Deep Learning and Its Application to Signal and Image Processing and Analysis

CLASS V - SPRING 2021

TAMMY RIKLIN RAVIV, ELECTRICAL AND COMPUTER ENGINEERING

Today's topics

Convolutional Neural Networks (CNN)

Batch Normalization

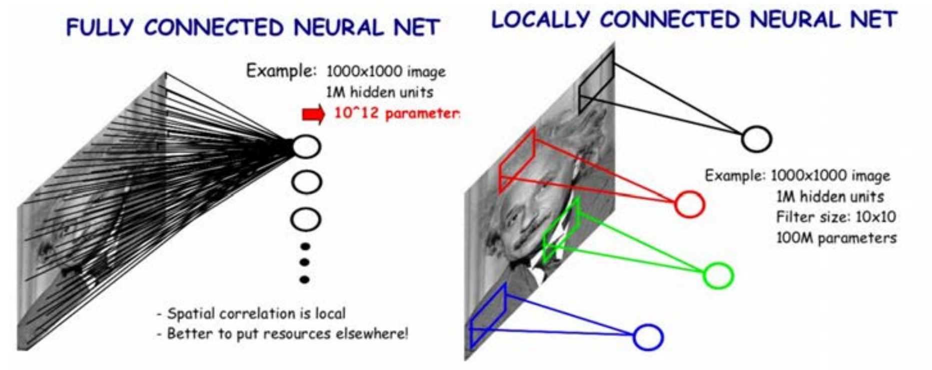
Convolutional Neural Networks

- Standard Convolution , Receptive Field
- Pointwise convolutions | 1×1 convolution
- Dilated convolution | Atrous Convolution
- Deconvolution | Transpose convolution | Fractionally Strided convolution
- Pixel Shuffle convolution for Super-resolution
- Depthwise separable convolution
- Spatially separable convolution

Deep learning book, chapter 9 Goodfellow and Bengio and Courville MIT Press, 2016
https://medium.com/hitchhikers-guide-to-deep-learning/
10-introduction-to-deep-learning-with-computer-vision-types-of-convolutions-atrous-convolutions-3cf142f77bc0

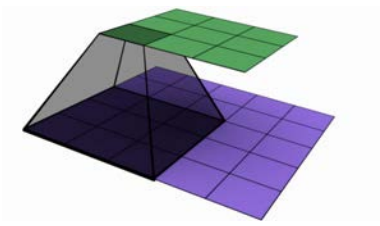
3

Fully Connected vs. locally connected



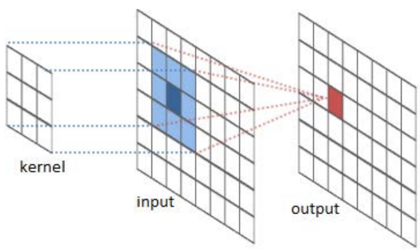
Convolution

- Convolution the process of extracting features from input data using kernels/filters.
- Convolution is a mathematical operation of two functions that produces a third function that expresses how the shape of one is modified by the other



Convolution

Convolution is an operation on two functions of a real-valued argument.



$$s(t) = \int x(a)w(t-a)da$$

$$s(t) = (x * w)t$$

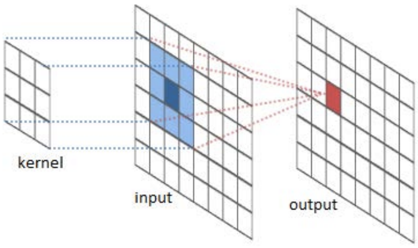
Feature space

Input

Kernel

Convolution

Convolution is an operation on two functions of a real-valued argument.



In fact

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

$$s(t) = (x * w)t$$

Feature space

Input

Kernel

Multi-dimension convolution

$$s(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

Commutative property:

$$s(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

In practice cross correlation is commonly used instead :

