Deep Learning for Computer Vision

Convolutional Neural Networks: An Introduction

Vineeth N Balasubramanian

Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Homework Exercises

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

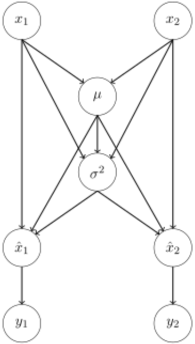
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Forward propagation is straight-forward:



Homework Exercises

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

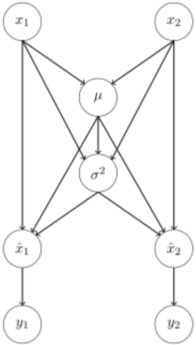
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

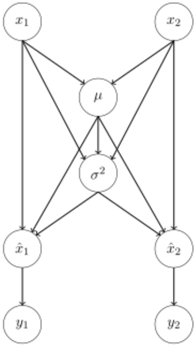
Forward propagation is straight-forward:



Backprop?

Image Credit: *Aditya Agrawal*

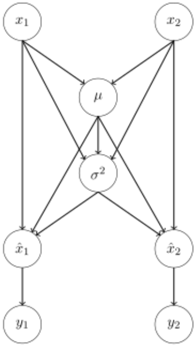
Homework: Backprop in Batch Normalization



$$\frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial \beta} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial \beta}$$

$$= \frac{\partial L}{\partial y_1} + \frac{\partial L}{\partial y_2} = \sum_{i=1}^{2} \frac{\partial L}{\partial y_i}$$

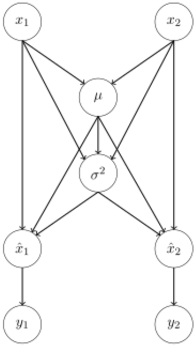
Homework: Backprop in Batch Normalization



$$\frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial \beta} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial \beta}$$
$$= \frac{\partial L}{\partial y_1} + \frac{\partial L}{\partial y_2} = \sum_{i=1}^{2} \frac{\partial L}{\partial y_i}$$

$$\frac{\partial L}{\partial \gamma} = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial \gamma} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial \gamma}$$
$$= \frac{\partial L}{\partial y_1}\hat{x}_1 + \frac{\partial L}{\partial y_2}\hat{x}_2 = \sum_{i=1}^{2} \frac{\partial L}{\partial y_i}\hat{x}_i$$

Homework: Backprop in Batch Normalization



$$\frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial \beta} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial \beta}$$
$$= \frac{\partial L}{\partial y_1} + \frac{\partial L}{\partial y_2} = \sum_{i=1}^{2} \frac{\partial L}{\partial y_i}$$

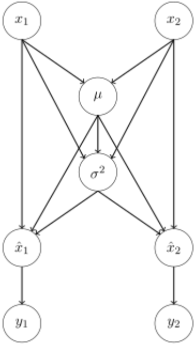
$$\frac{\partial L}{\partial \gamma} = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial \gamma} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial \gamma}$$
$$= \frac{\partial L}{\partial y_1}\hat{x}_1 + \frac{\partial L}{\partial y_2}\hat{x}_2 = \sum_{i=1}^{2} \frac{\partial L}{\partial y_i}\hat{x}_i$$

$$\frac{\partial L}{\partial \hat{x}_1} = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial \hat{x}_1} = \frac{\partial L}{\partial y_1}\gamma$$

$$\frac{\partial L}{\partial \hat{x}_2} = \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial \hat{x}_2} = \frac{\partial L}{\partial y_2}\gamma$$

Credit: Aditya Agrawal

Homework: Backprop in Batch Normalization

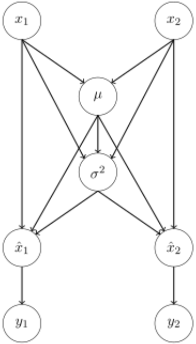


$$\frac{\partial L}{\partial \sigma^2} = \frac{\partial L}{\partial \hat{x}_1}\frac{\partial \hat{x}_1}{\partial \sigma^2} + \frac{\partial L}{\partial \hat{x}_2}\frac{\partial \hat{x}_2}{\partial \sigma^2} = \sum_{i=1}^{2} \frac{\partial L}{\partial \hat{x}_i}\frac{\partial \hat{x}_i}{\partial \sigma^2}$$

$$= \sum_{i=1}^{2} \frac{\partial L}{\partial \hat{x}_i}(x_i - \mu)\frac{-1}{2}(\sigma^2 + \epsilon)^{-3/2}$$

Credit: *Aditya Agrawal*

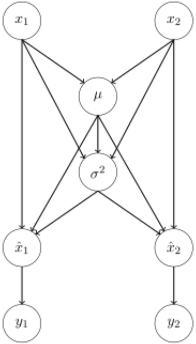
Homework: Backprop in Batch Normalization



$$
\begin{aligned}
\frac{\partial L}{\partial \mu} &= \frac{\partial L}{\partial \hat{x}_1}\frac{\partial \hat{x}_1}{\partial \mu} + \frac{\partial L}{\partial \hat{x}_2}\frac{\partial \hat{x}_2}{\partial \mu} + \frac{\partial L}{\partial \sigma^2}\frac{\partial \sigma^2}{\partial \mu} \\
&= \sum_{i=1}^{2} \frac{\partial L}{\partial \hat{x}_i}\frac{\partial \hat{x}_i}{\partial \mu} + \frac{\partial L}{\partial \sigma^2}\frac{\partial \sigma^2}{\partial \mu} \\
&= \sum_{i=1}^{2} \frac{\partial L}{\partial \hat{x}_i}\frac{-1}{\sqrt{\sigma^2+\epsilon}} + \frac{\partial L}{\partial \sigma^2}\frac{-2(x_1-\mu) \; - 2(x_2-\mu)}{2} \\
&= \sum_{i=1}^{2} \frac{\partial L}{\partial \hat{x}_i}\frac{-1}{\sqrt{\sigma^2+\epsilon}} + \frac{\partial L}{\partial \sigma^2}\frac{\sum_{i=1}^{2} -2(x_i-\mu)}{2}
\end{aligned}
$$

Credit: *Aditya Agrawal*

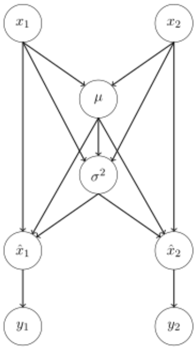
Homework: Backprop in Batch Normalization



$$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial \hat{x}_1}\frac{\partial \hat{x}_1}{\partial x_1} + \frac{\partial L}{\partial \sigma^2}\frac{\partial \sigma^2}{\partial x_1} + \frac{\partial L}{\partial \mu}\frac{\partial \mu}{\partial x_1}$$

$$= \frac{\partial L}{\partial \hat{x}_1}\frac{1}{\sqrt{\sigma^2+\epsilon}} + \frac{\partial L}{\partial \sigma^2}\frac{2(x_1-\mu)}{2} + \frac{\partial L}{\partial \mu}\frac{1}{2}$$

Homework: Backprop in Batch Normalization



$$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial \hat{x}_1}\frac{\partial \hat{x}_1}{\partial x_1} + \frac{\partial L}{\partial \sigma^2}\frac{\partial \sigma^2}{\partial x_1} + \frac{\partial L}{\partial \mu}\frac{\partial \mu}{\partial x_1}$$

$$= \frac{\partial L}{\partial \hat{x}_1}\frac{1}{\sqrt{\sigma^2+\epsilon}} + \frac{\partial L}{\partial \sigma^2}\frac{2(x_1-\mu)}{2} + \frac{\partial L}{\partial \mu}\frac{1}{2}$$

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial \hat{x}_i}\frac{1}{\sqrt{\sigma^2+\epsilon}} + \frac{\partial L}{\partial \sigma^2}\frac{2(x_i-\mu)}{m} + \frac{\partial L}{\partial \mu}\frac{1}{m}$$

Credit: *Aditya Agrawal*

Acknowledgements

- This lecture's content is largely based on **Lecture 11** of **CS7015** course taught by Mitesh Khapra at IIT Madras

Review: Convolution Operation

- **Convolution** is a mathematical way of combining two signals to form a third signal

Review: Convolution Operation

- **Convolution** is a mathematical way of combining two signals to form a third signal
- As we saw in Part 5 of Week 1, it is one of the most important techniques in signal processing

Review: Convolution Operation

- **Convolution** is a mathematical way of combining two signals to form a third signal
- As we saw in Part 5 of Week 1, it is one of the most important techniques in signal processing
- In case of 2D data (grayscale images), the convolution operation between a filter $W^{k \times k}$ and an image $X^{N_1 \times N_2}$ can be expressed as:

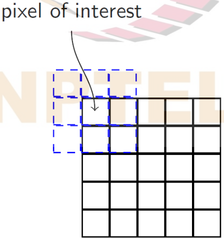
$$Y(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} W(u,v) X(i-u, j-v)$$

Convolution Operation

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

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

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

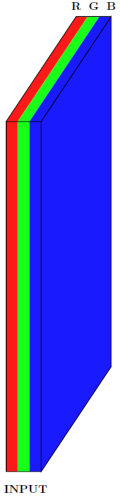


Pause and Ponder

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

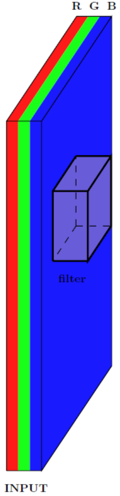
Convolution Operation

- What would a 3D filter look like?



