

CNN Architecture

ML Instruction Team, Fall 2022

CE Department
Sharif University of Technology

Ali Sharifi-Zarchi
Behrooz Azarkhalili
Arian Amani
Hamidreza Yaghoubi

Brief History of Computer Vision

■ What is Computer Vision?



Figure: Source

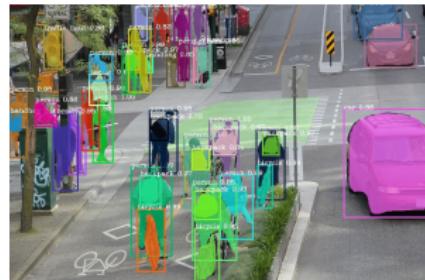


Figure: Source

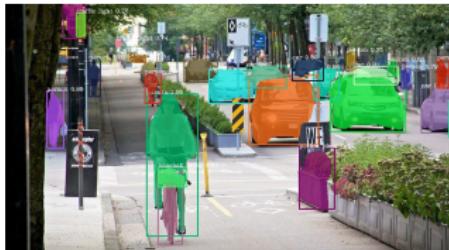


Figure: Source

Brief History of Computer Vision

■ Before Deep Learning: Hand-Crafted Features

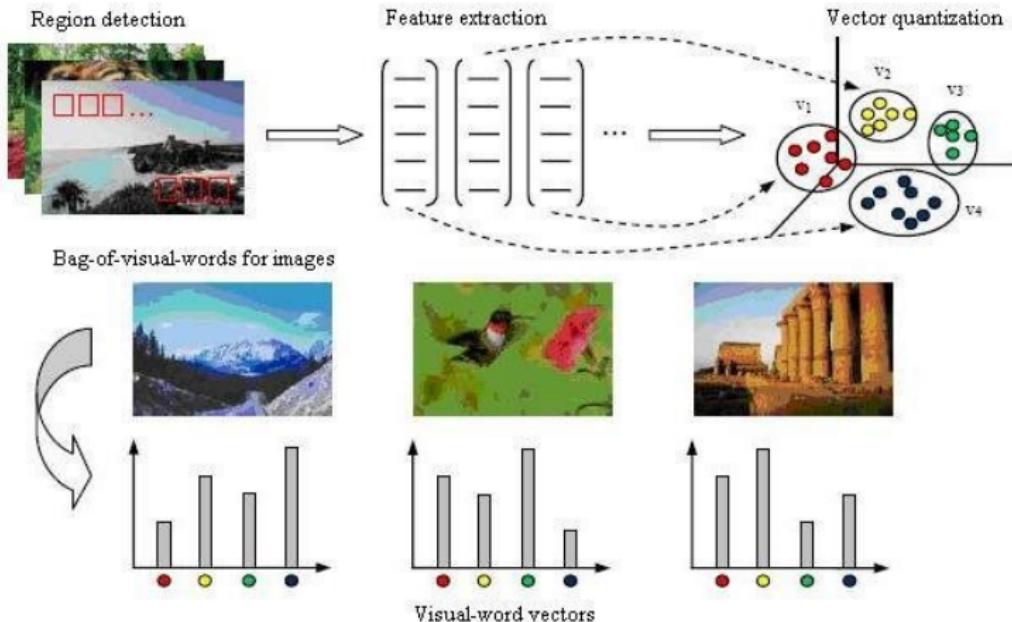


Figure: [1]

Brief History of Computer Vision

■ Before Deep Learning: Hand-Crafted Features

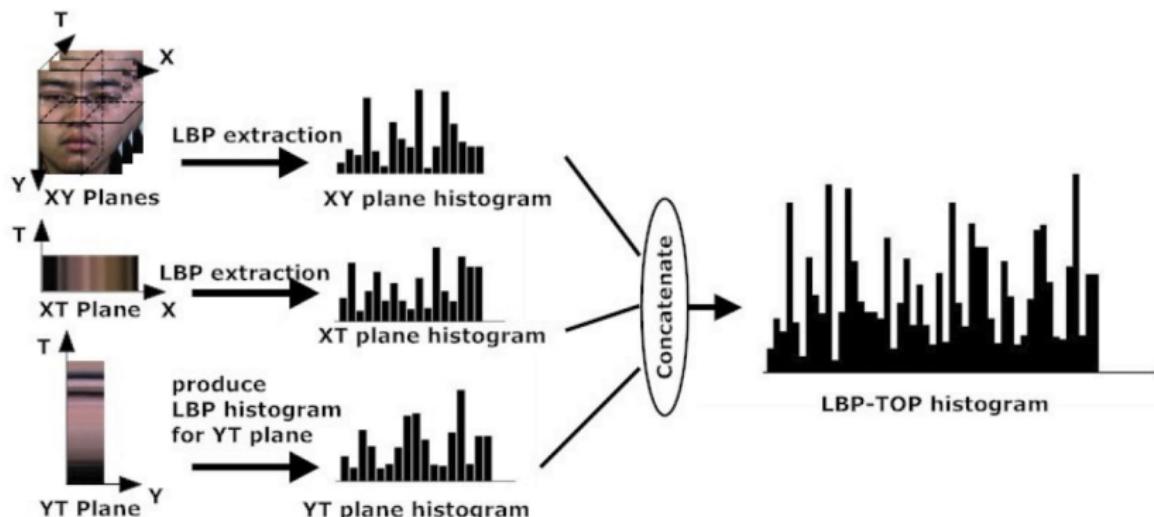


Figure: [2]

Brief History of Computer Vision

Effect of Deep Learning: Comparison between Deep Learning and Traditional Models

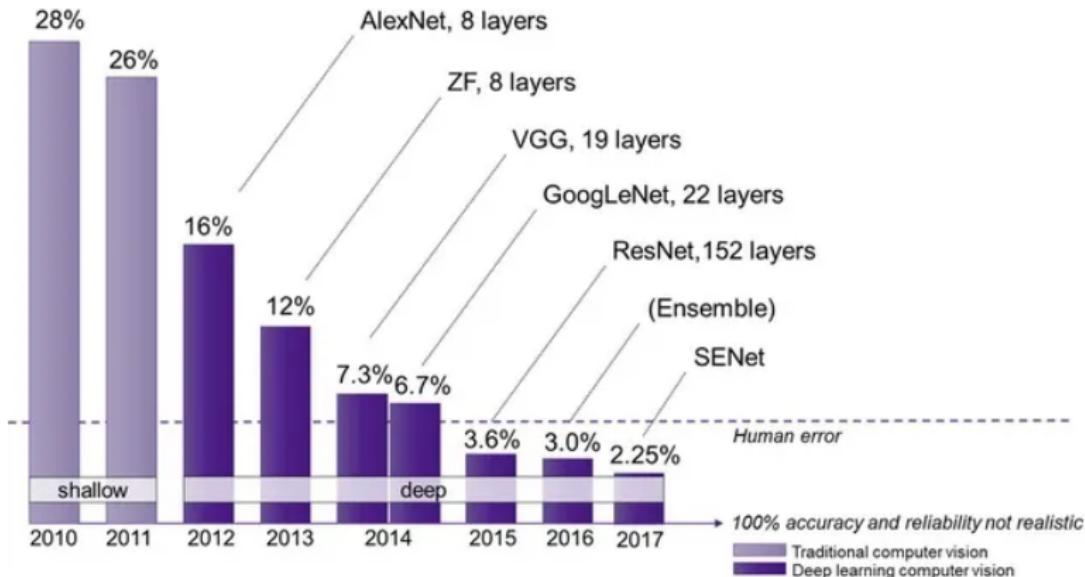


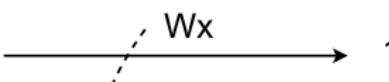
Figure: ImageNet Competition Results Over Time, Source

CNNs

- What we've been using:
 - ▶ Fully Connected Layers

32x32x3 Image Flattened to

3072



Output
10



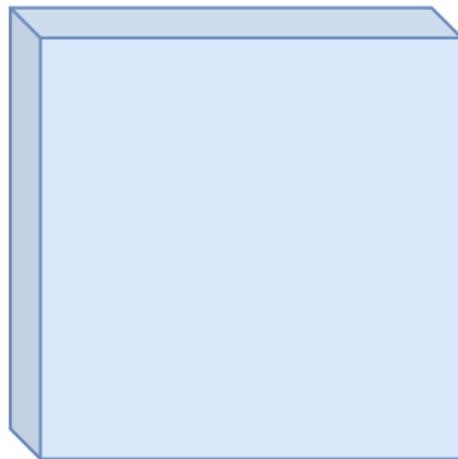
$W \Rightarrow \text{Weights } (10 \times 3072)$

CNNs

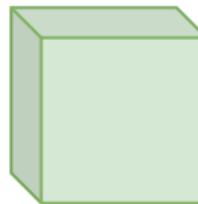
- What we're going to learn:

- ▶ Convolutional Layer

- 32x32x3 Image



5x5x3 Filter



CNNs vs. FCNs

- First of all...

CNNs vs. FCNs

- First of all...

- ▶ Why use ConvNets instead of Fully Connected Networks with images?

CNNs vs. FCNs

- First of all...
 - ▶ Why use ConvNets instead of Fully Connected Networks with images?
 - Let's do an experiment.

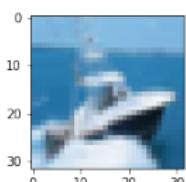
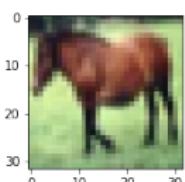
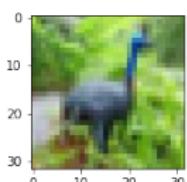
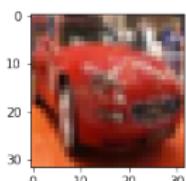
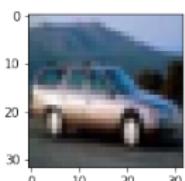
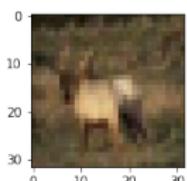
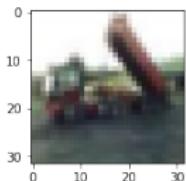
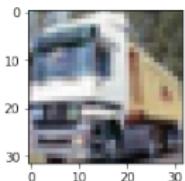
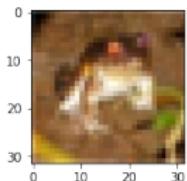
CNNs vs. FCNs

- We're going to use the CIFAR10 dataset

```
# example of loading the cifar10 dataset
import matplotlib.pyplot as plt
from keras.datasets import cifar10
# load dataset
(trainX, trainy), (testX, testy) = cifar10.load_data()
# summarize loaded dataset
print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
print('Test: X=%s, y=%s' % (testX.shape, testy.shape))
# define subplot
fig, ax = plt.subplots(3, 3)
# plot first few images
for i in range(9):
    # plot raw pixel data
    ax[i//3, i%3].imshow(trainX[i])
# show the figure
fig.tight_layout()
fig.set_figheight(8)
fig.set_figwidth(8)
plt.show()
```

CNNs vs. FCNs

- We're going to use the CIFAR10 dataset



CNNs vs. FCNs

- First, we train on a Fully Connected Network

```
dense_model = Sequential(  
    [  
        Dense(2048, input_dim=32*32*3, activation='relu', name='Dense_Layer_1'),  
        Dense(1024, activation='relu', name='Dense_Layer_2'),  
        Dense(512, activation='relu', name='Dense_Layer_3'),  
        Dense(128, activation='relu', name='Dense_Layer_4'),  
        Dense(10, activation='softmax', name='Softmax_Output_Layer'),  
    ],  
    name='Dense_Model'  
)  
dense_model.compile(loss='categorical_crossentropy',  
                     optimizer='adam',  
                     metrics=['accuracy'])  
  
dense_model.summary()
```

