CNN Architecture

ML Instruction Team, Fall 2022

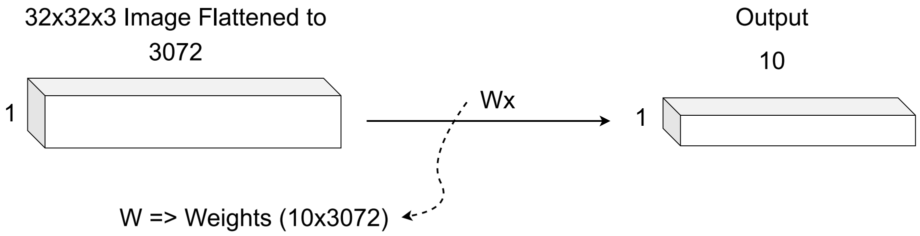
CE Department
Sharif University of Technology

Ali Sharifi-Zarchi
Behrooz Azarkhalili
Arian Amani
Hamidreza Yaghoubi

CNNs

What we've been using:

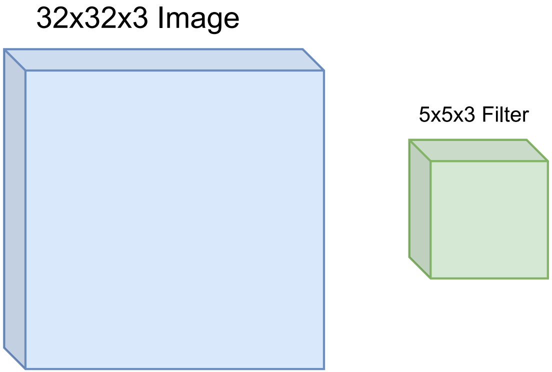
- Fully Connected Layers



CNNs

What we're going to learn:

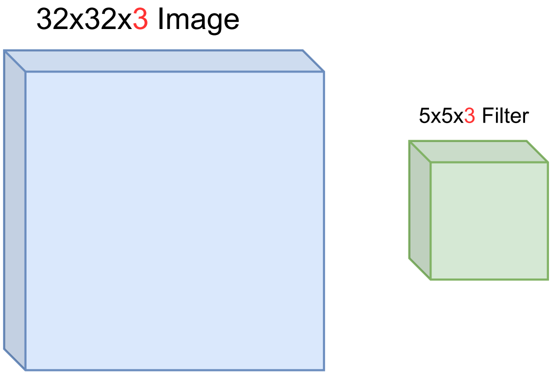
- Convolutional Layer



CNNs

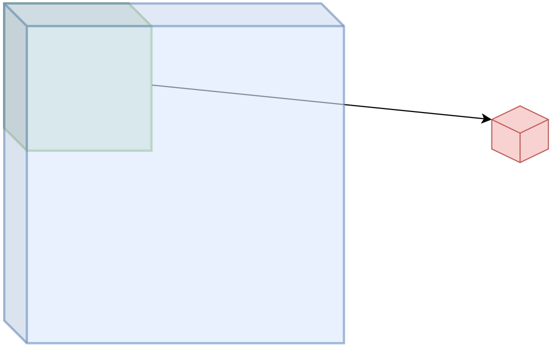
Convolutional Layer

- Filters always extend the full depth of the input volume. (#Input channels == #Filter Channels)



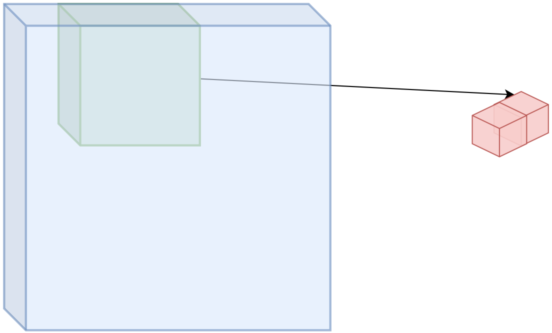
What is Convolution and how does it work?

- We apply the same filter on different regions of the input



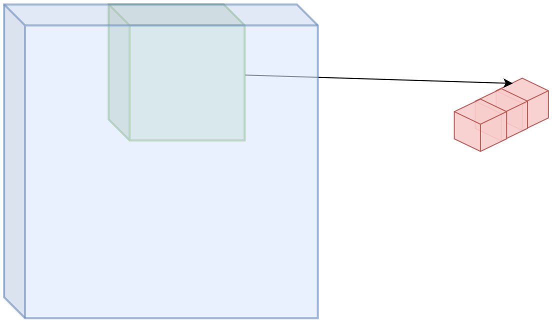
What is Convolution and how does it work?

- We apply the same filter on different regions of the input
- Convolutional filters learn to make decisions based on local spatial input which is an important attribute working with images



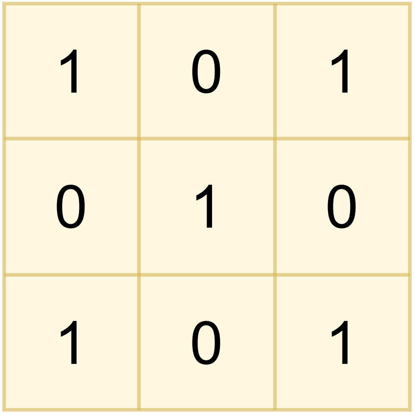
What is Convolution and how does it work?

- We apply the same filter on different regions of the input
- Convolutional filters learn to make decisions based on local spatial input which is an important attribute working with images.
- Uses a lot fewer parameters compared to Fully Connected Layers.



