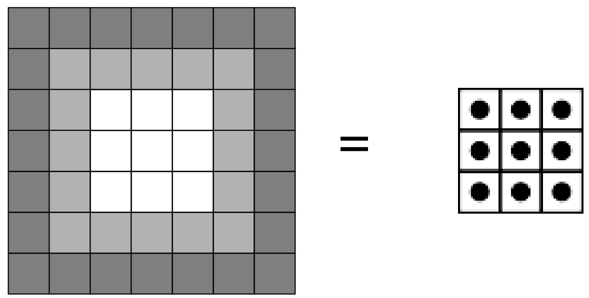
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input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



$$In\ general,\ W_2 = W_1 - F + 1$$

$$H_2 = H_1 - F + 1$$

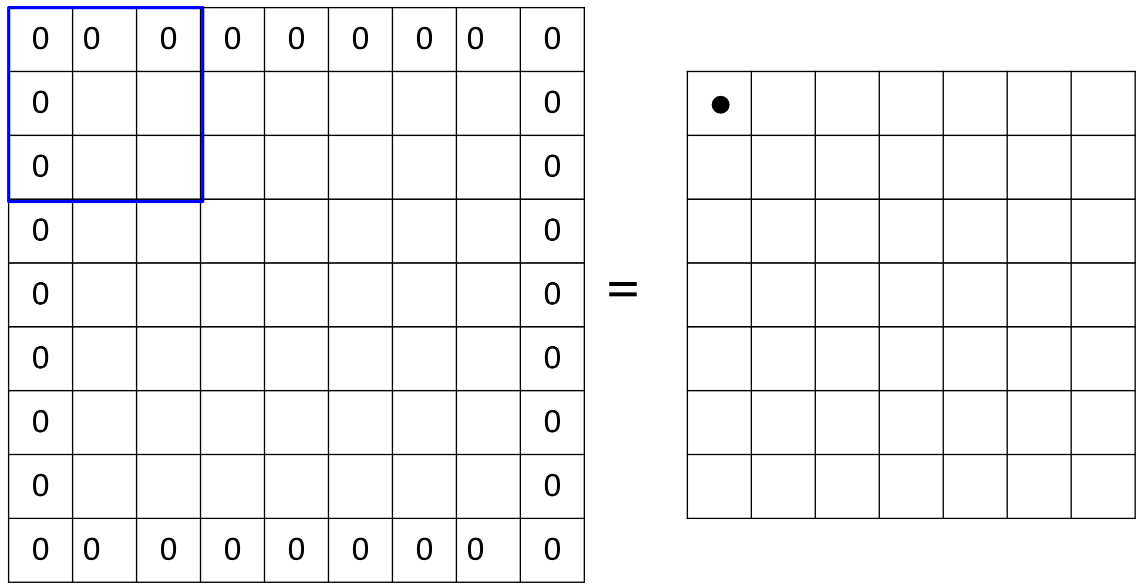
We will refine this formula further

- 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

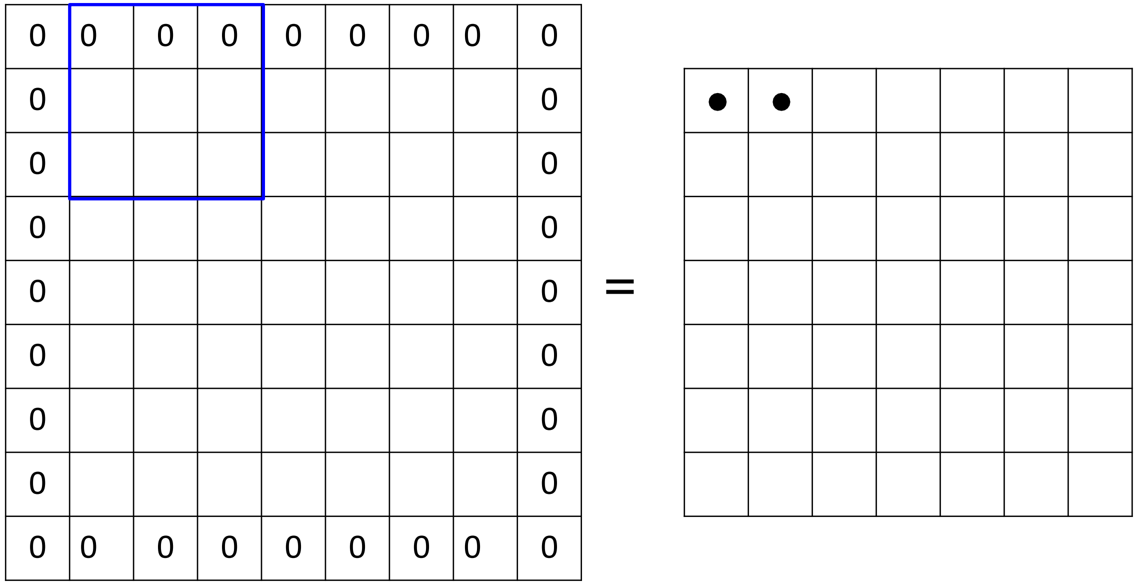
- What if we want the output to be of same size as the input?

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

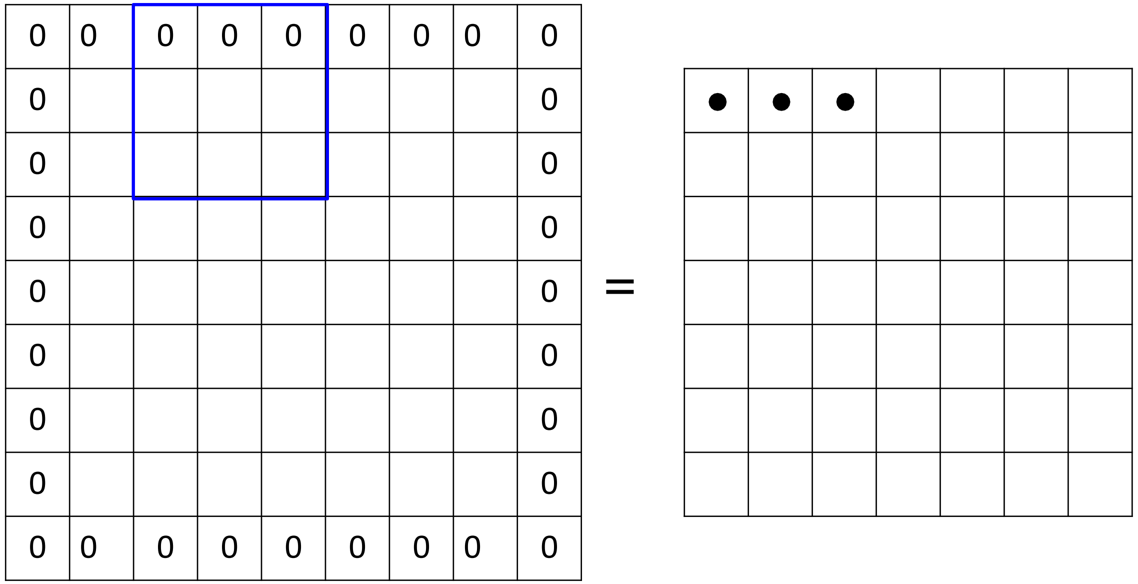
- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners



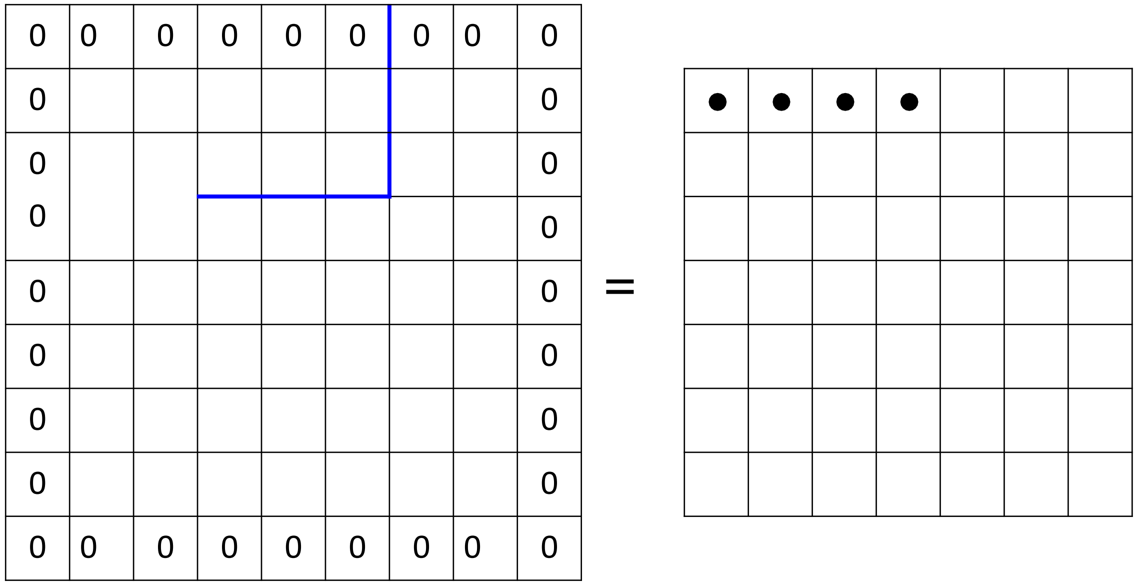
- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad $P = 1$ with a 3×3 kernel



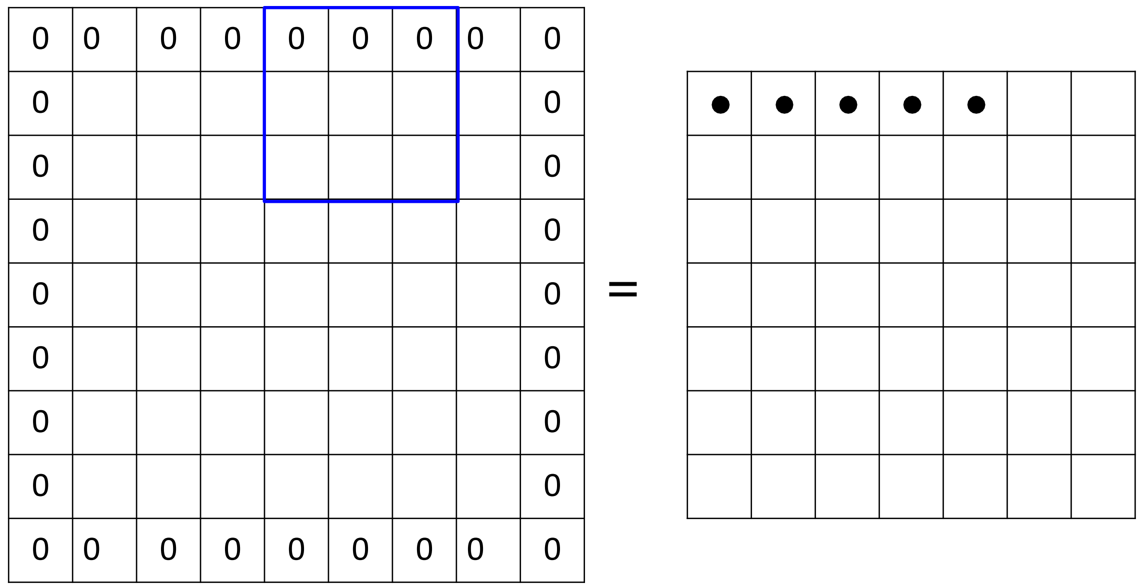
- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of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



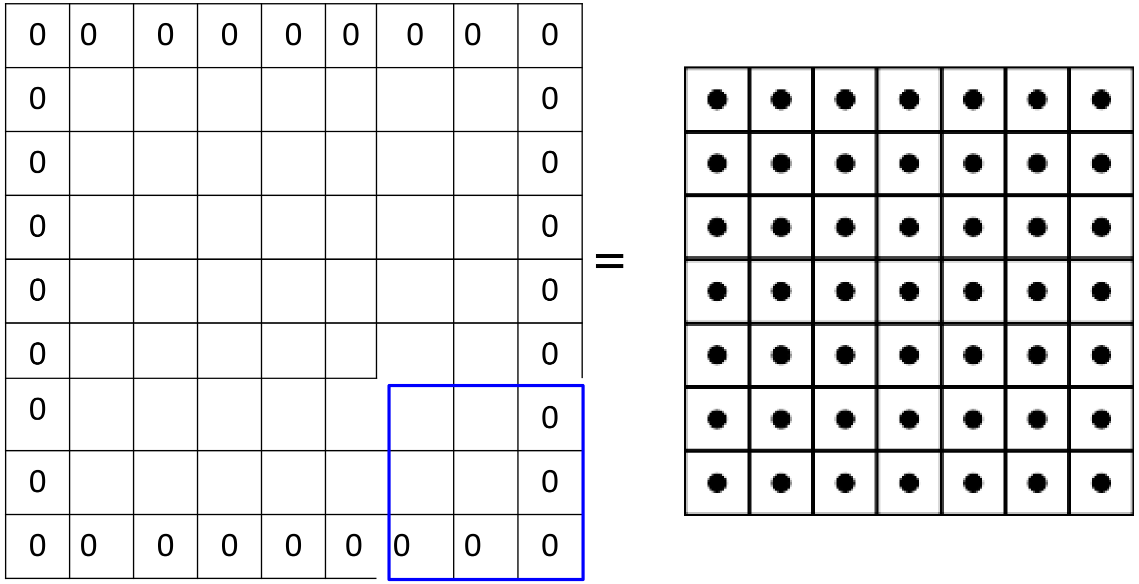
- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of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



- 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

We now have,

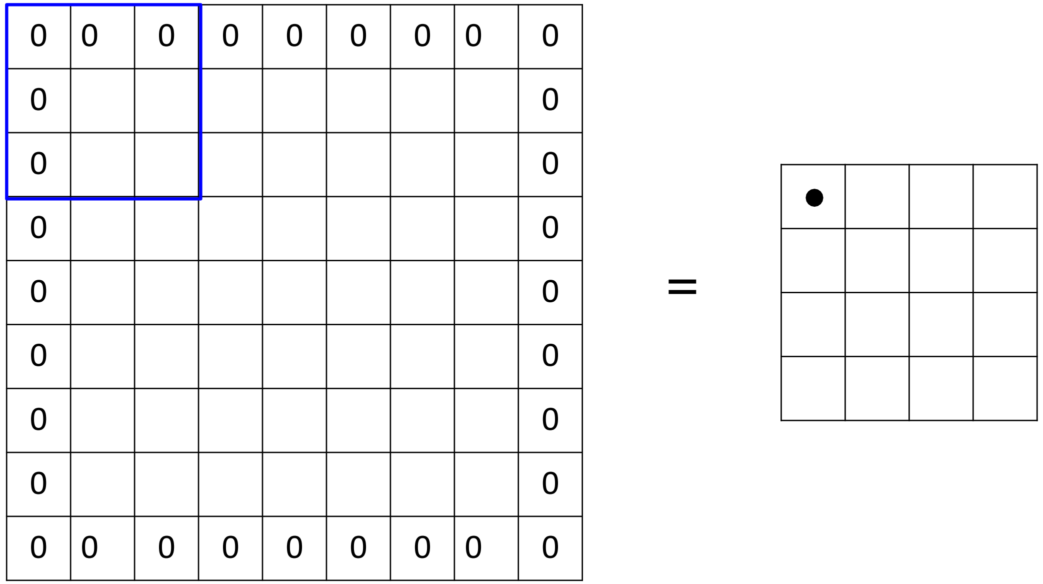
$$W_2 = W_1 - F + 2P + 1$$

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

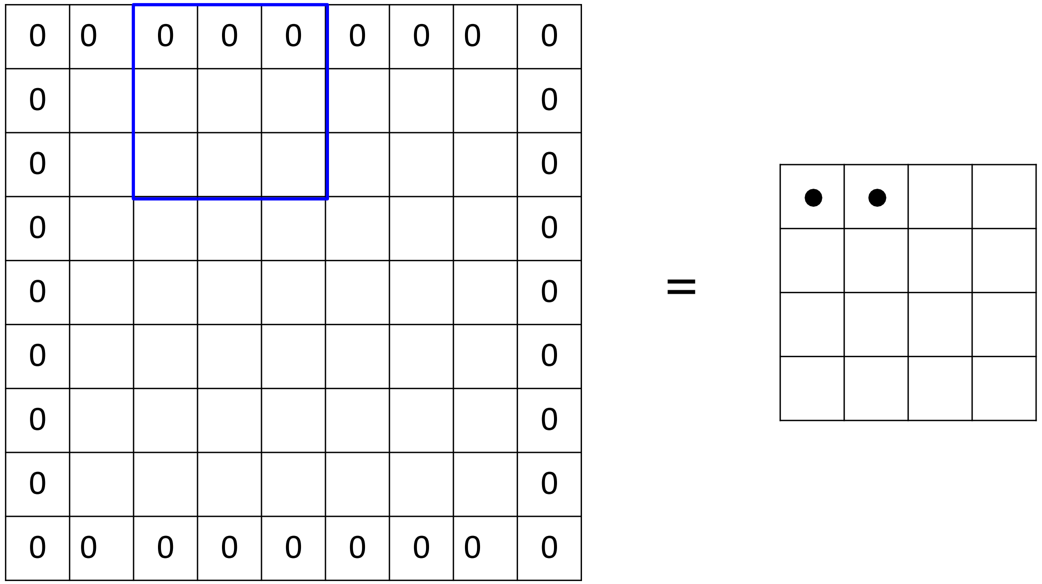
We will refine this formula further

- What does the stride S do?