CNNs vs. FCNs

- First, we train on a Fully Connected Network

Model: "Dense_Model"

Layer (type)	Output Shape	Param #
Dense_Layer_1 (Dense)	(None, 2048)	6293504
Dense_Layer_2 (Dense)	(None, 1024)	2098176
Dense_Layer_3 (Dense)	(None, 512)	524800
Dense_Layer_4 (Dense)	(None, 128)	65664
Softmax_Output_Layer (Dense)	(None, 10)	1290
)		

Total params: 8,983,434

Trainable params: 8,983,434

Non-trainable params: 0

[Listing:](#) FCN with around 9M parameters

CNNs vs. FCNs

- First, we train on a Fully Connected Network

```
dense_test_accuracy = dense_model.evaluate(dense_test, testY)[1]
print("Test Accuracy", np.round((dense_test_accuracy)*100,2))
```

```
313/313 [=====] - 1s 3ms/ step - loss: 1.4785 - accuracy: 0.5059
Test Accuracy 50.59
```

Listing: Test results after the training is done; Test Accuracy: 50.6%

CNNs vs. FCNs

- Now, let's train on a Convolutional Neural Network

```
Convolutional_model = Sequential(  
    [  
        Conv2D(32, (3, 3), input_shape=(32, 32, 3), activation='relu', name='Conv_Layer_1'),  
        MaxPool2D(pool_size=(2, 2), name='Max_Pool_1'),  
        Conv2D(64, (3, 3), activation='relu', name='Conv_Layer_2'),  
        Conv2D(128, (3, 3), activation='relu', name='Conv_Layer_3'),  
        Flatten(name='Flatten'),  
        Dense(128, activation='relu', name='Dense_Flat_1'),  
        Dense(10, activation='softmax', name='Softmax_Output_Layer')  
    ],  
    name='Convolutional_Model'  
)  
  
Convolutional_model.compile(loss='categorical_crossentropy',  
                            optimizer='adam',  
                            metrics=['accuracy'])  
  
Convolutional_model.summary()
```

CNNs vs. FCNs

- Now, let's train on a Convolutional Neural Network

Model: "Convolutional_Model"

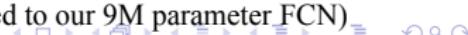
Layer (type)	Output Shape	Param #
Conv_Layer_1 (Conv2D)	(None, 30, 30, 32)	896
Max_Pool_1 (MaxPooling2D)	(None, 15, 15, 32)	0
Conv_Layer_2 (Conv2D)	(None, 13, 13, 64)	18496
Conv_Layer_3 (Conv2D)	(None, 11, 11, 128)	73856
Flatten (Flatten)	(None, 15488)	0
Dense_Flat_1 (Dense)	(None, 128)	1982592
Softmax_Output_Layer (Dense)	(None, 10)	1290
)		

Total params: 2,077,130

Trainable params: 2,077,130

Non-trainable params: 0

Listing: CNN with around 2M parameters (A lot less compared to our 9M parameter FCN)



CNNs vs. FCNs

Now, let's train on a Convolutional Neural Network

```
dense_test_accuracy = dense_model.evaluate(dense_test, testY)[1]
print("Test Accuracy", np.round((dense_test_accuracy)*100,2))
```

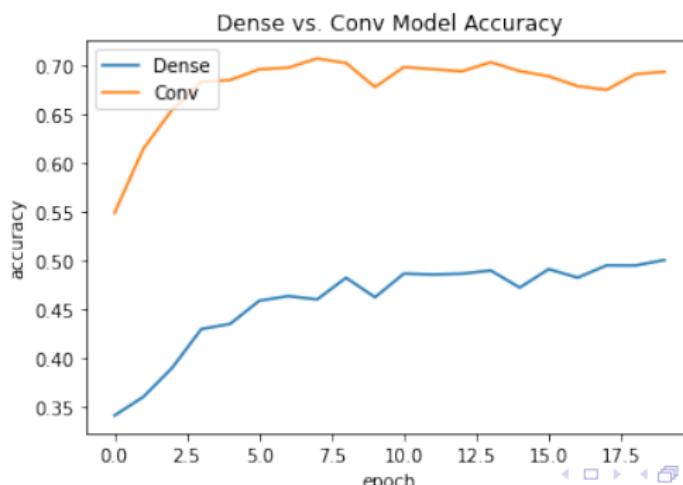
```
313/313 [=====] - 1s 3ms/ step - loss: 1.4785 - accuracy: 0.5059
Test Accuracy 50.59
```

Listing: Test results after the training is done; Test Accuracy: 67.8%

CNNs vs. FCNs

- As you can see, CNNs have a great effect on both results and computation efficiency while working with visual imagery.

Model Architecture	# Params	Test Accuracy
CNN	2M	67.8%
FCN	9M	50.6%



CNNs vs. FCNs

- Now that we understand the importance of Deep Learning and Convolutional Neural Networks, let's learn the details.

What is Convolution and how does it work?

- Consider this, an array of numbers

1	0	1
0	1	0
1	0	1

What is Convolution and how does it work?

- Consider this, an array of numbers.
 - ▶ Weights for a convolutional layer, called a filter or a kernel.

1	0	1
0	1	0
1	0	1

What is Convolution and how does it work?

- Consider this as 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:

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

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

Figure:
Convolving
Kernel

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

Image

4		

Convolved
Feature

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:

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

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

Figure:
Convolving
Kernel

1	1 _{x1}	1 _{x0}	0 _{x1}	0
0	1 _{x0}	1 _{x1}	1 _{x0}	0
0	0 _{x1}	1 _{x0}	1 _{x1}	1
0	0	1	1	0
0	1	1	0	0

Image

4	3	

Convolved
Feature

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:

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

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

Figure:
Convolving
Kernel

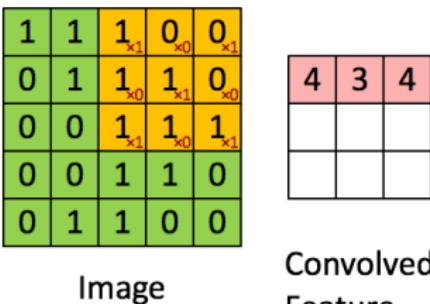


Figure: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature,
Source

What is Convolution and how does it work?

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

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

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

Figure:
Convolving
Kernel

1	1	1	0	0
0	\times_1	\times_0	\times_1	1
0	\times_0	\times_1	\times_0	1
0	\times_1	\times_0	\times_1	1
0	1	1	0	0

Image

4	3	4
2		

Convolved
Feature

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:

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

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

Figure:
Convolving
Kernel

1	1	1	0	0
0	1	\times_1	\times_0	\times_1
0	0	\times_0	\times_1	\times_0
0	0	\times_1	\times_0	\times_1
0	1	1	0	0

Image

4	3	4
2	4	

Convolved
Feature

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:

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

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

Figure:
Convolving
Kernel

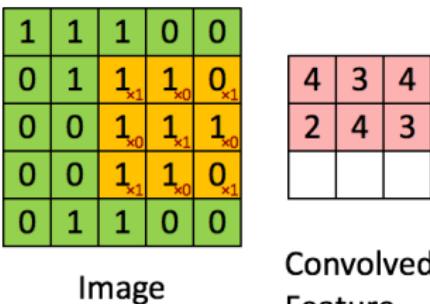


Figure: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature,
Source

What is Convolution and how does it work?

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

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

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

Figure:
Convolving
Kernel

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

Image

4	3	4
2	4	3
2		

Convolved
Feature

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:

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

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

Figure:
Convolving
Kernel

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

Image

4	3	4
2	4	3
2	3	

Convolved
Feature

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:

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

I: Input Image

K: Our Kernel

k_h and k_w : The height and width of the Kernel

1	0	1
0	1	0
1	0	1

Figure:
Convolving
Kernel

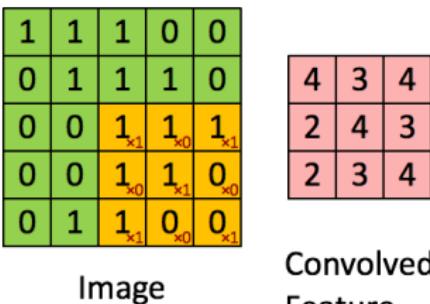


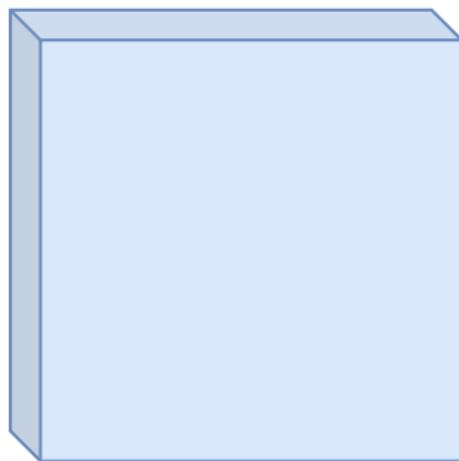
Figure: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature,
Source

CNNs

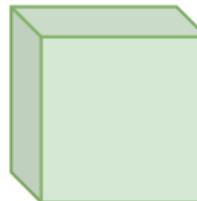
Convolutional Layer

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

32x32x3 Image

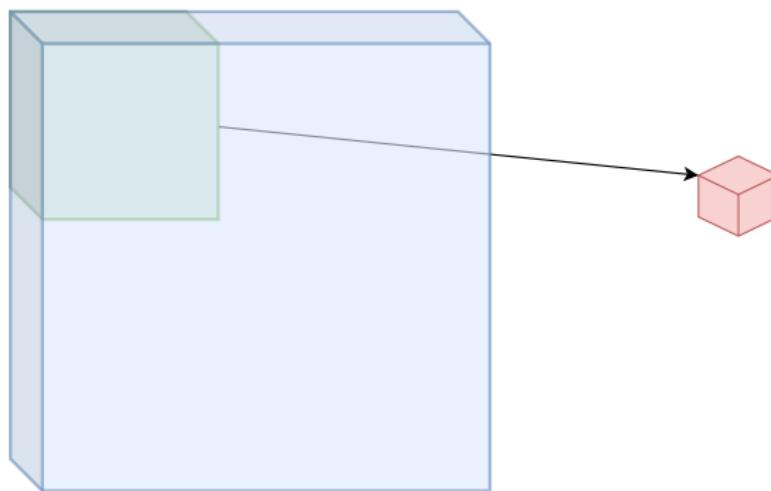


5x5x3 Filter



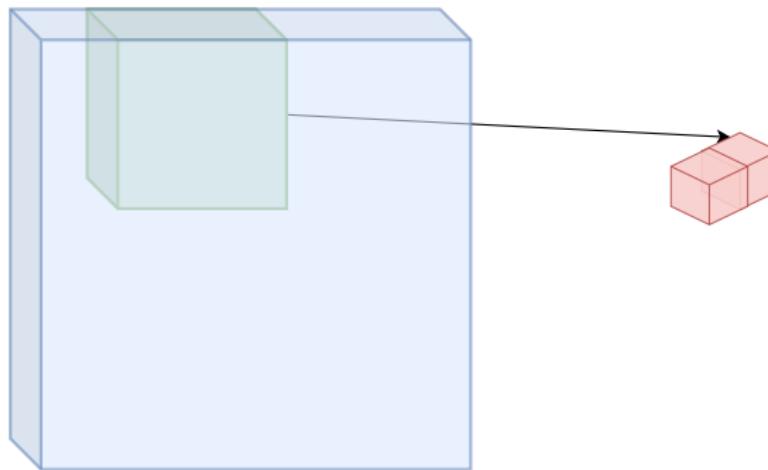
What is Convolution and how does it work?