What is Convolution and how does it work?

Consider this, the filter we are going to use:



1	0	1
0	1	0
1	0	1

What is Convolution and how does it work?

This is how we calculate the convolutional layer's output:

$$ConvolvedFeature(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image
K: Our Kernel
k_h and k_w: The height and width of the Kernel

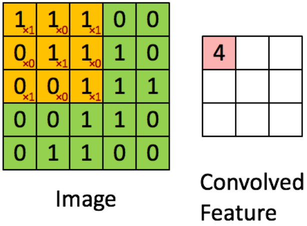


Figure: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature, Source

What is Convolution and how does it work?

This is how we calculate the convolutional layer's output:

$$ConvolvedFeature(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1}\sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image
K: Our Kernel
k_h and k_w: The height and width of the Kernel

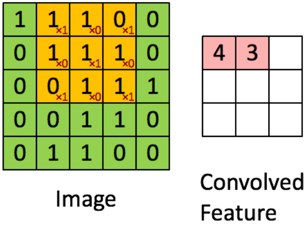


Figure: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature, Source

What is Convolution and how does it work?

This is how we calculate the convolutional layer's output:

$$ConvolvedFeature(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image
K: Our Kernel
k_h and k_w: The height and width of the Kernel

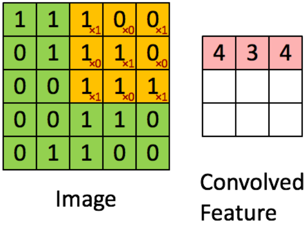


Figure: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature, Source

What is Convolution and how does it work?

This is how we calculate the convolutional layer's output:

$$ConvolvedFeature(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image
K: Our Kernel
k_h and k_w: The height and width of the Kernel

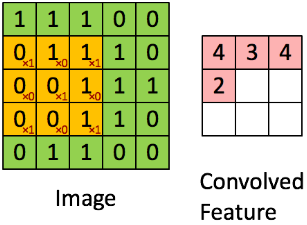


Figure: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature, Source

What is Convolution and how does it work?

This is how we calculate the convolutional layer's output:

$$ConvolvedFeature(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image
K: Our Kernel
k_h and k_w: The height and width of the Kernel

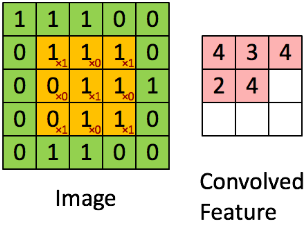


Figure: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature, Source

What is Convolution and how does it work?

This is how we calculate the convolutional layer's output:

$$ConvolvedFeature(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image
K: Our Kernel
k_h and k_w: The height and width of the Kernel

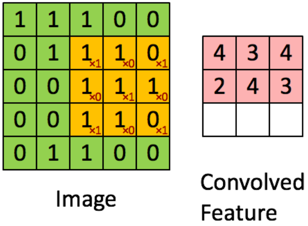


Figure: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature, Source

What is Convolution and how does it work?

This is how we calculate the convolutional layer's output:

$$ConvolvedFeature(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image
K: Our Kernel
k_h and k_w: The height and width of the Kernel

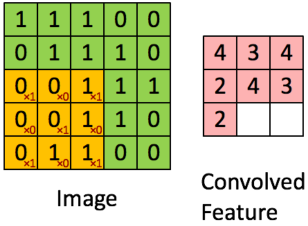


Figure: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature, Source

What is Convolution and how does it work?

This is how we calculate the convolutional layer's output:

$$ConvolvedFeature(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image
K: Our Kernel
k_h and k_w: The height and width of the Kernel

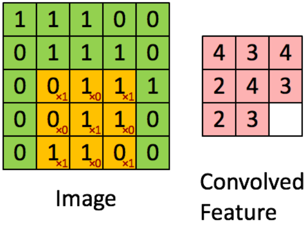


Figure: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature, Source

What is Convolution and how does it work?

This is how we calculate the convolutional layer's output:

$$ConvolvedFeature(i,j) = (I * K)(i,j) = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} I(i+a, j+b)K(a,b)$$

I: Input Image
K: Our Kernel
k_h and k_w: The height and width of the Kernel

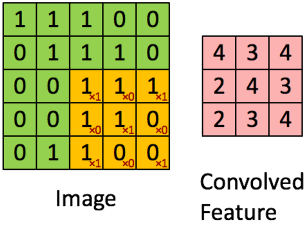


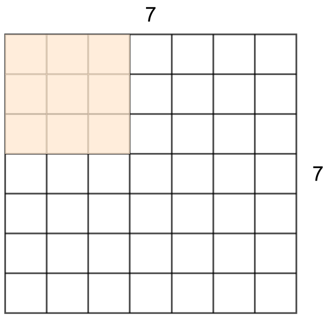
Figure: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature, Source

What is a Stride

The amount of movement between applications of the filter to the input image is referred to as the stride, and it is almost always symmetrical in height and width dimensions.

Closer look

- 7x7 input with 3x3 filter

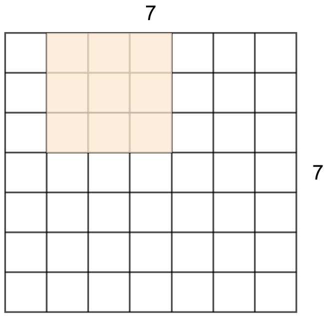


What is a Stride

The amount of movement between applications of the filter to the input image is referred to as the stride, and it is almost always symmetrical in height and width dimensions.