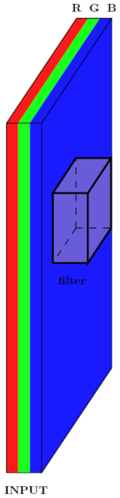
Convolution Operation

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



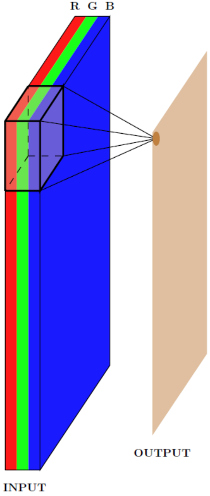
Convolution Operation

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



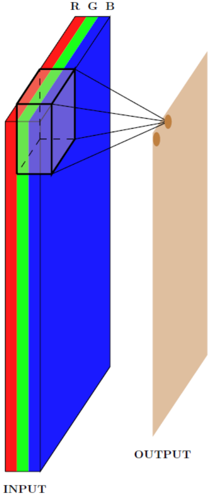
Convolution Operation

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



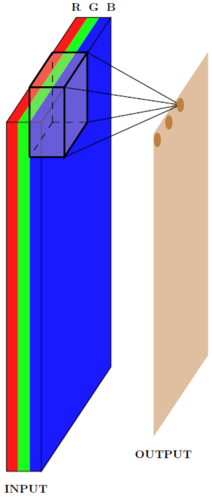
Convolution Operation

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



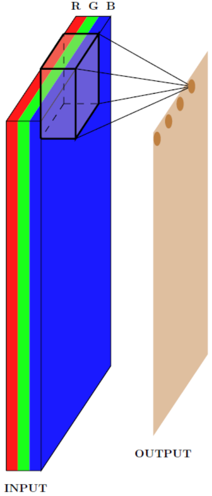
Convolution Operation

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



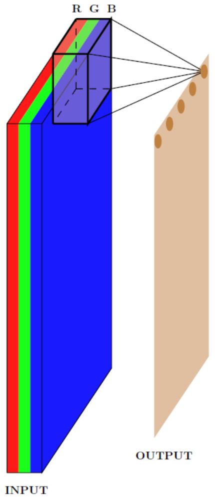
Convolution Operation

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



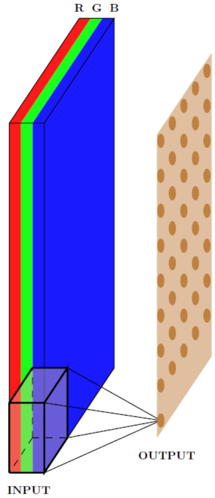
Convolution Operation

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



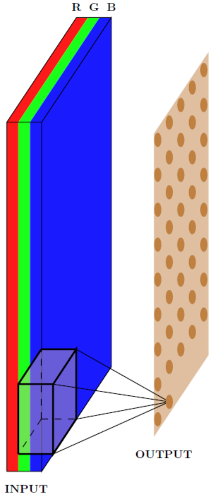
Convolution Operation

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



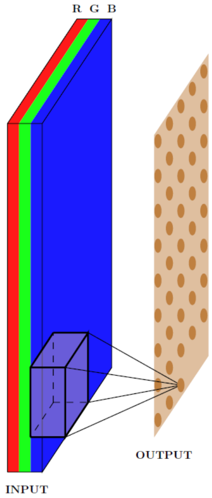
Convolution Operation

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



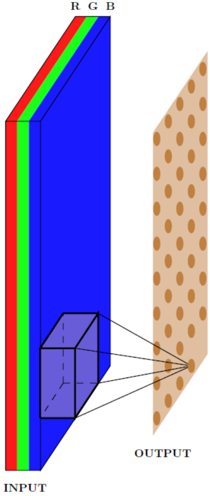
Convolution Operation

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



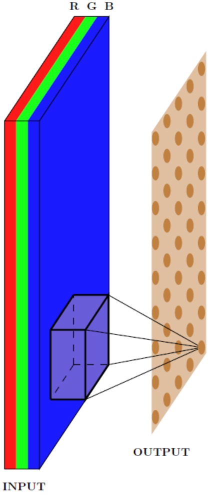
Convolution Operation

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



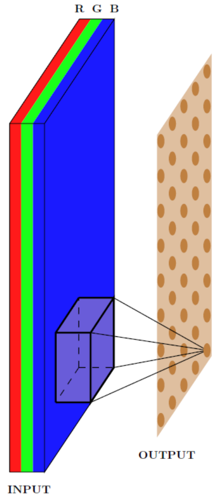
Convolution Operation

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



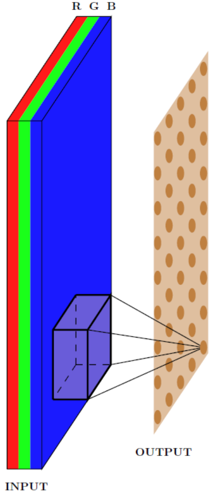
Convolution Operation

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



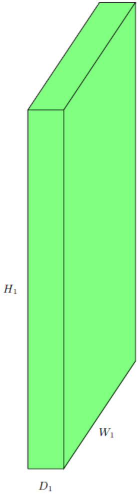
Convolution Operation

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



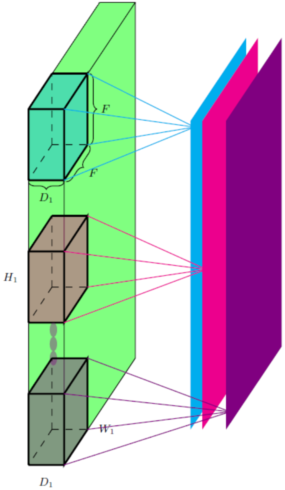
Convolution: Understanding the (Hyper)Parameters

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



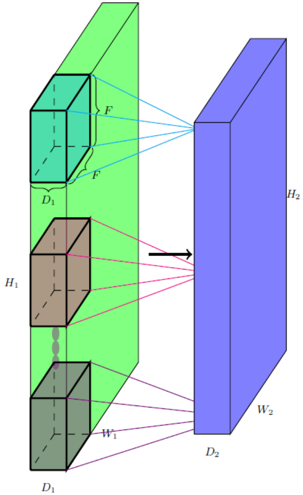
Convolution: Understanding the (Hyper)Parameters

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



Convolution: Understanding the (Hyper)Parameters

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

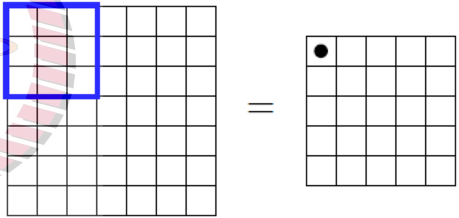


Convolution: Understanding the (Hyper)Parameters

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

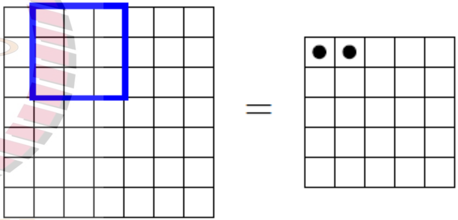
Convolution: Understanding the (Hyper)Parameters

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



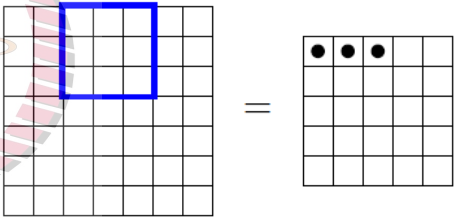
Convolution: Understanding the (Hyper)Parameters

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



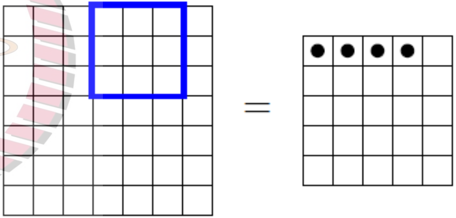
Convolution: Understanding the (Hyper)Parameters

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



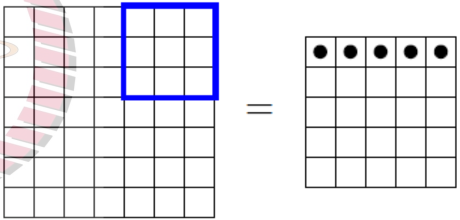
Convolution: Understanding the (Hyper)Parameters

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



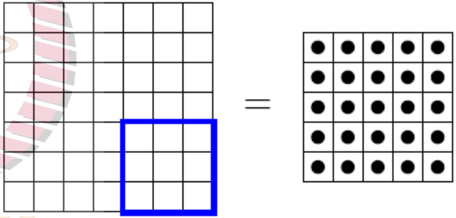
Convolution: Understanding the (Hyper)Parameters

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



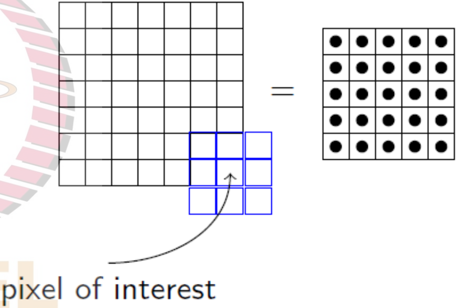
Convolution: Understanding the (Hyper)Parameters

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



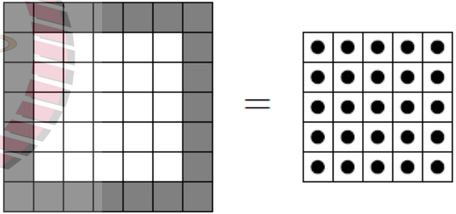
Convolution: Understanding the (Hyper)Parameters

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



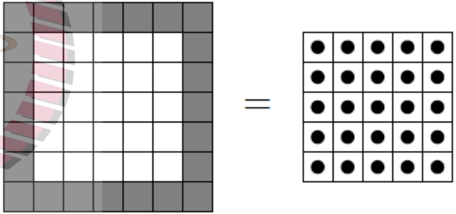
Convolution: Understanding the (Hyper)Parameters

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



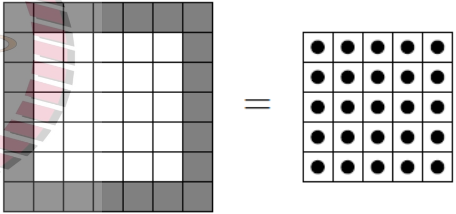
Convolution: Understanding the (Hyper)Parameters

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



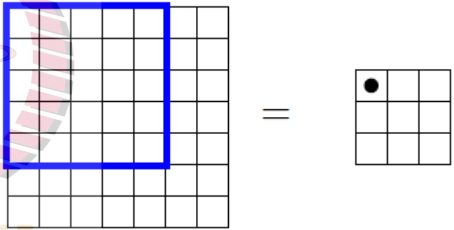
Convolution: Understanding the (Hyper)Parameters

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



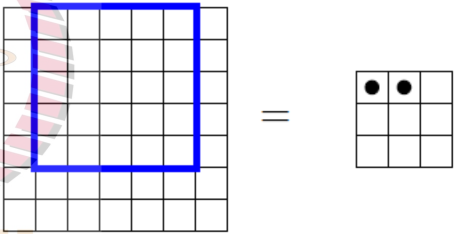
Convolution: Understanding the (Hyper)Parameters

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



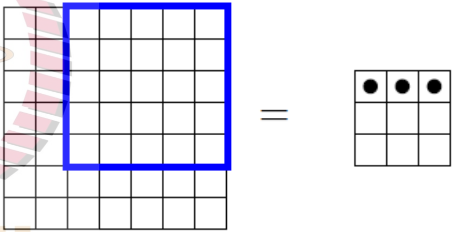
Convolution: Understanding the (Hyper)Parameters

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



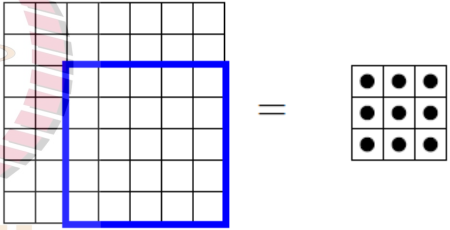
Convolution: Understanding the (Hyper)Parameters

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



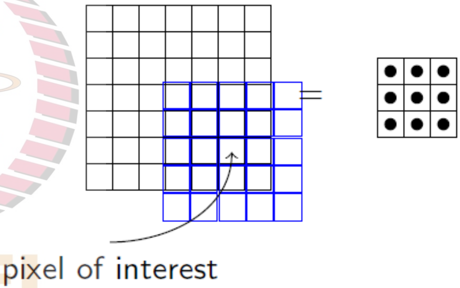
Convolution: Understanding the (Hyper)Parameters

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



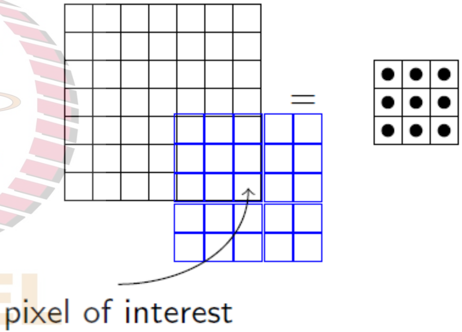
Convolution: Understanding the (Hyper)Parameters

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



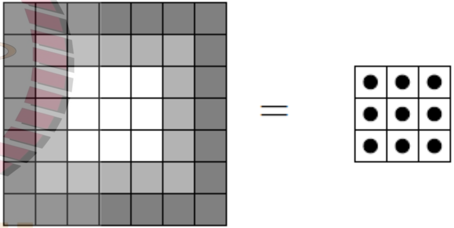
Convolution: Understanding the (Hyper)Parameters

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



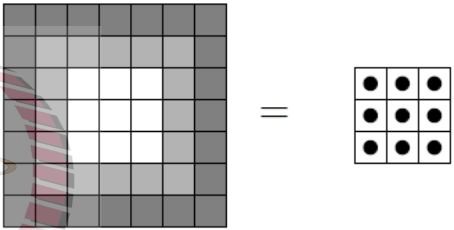
Convolution: Understanding the (Hyper)Parameters

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



Convolution: Understanding the (Hyper)Parameters

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



In general,

$$W_2 = W_1 - F + 1$$
$$H_2 = H_1 - F + 1$$

We will refine this formula further

Convolution: Understanding the (Hyper)Parameters

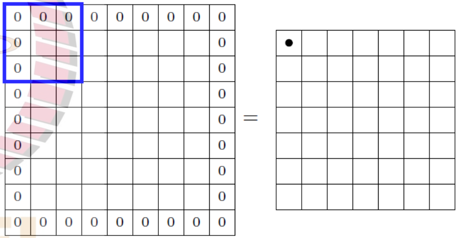
- What if we want output to be of same size as input?