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



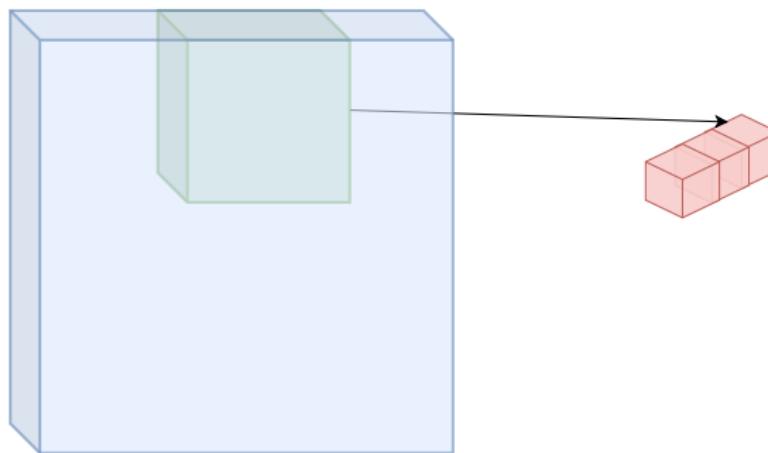
What is Convolution and how does it work?

- 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 when 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 when working with images.
- Uses a lot fewer parameters compared to Fully Connected Layers.

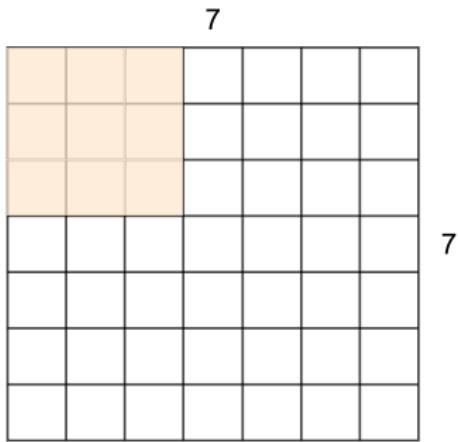


What is a Stride

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

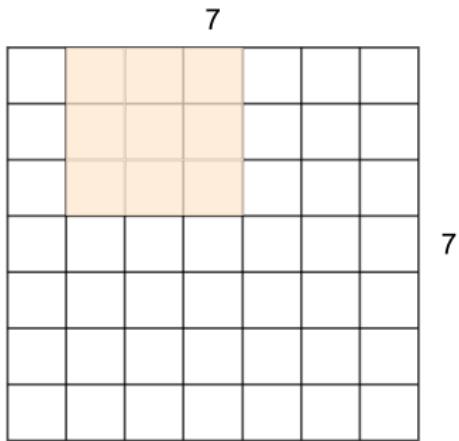


What is a Stride

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

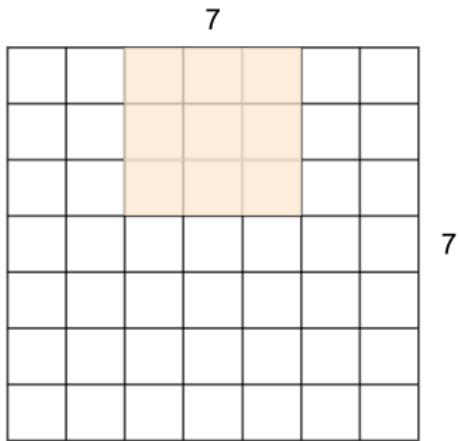


What is a Stride

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

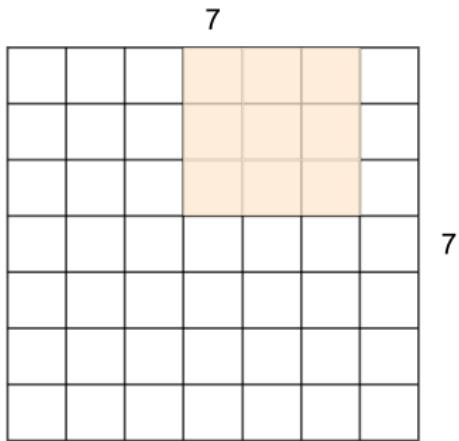


What is a Stride

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

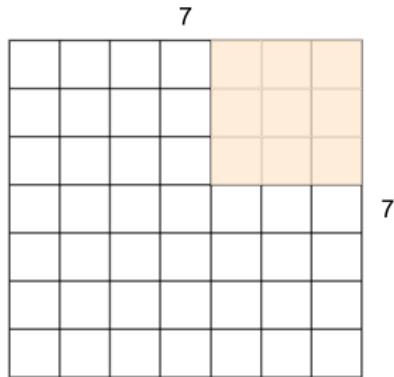


What is a Stride

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

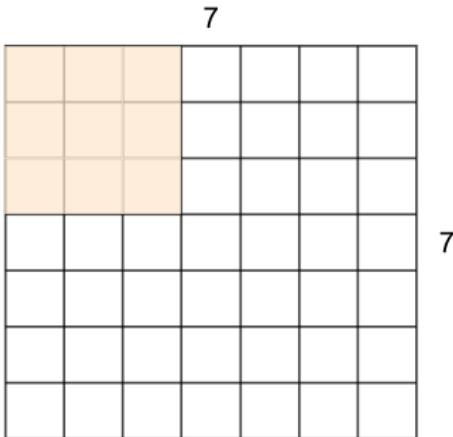
- 7x7 input with 3x3 filter
- This was a Stride 1 filter
- => Outputs 5x5



Strides

Now let's use Stride 2

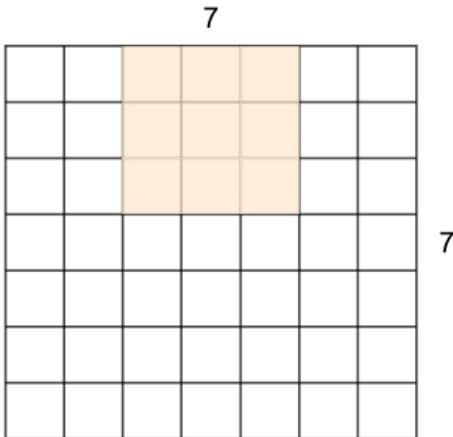
- 7x7 input with 3x3 filter



Strides

Now let's use Stride 2

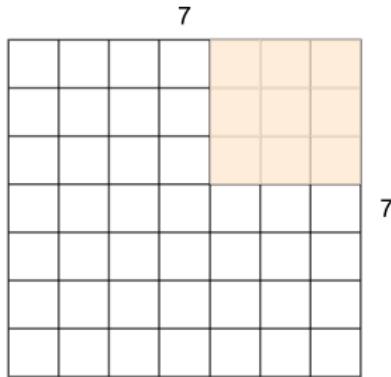
- 7x7 input with 3x3 filter



Strides

Now let's use Stride 2

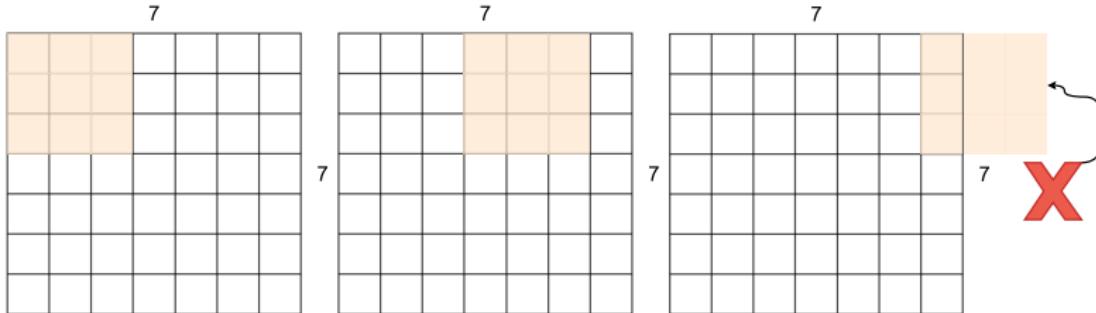
- 7x7 input with 3x3 filter
- This was a Stride 2 filter
- => Outputs 3x3



Strides

Stride 3?

- 7x7 input with 3x3 filter

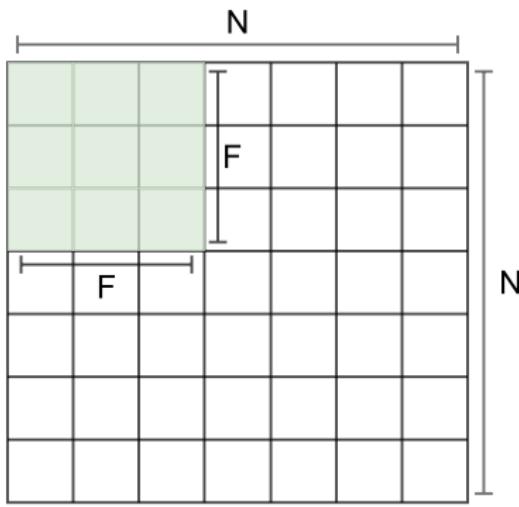


Strides

So 7×7 input with 3×3 filter and stride 3 doesn't work!

Let's do the calculations:

$$\text{OutputSize} = (N - F)/Stride + 1$$

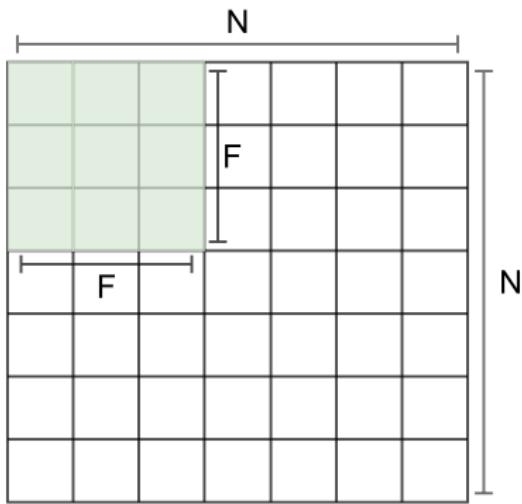


Strides

$$\text{OutputSize} = (N - F)/\text{Stride} + 1$$

$N = 7, F = 3 \Rightarrow$

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



Strides

Do you see any problems?

Strides

- 1. Borders don't get enough attention.

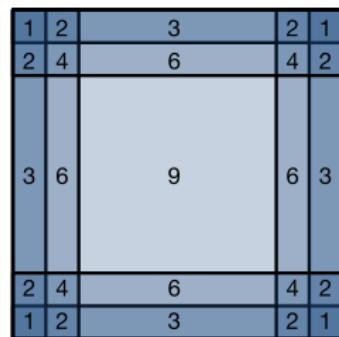
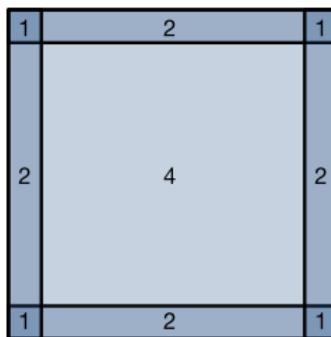
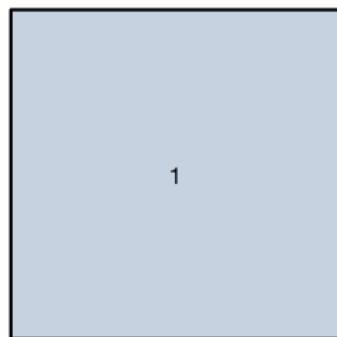


Figure: [3], Pixel utilization for convolutions of size 1×1 , 2×2 , and 3×3 respectively.