Closer look

- 7x7 input with 3x3 filter

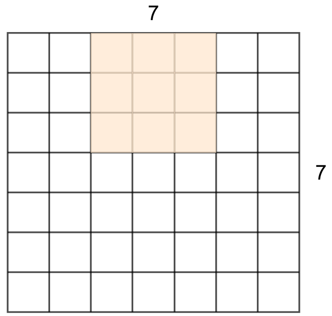


What is a Stride

The amount of movement between applications of the filter to the input image is referred to as the stride, and it is almost always symmetrical in height and width dimensions.

Closer look

- 7x7 input with 3x3 filter

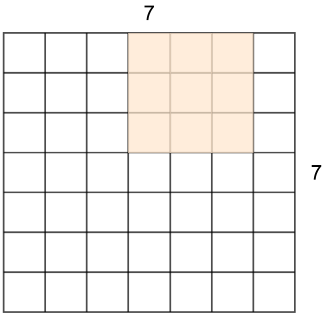


What is a Stride

The amount of movement between applications of the filter to the input image is referred to as the stride, and it is almost always symmetrical in height and width dimensions.

Closer look

- 7x7 input with 3x3 filter

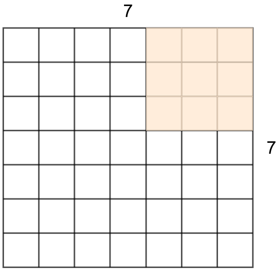


What is a Stride

The amount of movement between applications of the filter to the input image is referred to as the stride, and it is almost always symmetrical in height and width dimensions.

Closer look

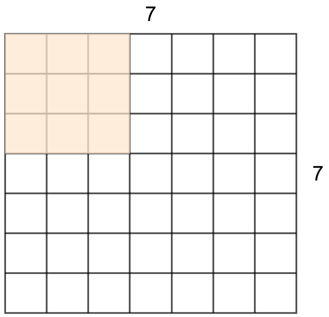
- 7x7 input with 3x3 filter
- This was a Stride 1 filter
- => Outputs 5x5



Strides

Now let's use Stride 2

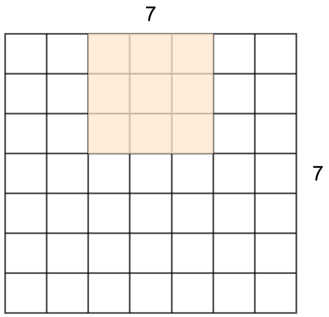
- 7x7 input with 3x3 filter



Strides

Now let's use Stride 2

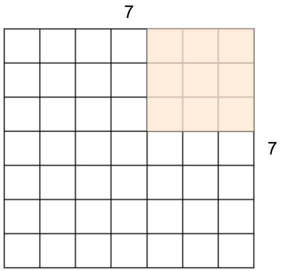
- 7x7 input with 3x3 filter



Strides

Now let's use Stride 2

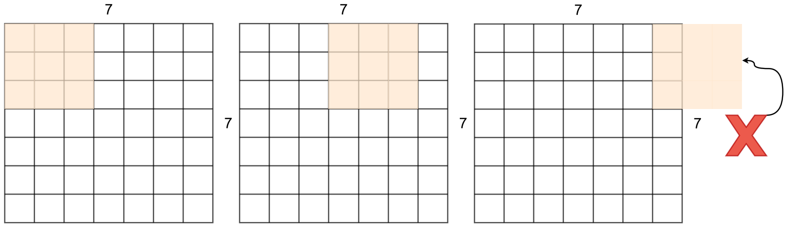
- 7x7 input with 3x3 filter
- This was a Stride 2 filter
- => Outputs 3x3



Strides

Stride 3?

- 7x7 input with 3x3 filter

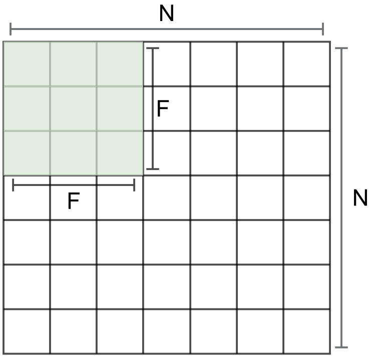


Strides

So 7x7 input with 3x3 filter and stride 3 doens't work!

Let't do the calculations:

$$OutputSize = (N - F)/Stride + 1$$

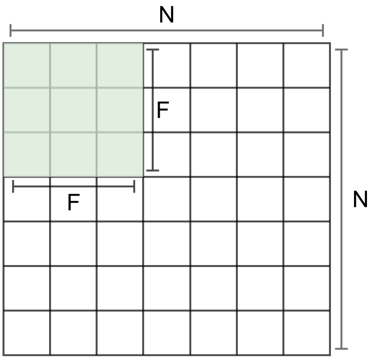


Strides

$OutputSize = (N - F)/Stride + 1$

N = 7, F = 3 =>

- Stride 1 ==> $(7 - 3)/1 + 1 = 5$
- Stride 2 ==> $(7 - 3)/2 + 1 = 3$
- Stride 3 ==> $(7 - 3)/3 + 1 = 2.33$:))



Strides

Do you see any problems?

Strides

- 1. Borders don't get enough attention.