Convolution: Understanding the (Hyper)Parameters

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

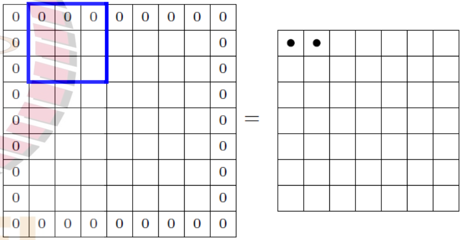
Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right; recall there are other ways of padding, see Week 1 Part 5 lecture



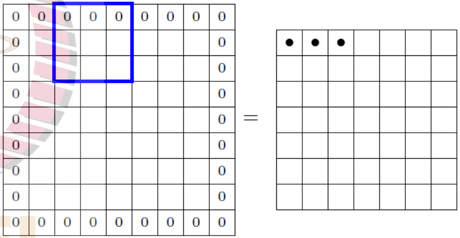
Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad P = 1 with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right; recall there are other ways of padding, see Week 1 Part 5 lecture



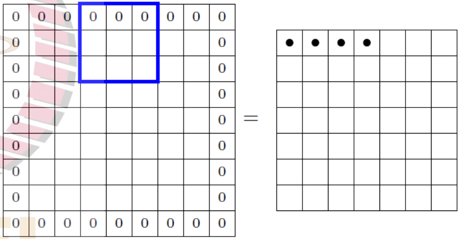
Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right; recall there are other ways of padding, see Week 1 Part 5 lecture



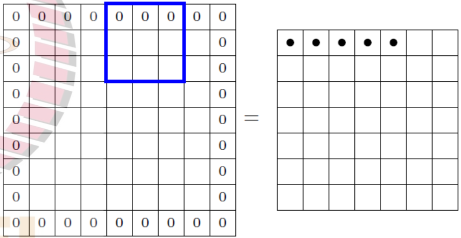
Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right; recall there are other ways of padding, see Week 1 Part 5 lecture



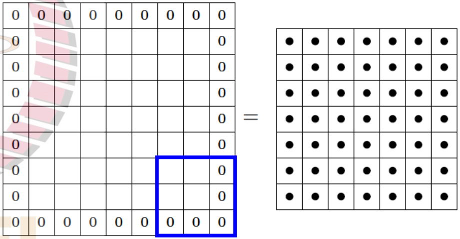
Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right; recall there are other ways of padding, see Week 1 Part 5 lecture



Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right; recall there are other ways of padding, see Week 1 Part 5 lecture



Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad $P = 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right; recall there are other ways of padding, see Week 1 Part 5 lecture

Convolution: Understanding the (Hyper)Parameters

- What if we want output to be of same size as input?
- Recall use of **padding**
- Pad inputs with appropriate number of inputs so you can now apply kernel at corners
- Let us use pad P $= 1$ with a 3×3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right; recall there are other ways of padding, see Week 1 Part 5 lecture

We now have:

$$W_2 = W_1 - F + 2P + 1$$
$$H_2 = H_1 - F + 2P + 1$$

We will refine this formula further

Convolution: Understanding the (Hyper)Parameters

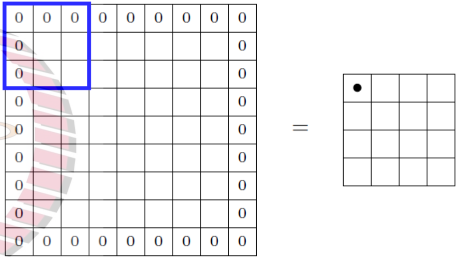
- What does **stride** S do?

Convolution: Understanding the (Hyper)Parameters

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

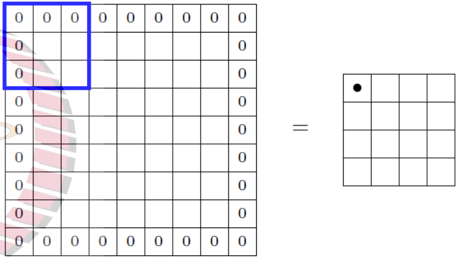
Convolution: Understanding the (Hyper)Parameters

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



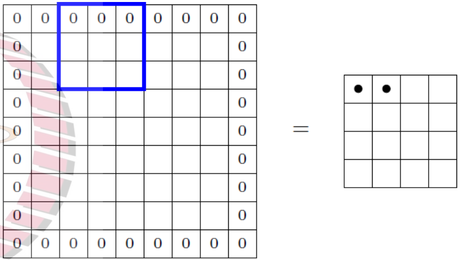
Convolution: Understanding the (Hyper)Parameters

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



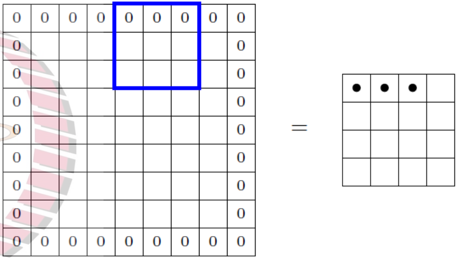
Convolution: Understanding the (Hyper)Parameters

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



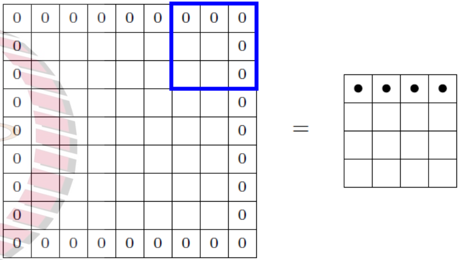
Convolution: Understanding the (Hyper)Parameters

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



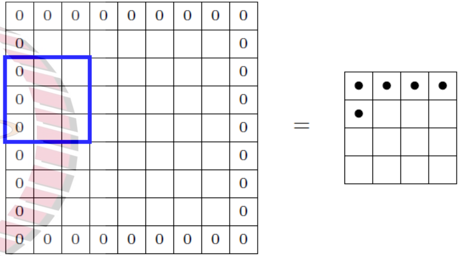
Convolution: Understanding the (Hyper)Parameters

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



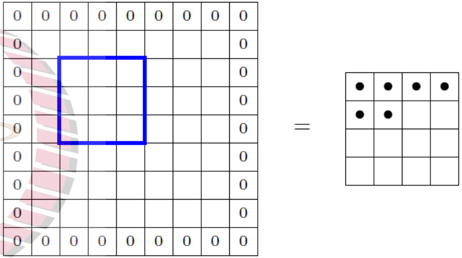
Convolution: Understanding the (Hyper)Parameters

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



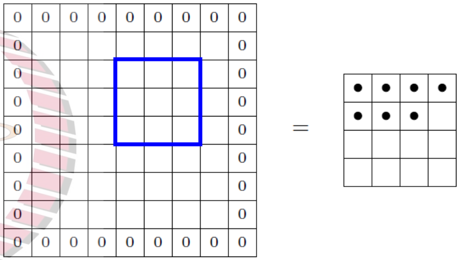
Convolution: Understanding the (Hyper)Parameters

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

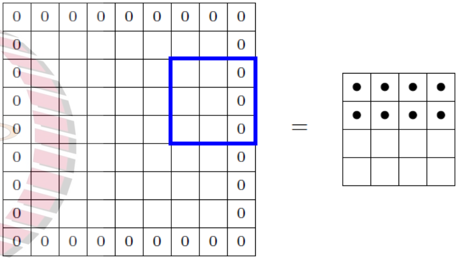


Convolution: Understanding the (Hyper)Parameters

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

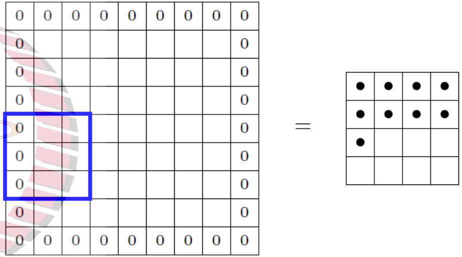


- What does **stride** S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Skip every 2nd pixel ($S = 2$) which will result in an output of smaller dimensions



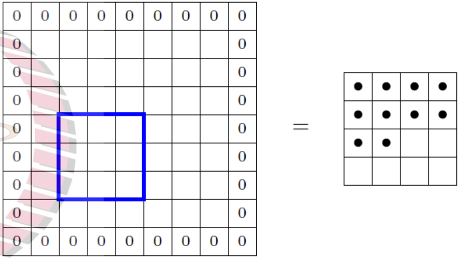
Convolution: Understanding the (Hyper)Parameters

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



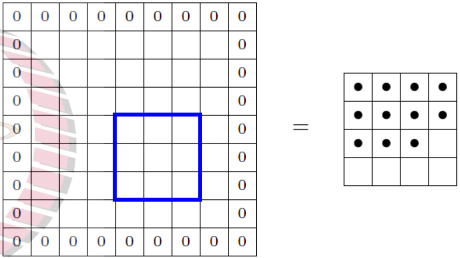
Convolution: Understanding the (Hyper)Parameters

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



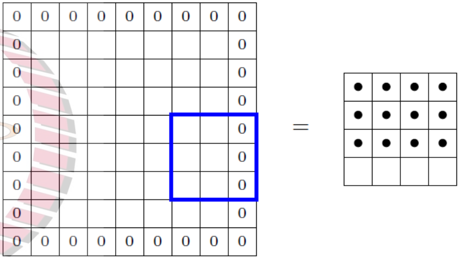
Convolution: Understanding the (Hyper)Parameters

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



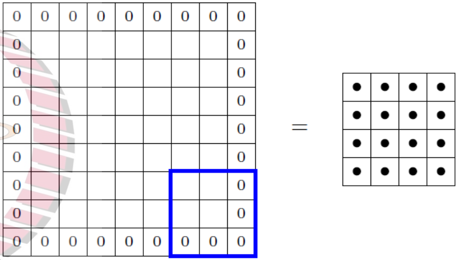
Convolution: Understanding the (Hyper)Parameters

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



Convolution: Understanding the (Hyper)Parameters

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



Convolution: Understanding the (Hyper)Parameters

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

So our final formula should mostly look like,

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

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

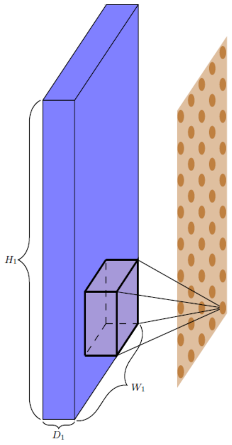
Not done yet, we will refine this formula further!