$$s(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

2	3	6	9	3
0	1	0	1	5
6	0	4	1	2
0	4	2	1	1
1	3	1	0	0

*

0	1
1	0

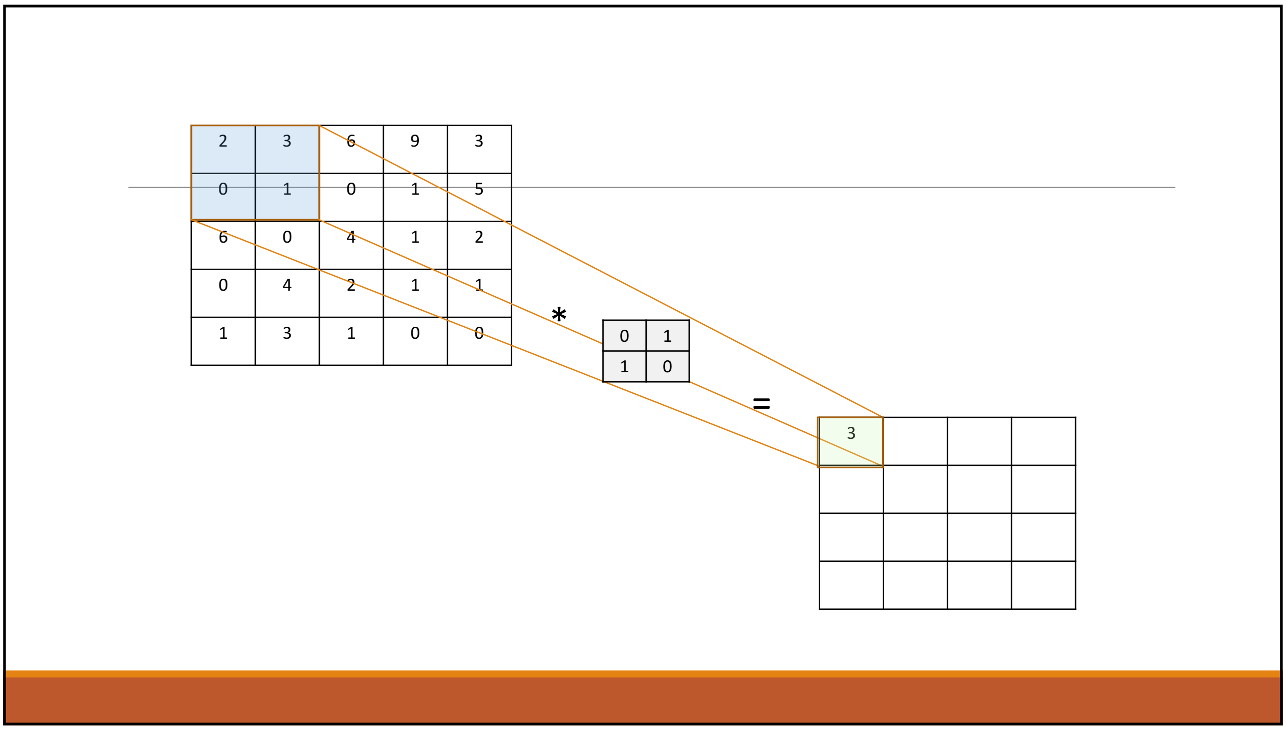
=

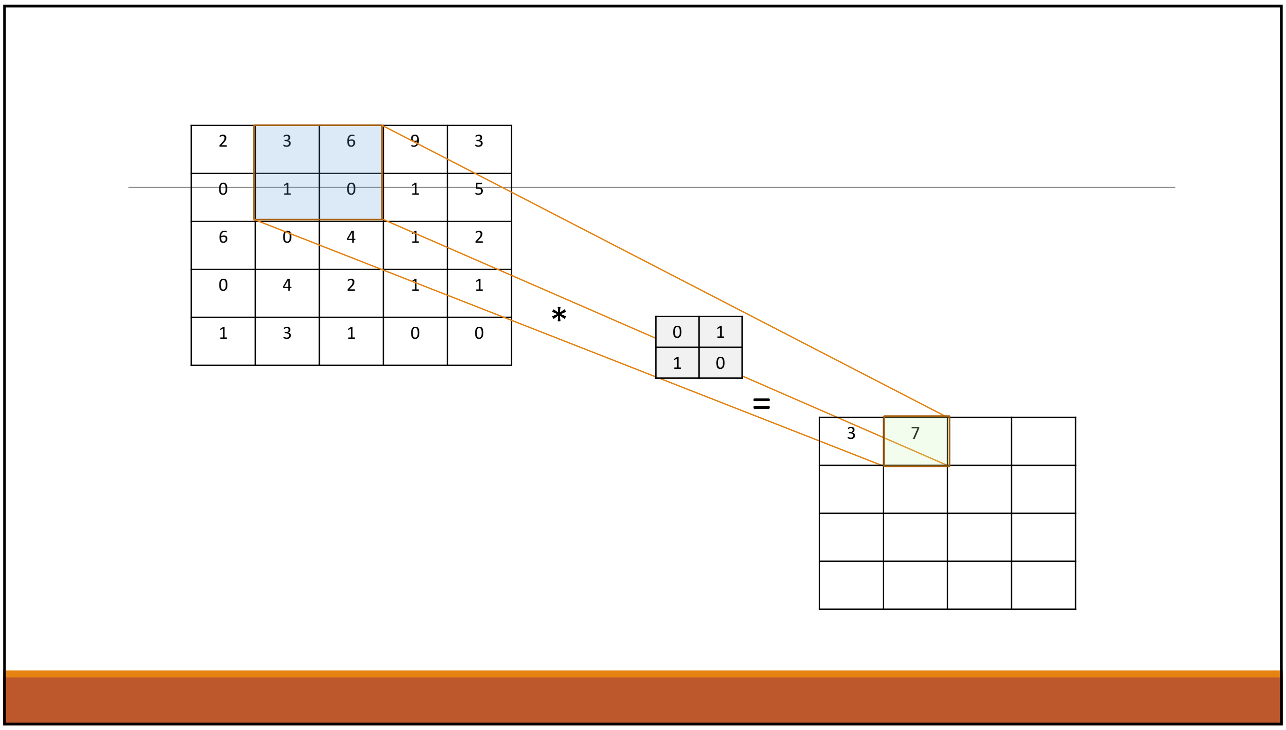
2	3	6	9	3
0	1	0	1	5
6	0	4	1	2
0	4	2	1	1
1	3	1	0	0