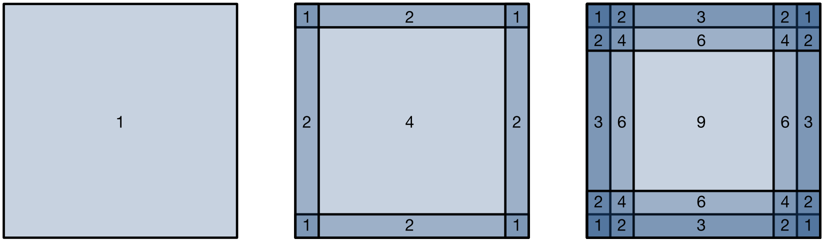


Figure: Dive into Deep Learning, Fig. 7.3.1: Pixel utilization for convolutions of size 1×1, 2×2, and 3×3 respectively.

Strides

- 1. Borders don't get enough attention.
- 2. Outputs shrink!

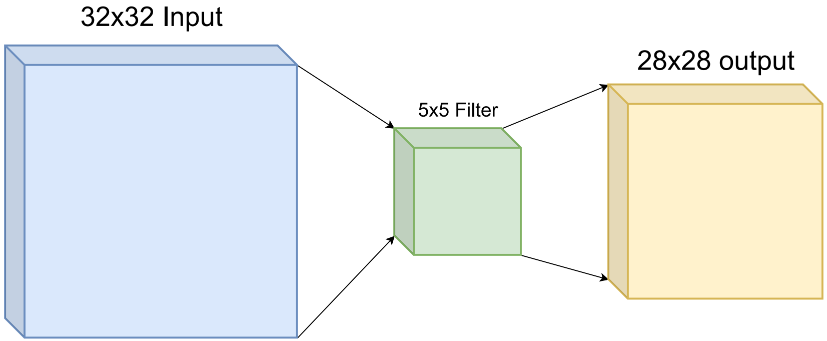


Figure: 32x32 input shrinks to 28x28 output. (information loss)

Strides

What is the solution?

Padding

Enters Padding

Padding

- We can use Padding to preserve the dimensionality
- It ensures that all pixels are used equally frequently

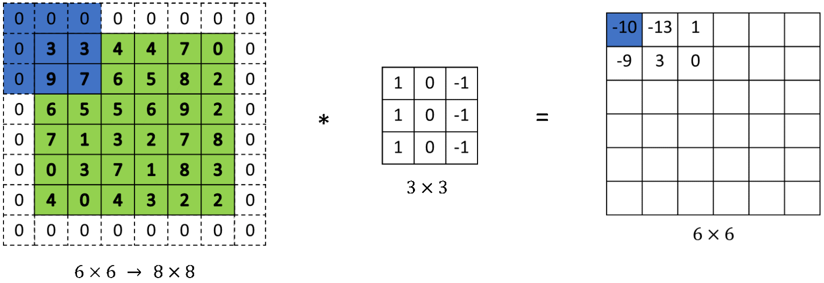


Figure: DataHacker.rs: CNN Padding - Applying padding of 1 before convolving with 3×3 filter

Padding

- It is common to use $P = (F - 1)/2$ with stride 1 to preserve the input size.

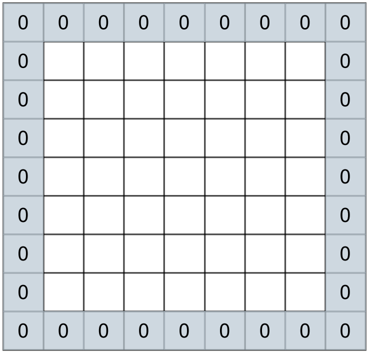


Figure: Applying zero-padding to a 7x7 input with padding 1

Padding

- It is common to use $P = (F - 1)/2$ with stride 1 to preserve the input size.
- $OutputSize = (N + 2P - F)/Stride + 1$

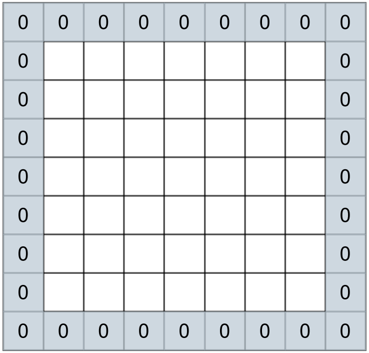


Figure: Applying zero-padding to a 7x7 input with padding 1

Padding

- It is common to use $P = (F-1)/2$ with stride 1 to preserve the input size.
- $OutputSize = (N + 2P - F)/Stride + 1$

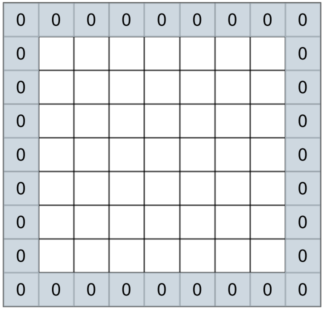


Figure: Applying zero-padding to a 7x7 input with padding 1

7x7 input, stride 1, P to preserve the dimentions?

- $F = 3 --> P = 1$
- $F = 5 --> P = 2$
- $F = 7 --> P = 3$

Pooling

- Pooling is performed in neural networks to reduce variance and computation complexity

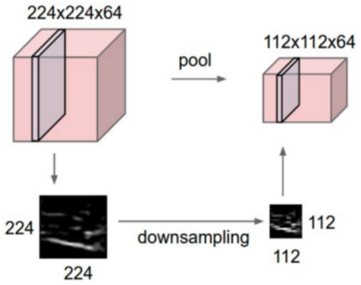


Figure: Pooling Layer

Pooling: How It Works

- Polling layers aggregate the inputs using an aggregation function such as the max or mean.