Convolution: Understanding the (Hyper)Parameters

- Finally, coming to depth of output

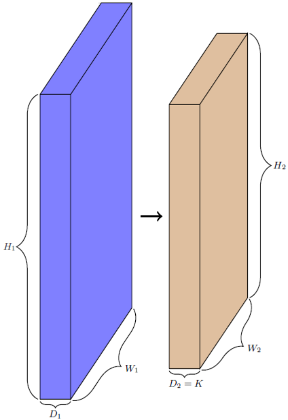
Convolution: Understanding the (Hyper)Parameters

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



Convolution: Understanding the (Hyper)Parameters

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



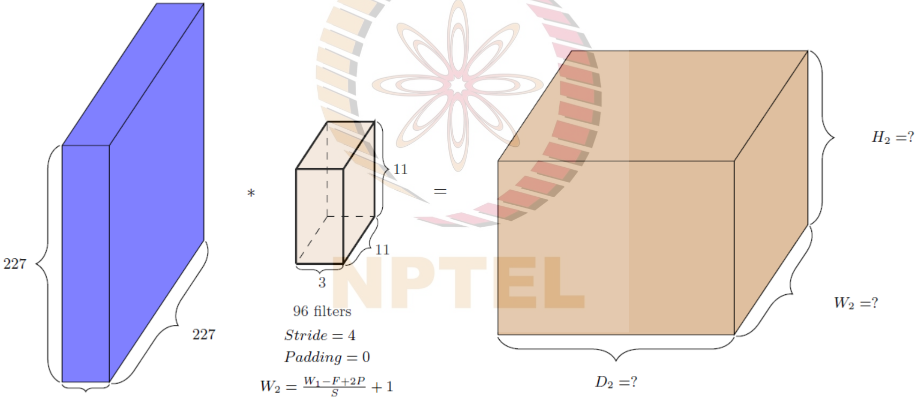
$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

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

$$D_2 = K$$

Quick Exercise

Work out output dimensions for the setting below!

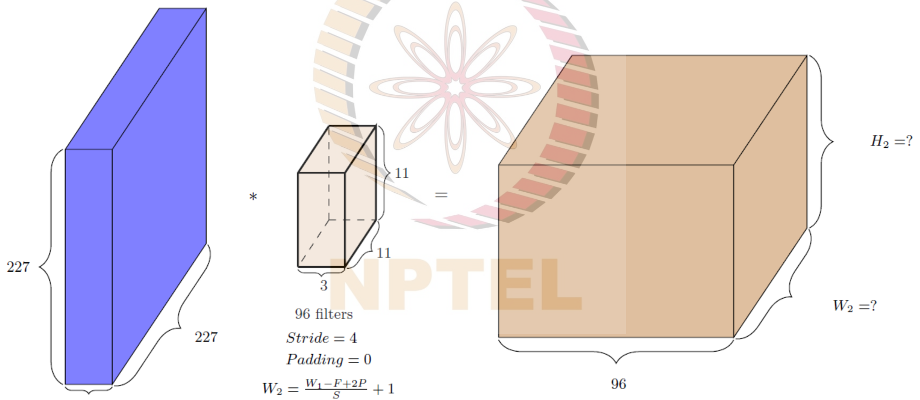


$W_2 = \frac{W_1 - F + 2P}{S} + 1$

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

Quick Exercise

Work out output dimensions for the setting below!



$Stride = 4$

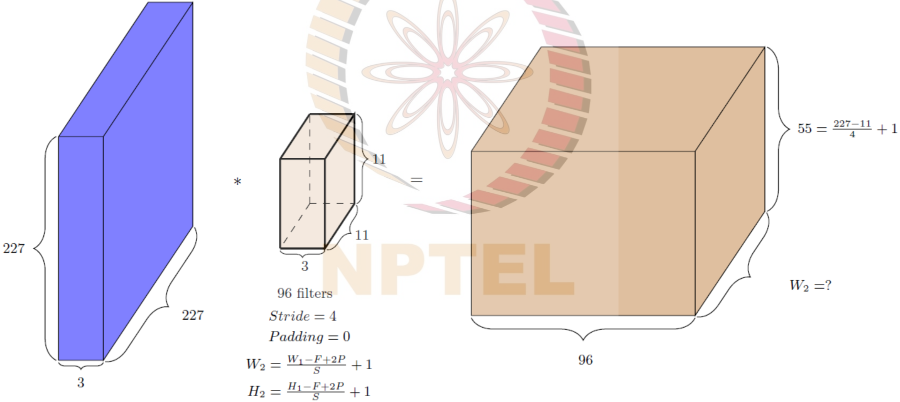
$Padding = 0$

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

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

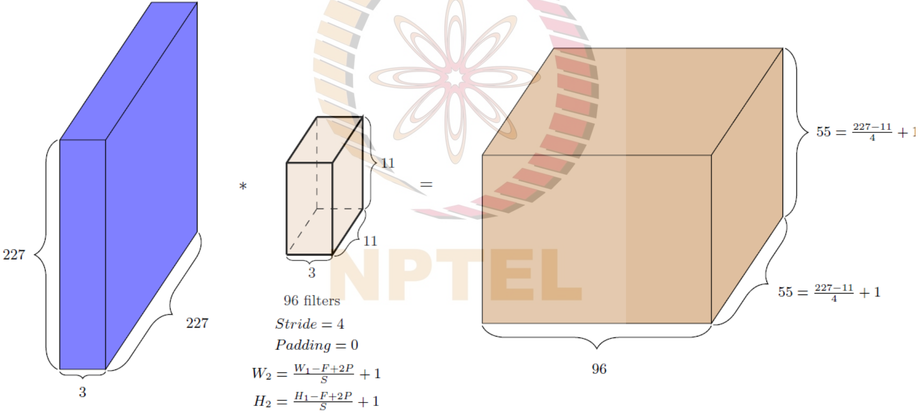
Quick Exercise

Work out output dimensions for the setting below!



Quick Exercise

Work out output dimensions for the setting below!

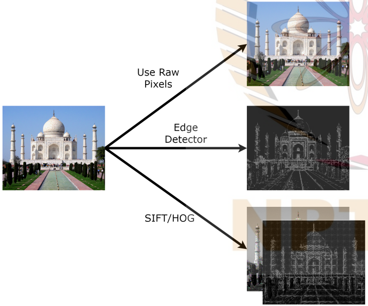


Pause and Ponder

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

Traditional Machine Learning for Vision

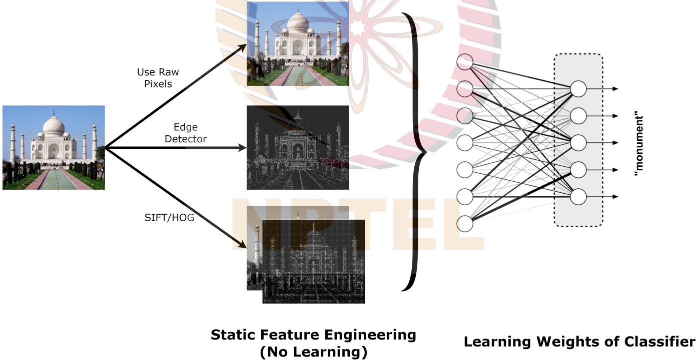
- Traditional ML-based computer vision solutions involve static feature engineering from images (e.g. recall SIFT, LBP, HoG, etc)
- Though effective, static feature engineering was a bottleneck of pre-DL vision solutions



Static Feature Engineering (No Learning)

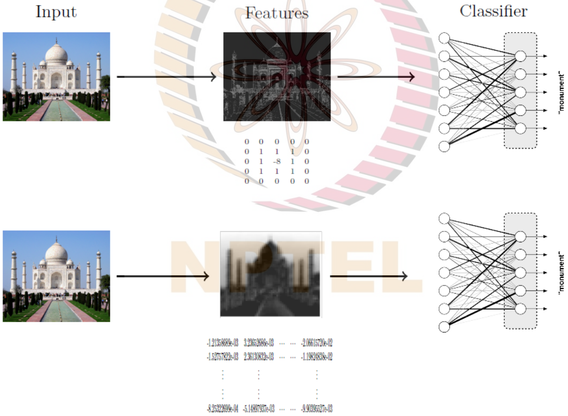
Traditional Machine Learning for Vision

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



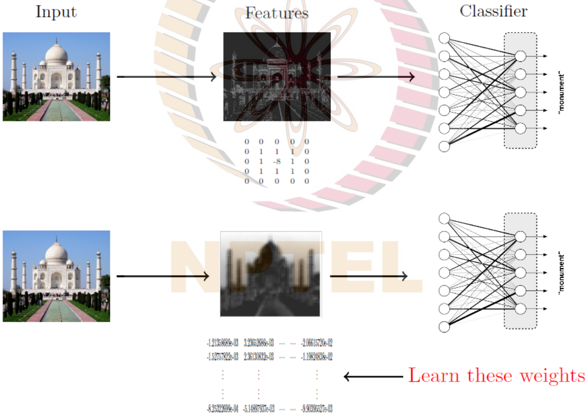
Beyond Static Feature Engineering

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



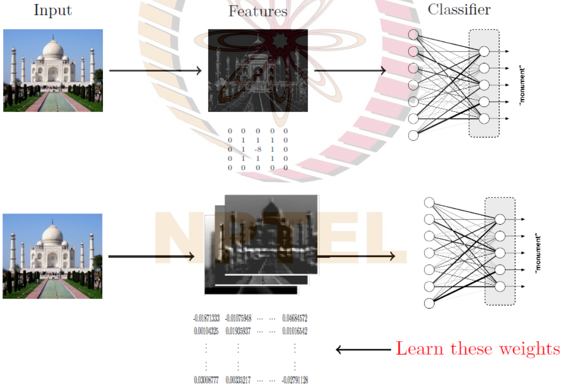
Beyond Static Feature Engineering

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



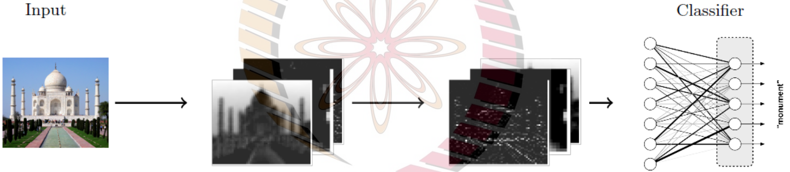
Beyond Static Feature Engineering

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



Beyond Static Feature Engineering

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

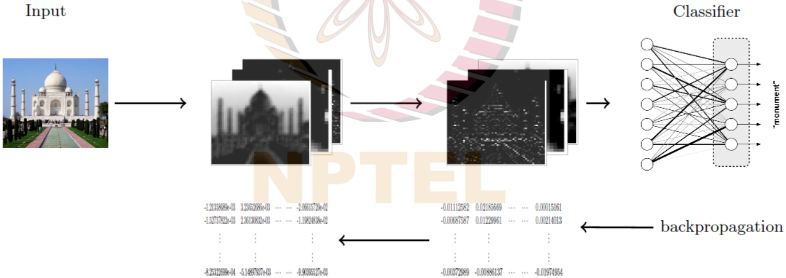


Beyond Static Feature Engineering

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

Beyond Static Feature Engineering

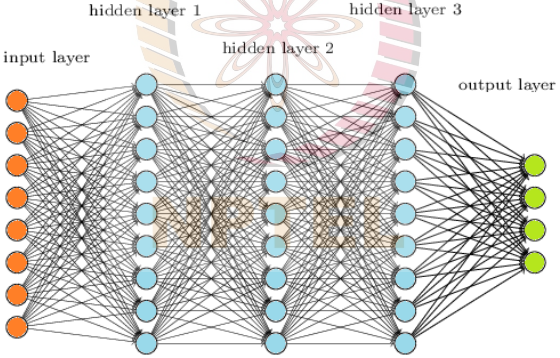
- Can we learn multiple layers of meaningful kernels/filters in addition to learning the weights of the classifier? **Yes, we can!**
- Simply by treating these kernels as parameters and learning them in addition to the weights of the classifier (using backpropagation, discussed in the next lecture)



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

Pause and Ponder

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



Challenges of Applying FNNs to Images

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 1MP color image with 20 neurons in the first hidden layer, how many weights in the first layer?

Challenges of Applying FNNs to Images

MNIST Dataset

Credit: Steve Renals

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

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

How do Convolutional Neural Networks Solve these Challenges?

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

How do Convolutional Neural Networks Solve these Challenges?

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

How do Convolutional Neural Networks Solve these Challenges?