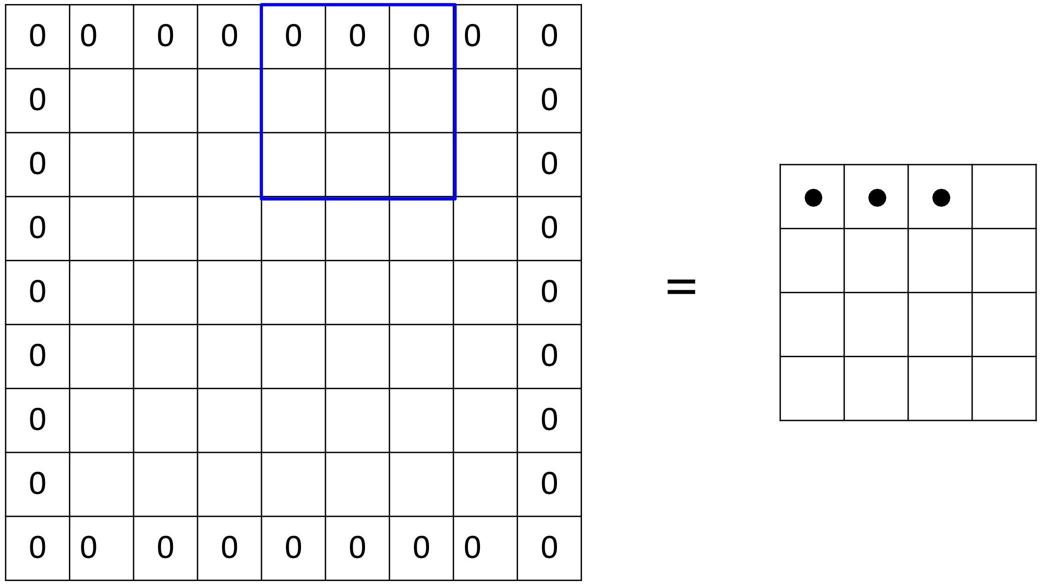
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)



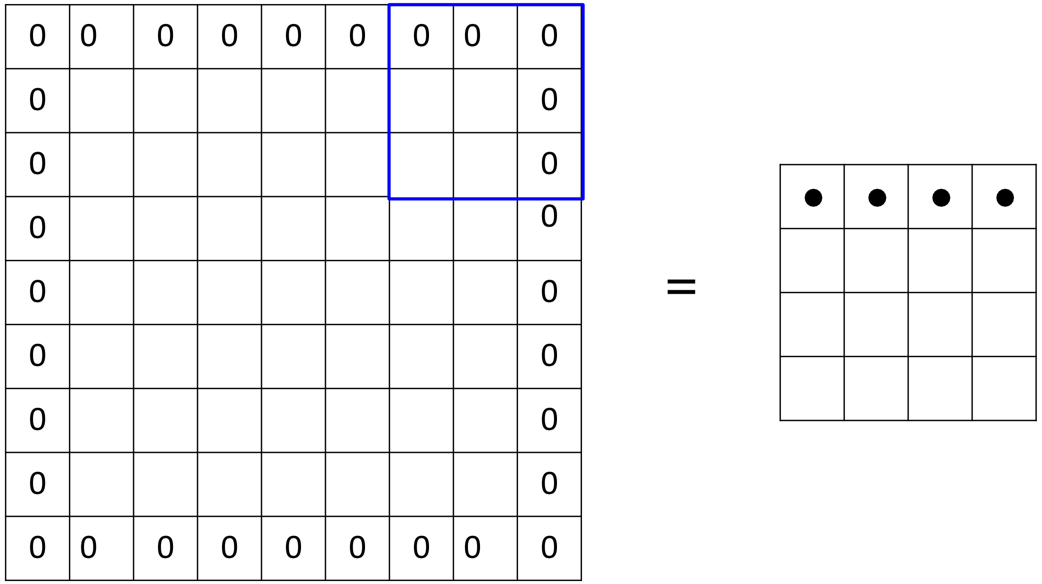
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



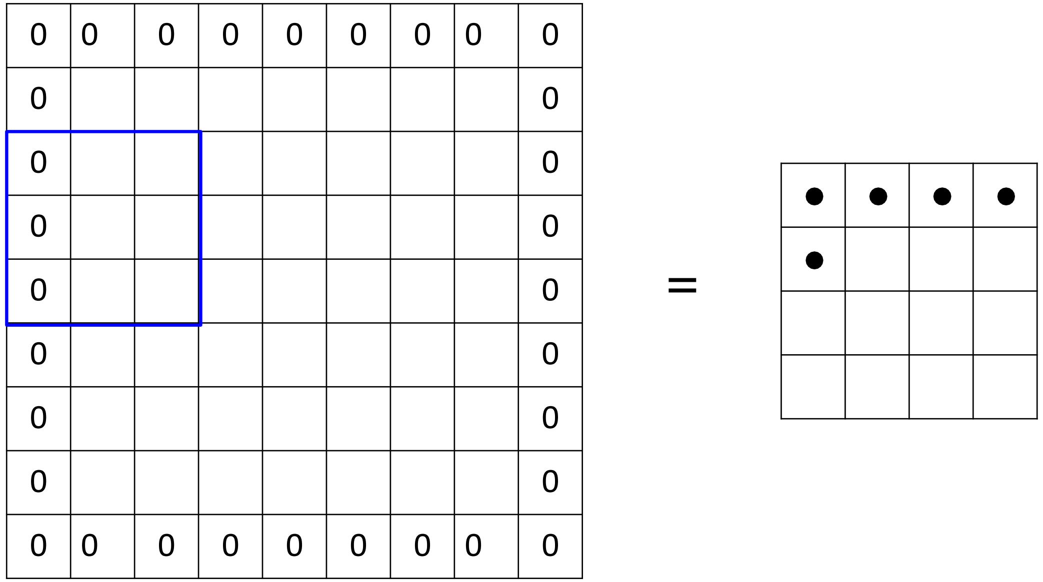
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



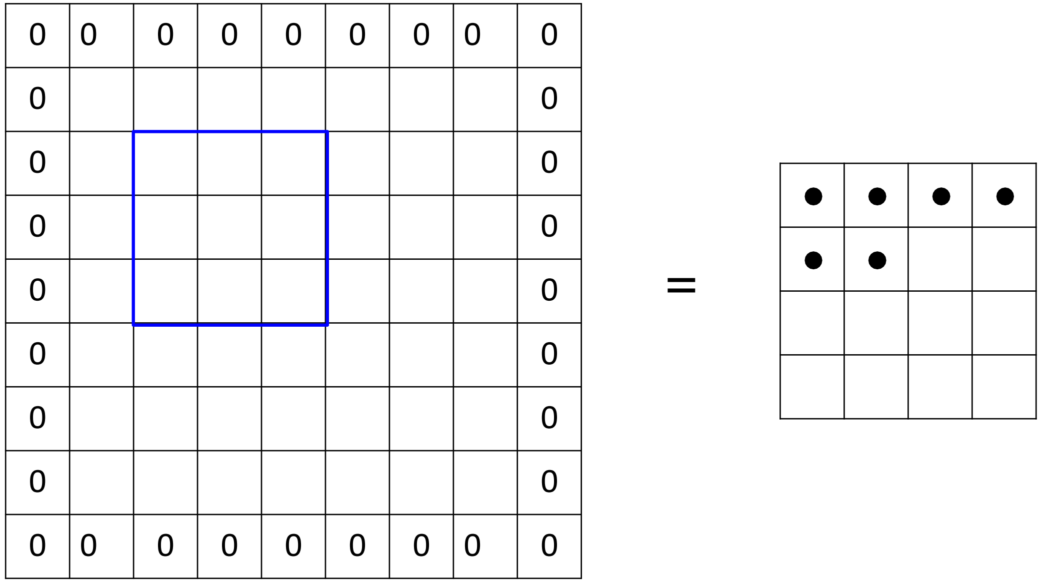
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



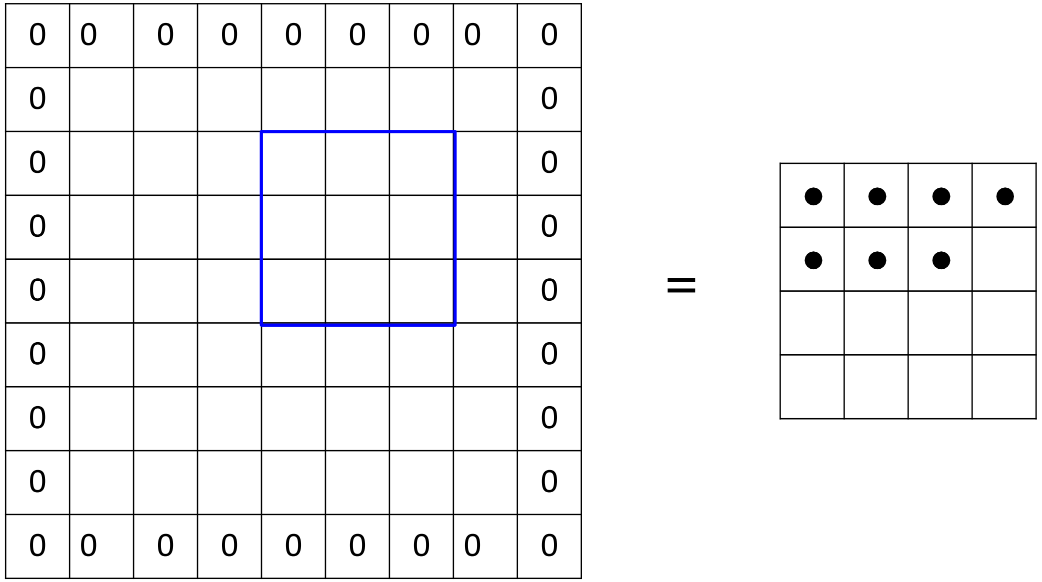
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



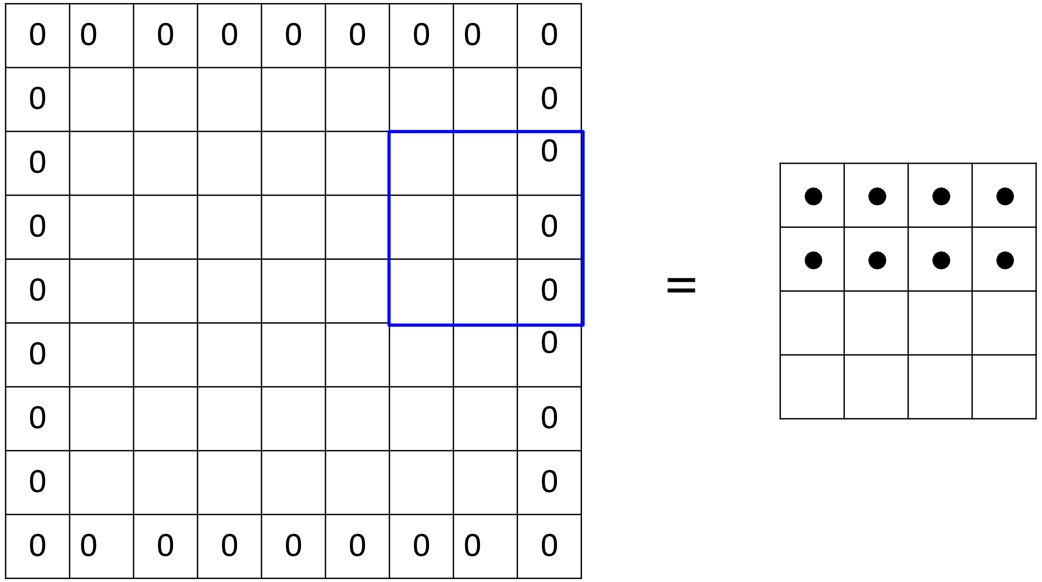
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



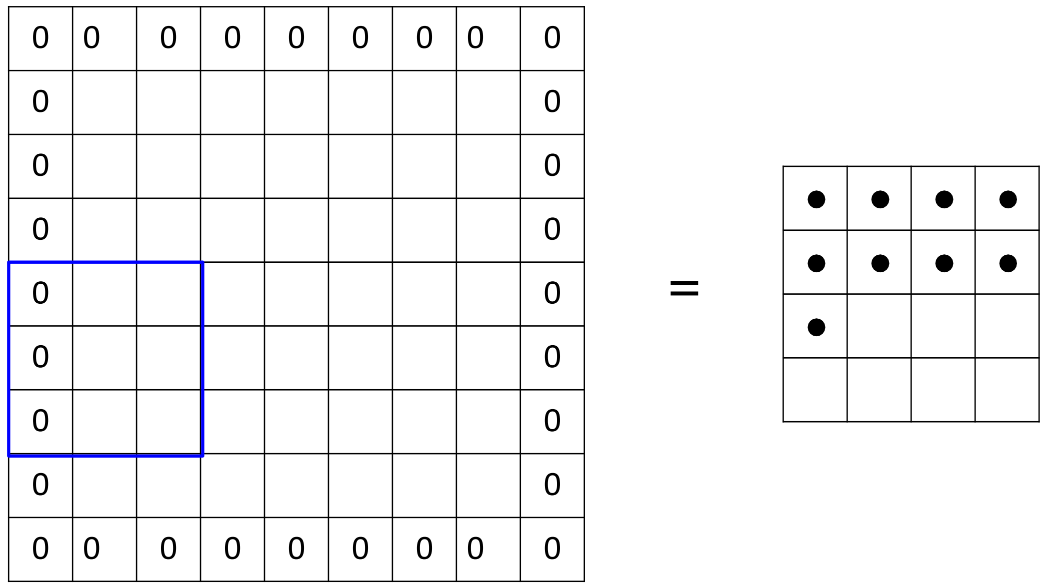
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



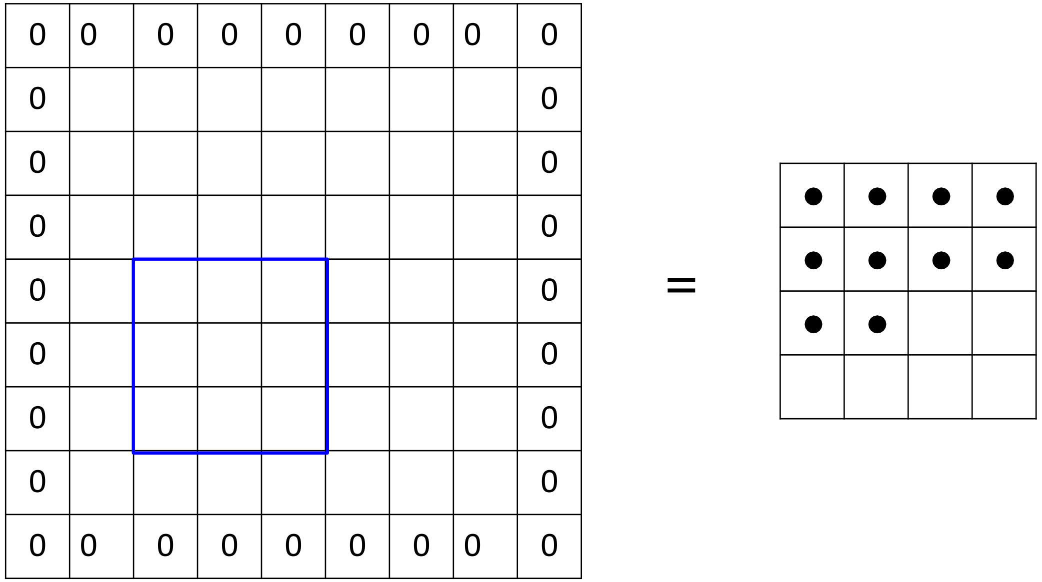
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