Strides

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

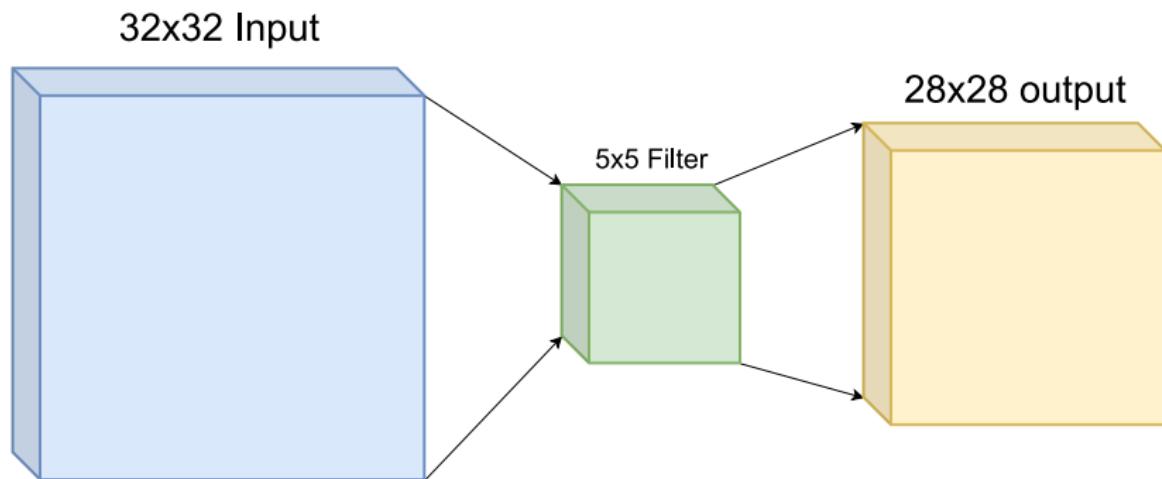


Figure: 32x32 input shrinks to 28x28 output. (information loss)

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

The diagram illustrates the convolution process with padding. On the left, a 6x6 input matrix is shown with values ranging from 0 to 9. The first row has values [0, 0, 0, 0, 0, 0]. The second row has values [0, 3, 3, 4, 4, 7, 0, 0]. The third row has values [0, 9, 7, 6, 5, 8, 2, 0]. The fourth row has values [0, 6, 5, 5, 6, 9, 2, 0]. The fifth row has values [0, 7, 1, 3, 2, 7, 8, 0]. The sixth row has values [0, 0, 3, 7, 1, 8, 3, 0]. The seventh row has values [0, 4, 0, 4, 3, 2, 2, 0]. The eighth row has values [0, 0, 0, 0, 0, 0, 0, 0]. Below this matrix is the text $6 \times 6 \rightarrow 8 \times 8$. To the right of the input matrix is a multiplication symbol (*). Next to it is a 3x3 filter matrix with values [1, 0, -1; 1, 0, -1; 1, 0, -1]. Below the filter is the text 3×3 . To the right of the filter is an equals sign (=). To the right of the equals sign is a 3x3 output matrix with values [-10, -13, 1; -9, 3, 0; 0, 0, 0]. Below this output matrix is the text 6×6 .

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.

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

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.
- $\text{OutputSize} = (N + 2P - F)/\text{Stride} + 1$

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

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.
- $OutputSize = (N + 2P - F)/Stride + 1$

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

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

- $F = 3 \rightarrow P = 1$
- $F = 5 \rightarrow P = 2$
- $F = 7 \rightarrow P = 3$

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

Pooling

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

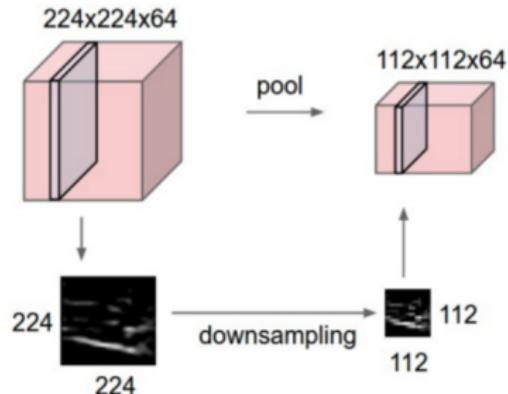


Figure: Pooling Layer

Pooling: How It Works

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

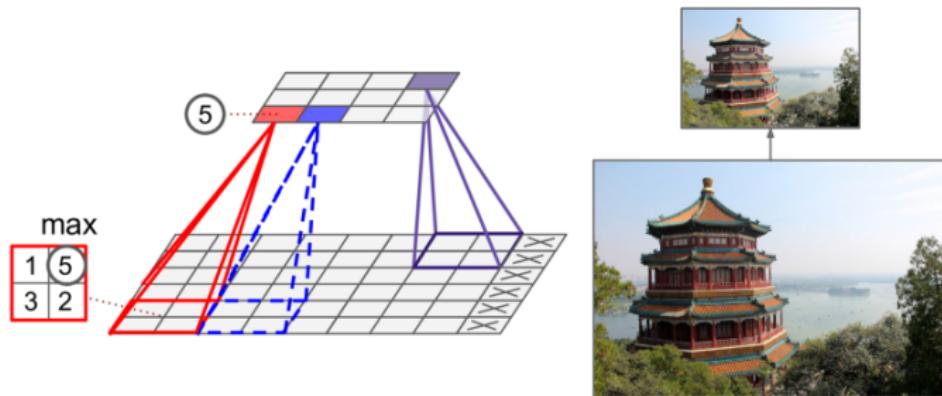


Figure: Max pooling layer (2×2 pooling kernel, stride 2, no padding)

Pooling: Benefits

- A Pooling layer's goal is to sub-sample the input in order to reduce:
 - ▶ The computational load
 - ▶ The memory usage
 - ▶ The number of parameters
 - ▶ Limiting the risk of overfitting

Pooling: Types

- Max Pooling: gives better results when the background is white and the object is dark
- Min Pooling: gives better results when the background is dark and the object is white
- Average Pooling: smoothes the picture

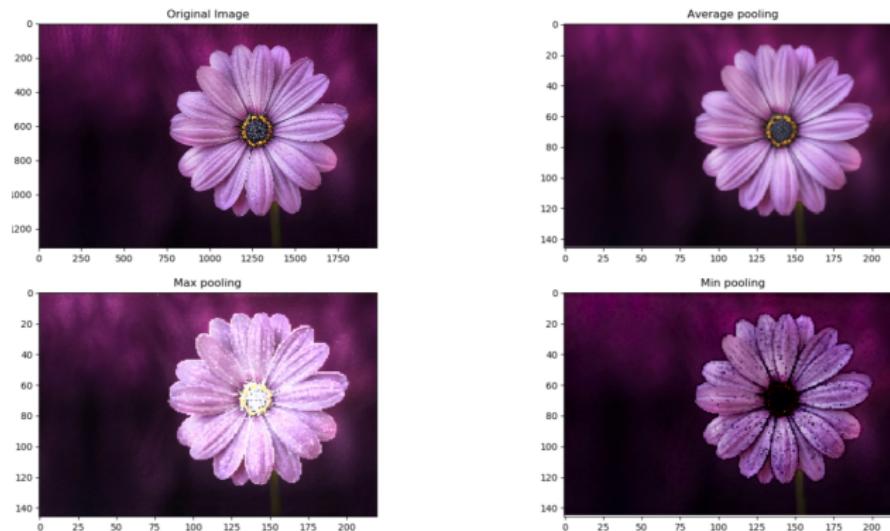


Figure: Effects of different pooling layers

Pooling: Max Pooling

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

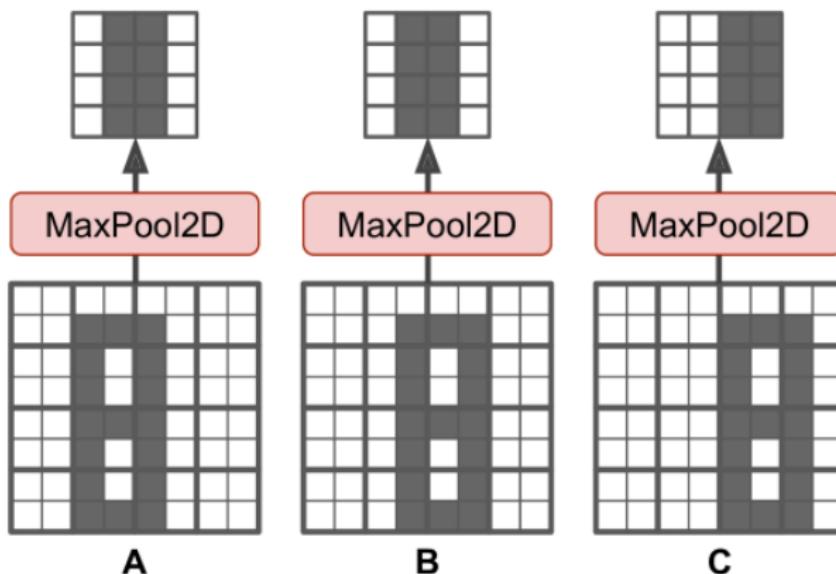


Figure: Invariance to small translations

Dilation

- It expands the kernel (input) by inserting spaces between its consecutive elements

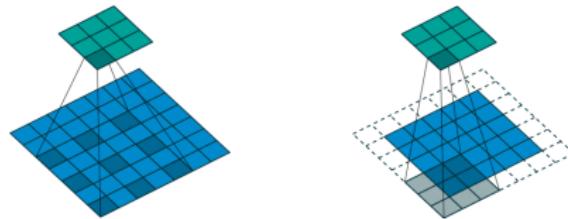


Figure: Dilated Convolution ($l = 2$) vs. Standard Convolution ($l = 1$)

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

So far, we have talked about 2D inputs, but although images are 2D, we need to represent them as 3D matrices to show their colors:

- Pixel values range from 0 to 255
- We cannot represent all the colors in a picture with only one channel of numbers from 0 to 255

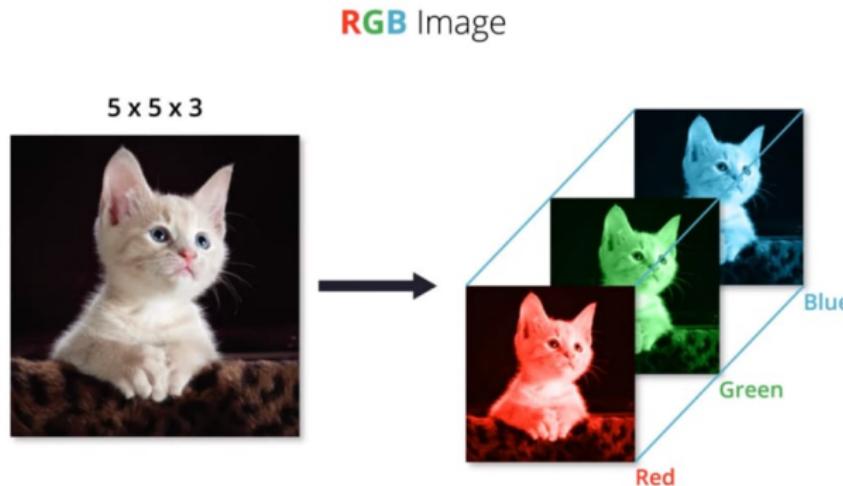


Figure: RGB Image, Source

Channels

So far, we have talked about 2D inputs, but although images are 2D, we need to represent them as 3D matrices to show their colors:

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

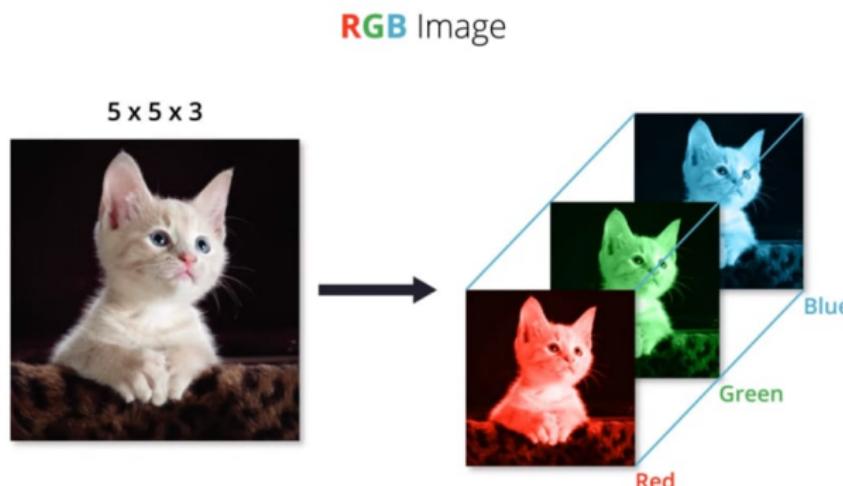


Figure: RGB Image, Source

Channels

So far, we have talked about 2D inputs, but although images are 2D, we need to represent them as 3D matrices to show their colors:

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

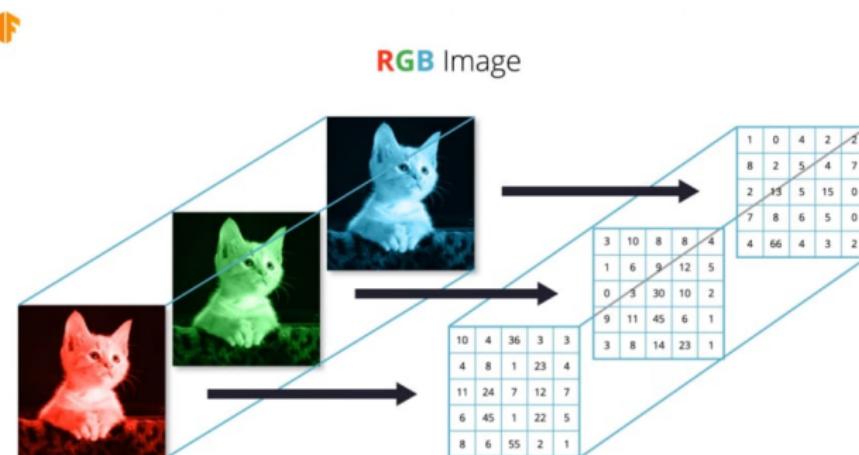


Figure: RGB Image, Source

Channels

So far, we have talked about 2D inputs, but although images are 2D, we need to represent them as 3D matrices to show their colors:

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

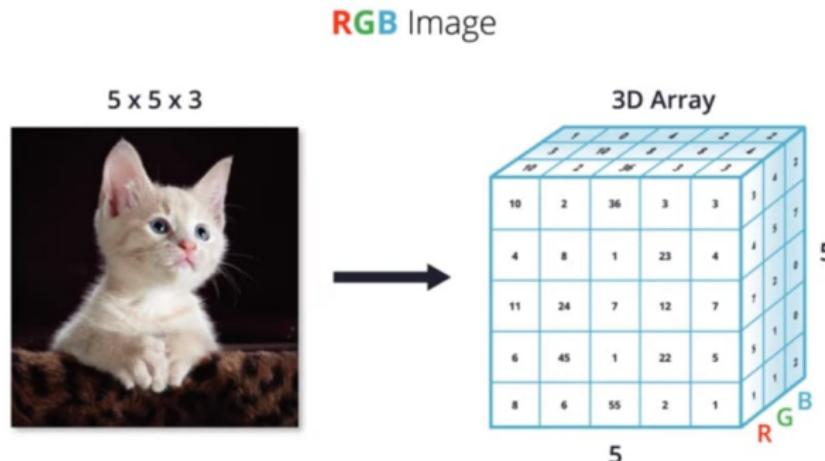


Figure: RGB Image, Source

Channels

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

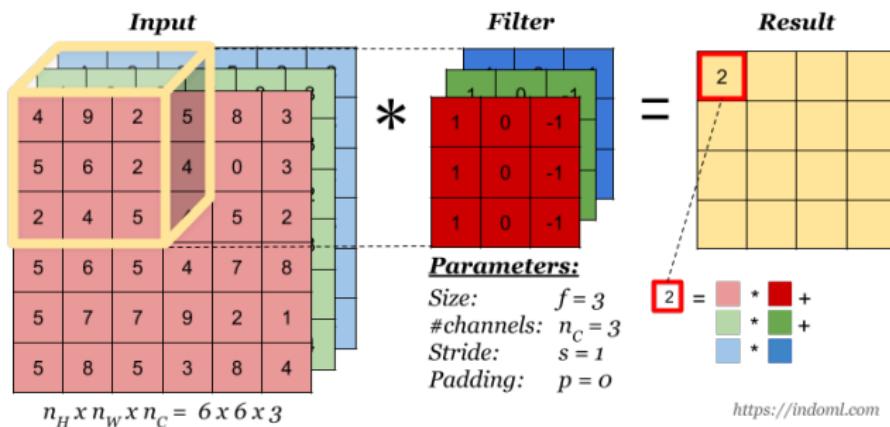


Figure: The total number of multiplications to calculate the result is $(4 \times 4) \times (3 \times 3 \times 3) = 432$, Source

Channels

- 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 obtain multiple output channels. Each channel can learn something distinct.

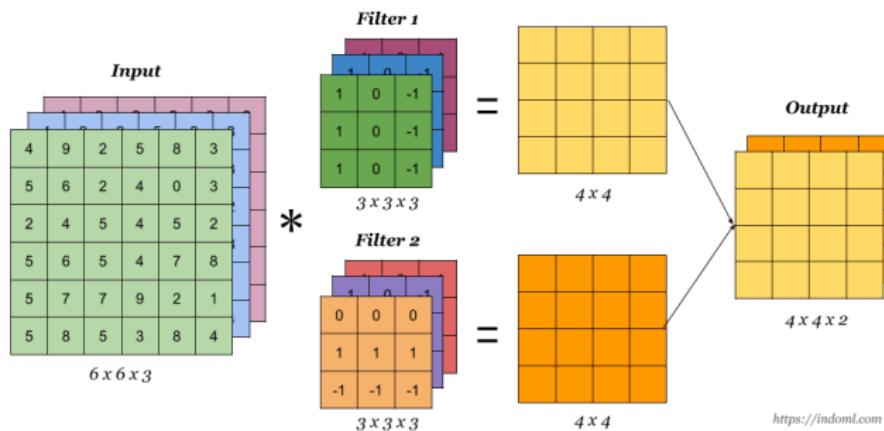


Figure: The total number of multiplications to calculate the result is $(4 \times 4 \times 2) \times (3 \times 3 \times 3) = 864$, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