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

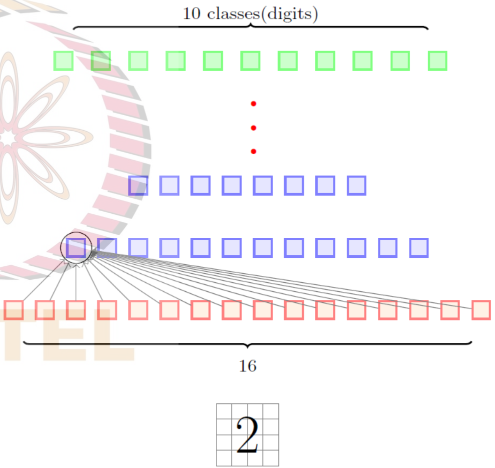
How do Convolutional Neural Networks Solve these Challenges?

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

Credit: Steve Renals

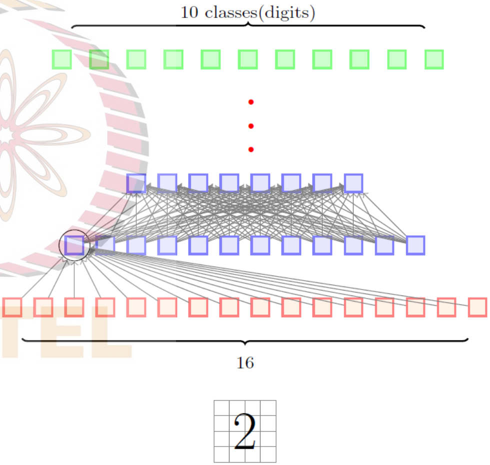
Local Receptive Fields

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



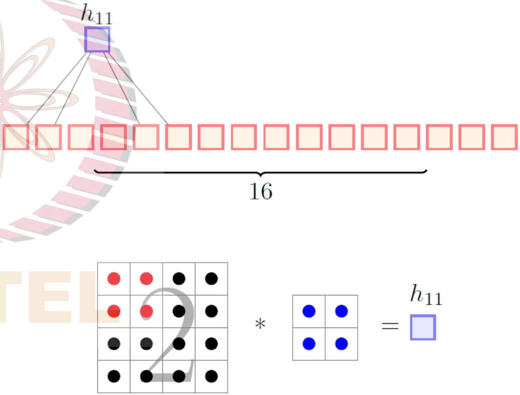
Local Receptive Fields

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



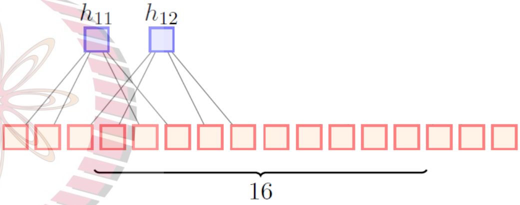
Local Receptive Fields

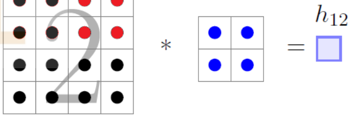
- Only a few local neurons participate in computation of h_{11}
- E.g. only pixels 1, 2, 5, 6 contribute to h_{11}



Local Receptive Fields

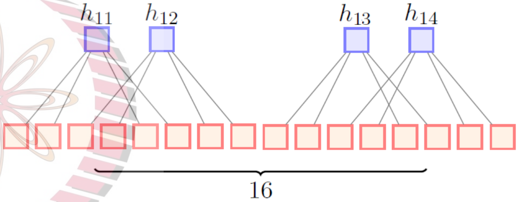
- Only a few local neurons participate in computation of h_{11}
- E.g. only pixels 1, 2, 5, 6 contribute to h_{11}

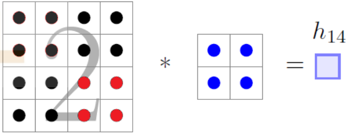




Local Receptive Fields

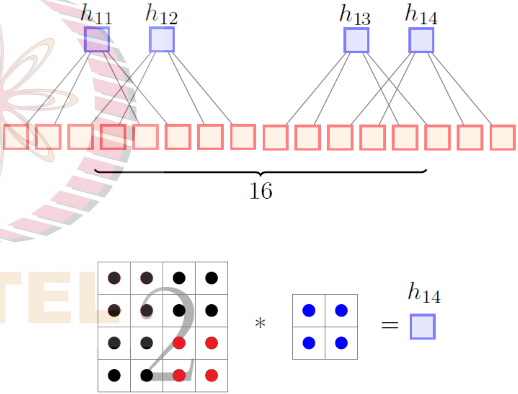
- Only a few local neurons participate in computation of h_{11}
- E.g. only pixels 1, 2, 5, 6 contribute to h_{11}
- Similar for other pixels





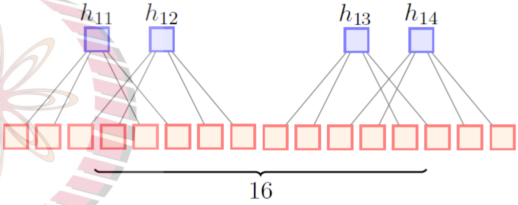
Local Receptive Fields

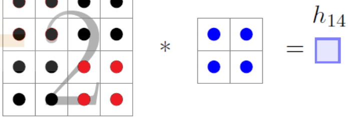
- Only a few local neurons participate in computation of h_{11}
- E.g. only pixels 1, 2, 5, 6 contribute to h_{11}
- Similar for other pixels
- The connections are much sparser
- This **sparse connectivity** reduces the number of parameters in the model



Local Receptive Fields

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





Local Receptive Fields

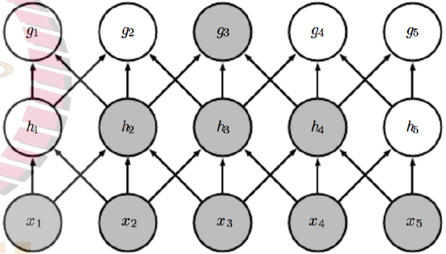
- But is sparse connectivity really a good thing?

Local Receptive Fields

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

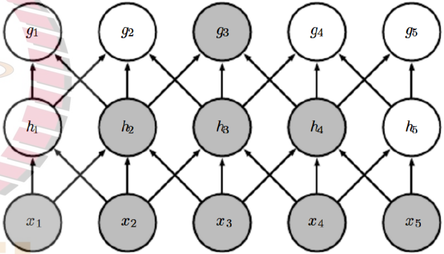
Local Receptive Fields

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



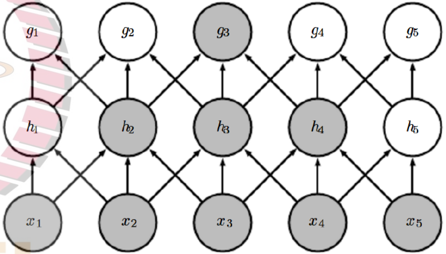
Local Receptive Fields

- But is sparse connectivity really a good thing?
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really
- The two highlighted neurons ($x_1 x_5$) do not interact in layer 1



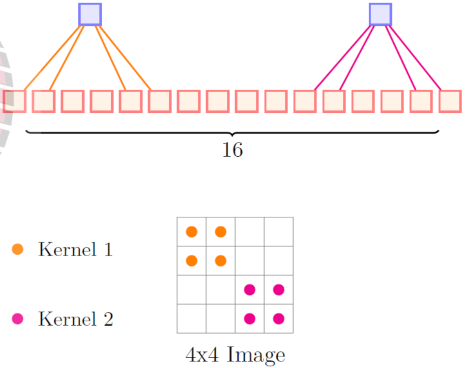
Local Receptive Fields

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



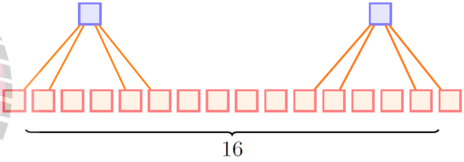
Weight Sharing

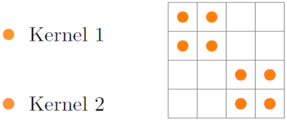
- Consider the following network; do we want the kernel weights to be different for different parts of the image?



Weight Sharing

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

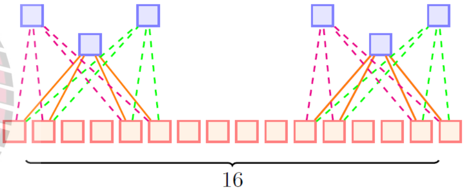




4x4 Image

Weight Sharing

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

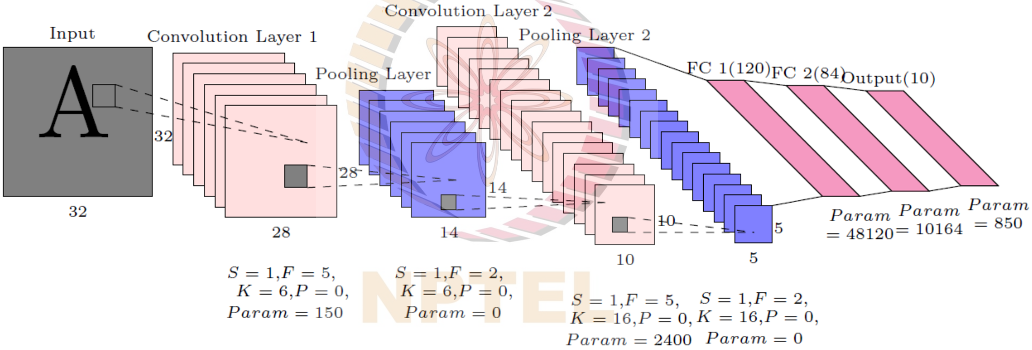


Convolutional Neural Network

- A typical CNN looks as follows:

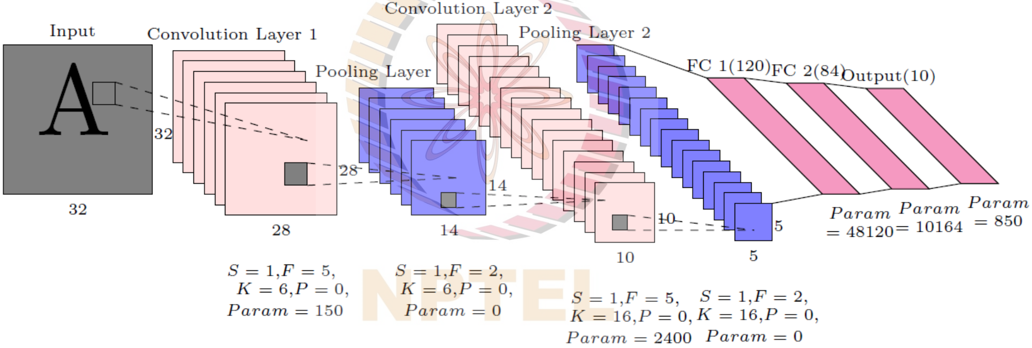
Convolutional Neural Network

- A typical CNN looks as follows:



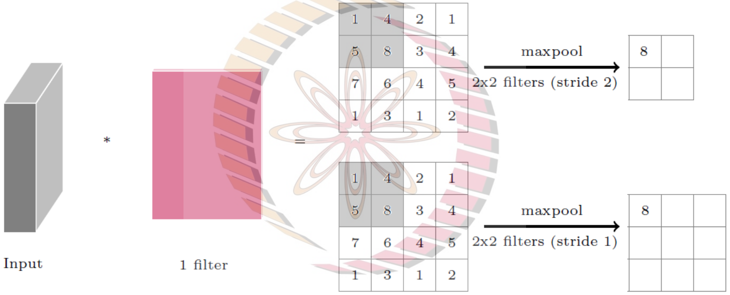
Convolutional Neural Network

- A typical CNN looks as follows:



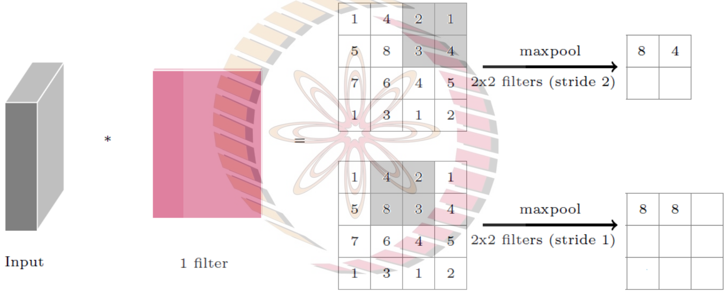
- It has alternate convolution and pooling layers
- What do pooling layers do?