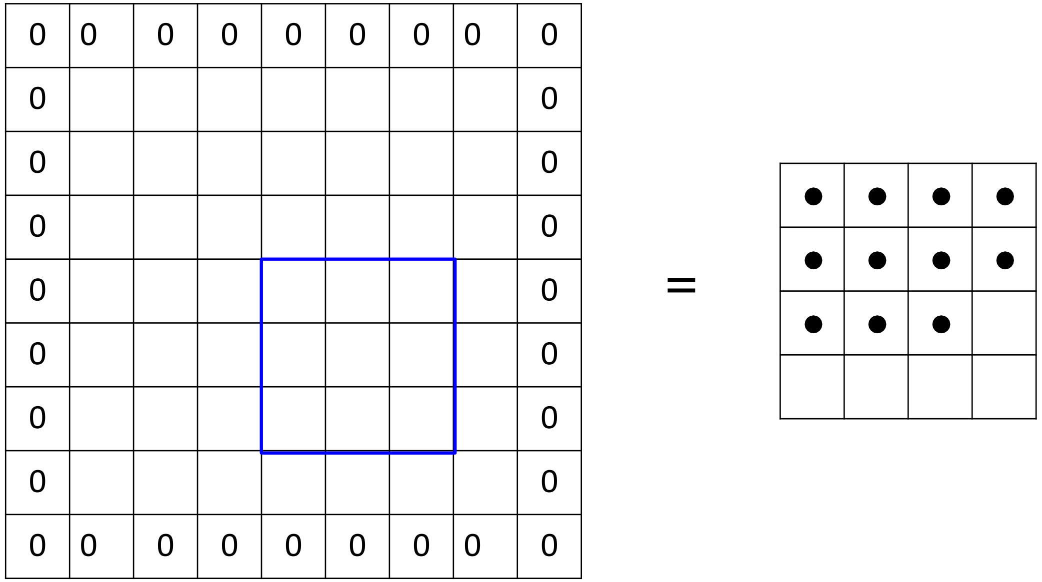
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



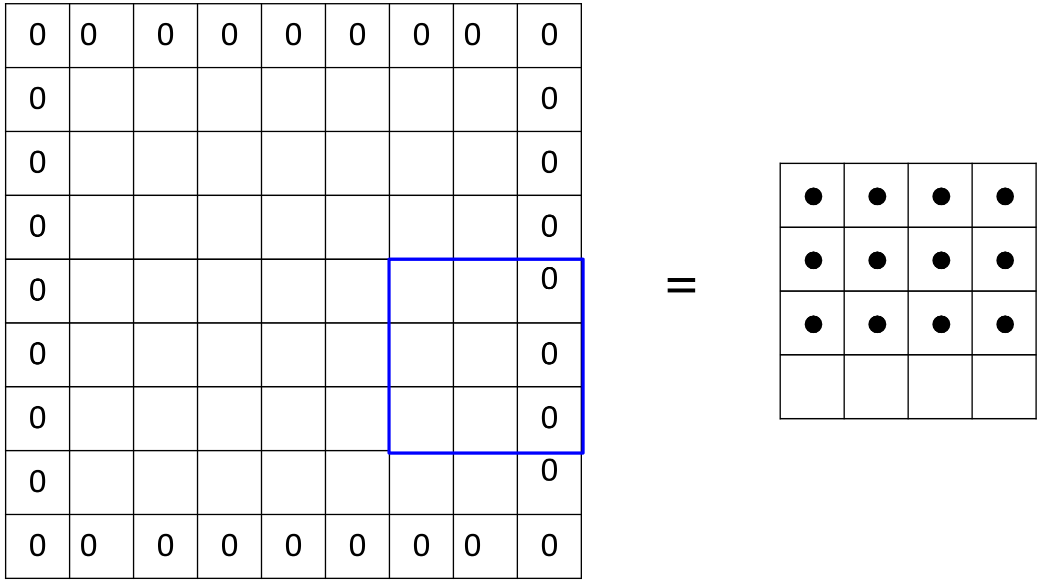
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



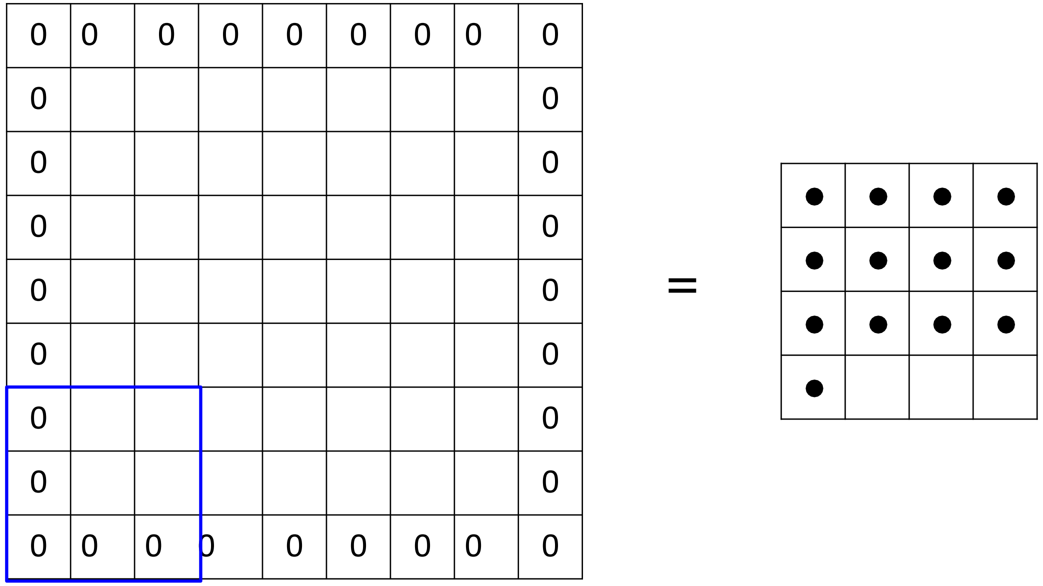
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



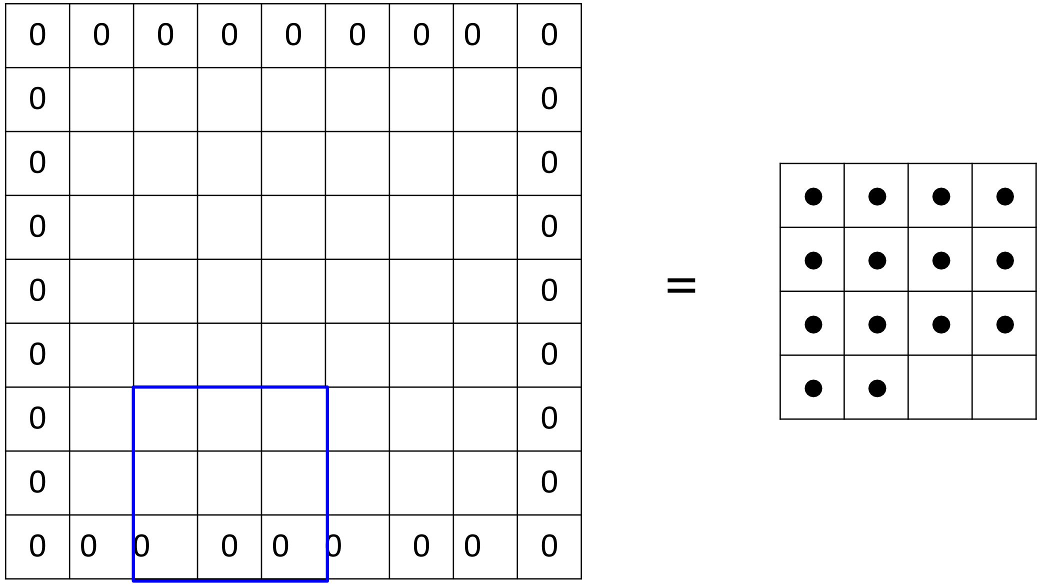
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



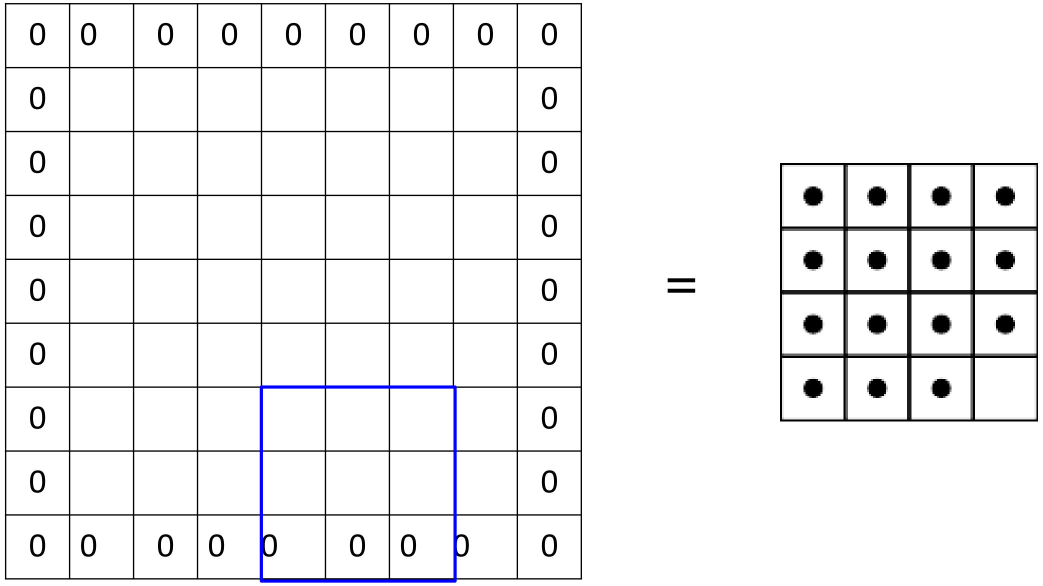
- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



- 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

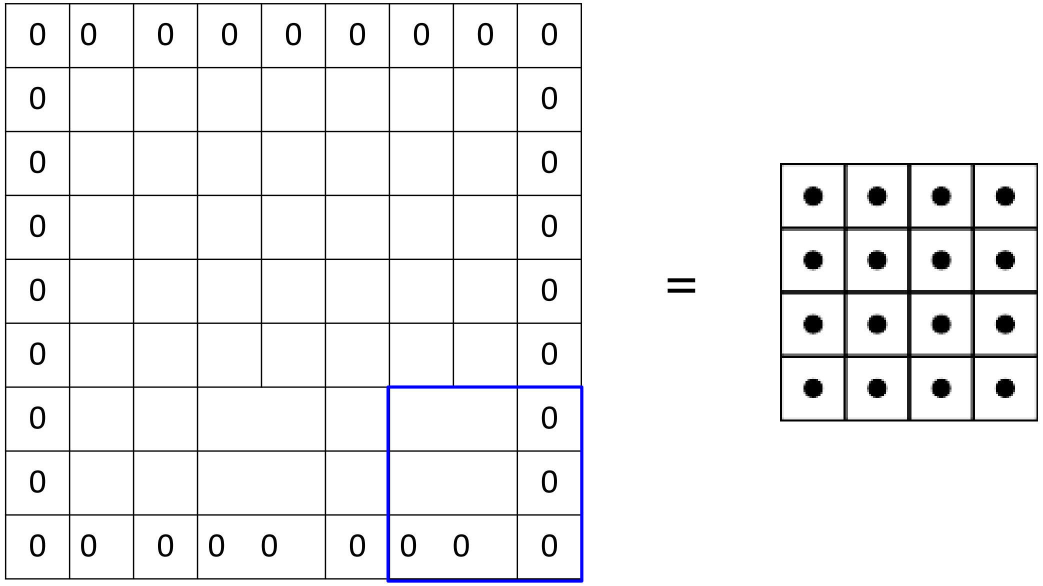


- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions



- 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

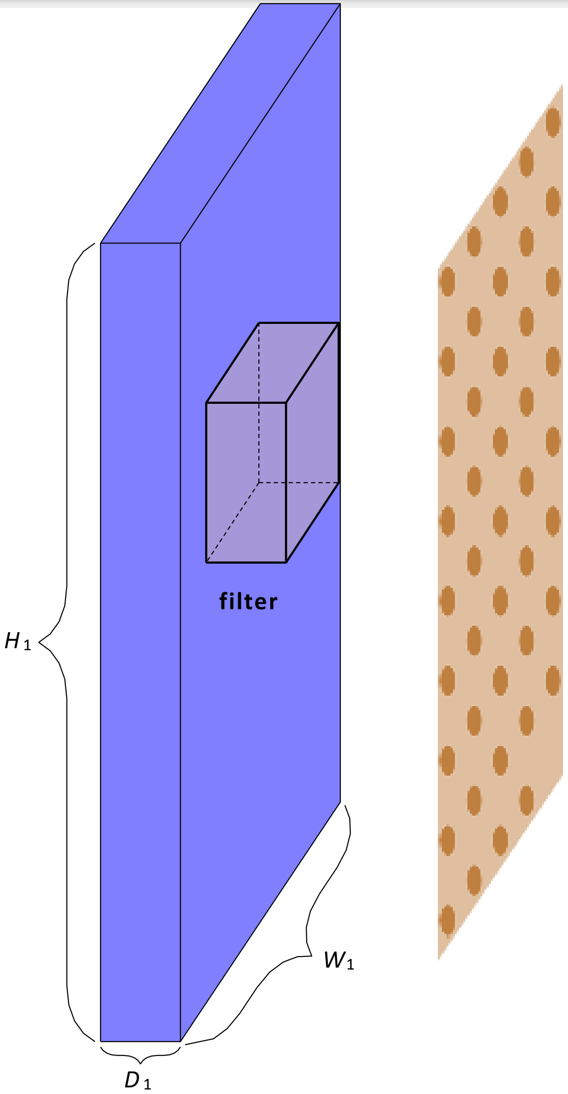
So what should our final formula look like,

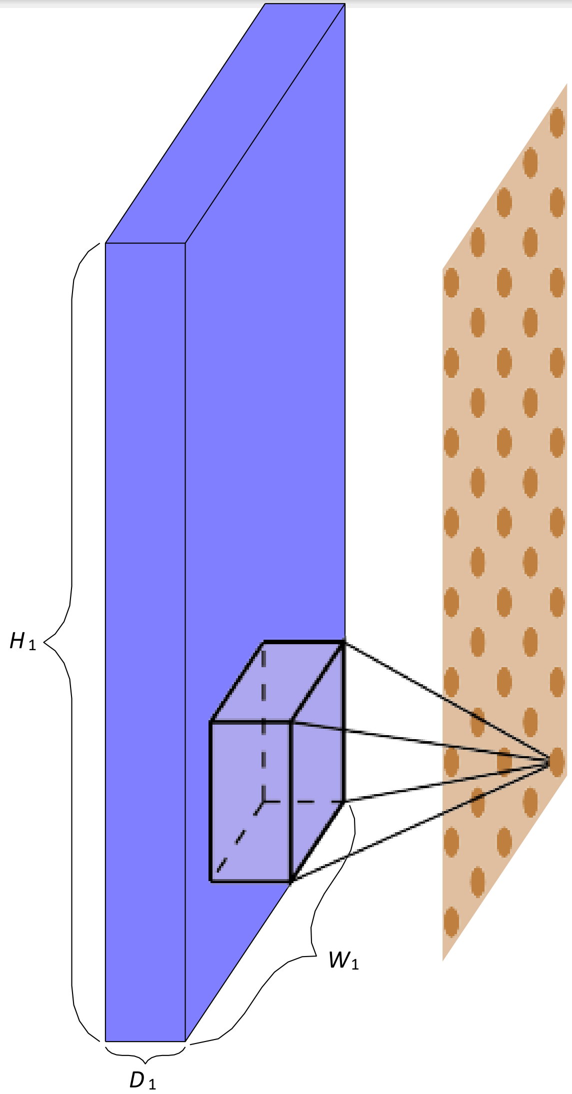


- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

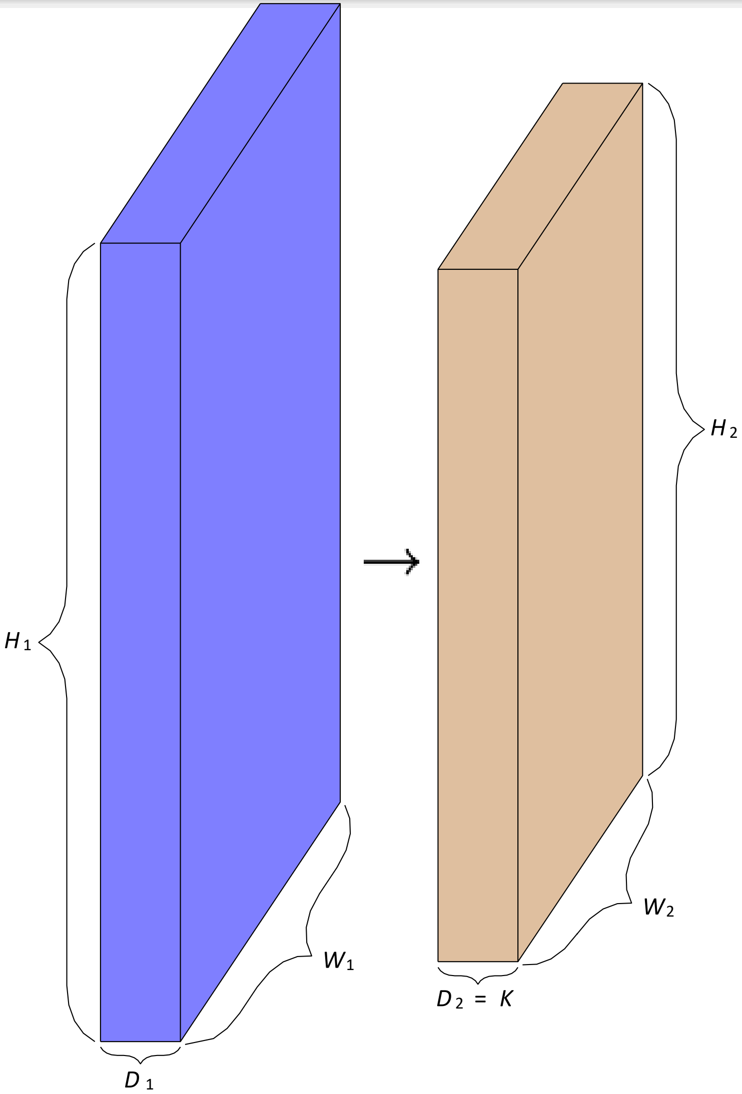
So what should our final formula look like,

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


filter
H_1
W_1
D_1



- Each filter gives us one 2D output.