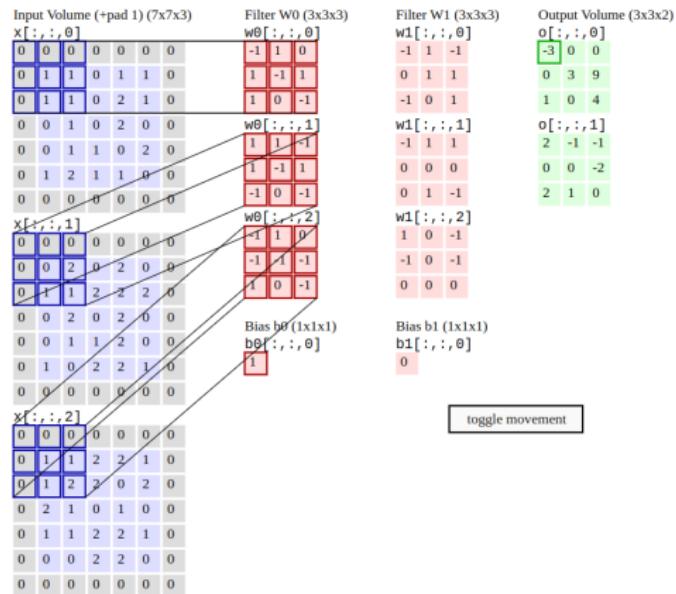


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

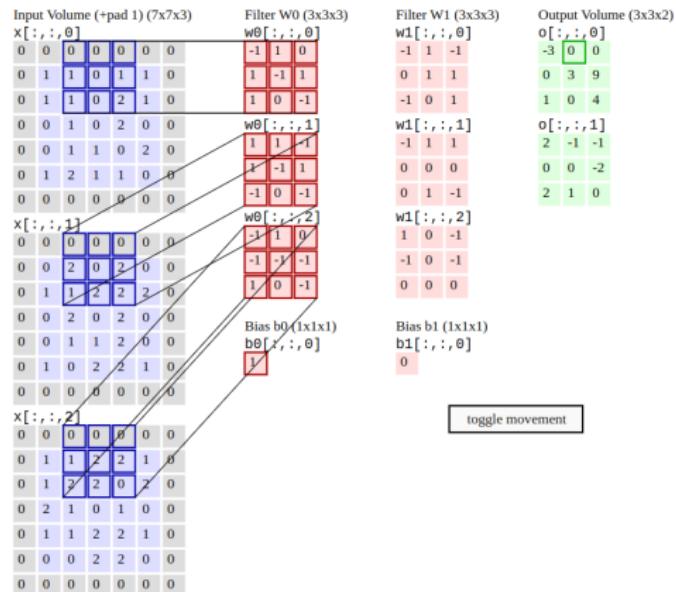


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

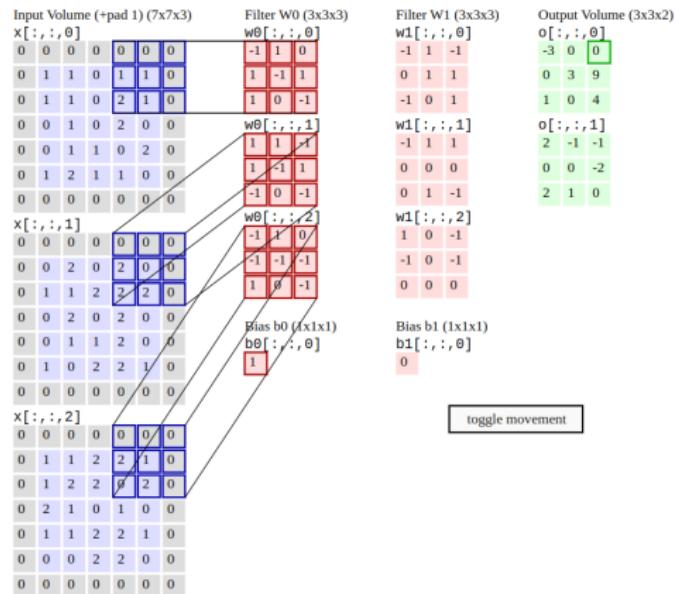


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

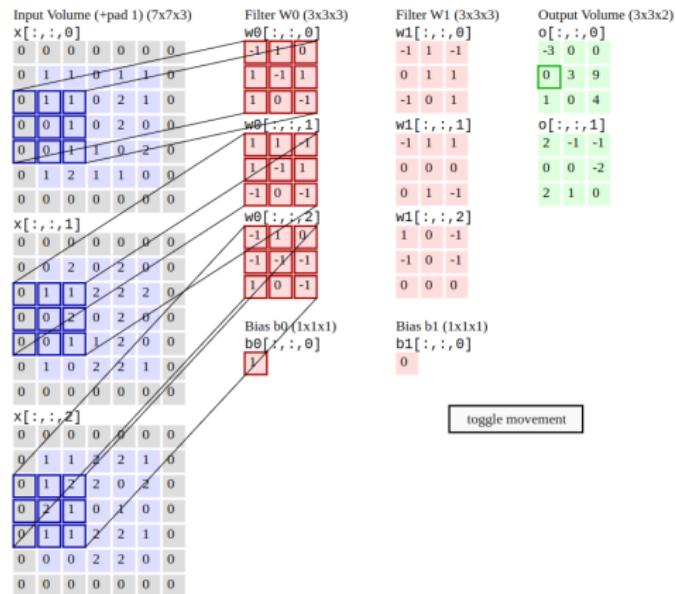


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

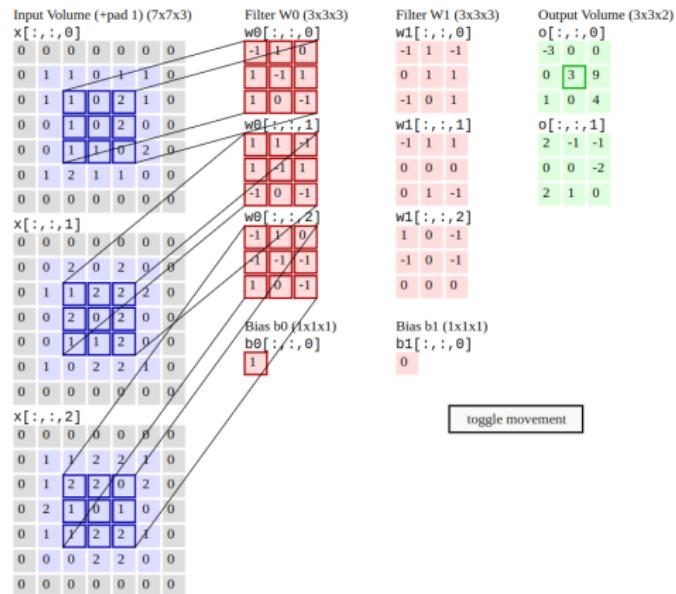


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

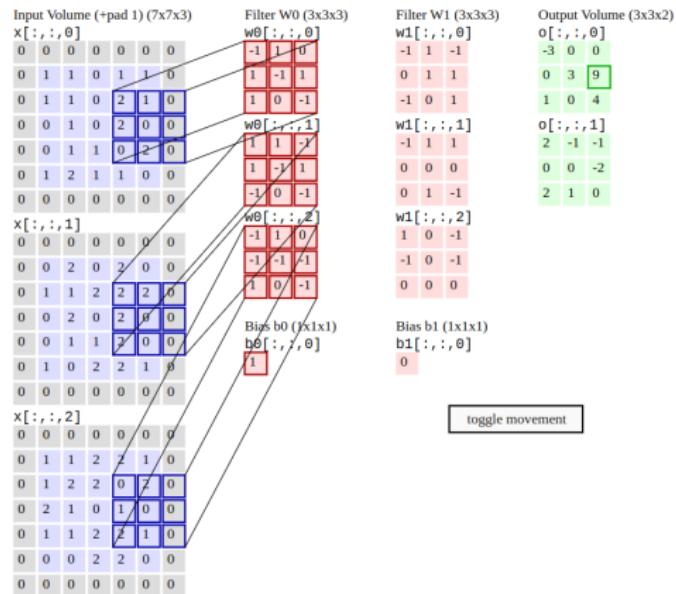


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

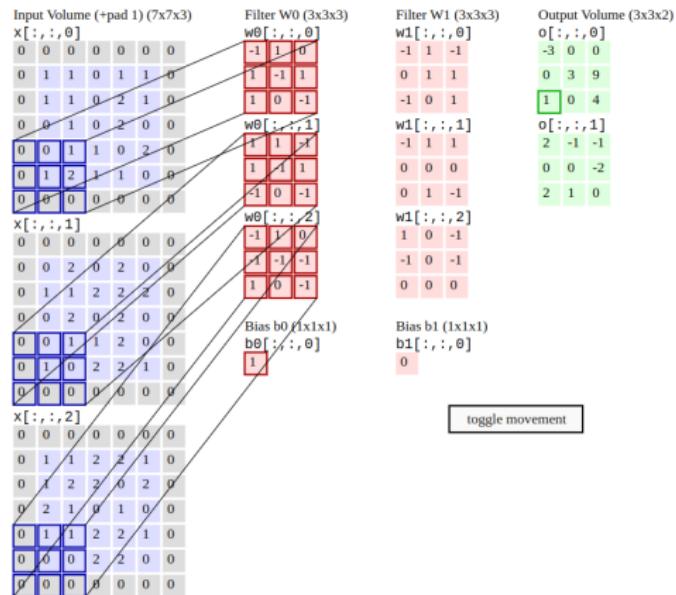


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

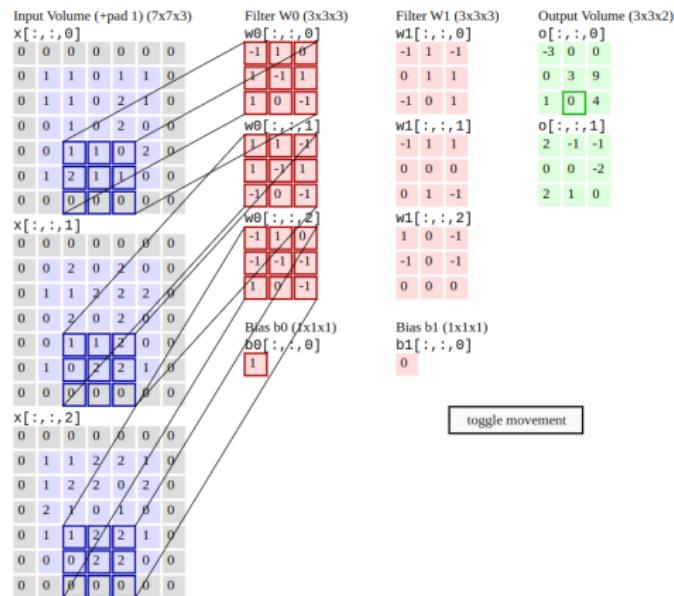


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

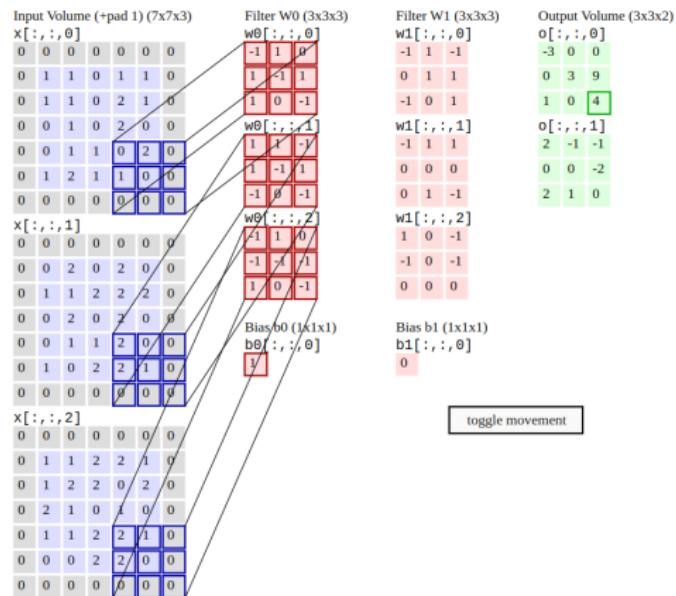


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

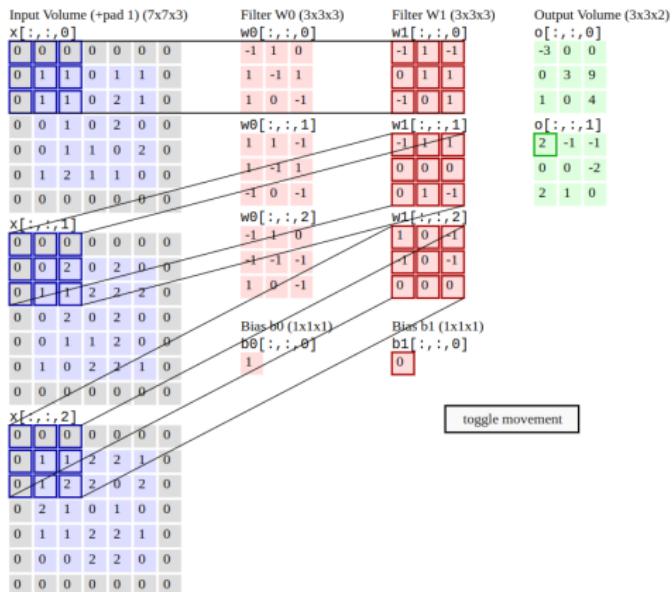


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

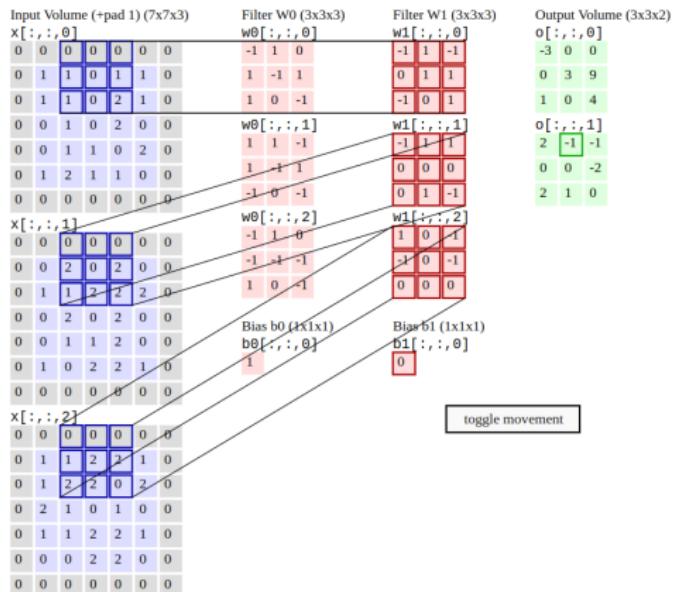


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

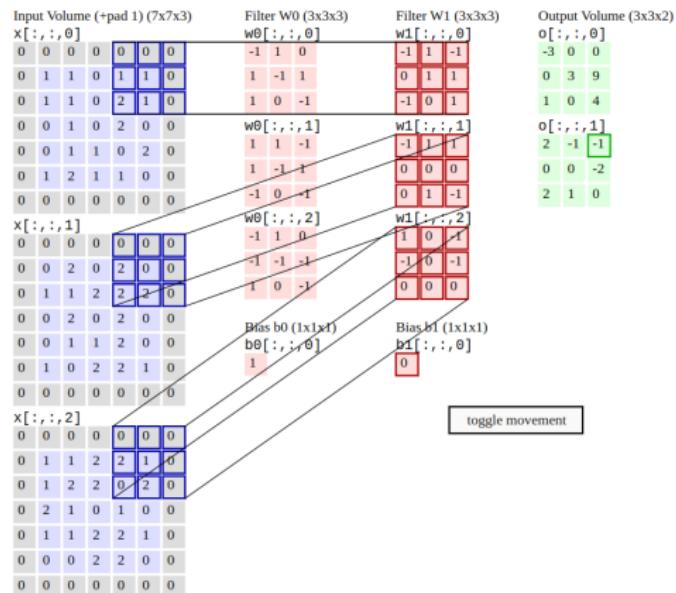


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

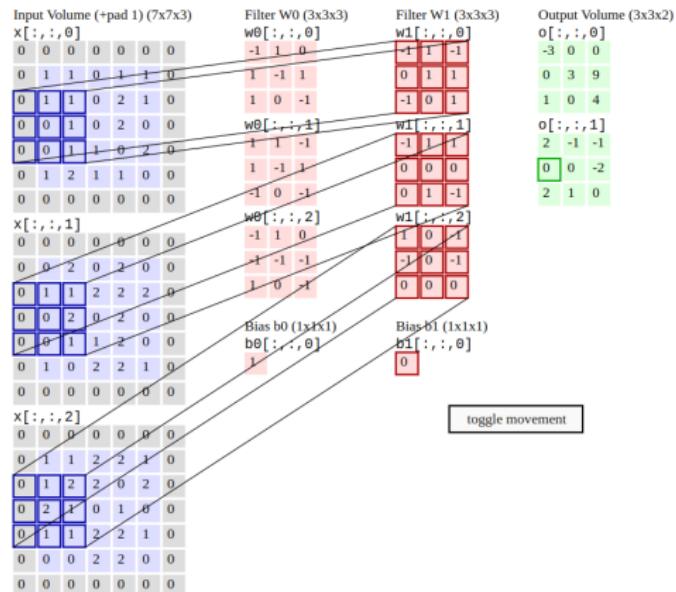