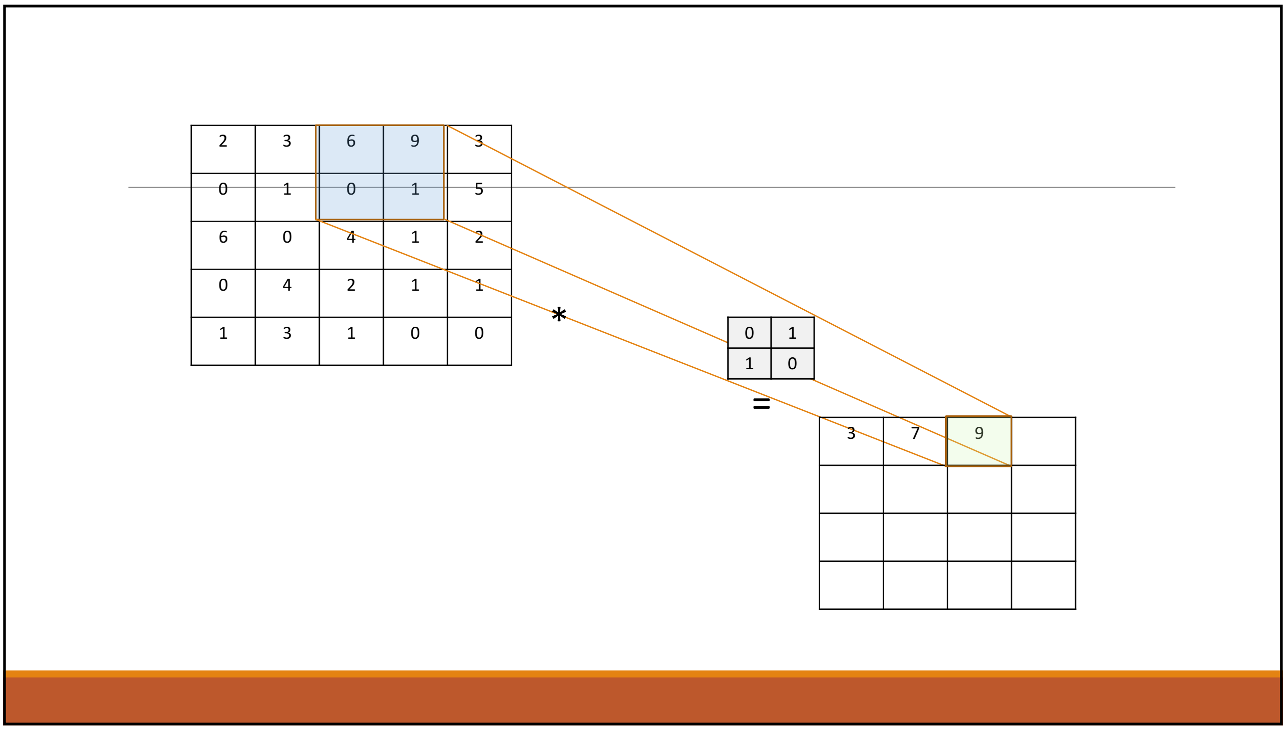
*

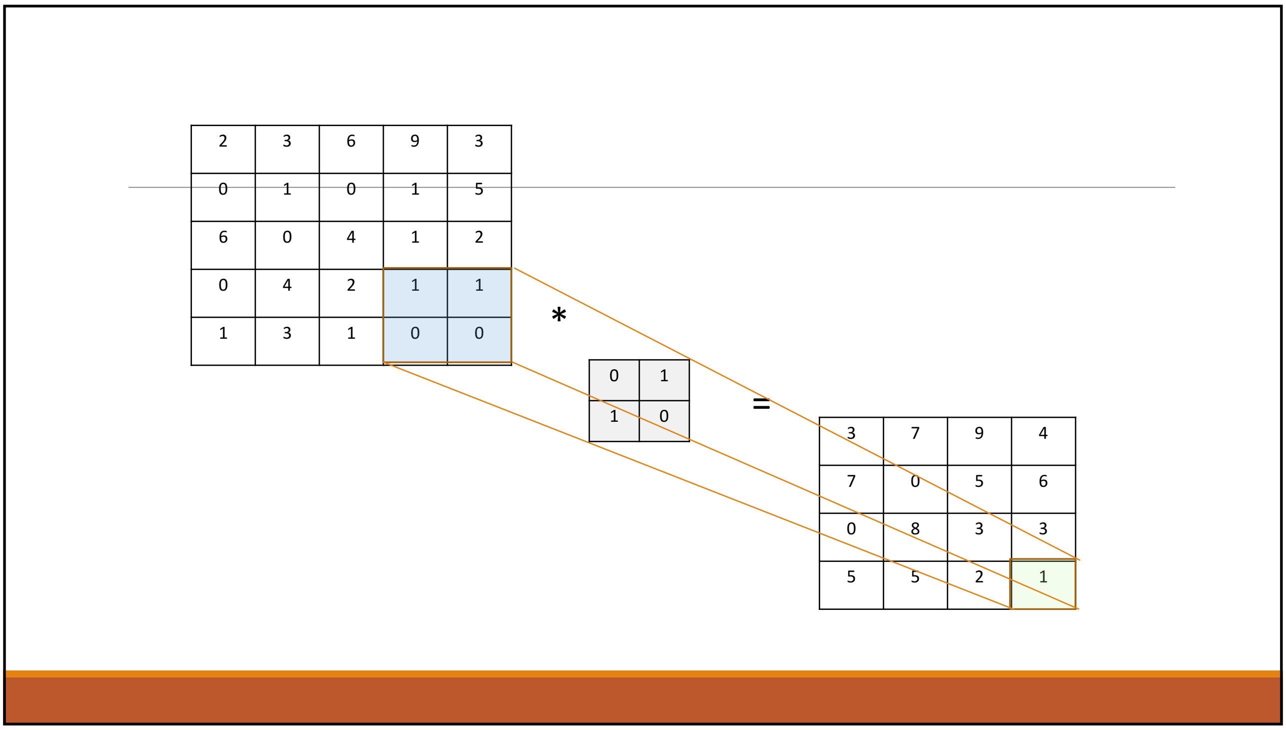
0	1
1	0

=


2 3 6 9 3
0 1 0 1 5
6 0 4 1 2
0 4 2 1 1
1 3 1 0 0
*
0 1
1 0
=
3


2 3 6 9 3
0 1 0 1 5
6 0 4 1 2
0 4 2 1 1
1 3 1 0 0
*
0 1
1 0
=
3 7


2 3 6 9 3
0 1 0 1 5
6 0 4 1 2
0 4 2 1 1
1 3 1 0 0
*
0 1
1 0
=
3 7 9


2 3 6 9 3
0 1 0 1 5
6 0 4 1 2
0 4 2 1 1
1 3 1 0 0
*
0 1
1 0
=
3 7 9 4
7 0 5 6
0 8 3 3
5 5 2 1

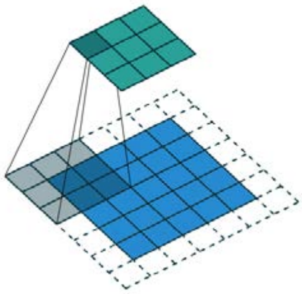
$$\times \begin{array}{|c|c|c|c|c|}\hline 0&0&0&0&0\\\hline 0&1&1&1&0\\\hline 0&1&1&1&0\\\hline 0&1&1&1&0\\\hline 0&0&0&0&0\\\hline\end{array} =$$