Pooling Layer



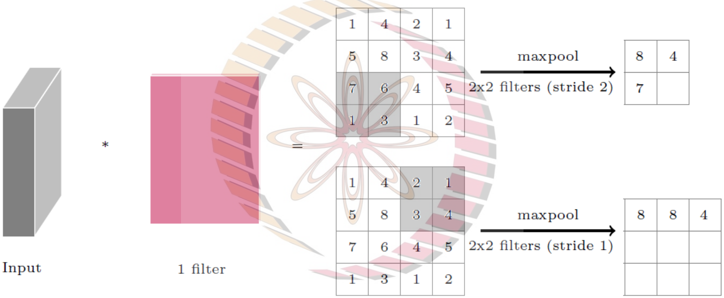
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



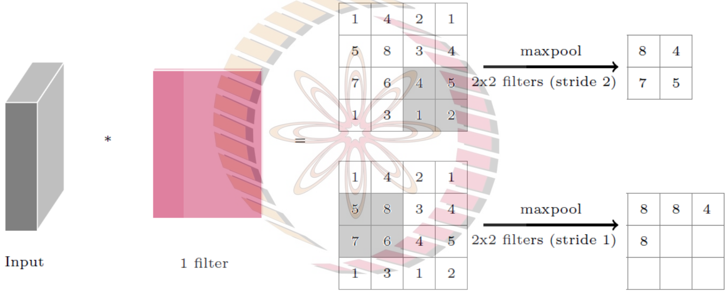
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



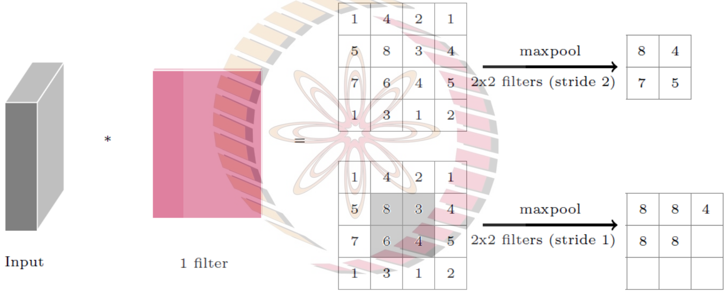
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



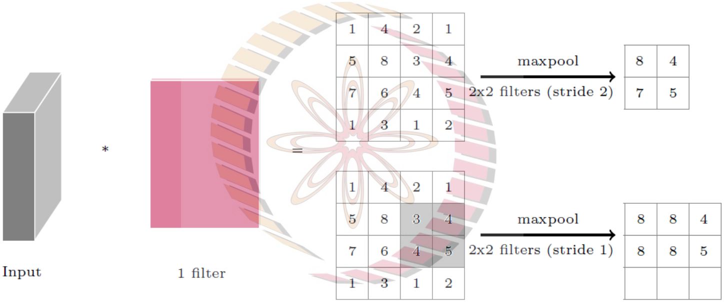
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



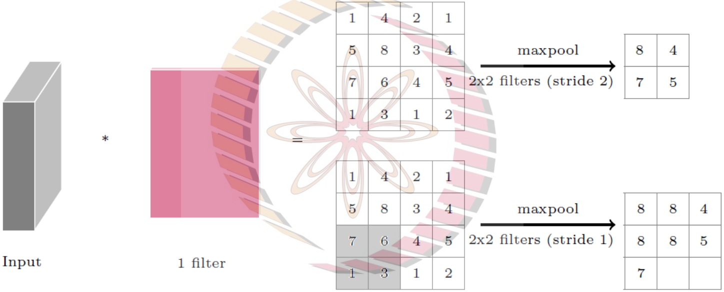
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



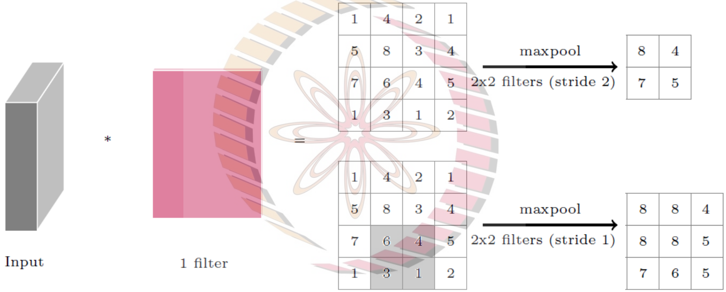
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



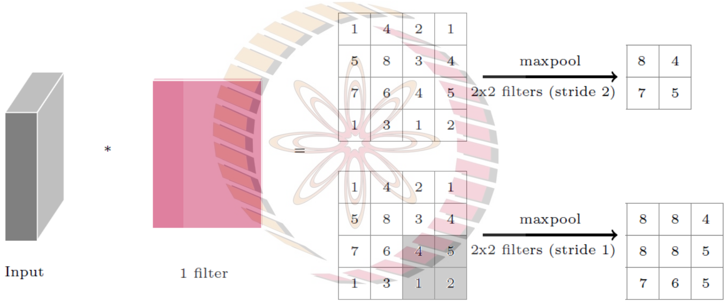
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



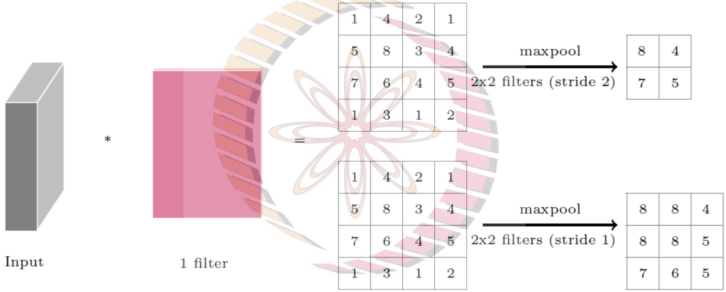
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



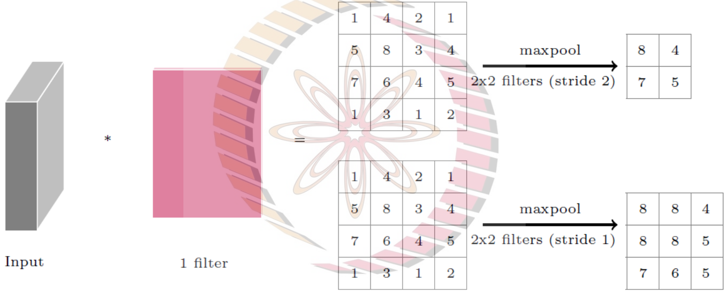
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



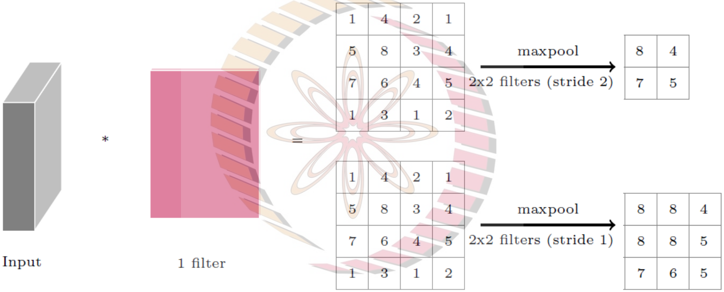
- **Pooling** is a parameter-free down sampling operation