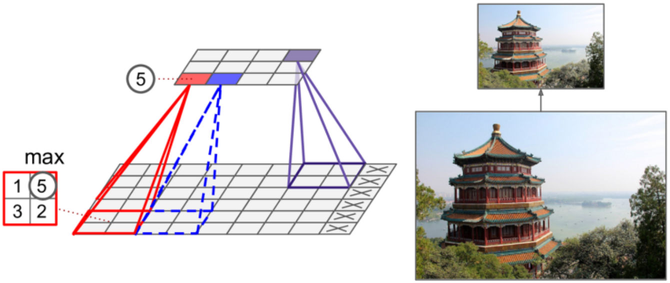


Figure: Max pooling layer (2 × 2 pooling kernel, stride 2, no padding)

Pooling: Benefits

- Polling layers goal is to subsample the input on order to Reduce:
  - The computational load
  - The memory usage
  - The number of parameters
  - Limiting the risk of overfitting

Pooling: Types

- Max Pooling: give a better result when the background is white and the object is dark
- Min Pooling: give a better result when the background is dark and the object is white
- Average Pooling: smoothes the picture

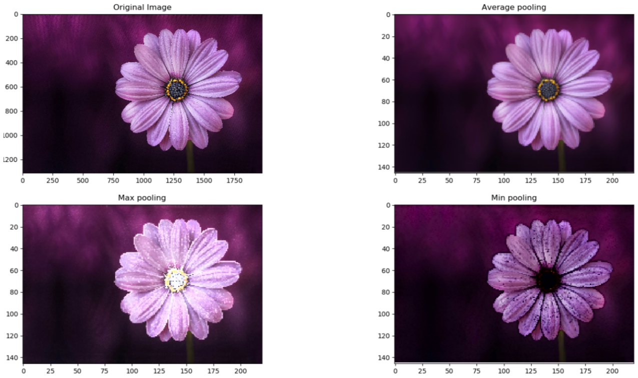


Figure: Different effects of different pooling layers

Pooling: Max Pooling

- Most common type of pooling layer
- Invariance to small translations

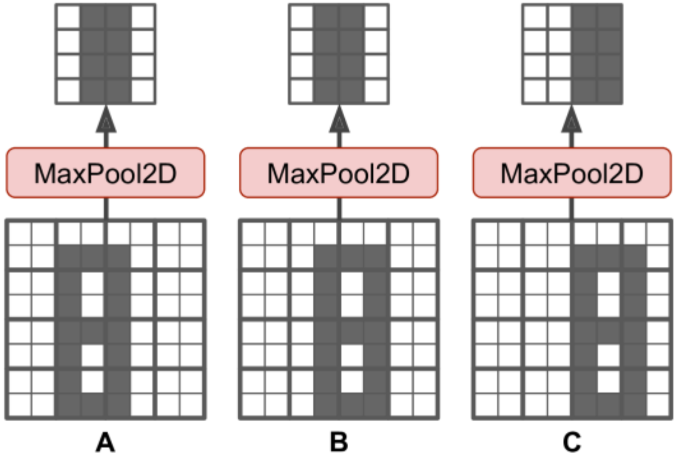


Figure: Invariance to small translations

Dilation

- It is a technique that expands the kernel (input) by inserting holes between its consecutive elements

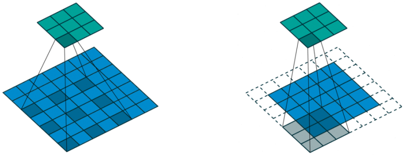


Figure: Dilated Convolution ($l = 2$) vs. Standard Convolution ($l = 1$)

Dilation: Benefits

- Since dilated convolutions support exponential expansion in the context of the receptive field, there is no loss of resolution.
- Dilated convolutions use ’l’ as the parameter for the dilation rate. As we increase the value of ’l’ it allows one to have a larger receptive field which is really helpful as we are able to view more data points thus saving computation and memory costs

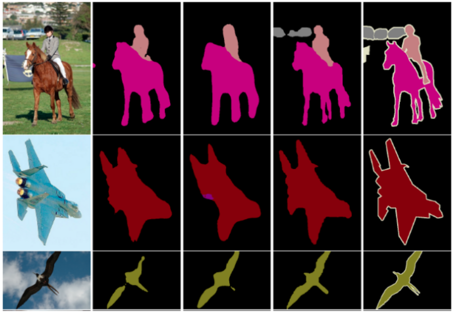


Figure: Image - FCN-8s - DeepLab - DilatedNet - Ground Truth

Channels, etc.

By now, we talked about 2D inputs, but images are 3D:

- They consist of 3 channels: Red, Green and Blue (RGB images)
- Shape: (height x width x 3)

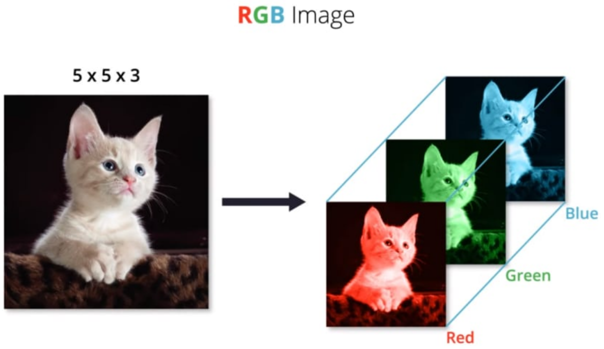
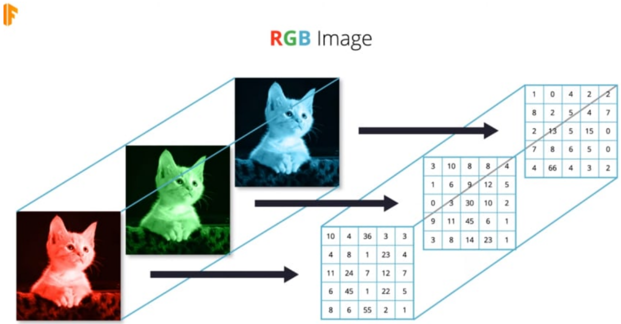


Figure: RGB Image, Source

Channels, etc.