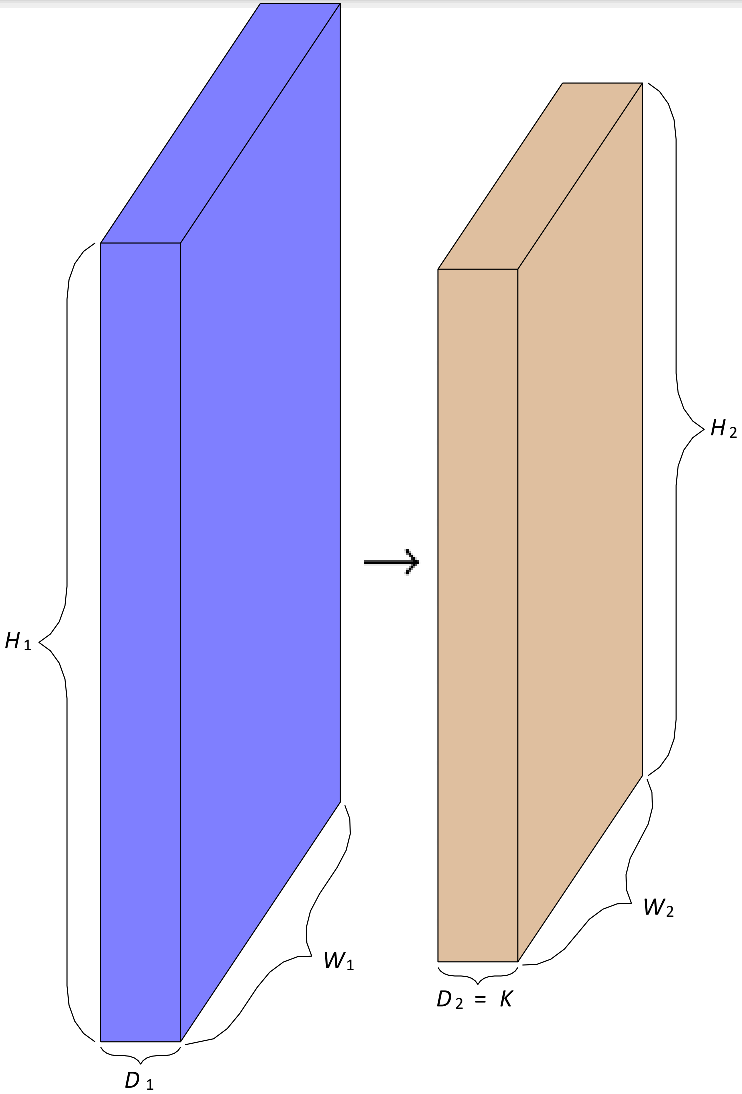


$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

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

$$D_2 = K$$

- Each filter gives us one 2D output.
- K filters will give us K such 2D outputs

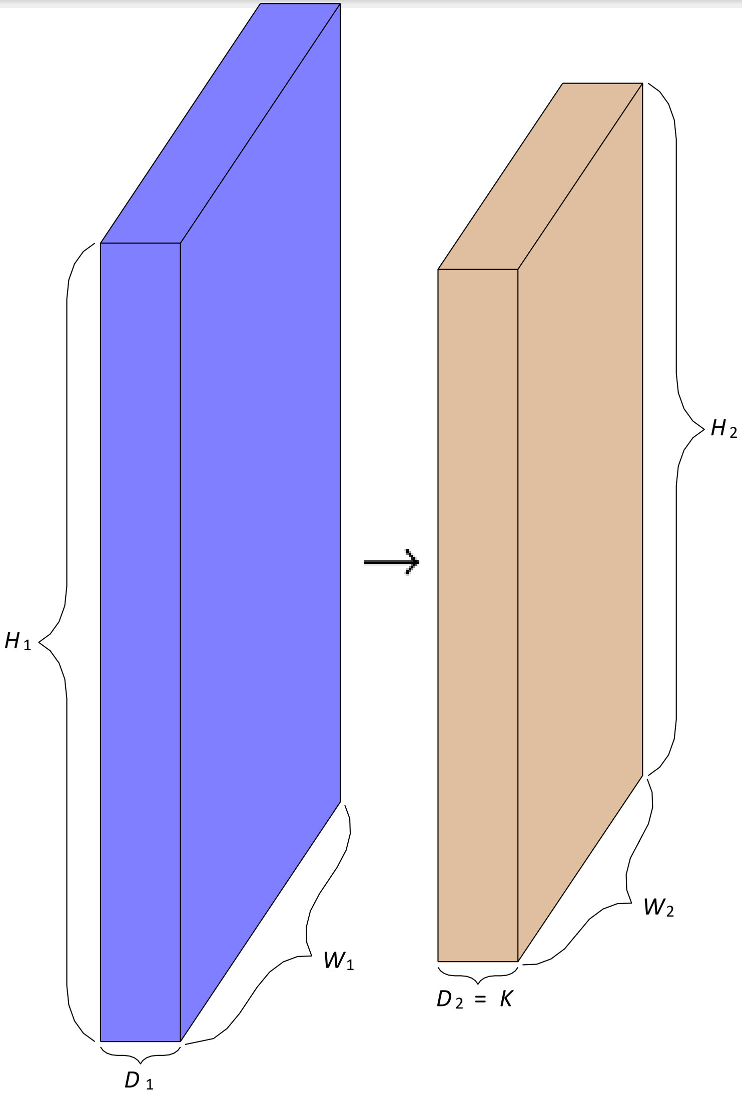


$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

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

$$D_2 = K$$

- 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



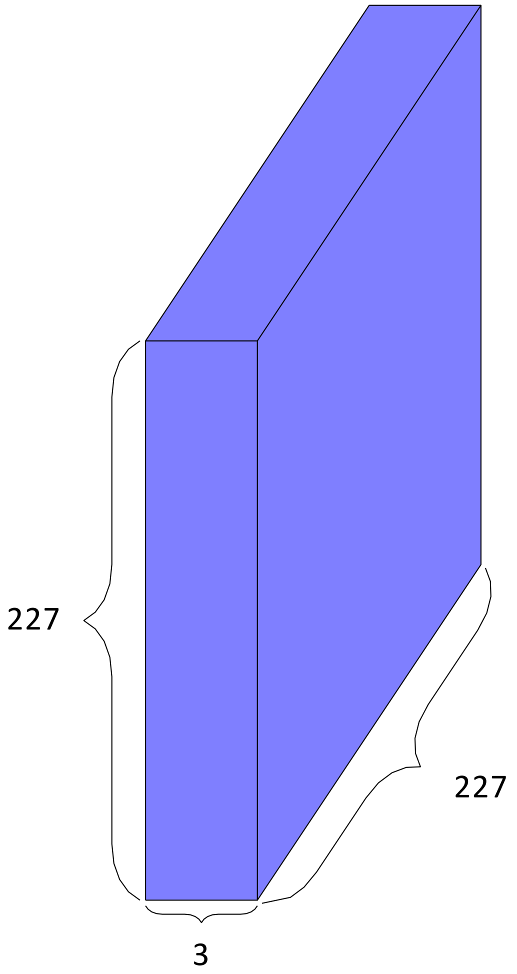
$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

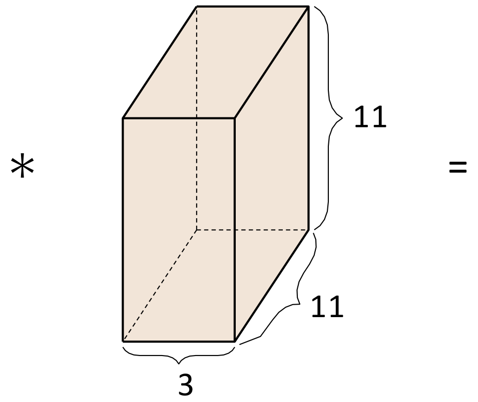
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

$$D_2 = K$$

- 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





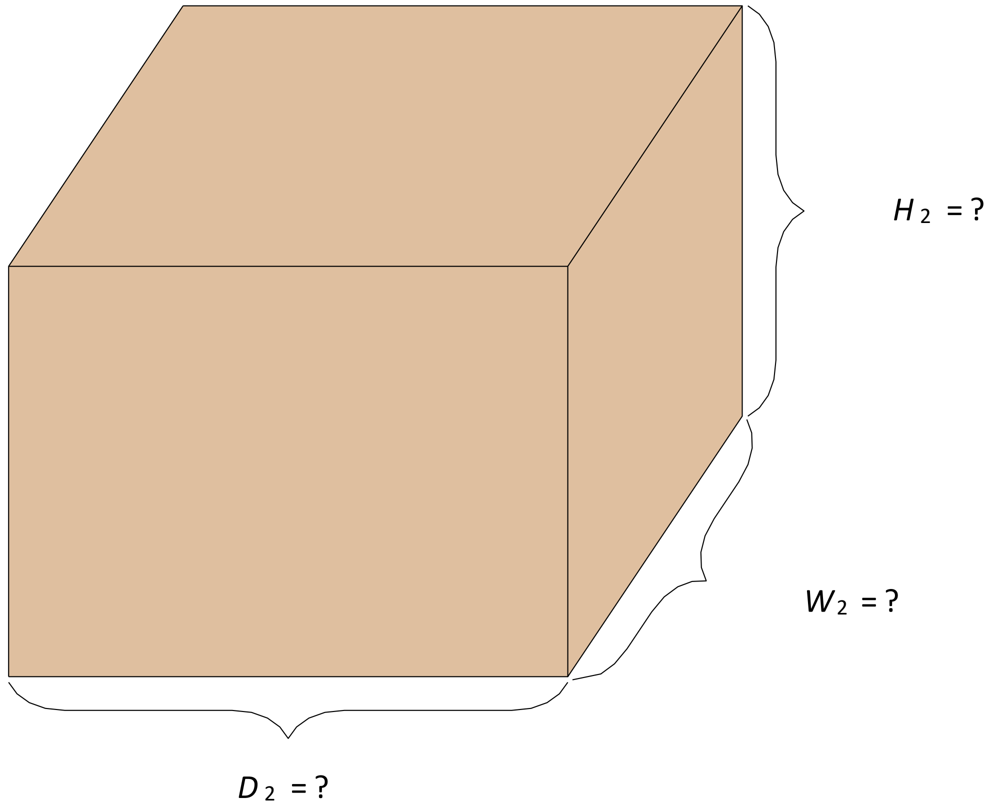
96 filters

$Stride = 4$

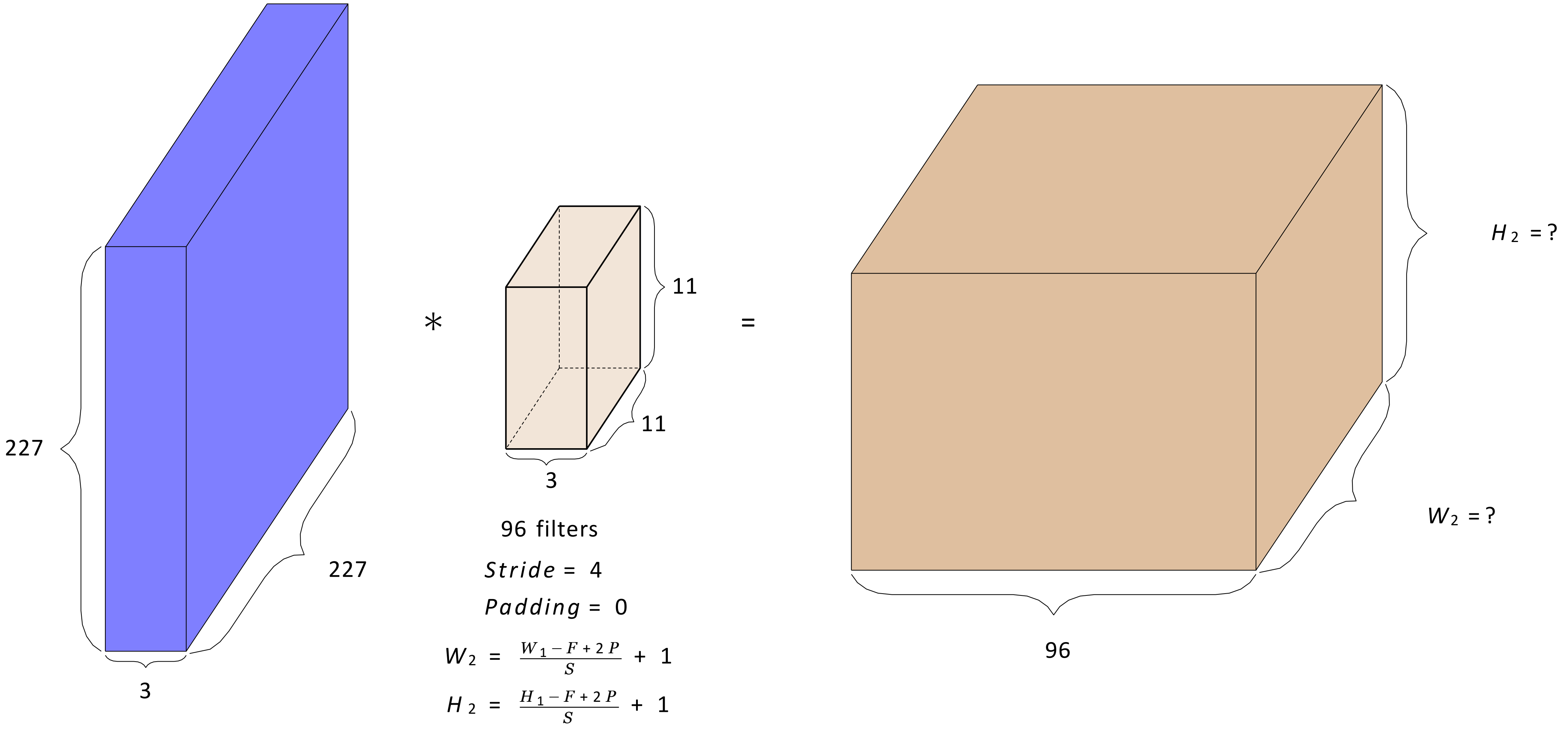
$Padding = 0$

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

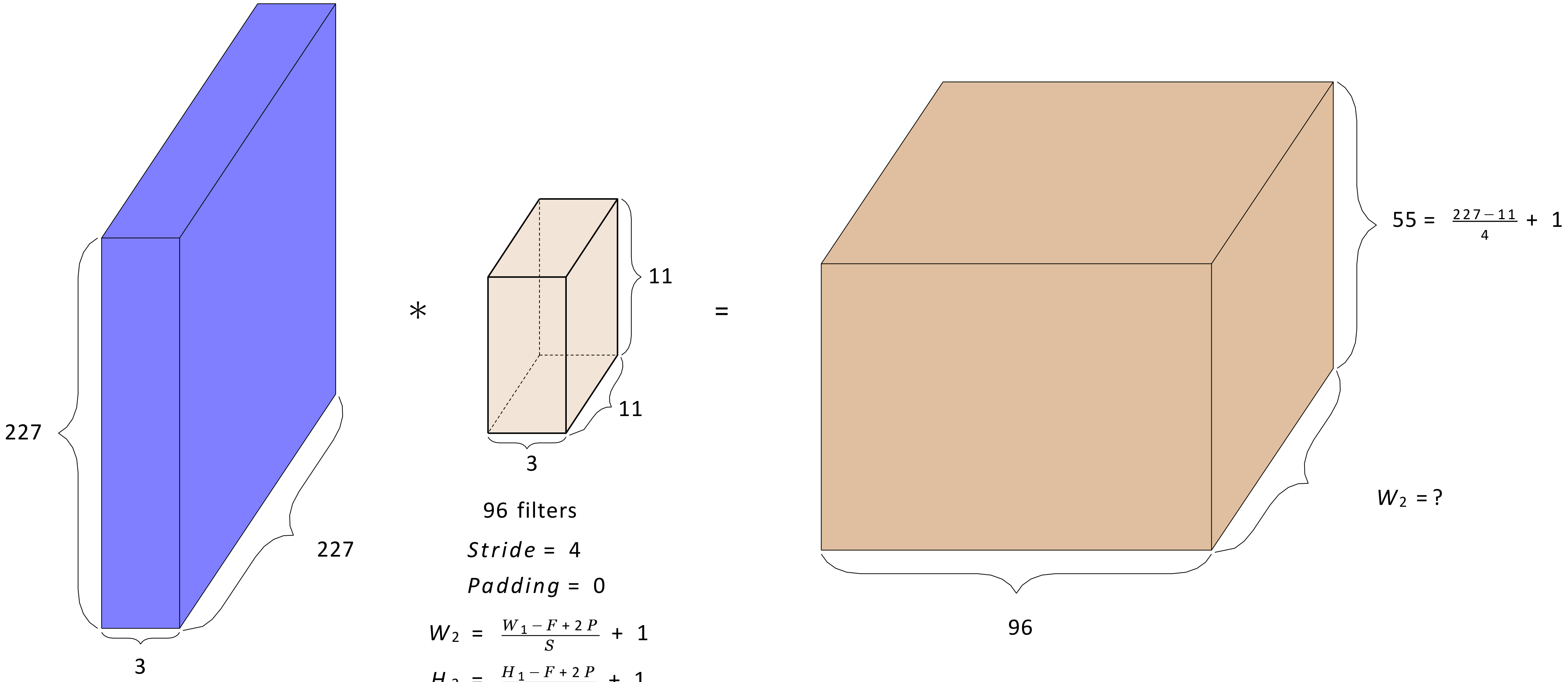
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$