Pooling Layer



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

Pooling Layer



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

Other Variants of Convolution: Dilated Convolution

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

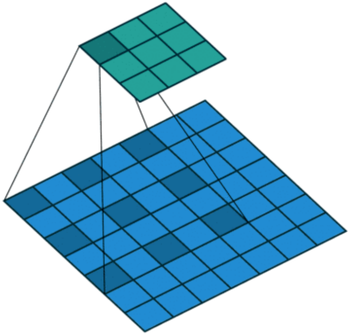


Image Credit: Vincent Dumoulin

Other Variants of Convolution: Dilated Convolution

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

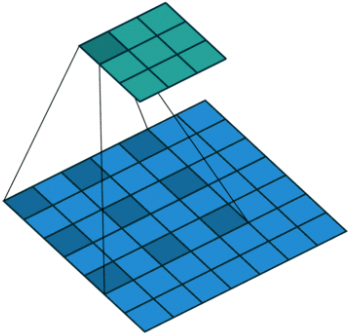


Image Credit: Vincent Dumoulin

Other Variants of Convolution: Dilated Convolution

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

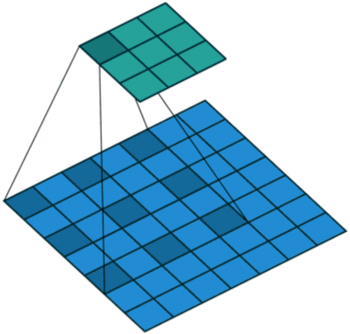


Image Credit: Vincent Dumoulin

Other Variants of Convolution: Dilated Convolution

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

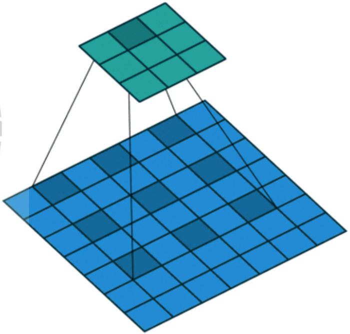


Image Credit: Vincent Dumoulin

Other Variants of Convolution: Dilated Convolution

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

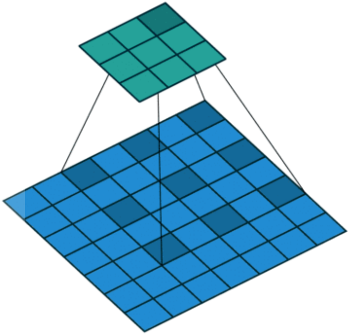


Image Credit: Vincent Dumoulin

Other Variants of Convolution: Dilated Convolution

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

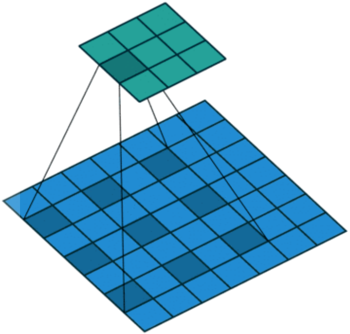


Image Credit: Vincent Dumoulin

Other Variants of Convolution: Dilated Convolution

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

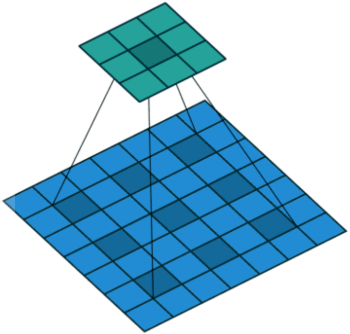


Image Credit: Vincent Dumoulin

Other Variants of Convolution: Dilated Convolution

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

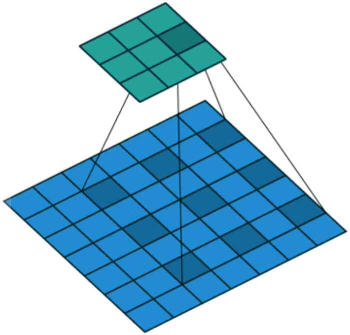


Image Credit: Vincent Dumoulin

Other Variants of Convolution: Dilated Convolution

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

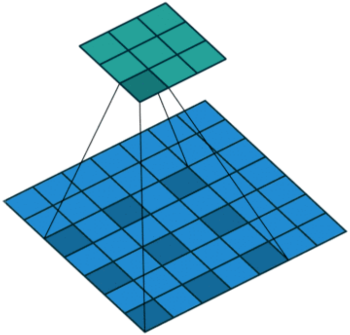


Image Credit: Vincent Dumoulin

Other Variants of Convolution: Dilated Convolution

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

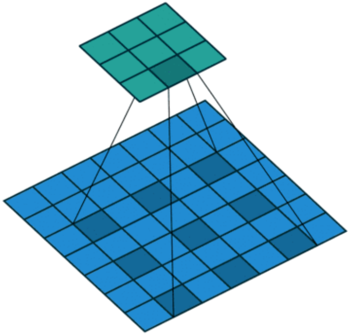


Image Credit: Vincent Dumoulin

Other Variants of Convolution: Dilated Convolution

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

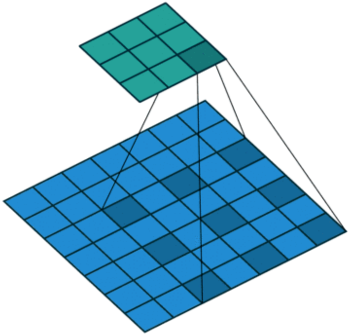


Image Credit: Vincent Dumoulin

Other Variants of Convolution: Dilated Convolution

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

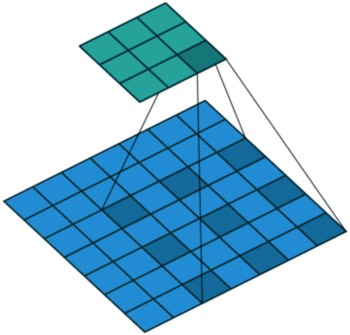


Image Credit: Vincent Dumoulin

Other Variants of Convolution: Dilated Convolution

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

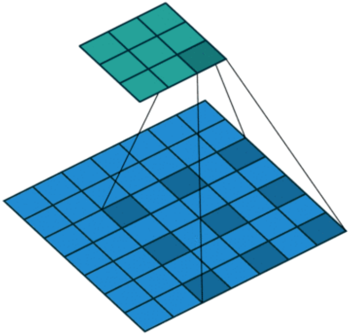


Image Credit: Vincent Dumoulin

Other Variants of Convolution: Transpose Convolution

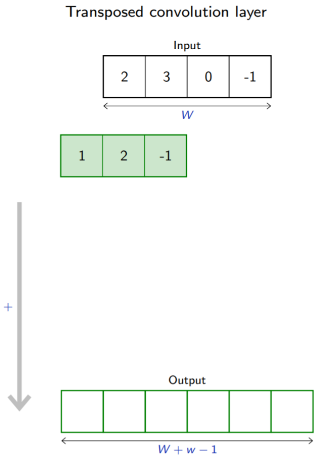
- Allows for learnable upsampling
- Also known as Deconvolution (bad) or Upconvolution

Other Variants of Convolution: Transpose Convolution

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

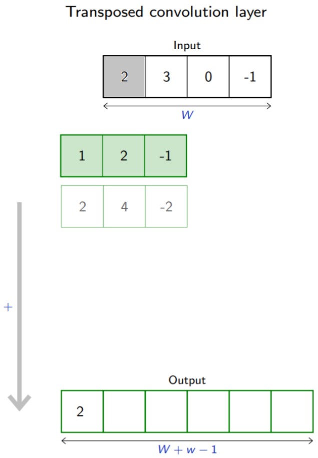
Other Variants of Convolution: Transpose Convolution

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



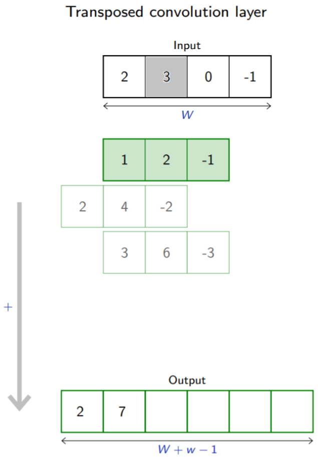
Other Variants of Convolution: Transpose Convolution

- 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

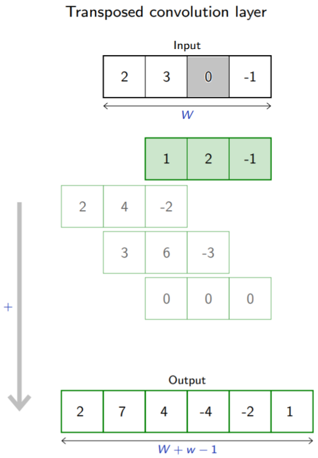
- Allows for learnable upsampling
- Also known as Deconvolution (bad) or Upconvolution
- Traditionally, we could achieve upsampling through interpolation or similar rules
- Why not allow the network to learn the rules by itself?
- Let us see a 1D example



Credit: Francois Fleuret

Other Variants of Convolution: Transpose Convolution

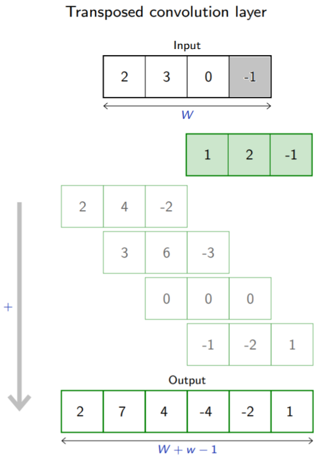
- Allows for learnable upsampling
- Also known as Deconvolution (bad) or Upconvolution
- Traditionally, we could achieve upsampling through interpolation or similar rules
- Why not allow the network to learn the rules by itself?
- Let us see a 1D example



Credit: Francois Fleuret

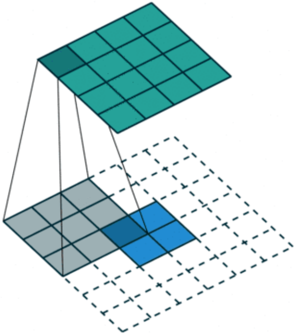
Other Variants of Convolution: Transpose Convolution

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

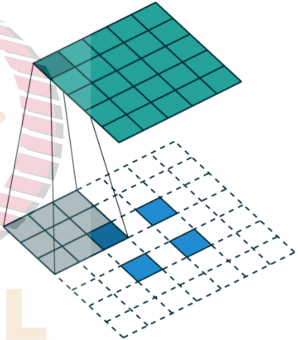


Credit: Francois Fleuret

Other Variants of Convolution: Transpose Convolution



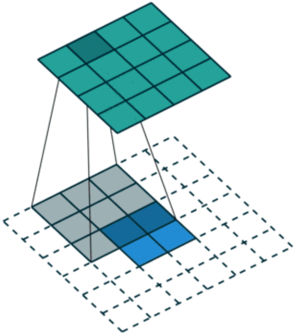
Upsampling 2 × 2 input to a 4 × 4 output



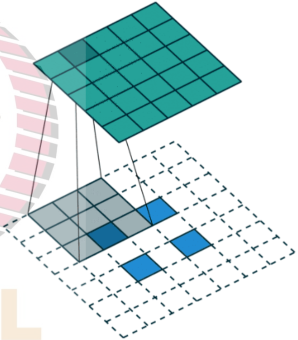
Upsampling 2 × 2 input to a 5 × 5 output

GIF Credit: *Vincent Dumoulin*

Other Variants of Convolution: Transpose Convolution



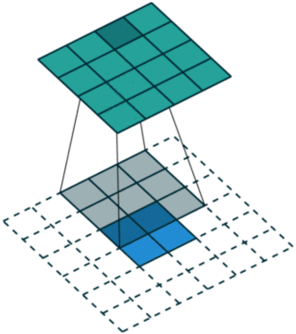
Upsampling 2 × 2 input to a 4 × 4 output



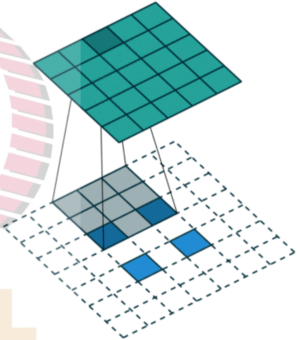
Upsampling 2 × 2 input to a 5 × 5 output

GIF Credit: Vincent Dumoulin

Other Variants of Convolution: Transpose Convolution



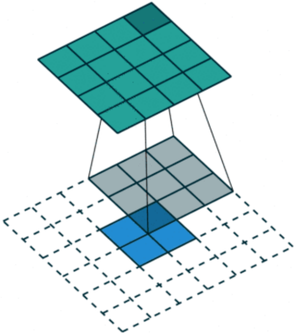
Upsampling 2 × 2 input to a 4 × 4 output



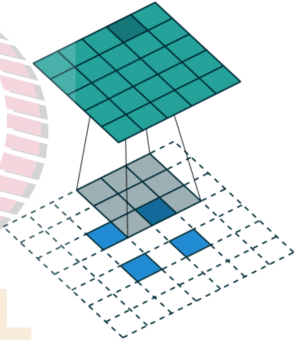
Upsampling 2 × 2 input to a 5 × 5 output

GIF Credit: *Vincent Dumoulin*

Other Variants of Convolution: Transpose Convolution



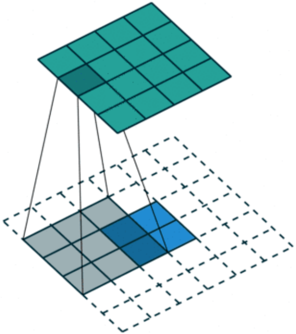
Upsampling 2×2 input to a 4×4 output



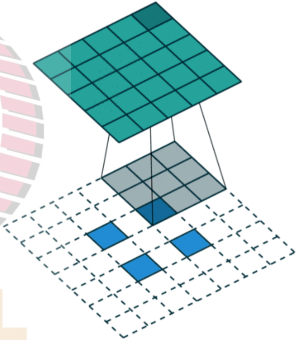
Upsampling 2×2 input to a 5×5 output

GIF Credit: Vincent Dumoulin

Other Variants of Convolution: Transpose Convolution



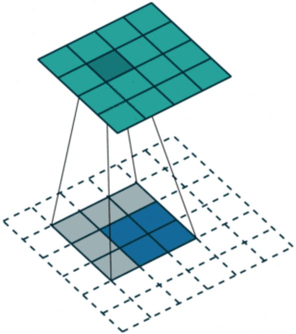
Upsampling 2 × 2 input to a 4 × 4 output



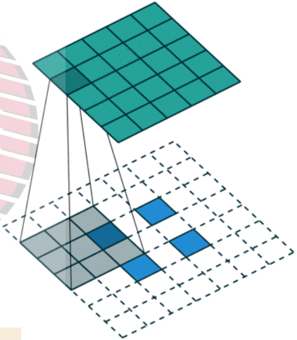
Upsampling 2 × 2 input to a 5 × 5 output

GIF Credit: *Vincent Dumoulin*

Other Variants of Convolution: Transpose Convolution



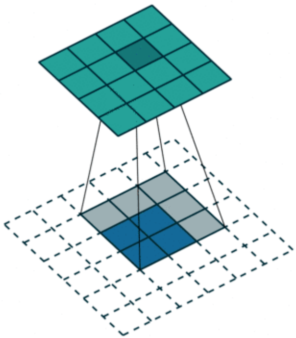
Upsampling 2 × 2 input to a 4 × 4 output