By now, we talked about 2D inputs, but images are 3D:

- They consist of 3 channels: Red, Green and Blue (RGB images)
- Shape: (height x width x 3)



10	4	36	3	3
4	8	1	23	4
11	24	7	12	7
6	45	1	22	5
8	6	55	2	1

3	10	8	8	4
1	6	9	12	5
0	3	30	10	2
9	11	45	6	1
3	8	14	23	1

1	0	4	2	2
8	2	5	4	7
2	13	5	15	0
7	8	6	5	0
4	66	4	3	2

Figure: RGB Image, Source

Channels, etc.

By now, we talked about 2D inputs, but images are 3D:

- They consist of 3 channels: Red, Green and Blue (RGB images)
- Shape: (height x width x 3)

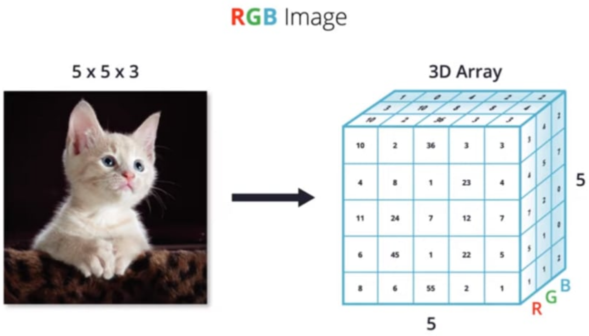


Figure: RGB Image, Source

Channels, etc.

- As said before: ”Filters always extend the full depth of the input volume. (#Input channels == #Filter Channels)”

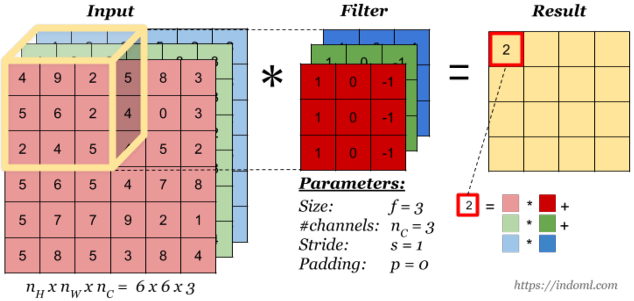


Figure: The total number of multiplications to calculate the result is (4 x 4) x (3 x 3 x 3) = 432, Source

Channels, etc.

- As said before: ”Filters always extend the full depth of the input volume. (#Input channels == #Filter Channels)”
- Each filter results in one output channel, we can apply multiple different filters to get multiple output channels. Each channel can learn something different.

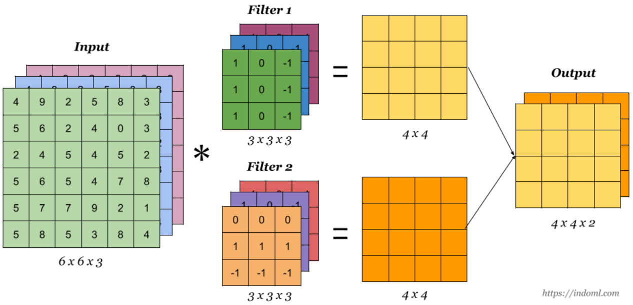


Figure: The total number of multiplications to calculate the result is (4 x 4 x 2) x (3 x 3 x 3) = 864, Source

Channels, etc.

- We apply several filters and stack the output together to get a multilayer output, with each layer representing something it learnt.

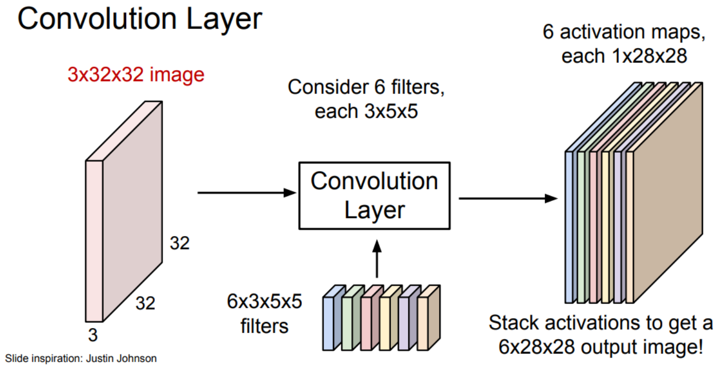

Convolution Layer
3x32x32 image
32
32
3
Consider 6 filters, each 3x5x5
Convolution Layer
6x3x5x5 filters
6 activation maps, each 1x28x28
Stack activations to get a 6x28x28 output image!
Slide inspiration: Justin Johnson

Figure: Slide from CS231n

Channels, etc.