Let us do a few exercises



Let us do a few exercises



96 filters

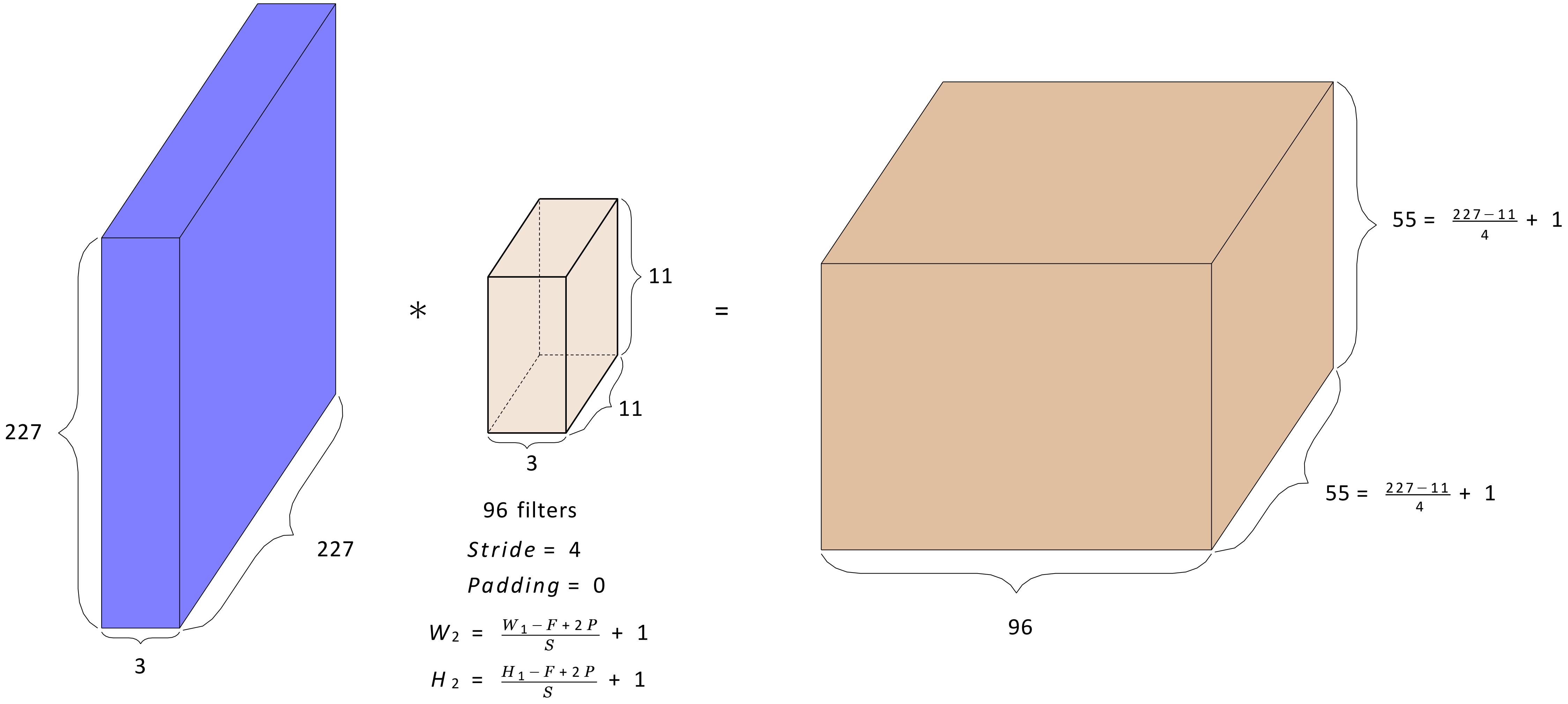
Stride = 4

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

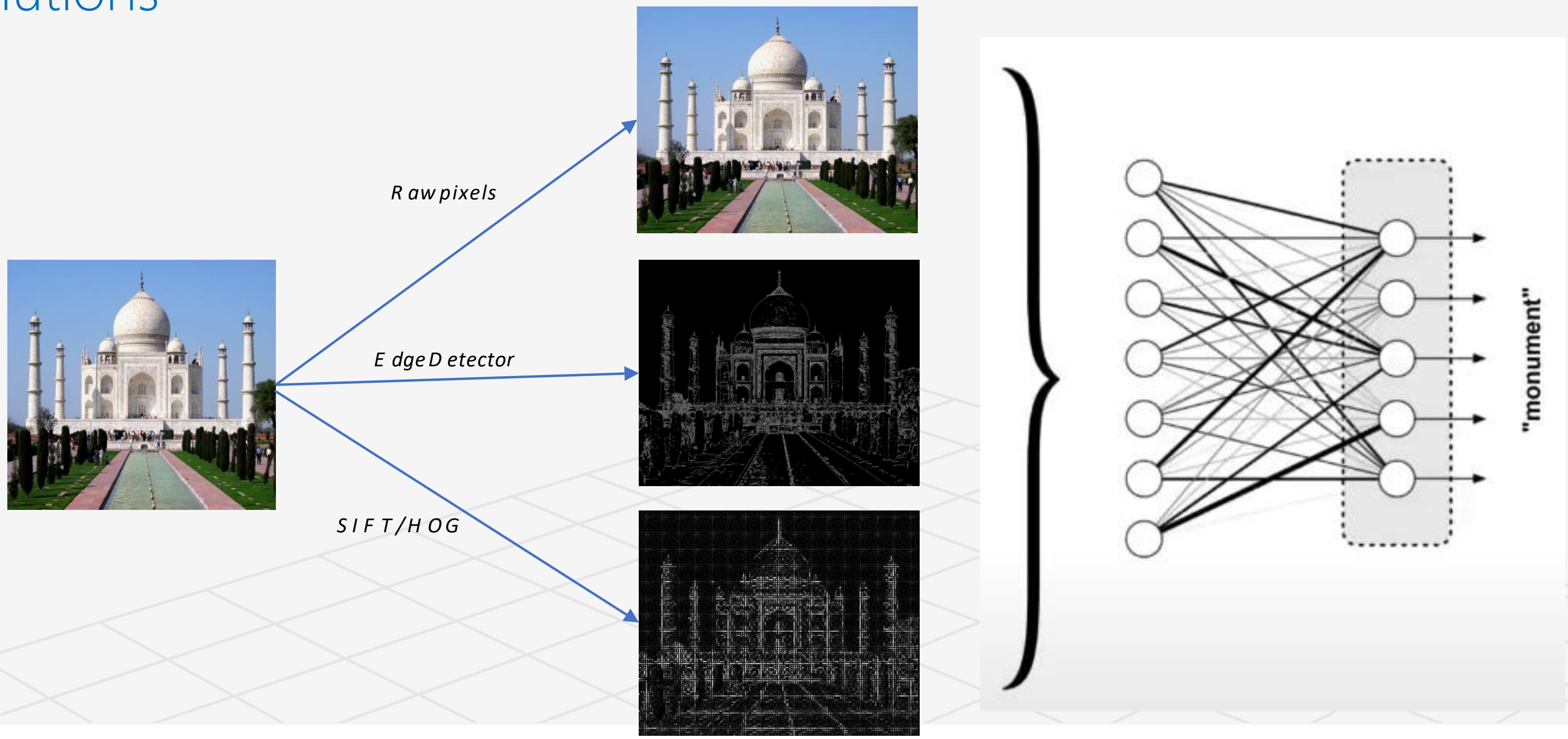


Putting things into perspective

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

Traditional ML based Vision

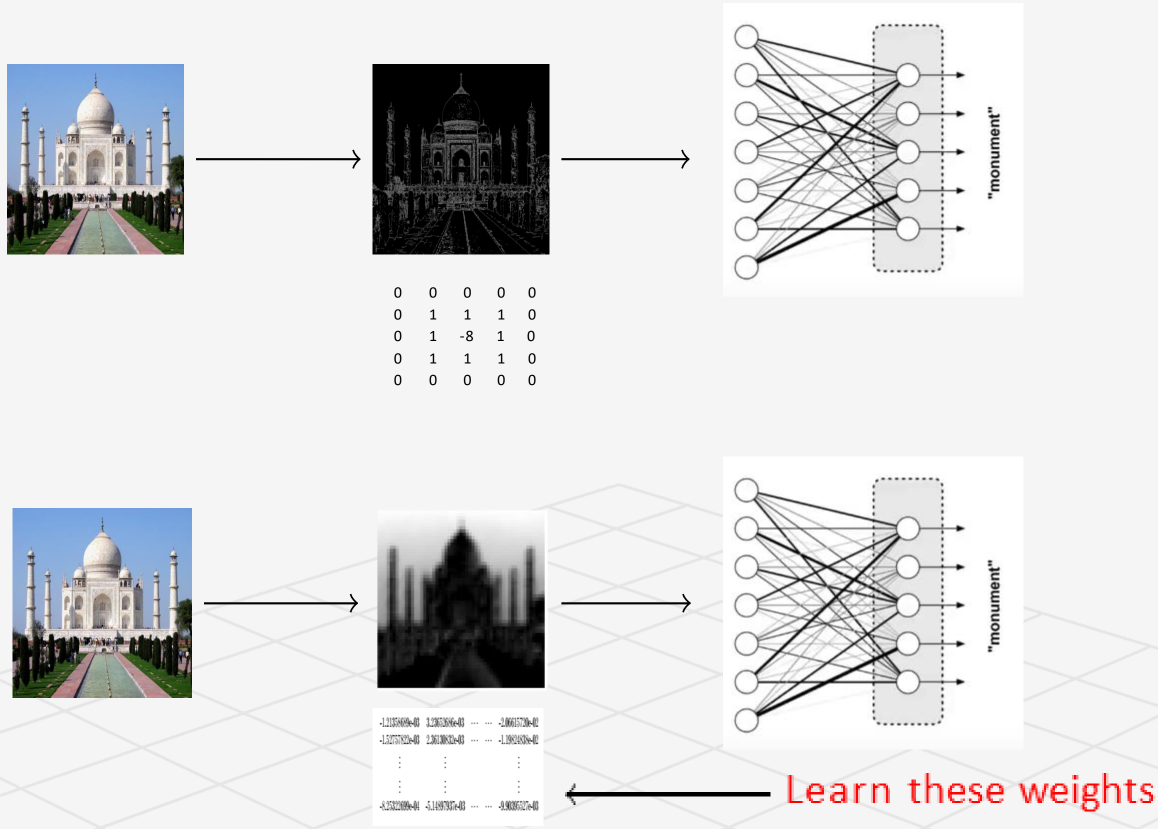
- Traditional ML-based computer vision solutions involve static feature engineering from images (e.g. recall SIFT, LBP, HoG, etc)
- Though effective, static feature engineering was a bottleneck of pre-DL vision solutions



static feature extraction (no learning) learning weights of classifier

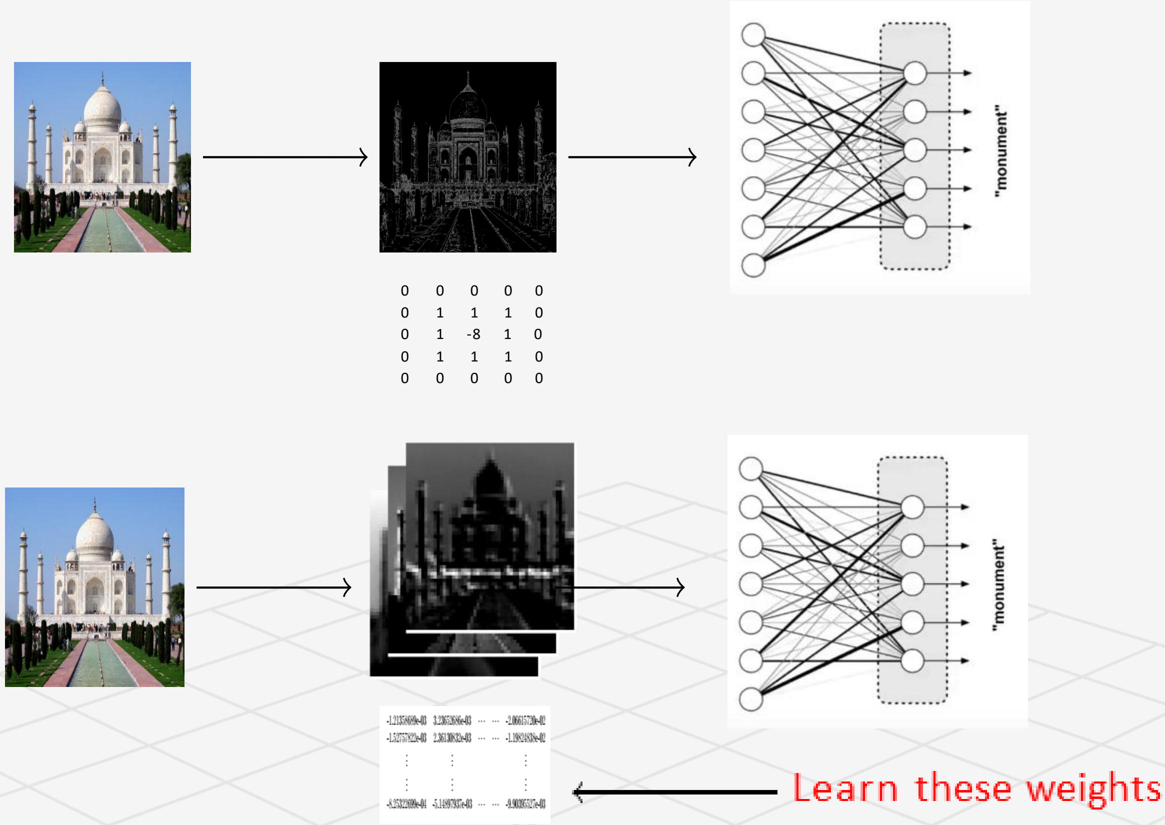
Beyond static feature engineering

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



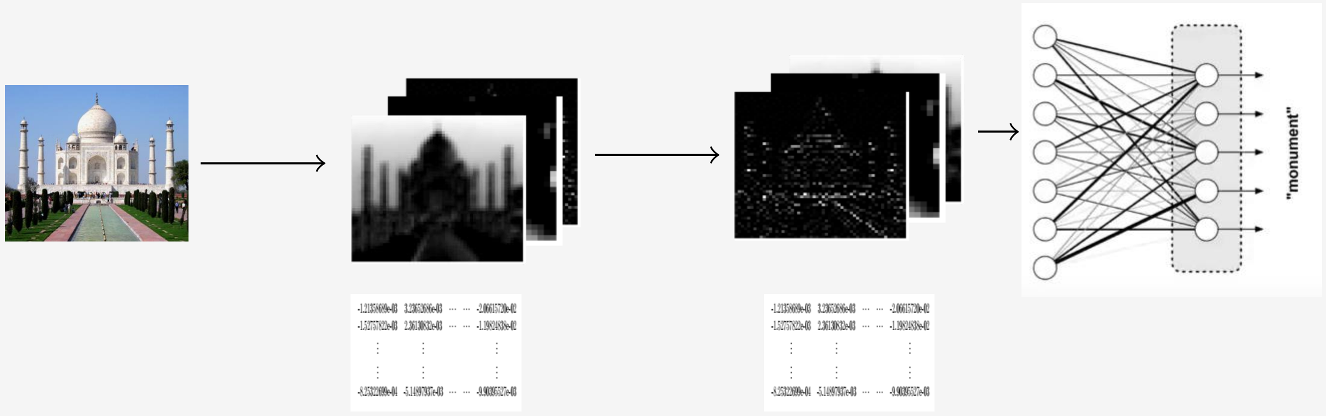
Beyond static feature engineering

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



Beyond static feature engineering

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

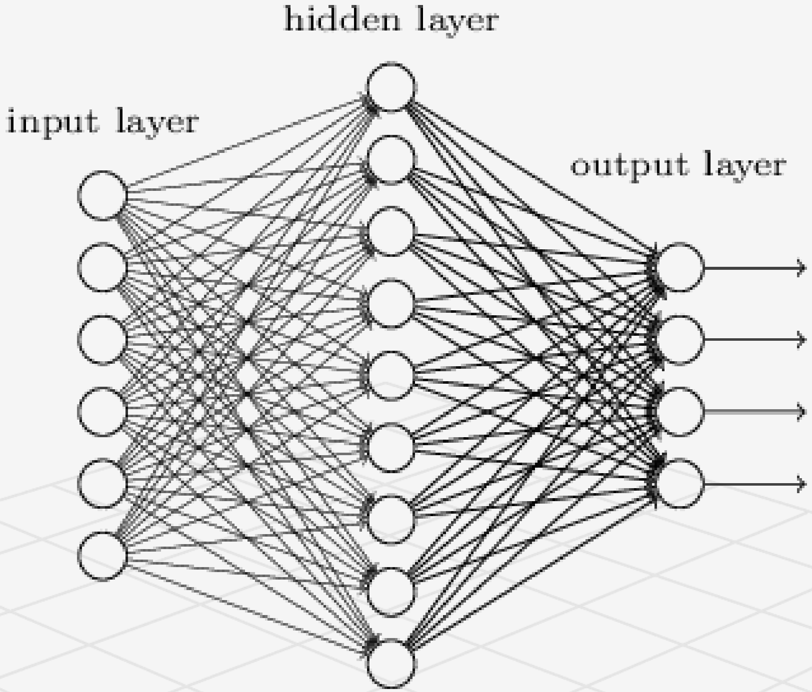


Yes, we can !