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

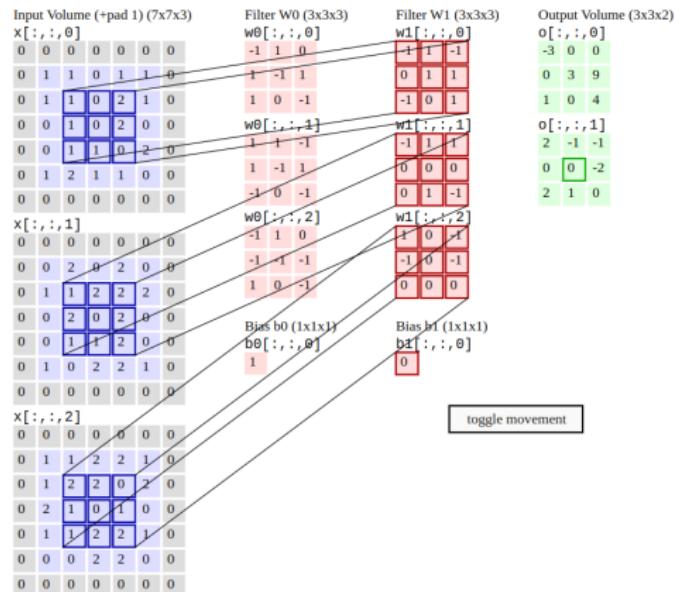


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

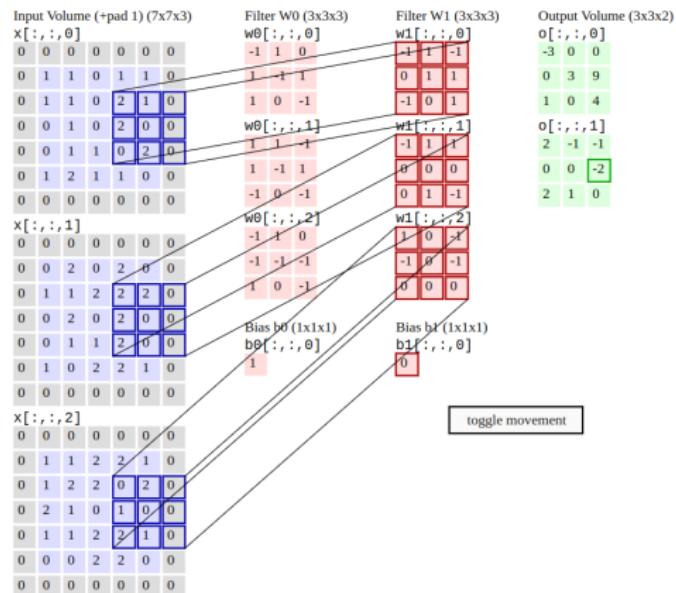


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

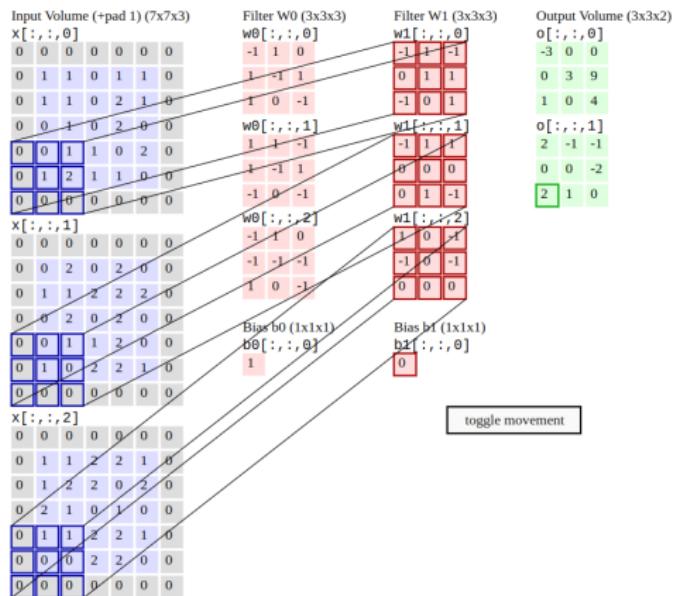


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

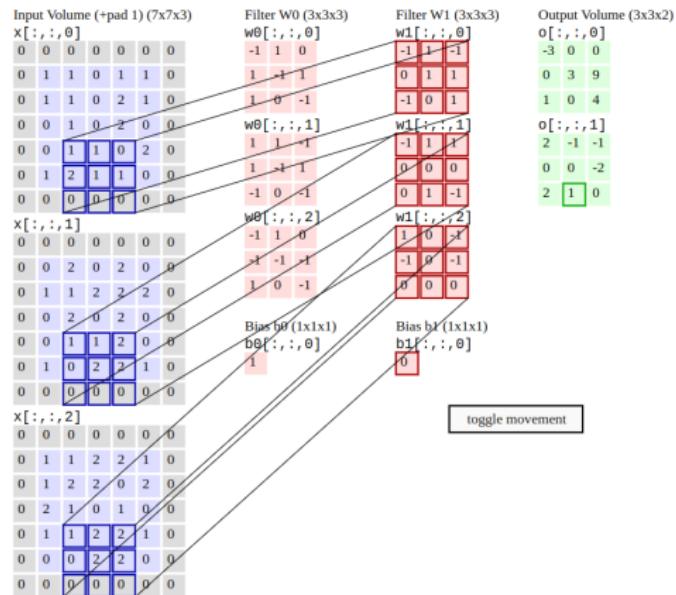


Figure: Great Visualization by CS231n, Source

Channels

Let's have a closer look to how the calculations with more than one filter work:

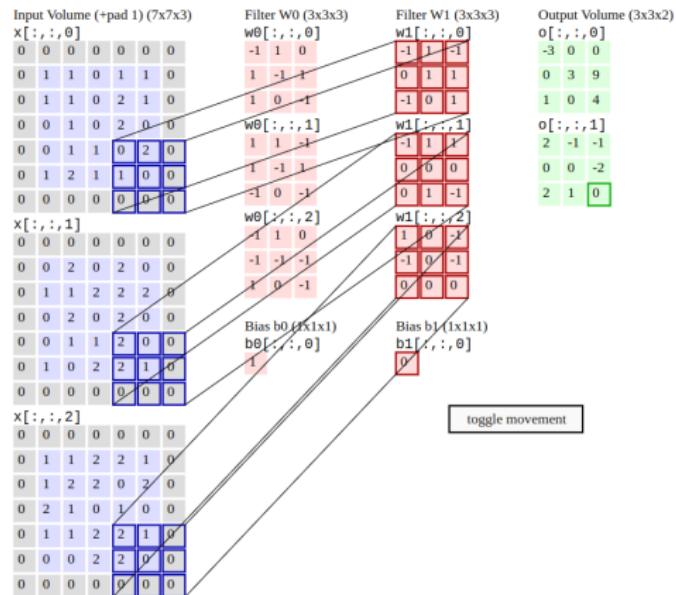
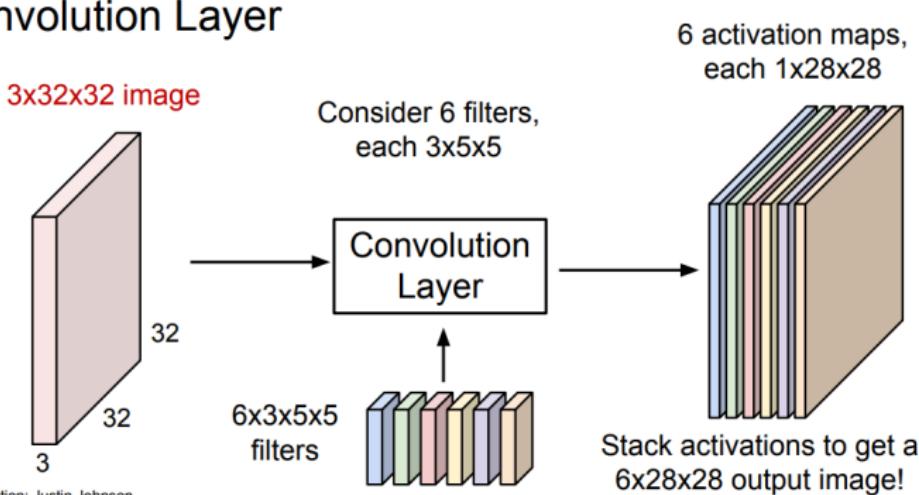


Figure: Great Visualization by CS231n, Source

Channels

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

Convolution Layer



Slide inspiration: Justin Johnson

Figure: Slide from CS231n

Channels

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

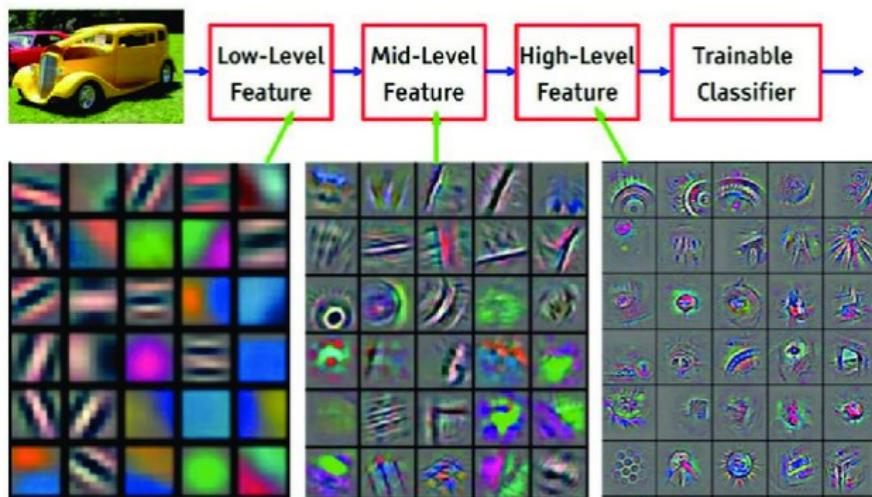


Figure: Each kernel learning something different, Source

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 $p = 0$:

$$o = (i - k) + 1 \quad (1)$$

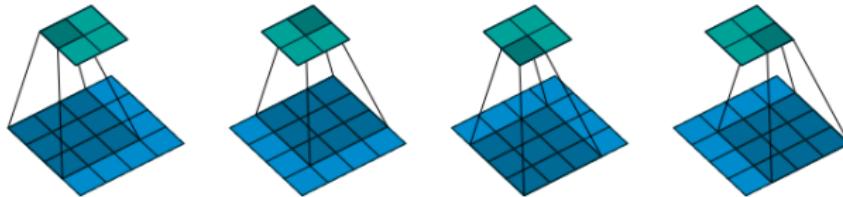


Figure: Convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$)