- We apply several filters and stack the output together to get a multilayer output, with each layer representing something it learnt.
- Some kernels learning to recognize vertical lines, some circles, and some, specific objects:

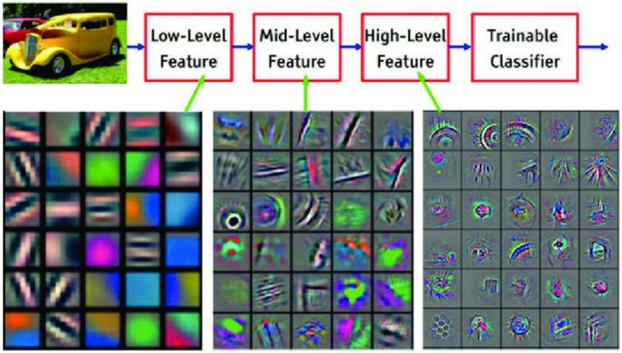


Figure: Each kernel learning something different, Source

Arithmetic of CNNs

We will focus on the following simplified setting:

- 2-D discrete convolutions ($N = 2$)
- Square inputs ($i_1 = i_2 = i$)
- Square kernel size ($k_1 = k_2 = k$)
- Same strides along both axes ($s_1 = s_2 = s$)
- Same zero padding along both axes ($p_1 = p_2 = p$)

Arithmetic of CNNs: No zero padding, unit strides

- For any i and k, and for $s = 1$ and $p = 0$:

$$o = (i - k) + 1 \tag{1}$$

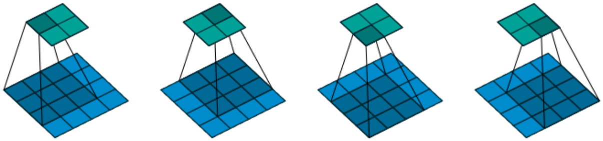


Figure: Convolving a 3 × 3 kernel over a 4 × 4 input using unit strides (i.e., i = 4, k = 3, s = 1 and p = 0)

Arithmetic of CNNs: Zero padding, unit strides

- For any i, k and p, and for $s = 1$:

$$o = (i - k) + 2p + 1 \tag{2}$$

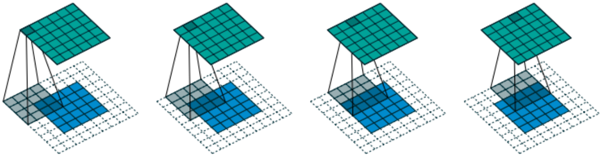


Figure: Convolving a 4 × 4 kernel over a 5 × 5 input padded with a 2 × 2 border of zeros using unit strides (i.e., i = 5, k = 4, s = 1 and p = 2)

Arithmetic of CNNs: Half (same) padding

- For any i and for k odd ($k = 2n + 1, n \in \mathbb{N}$), $s = 1$ and $p = \lfloor k/2 \rfloor = n$:

$$\begin{aligned} o &= i + 2\lfloor k/2 \rfloor - (k - 1) \\ &= i + 2n - 2n \\ &= i \end{aligned} \tag{3}$$

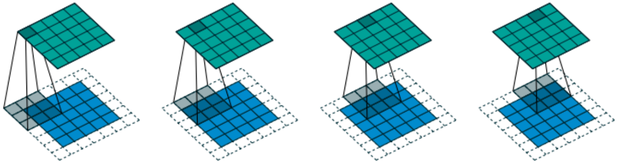


Figure: Convolving a 3 × 3 kernel over a 5 × 5 input using half padding and unit strides (i.e., i = 5, k = 3, s = 1 and p = 1)

Arithmetic of CNNs: Full padding

- For any i and k, and for $p = k - 1$ and $s = 1$:

$$\begin{aligned} o &= i + 2(k-1) - (k-1) \\ &= i + 2(k-1 \end{aligned} \tag{4}$$

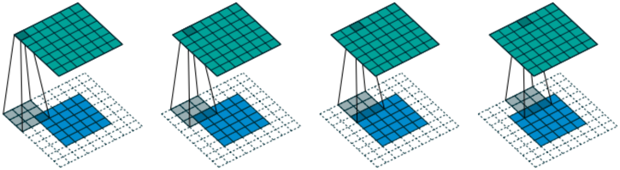


Figure: Convolving a 3 × 3 kernel over a 5 × 5 input using full padding and unit strides (i.e., i = 5, k = 3, s = 1 and p = 2)

Arithmetic of CNNs: No zero padding, non-unit strides

- For any i, k and s, and for $p = 0$:

$$o = \lfloor \frac{i-k}{s} \rfloor + 1 \tag{5}$$

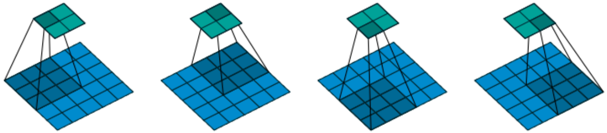


Figure: Convolving a 3 × 3 kernel over a 5 × 5 input using 2 × 2 strides (i.e., i = 5, k = 3, s = 2 and p = 0)

Arithmetic of CNNs: Zero padding, non-unit strides

- For any i, k and s, and for $p = 0$:

$$o = \lfloor \frac{i + 2p - k}{s} \rfloor + 1 \tag{6}$$

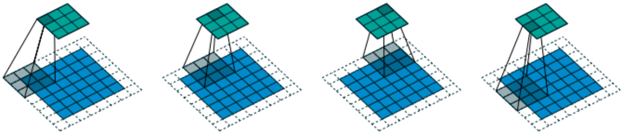


Figure: Convolving a 3 × 3 kernel over a 6 × 6 input padded with a 1 × 1 border of zeros using 2 × 2 strides (i.e., i = 6, k = 3, s = 2 and p = 1). In this case, the bottom row and right column of the zero padded input are not covered by the kernel