Simply by treating these kernels as parameters and learning them in addition to the weights of the classifier (using back propagation)

Pause and ponder

- Okay, I get it that the idea is to learn the kernel/filters by just treating them as parameters of the classification model
- But why not directly use flattened images with fully connected neural network or FNN instead?



Challenges of applying FNNs to images

MNIST Dataset

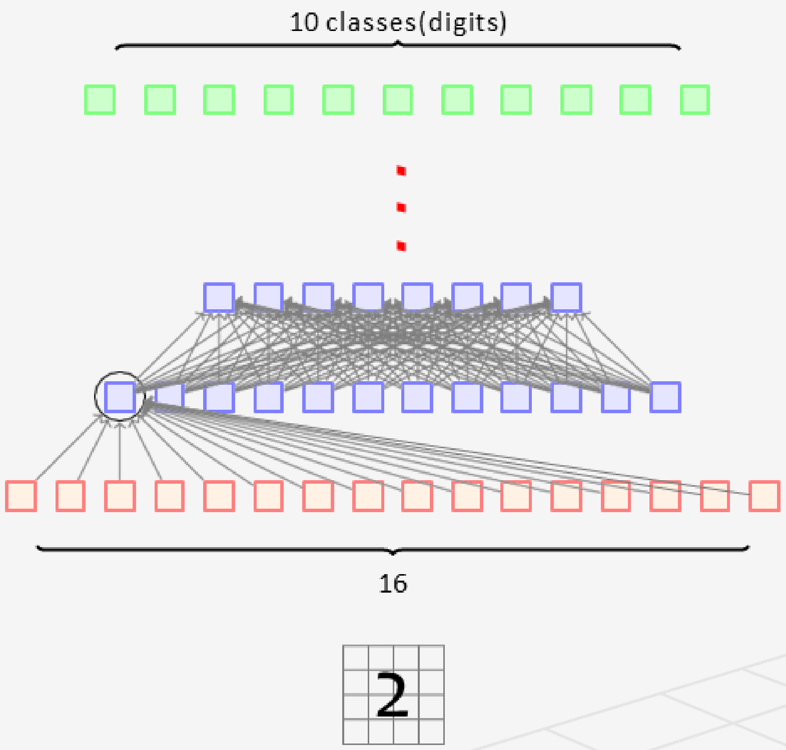
- On a reasonably simple dataset like MNIST, we can get about **2%** error (or even better) using FNNs, but
  - Ignores spatial (2-D) structure of input images - unroll each 28×28 image into a 784-D vector
    - Pixels that are spatially separate are treated the same way as pixels that are adjacent
- No obvious way for networks to learn same features (e.g. edges) at different places in the input image
- Can get computationally expensive for large images
  - For a $1\mathrm{MP}$ color image with 20 neurons in the first hidden layer, how many weights in the first layer?

60 million

How CNN solves these limitation

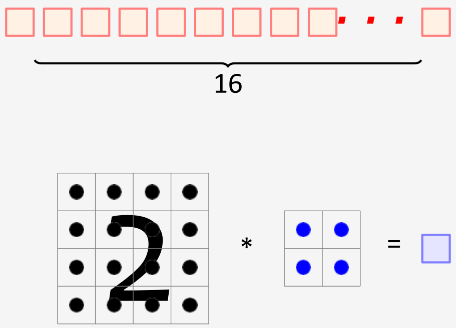
- Local receptive fields, in which hidden units are connected to local patches of the layer below, serve two purposes:
  - Capture local spatial relationships in pixels (which would not be captured by FNNs)
  - Greatly reduces number of parameters in the model
  - For a $\mathbf{1MP}$ color image a filter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

Local receptive field

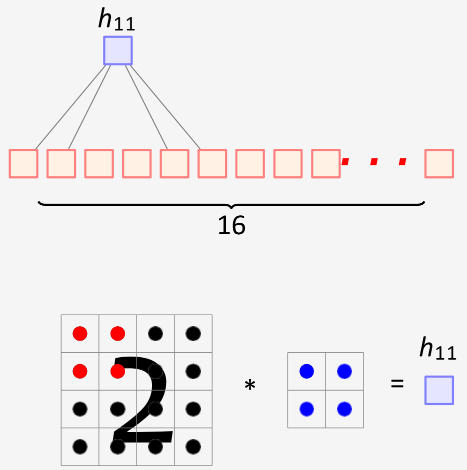


- This is what a regular feed-forward neural network
- There are many dense connections here
- For example all the 16 input neurons are contributing to the computation of h_{11}
- Contrast this to what happens in the case of convolution

Local receptive field

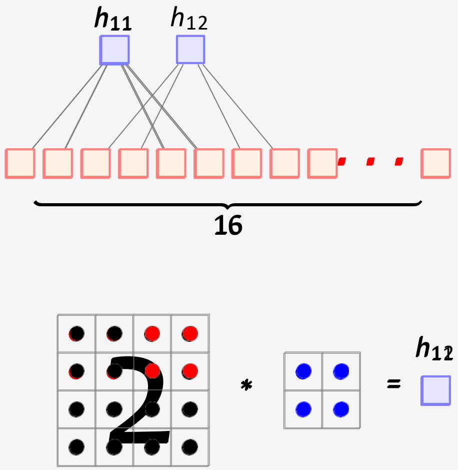


Local receptive field



- Only a few local neurons participate in the computation of h11
- For example, only pixels 1, 2, 5, 6 contribute to h11

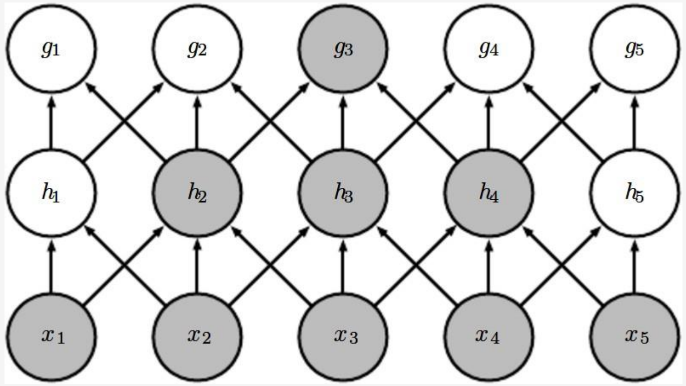
Local receptive field



- Only a few local neurons participate in the computation of h11
- For example, only pixels 1, 2, 5, 6 contribute to h11
- 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 ?

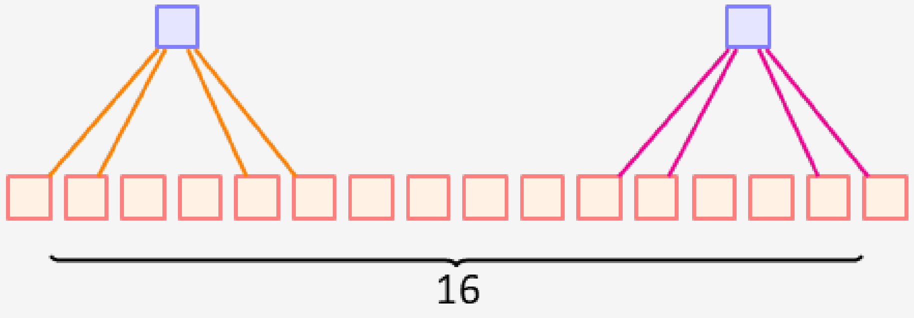
Local receptive field

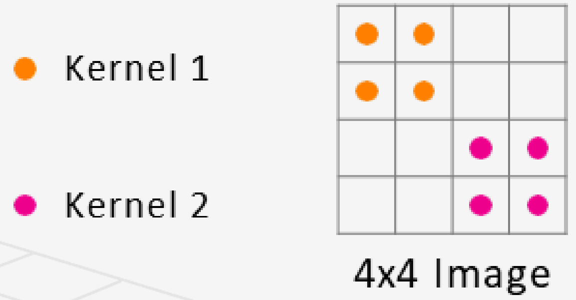
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really
- The two highlighted neurons (x1 &`x5)∗ do not interact in layer 1
- But they indirectly contribute to the computation of g3 and hence interact indirectly



Weight sharing

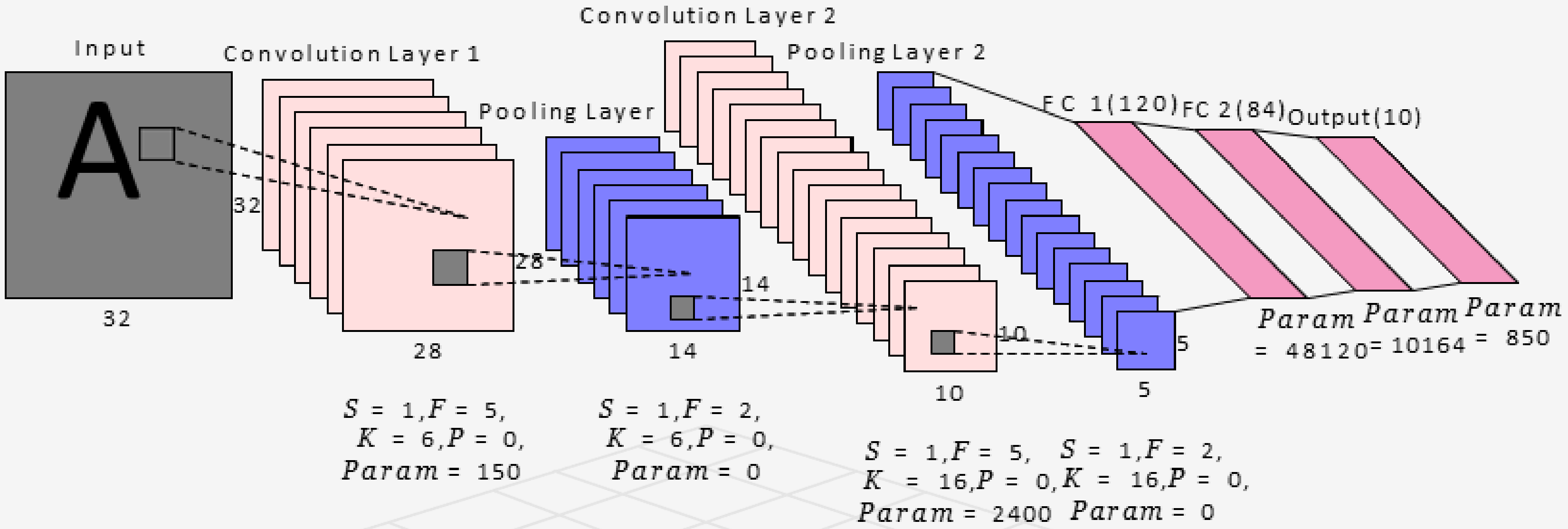
- Consider the following network
- Do we want the kernel weights to be different for different portions of the image?
  - We would want the filter to respond to an object/artifact same way irrespective of its position
- Imagine that we are trying to learn a kernel that detects edges
- Shouldn't we be applying the same kernel at all the portions of the image?





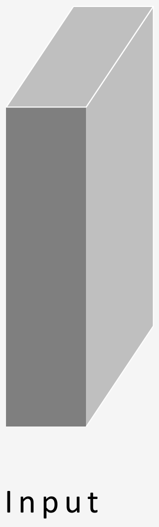
Convolutional neural network

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

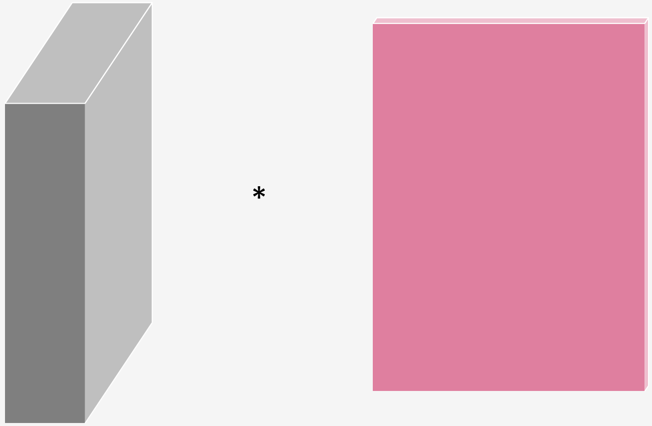


Pooling Layer

- Parameter free down sampling operation



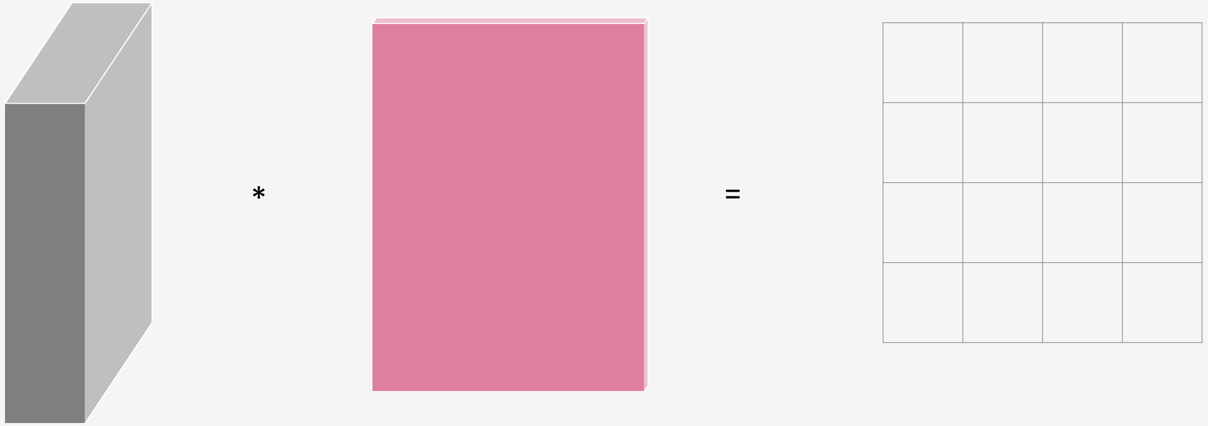
Pooling Layer



Input

1 filter

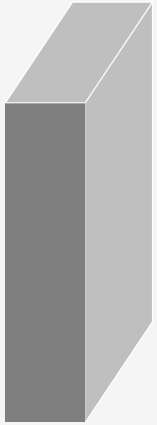
Pooling Layer



Input

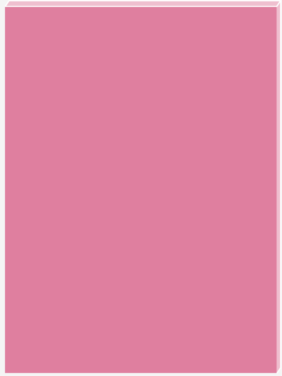
1 filter

Pooling Layer



Input

*

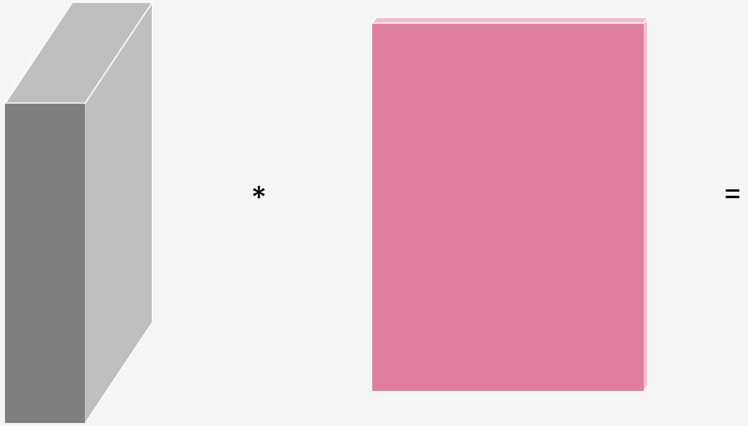


1 filter

=

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

Pooling Layer



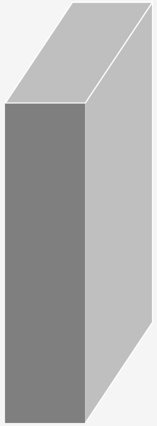
Input

1 filter

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

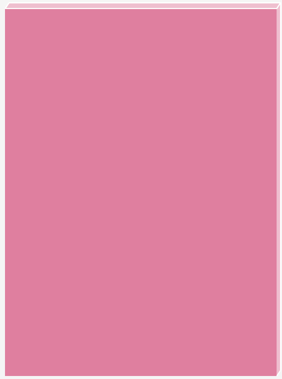
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