Arithmetic of CNNs: Zero padding, unit strides

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

$$o = (i - k) + 2p + 1 \quad (2)$$

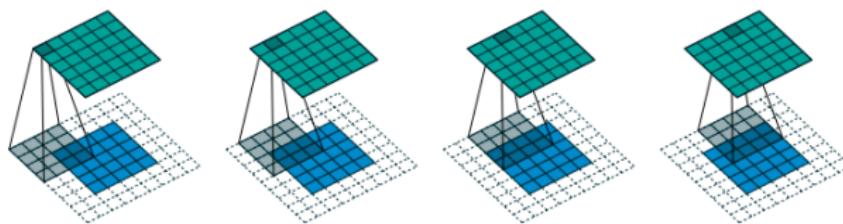


Figure: Convolving a 4×4 kernel over a 5×5 input padded with a 2×2 border of zeros using unit strides (i.e., $i = 5$, $k = 4$, $s = 1$ and $p = 2$)

Arithmetic of CNNs: Half (same) padding

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

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

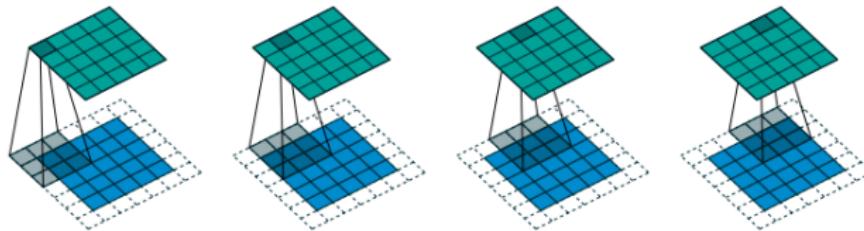


Figure: Convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 1$)

Arithmetic of CNNs: Full padding

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

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

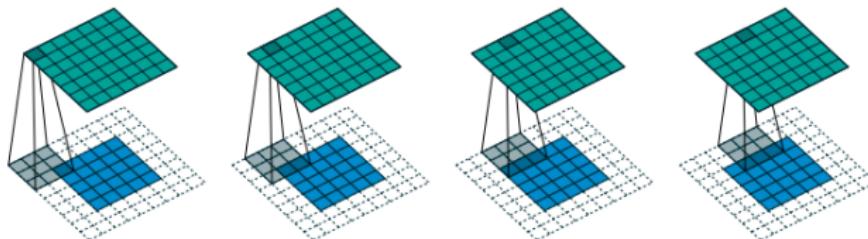


Figure: Convolving a 3×3 kernel over a 5×5 input using full padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 2$)

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

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

$$o = \frac{i - k}{s} + 1 \quad (5)$$

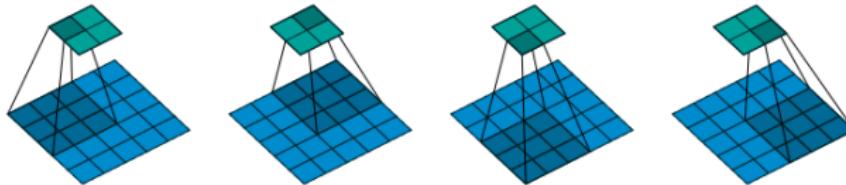


Figure: Convolving a 3×3 kernel over a 5×5 input using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 0$)

Arithmetic of CNNs: Zero padding, non-unit strides

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

$$o = \frac{i + 2p - k}{s} + 1 \quad (6)$$

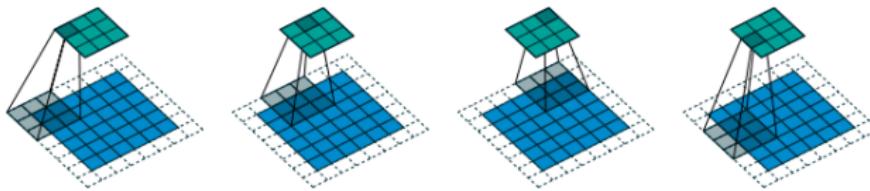


Figure: Convolving a 3×3 kernel over a 6×6 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 6$, $k = 3$, $s = 2$ and $p = 1$). In this case, the bottom row and right column of the zero-padded input are not covered by the kernel

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

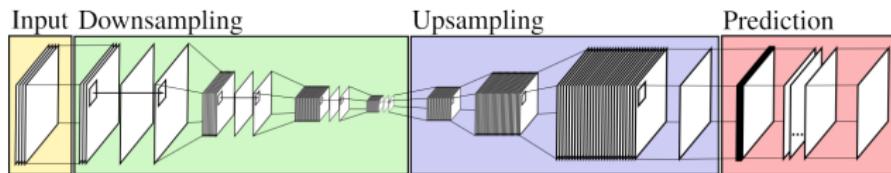


Figure: Schematic of the Downsampling and Upsampling

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.

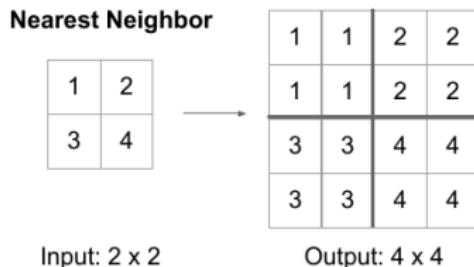


Figure: Nearest Neighbors Upsampling

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

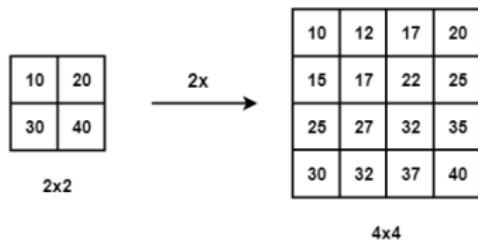


Figure: Bi-Linear Interpolation

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.

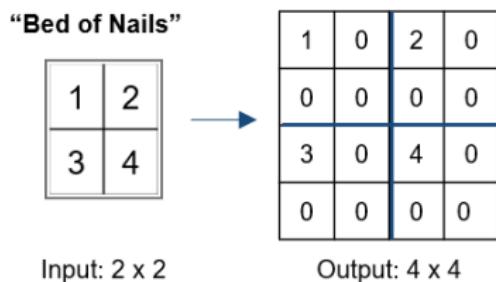


Figure: Bed of Nails Upsampling

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

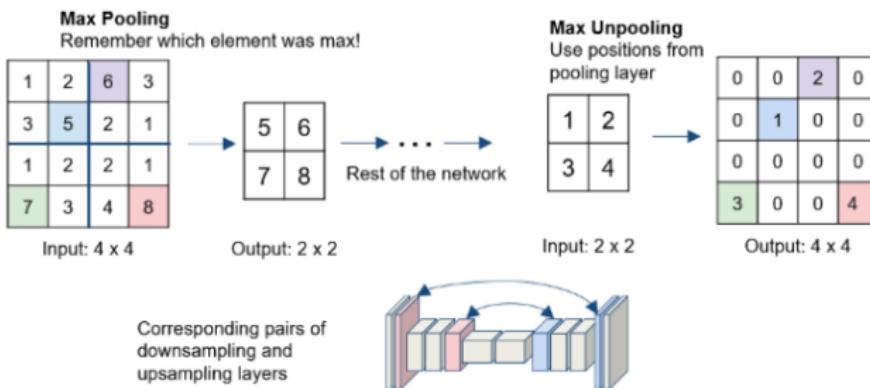


Figure: Max-Unpooling Upsampling

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

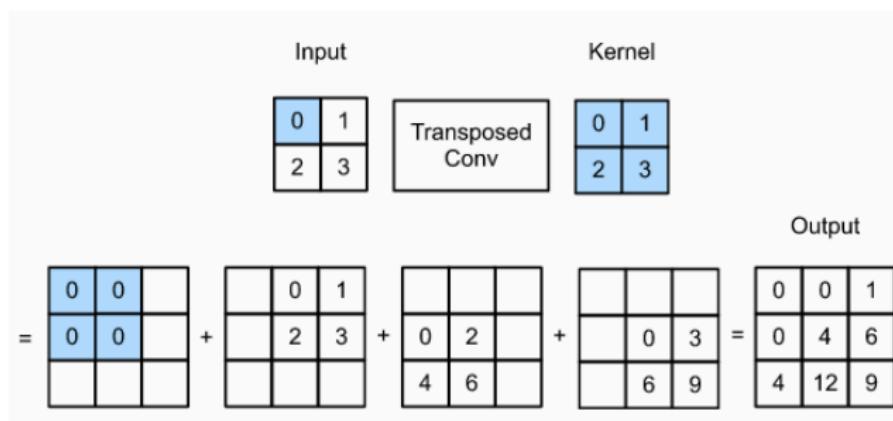


Figure: Transposed convolution with a 2×2 kernel. The shaded portions are a portion of an intermediate tensor as well as the input and kernel tensor elements used for the computation.

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.

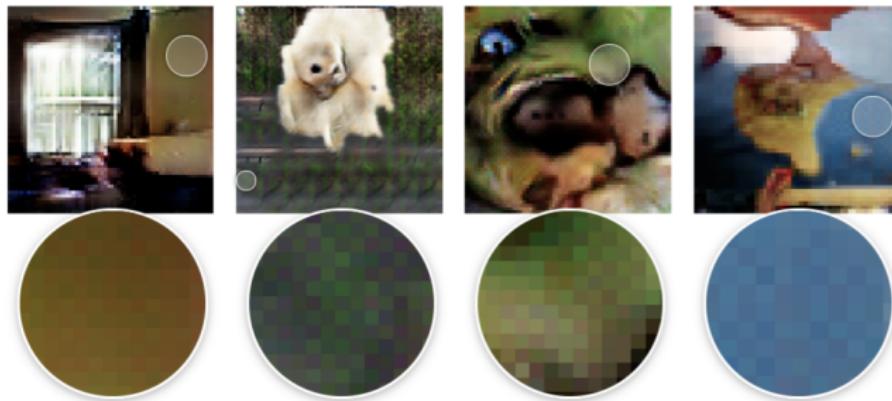


Figure: Checkerboard artifacts effect

Final Notes

Thank You!

Any Question?

References

- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow - Aurélien Géron - 2019 - [Link](#)
- Deep Learning for Vision Systems - Mohamed Elgendi - 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](#)

References



Dongping Tian.

A review on image feature extraction and representation techniques.

International Journal of Multimedia and Ubiquitous Engineering, 8:385–395, 01 2013.



Yandan Wang, John See, Raphael C.-W. Phan, and Yee-Hui Oh.

Efficient spatio-temporal local binary patterns for spontaneous facial micro-expression recognition.

PLOS ONE, 05 2015.



Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola.

Dive into deep learning.

arXiv preprint arXiv:2106.11342, 2021.