Upsample CNN

- Resizing feature maps is a common operation in many neural networks, especially those that perform some kind of image segmentation task
- This kind of architecture is famously known as the Encoder-Decoder network

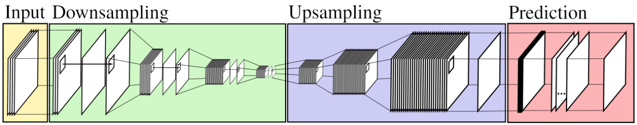


Figure: Schematic of the Downsampling and Upsampling

Upsample CNN: Techniques

- 1- Nearest Neighbors: In Nearest Neighbors, as the name suggests we take an input pixel value and copy it to the K-Nearest Neighbors where K depends on the expected output

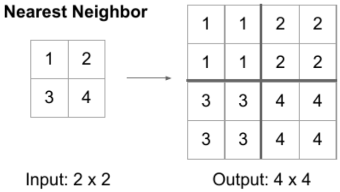


Figure: Nearest Neighbors Upsampling

Upsample CNN: Techniques

- 2- Bi-Linear Interpolation: In Bi-Linear Interpolation, we take the 4 nearest pixel value of the input pixel and perform a weighted average based on the distance of the four nearest cells smoothing the output

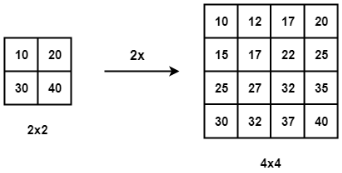


Figure: Bi-Linear Interpolation

Upsample CNN: Techniques

- 3- Bed Of Nails: In Bed of Nails, we copy the value of the input pixel at the corresponding position in the output image and filling zeros in the remaining positions.

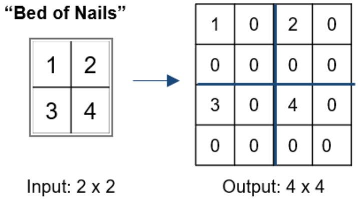


Figure: Bed of Nails Upsampling

Upsample CNN: Techniques

- 4- Max-Unpooling: To perform max-unpooling, first, the index of the maximum value is saved for every max-pooling layer during the encoding step. The saved index is then used during the Decoding step where the input pixel is mapped to the saved index, filling zeros everywhere else

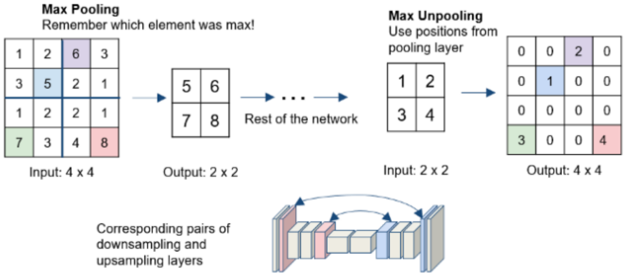


Figure: Max-Unpooling Upsampling

Transposed CNN

- Transposed Convolutions are used to upsample the input feature map to a desired output feature map using some learnable parameters
- They are the backbone of the modern segmentation and super-resolution algorithms
- They provide the best and most generalized upsampling of abstract representations

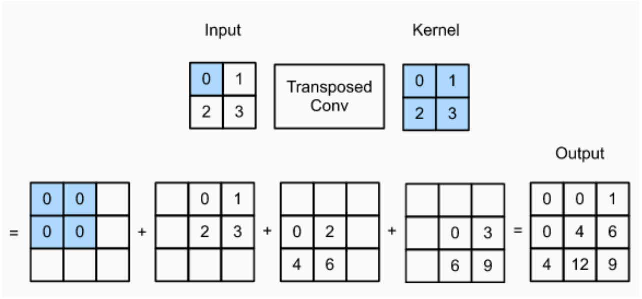


Figure: Transposed convolution with a 2×2 kernel. The shaded portions are a portion of an intermediate tensor as well as the input and kernel tensor elements used for the computation.

Transposed CNN: Problem

- Transposed convolutions suffer from chequered board effects as shown below. The main cause of this is uneven overlap at some parts of the image causing artifacts. This can be fixed or reduced by using kernel-size divisible by the stride, for e.g taking a kernel size of 2×2 or 4×4 when having a stride of 2.

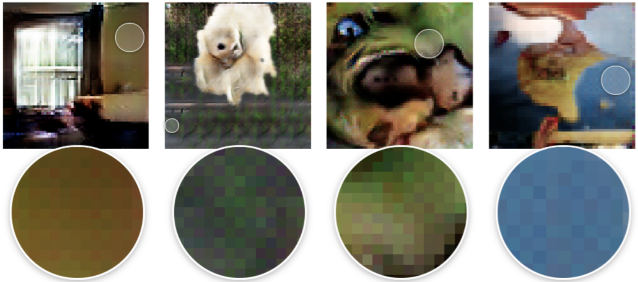


Figure: Checkerboard artifacts effect

Final Notes

Thank You!

Any Question?

Refrences

- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow - Aurélien Géron - 2019 - Link
- Deep Learning for Vision Systems - Mohamed Elgendy - 2020 - Link
- Dense semantic labeling of sub-decimeter resolution images with convolutional neural networks - Michele Volpi, Devis Tuia - 2016 - Link
- Fully Convolutional Networks for Semantic Segmentation - Jonathan Longm, Evan Shelhamer, Trevor Darrell - 2015 - Link
- Deconvolution and Checkerboard Artifacts - Augustus Odena, Vincent Dumoulin, Chris Olah - 2016 - Link