Pooling Layer



Input

*



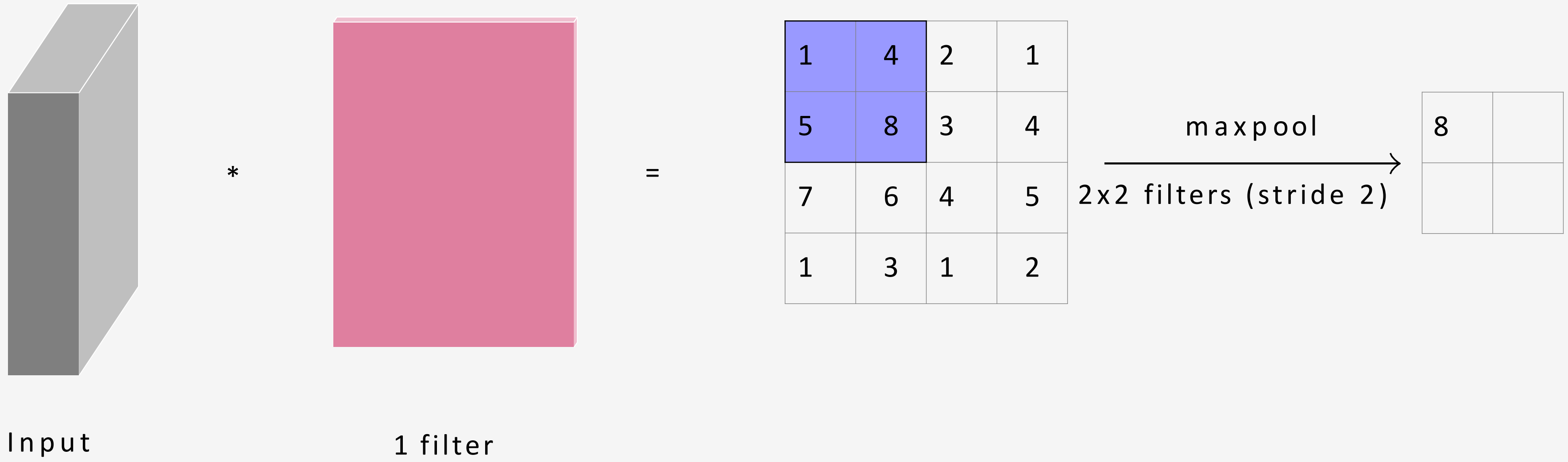
1 filter

=

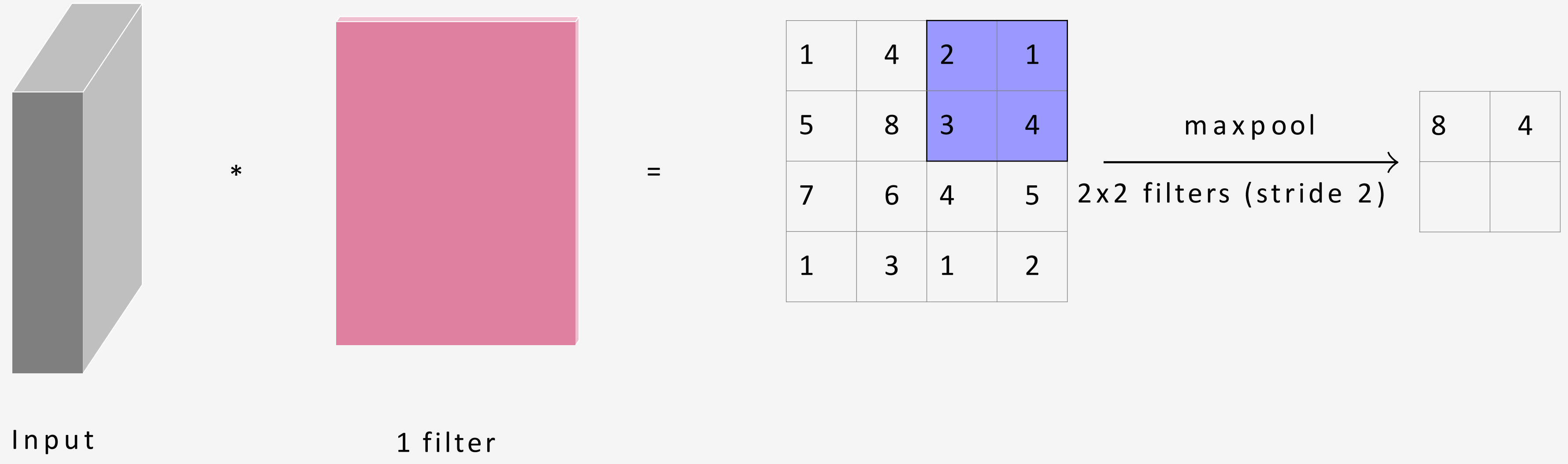
1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

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

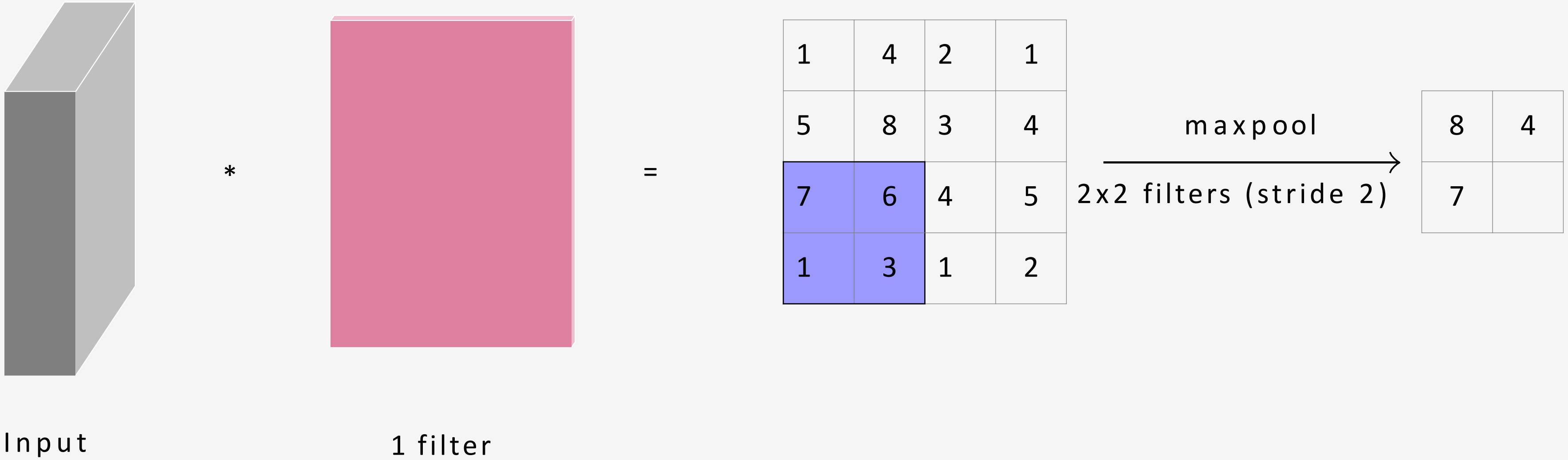
Pooling Layer



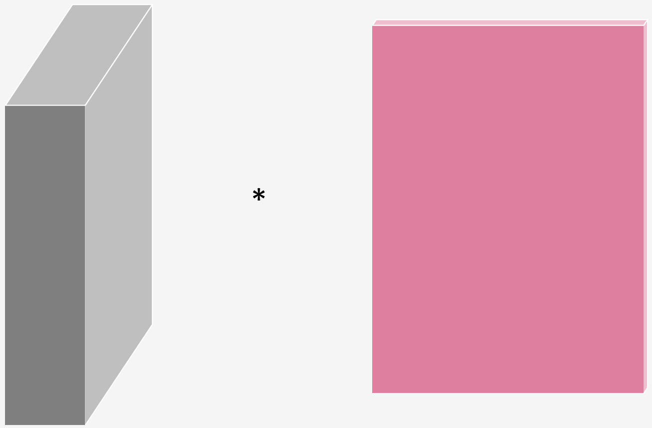
Pooling Layer



Pooling Layer



Pooling Layer



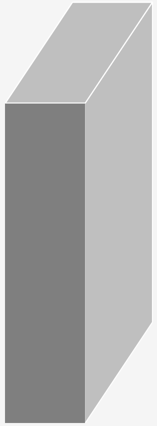
Input * 1 filter =

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

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

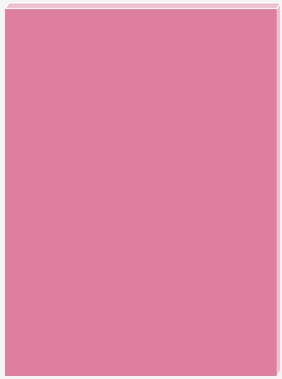
8	4
7	5

Pooling Layer



Input

*



1 filter

=

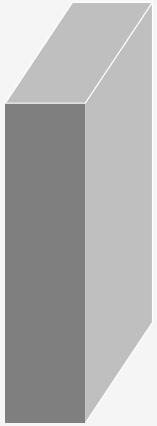
1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

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

8	4
7	5

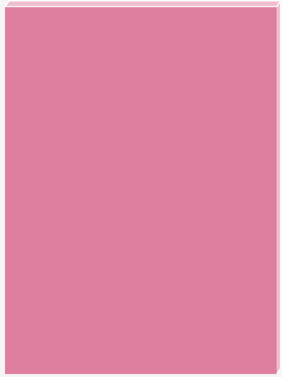
1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

Pooling Layer



Input

*



1 filter

=

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

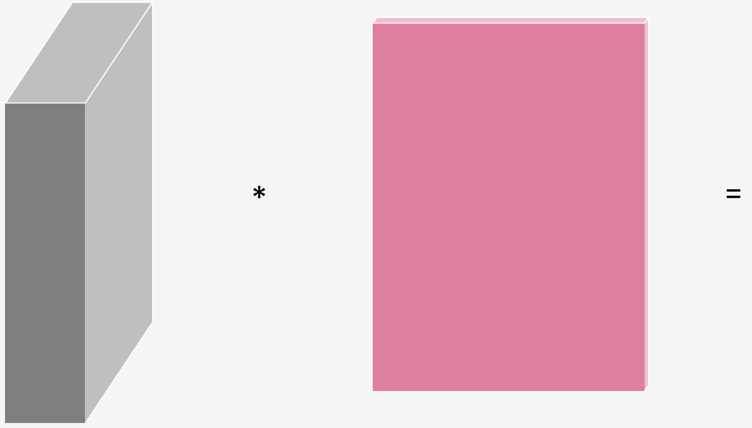
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

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

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

Pooling Layer



Input

1 filter

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

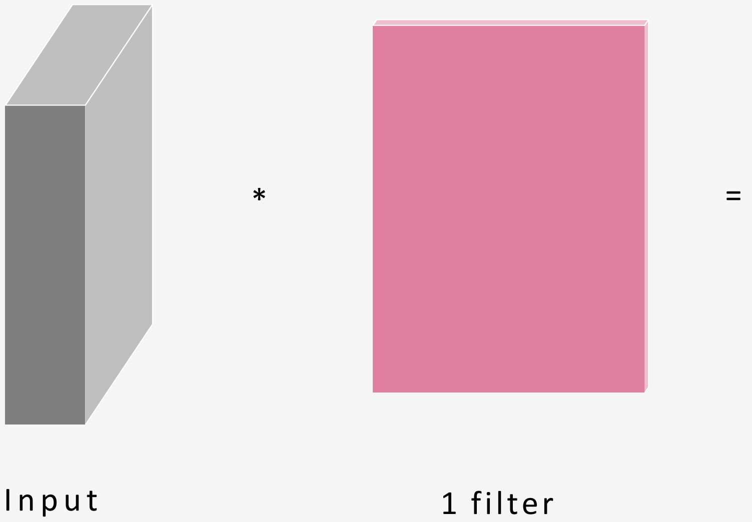
$\xrightarrow[\text{2x2 filters (stride 2)}]{\text{maxpool}}$

8	4
7	5

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

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

Pooling Layer



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

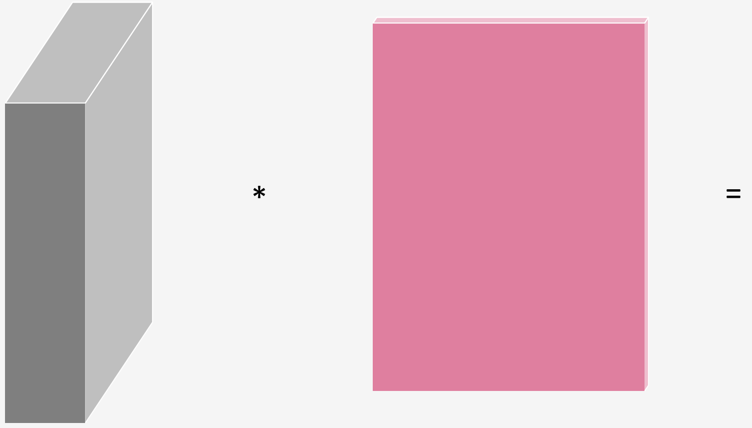
1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

maxpool

2x2 filters (stride 1)

8		

Pooling Layer



Input

1 filter

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

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

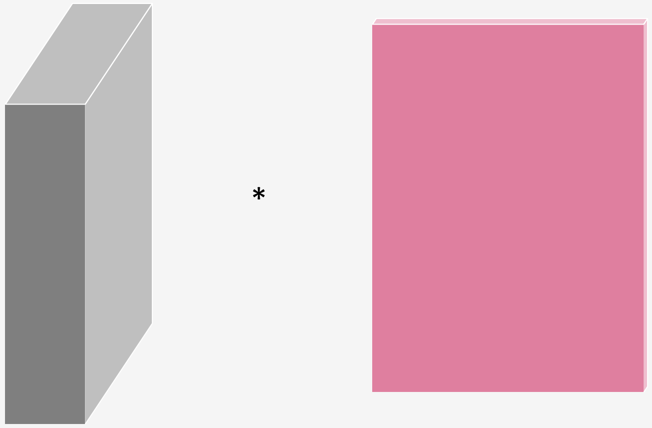
8	4
7	5

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	

Pooling Layer



Input * 1 filter =

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

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

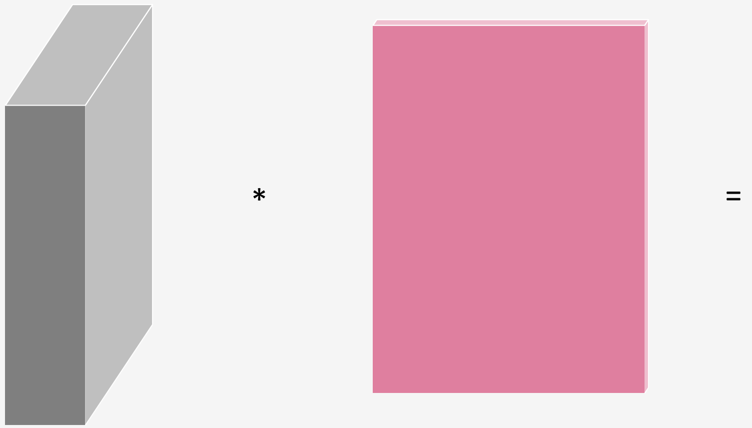
8	4
7	5

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	

Pooling Layer



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

Instead of max pooling we can also do average pooling

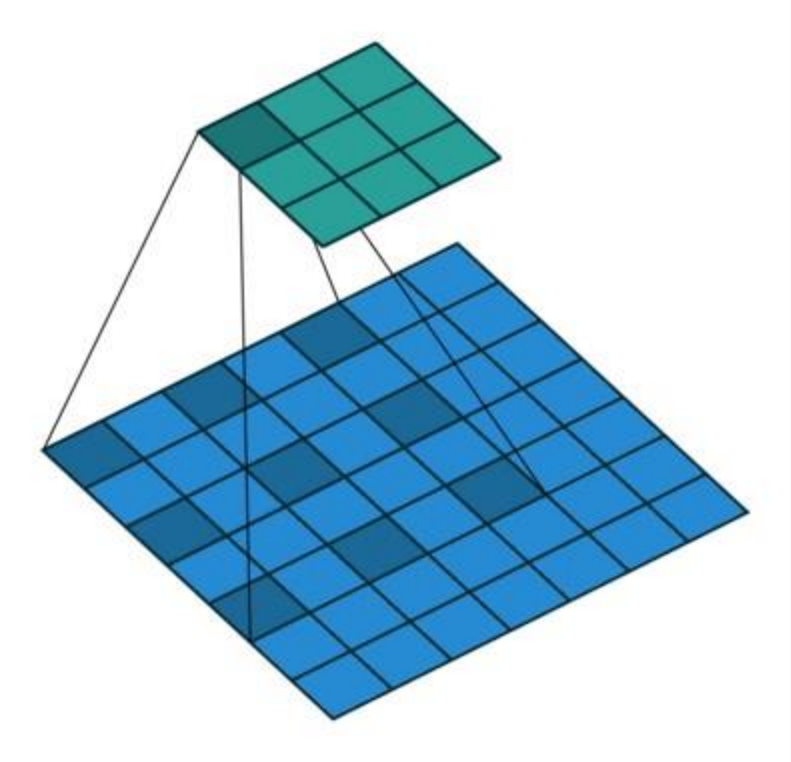
1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