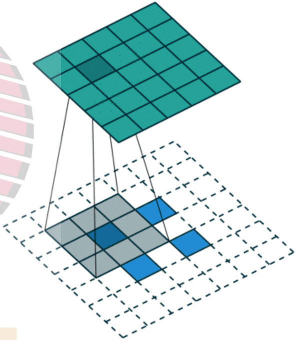


Upsampling 2 × 2 input to a 5 × 5 output

GIF Credit: Vincent Dumoulin

Other Variants of Convolution: Transpose Convolution

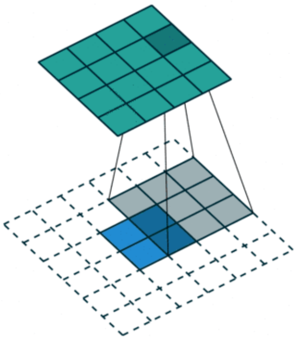


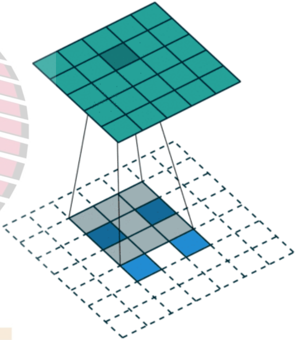


Upsampling 2 × 2 input to a 4 × 4 output   Upsampling 2 × 2 input to a 5 × 5 output

GIF Credit: Vincent Dumoulin

Other Variants of Convolution: Transpose Convolution



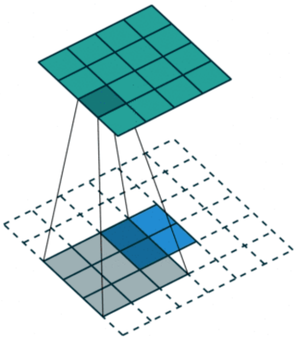


Upsampling 2 $\times$ 2 input to a 4 $\times$ 4 output

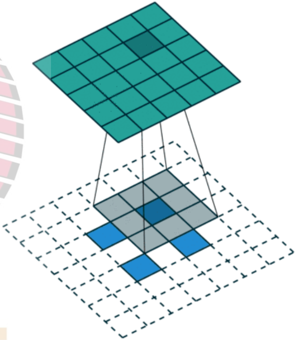
Upsampling 2 $\times$ 2 input to a 5 $\times$ 5 output

GIF Credit: *Vincent Dumoulin*

Other Variants of Convolution: Transpose Convolution



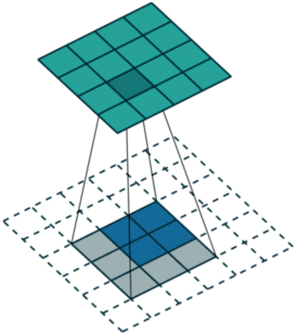
Upsampling 2 × 2 input to a 4 × 4 output



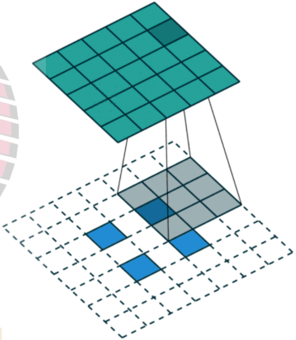
Upsampling 2 × 2 input to a 5 × 5 output

GIF Credit: Vincent Dumoulin

Other Variants of Convolution: Transpose Convolution



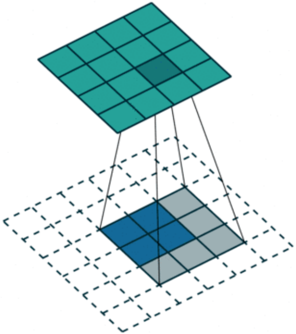
Upsampling 2 $\times$ 2 input to a 4 $\times$ 4 output



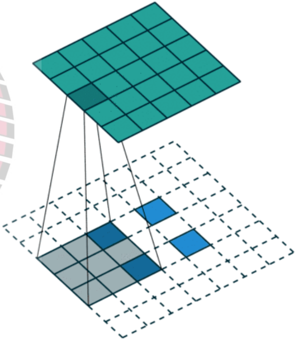
Upsampling 2 $\times$ 2 input to a 5 $\times$ 5 output

GIF Credit: Vincent Dumoulin

Other Variants of Convolution: Transpose Convolution



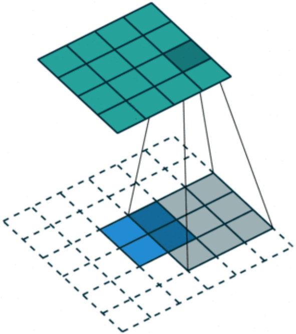
Upsampling 2 $\times$ 2 input to a 4 $\times$ 4 output



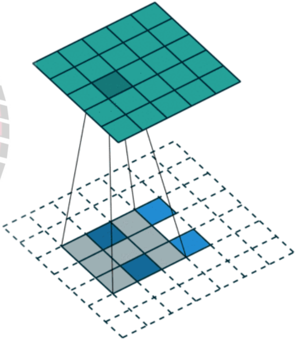
Upsampling 2 $\times$ 2 input to a 5 $\times$ 5 output

GIF Credit: *Vincent Dumoulin*

Other Variants of Convolution: Transpose Convolution



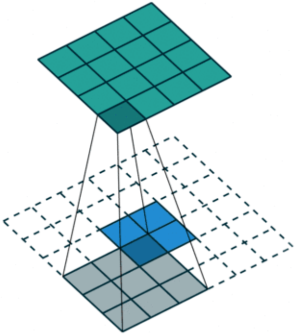
Upsampling 2×2 input to a 4×4 output

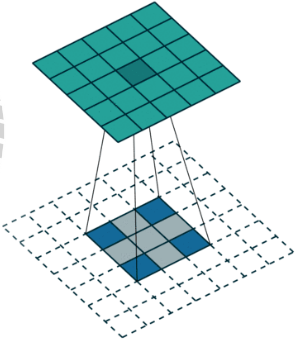


Upsampling 2×2 input to a 5×5 output

GIF Credit: Vincent Dumoulin

Other Variants of Convolution: Transpose Convolution



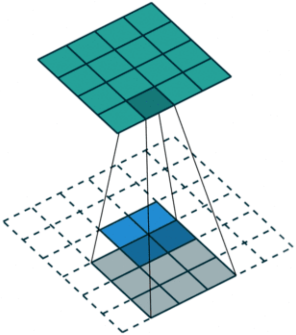


Upsampling 2×2 input to a 4×4 output

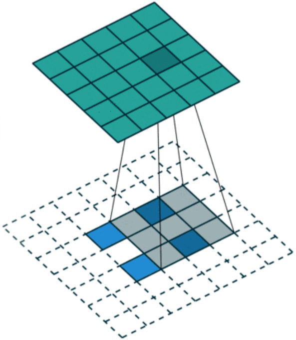
Upsampling 2×2 input to a 5×5 output

GIF Credit: Vincent Dumoulin

Other Variants of Convolution: Transpose Convolution



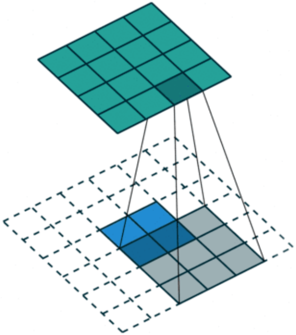
Upsampling 2 × 2 input to a 4 × 4 output



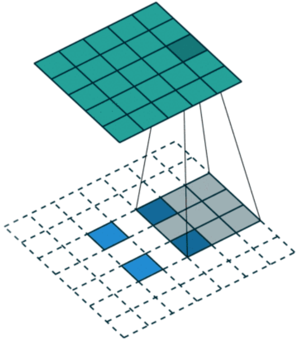
Upsampling 2 × 2 input to a 5 × 5 output

GIF Credit: Vincent Dumoulin

Other Variants of Convolution: Transpose Convolution



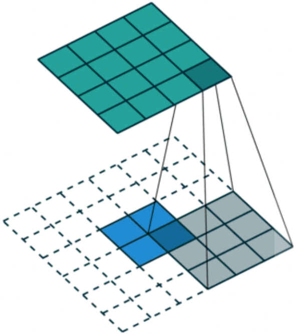
Upsampling 2 × 2 input to a 4 × 4 output



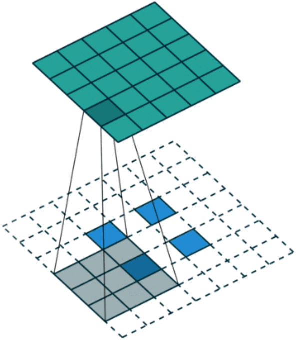
Upsampling 2 × 2 input to a 5 × 5 output

GIF Credit: Vincent Dumoulin

Other Variants of Convolution: Transpose Convolution



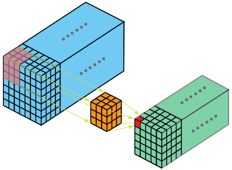
Upsampling 2 × 2 input to a 4 × 4 output



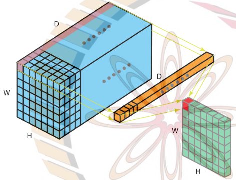
Upsampling 2 × 2 input to a 5 × 5 output

GIF Credit: *Vincent Dumoulin*

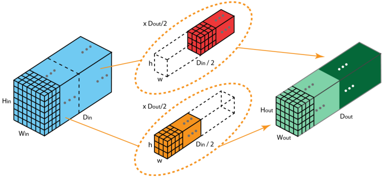
Other Variants of Convolution



3D Convolution



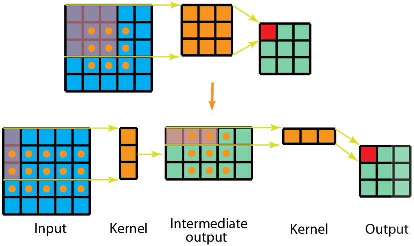
1 × 1 **Convolution**
Pointwise Convolution



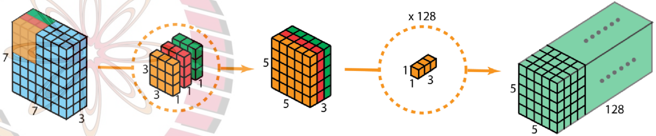
Grouped Convolution

Credit: Illarion Khlestov, Chi-Feng Wang

Other Variants of Convolution



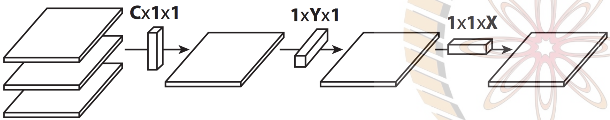
Spatial Separable Convolution



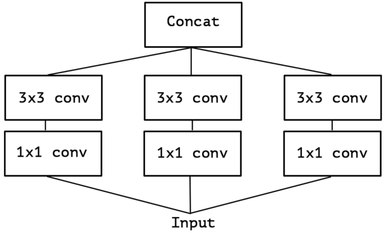
Depthwise Separable Convolution

Credit: Chi-Feng Wang

Other Variants of Convolutions



Flattened Convolutions



Spatial and Cross-Channel Convolutions

Credit: Illarion Khlestov

Homework

Readings

- For an interactive illustration of the convolution operation, visit `https://setosa.io/ev/image-kernels/`
- Read more about deconvolution operation at Distill
- Other good resources:
  - 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?

References

[1] David Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision* 60 (Nov. 2004), pp. 91–110.

[2] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)* 1 (2005), 886–893 vol. 1.

[3] Svetlana Lazebnik, Cordelia Schmid, and J. Ponce. “Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories”. In: vol. 2. Feb. 2006, pp. 2169 –2178.

[4] Kaiming He et al. “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition”. In: *Lecture Notes in Computer Science* (2014), 346–361.

[5] Dingjun Yu et al. “Mixed Pooling for Convolutional Neural Networks”. In: Oct. 2014, pp. 364–375.

[6] Oren Rippel, Jasper Snoek, and Ryan P. Adams. “Spectral Representations for Convolutional Neural Networks”. In: *NIPS*. 2015.

[7] Nal Kalchbrenner et al. “Neural Machine Translation in Linear Time”. In: *ArXiv* abs/1610.10099 (2016).