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

8	8	4
8	8	5
7	6	5

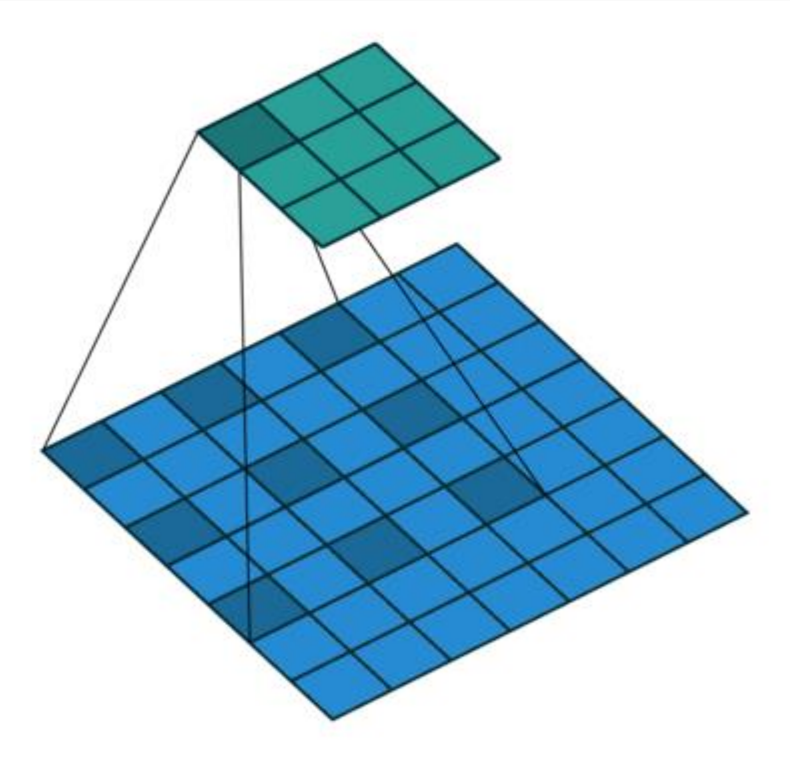
Other variants of convolution: Dilated Convolution

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



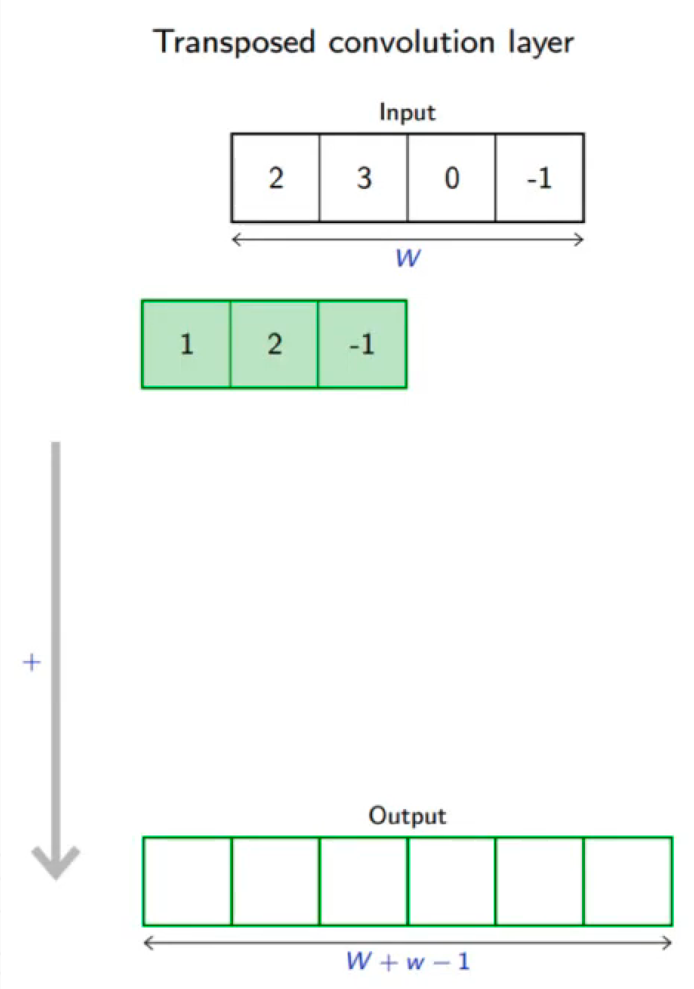
Other variants of convolution: Dilated Convolution

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



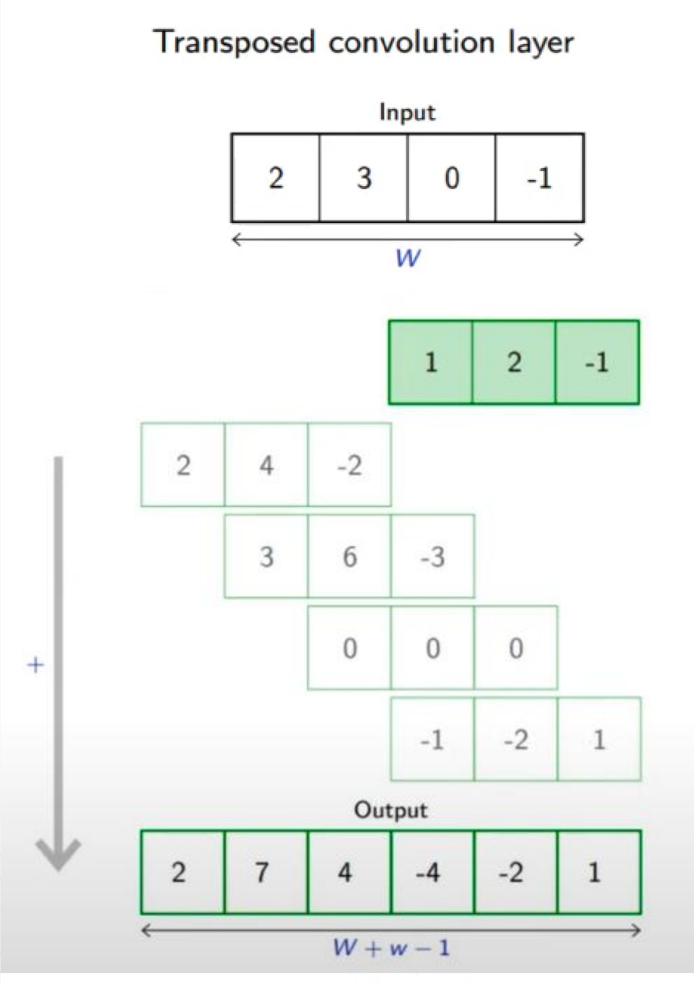
Other variants of convolution: Transpose Convolution

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



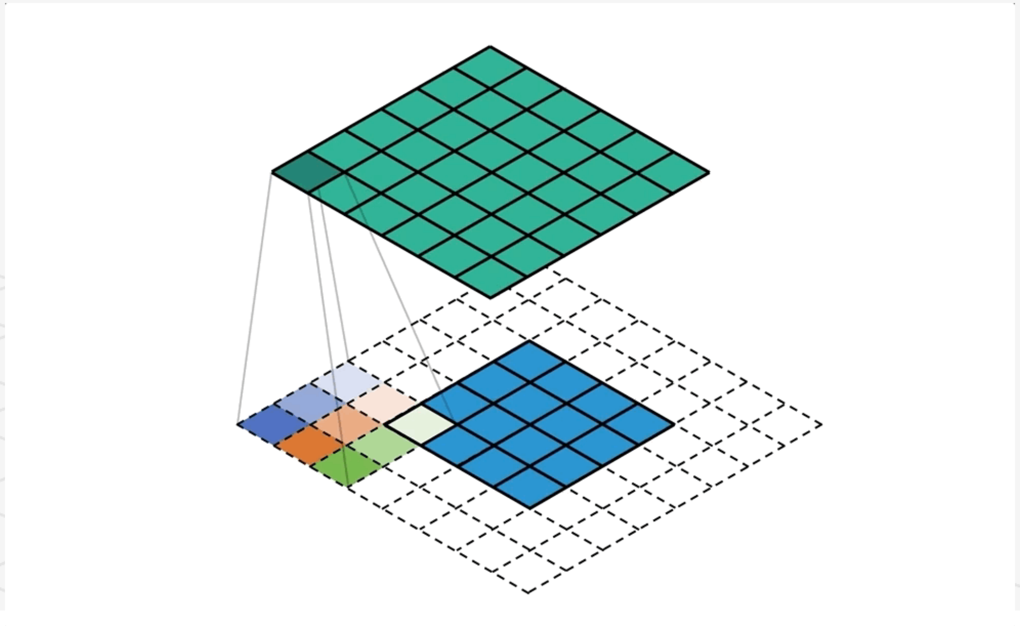
Other variants of convolution: Transpose Convolution

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

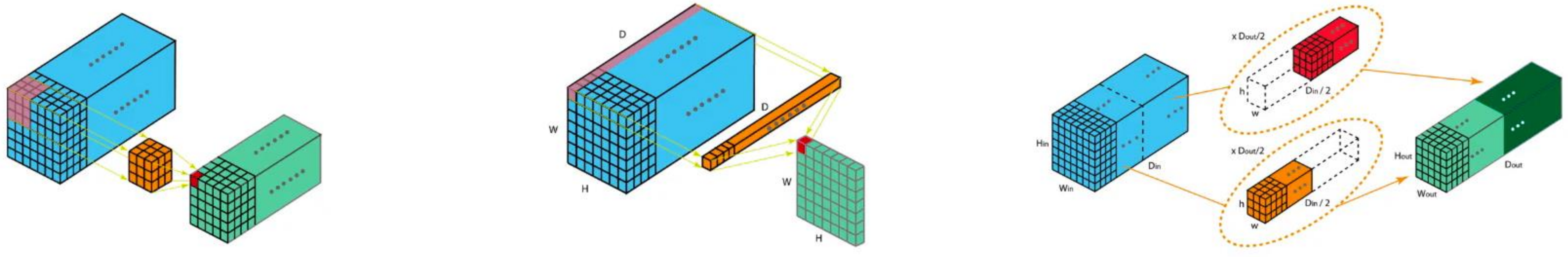


Other variants of convolution: Transpose Convolution

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

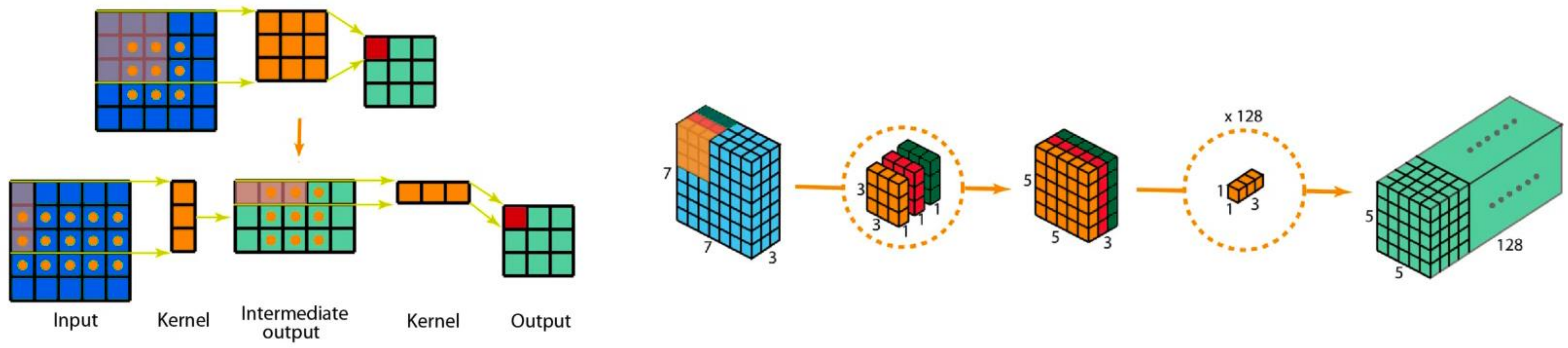


Other variants of convolution



3D Convolution | **1×1 Convolution** **Pointwise Convolution** | **Grouped Convolution**

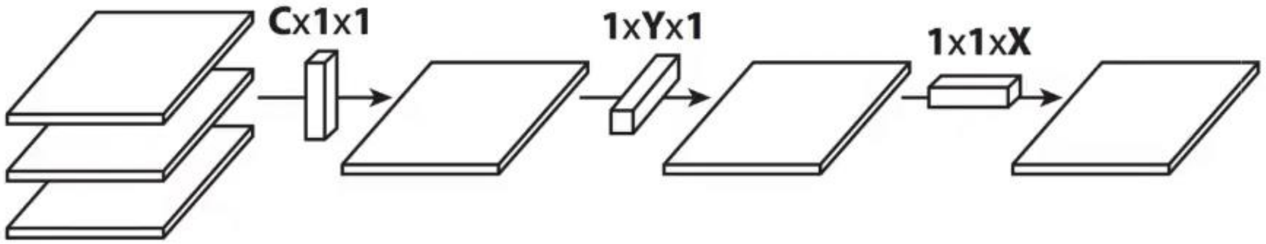
Other variants of convolution



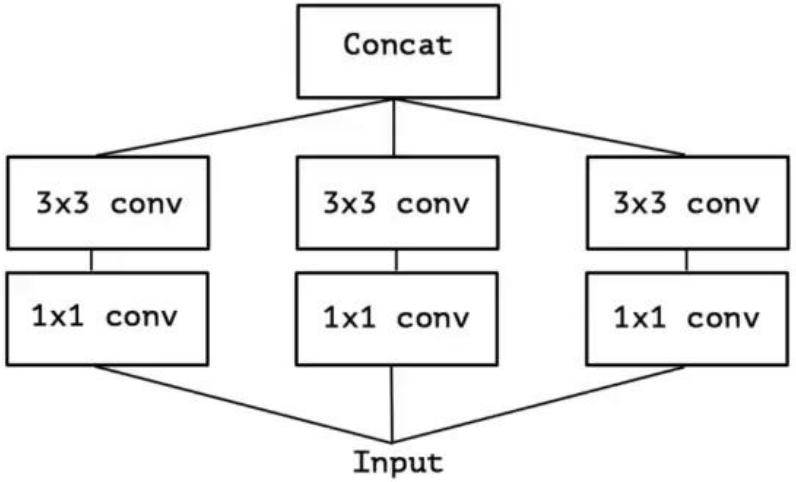
Spatial Separable Convolution

Depthwise Separable Convolution

Other variants of convolution



Flattened Convolutions



Spatial and Cross-Channel Convolutions

Resources and homework

- For an interactive illustration of the convolution operation, visit https://setosa.io/ev/image-kernels/
- Deep Learning Book: Chapter 9 - Convolutional Networks
- Stanford CS231n Notes

- Questions
  - Given a $32 \times 32 \times 3$ image and 6 filters of size $5 \times 5 \times 3$, what will be the dimension of the output volume when a stride of 1 and a padding of 0 is considered?
  - Is the max-pooling layer differentiable? How to backpropagate across it?