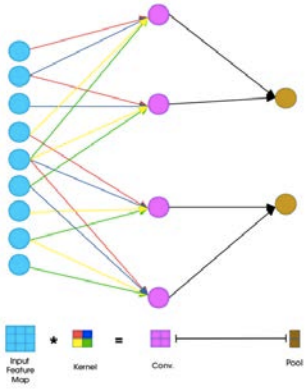
$$\times \begin{array}{|c|c|c|c|c|}\hline 0&0&0&0&0\\\hline 0&1&1&1&0\\\hline 0&1&1&1&0\\\hline 0&1&1&1&0\\\hline 0&0&0&0&0\\\hline\end{array} =$$

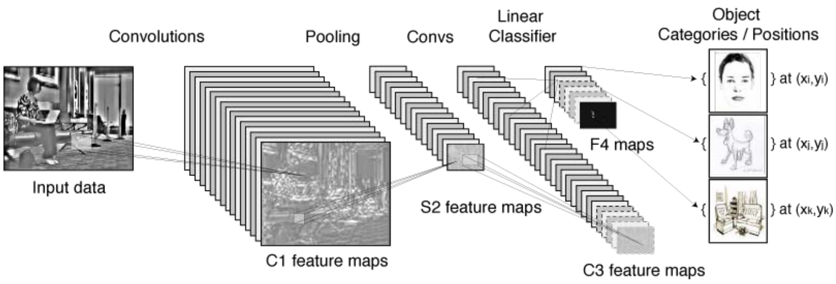
X $\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}$ =

X $\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}$ =

Convolutional Neural Networks







Convolution as a matrix

Discrete convolution can be viewed as multiplication by a matrix.

1. Entries constrained to be equal to other entries.

2. Convolution usually corresponds to a very sparse matrix $m << n$

$$y = h * x = \begin{bmatrix} h_1 & 0 & \dots & 0 & 0 \\ h_2 & h_1 & \dots & \vdots & \vdots \\ h_3 & h_2 & \dots & 0 & 0 \\ \vdots & h_3 & \dots & h_1 & 0 \\ h_{m-1} & \vdots & \dots & h_2 & h_1 \\ h_m & h_{m-1} & \vdots & \vdots & h_2 \\ 0 & h_m & \dots & h_{m-2} & \vdots \\ 0 & 0 & \dots & h_{m-1} & h_{m-2} \\ \vdots & \vdots & \vdots & h_m & h_{m-1} \\ 0 & 0 & 0 & \dots & h_m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

Toeplitz matrix

Convolution – three key ideas

1. Sparse interactions
2. Parameter sharing
3. Equivariant representations

.

Sparse interactions

The kernel is smaller than the input

millions of units -> hundreds of units

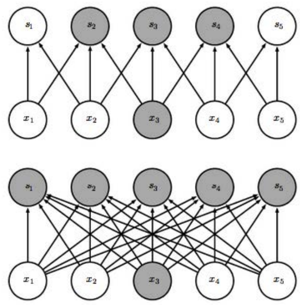
(e.g. pixels -> edges)

reduction of memory capacity

statistical/computational efficiency

.

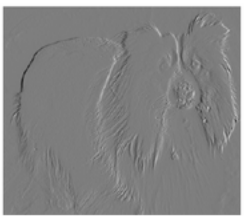
CNN



Traditional NN

A simple example

280



320

Original image

Vertical edge map

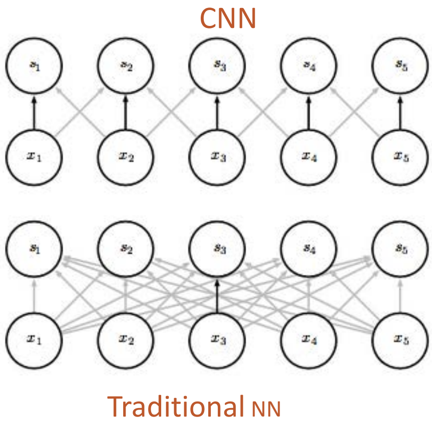
Forward operation

$$O(n \times k)$$

But efficient in terms of storage

Parameter sharing (tied weights)

Parameter sharing:
using the same parameter for more than one function in a model



Equivariance

In the case of convolution, the particular form of parameter sharing causes the layer to have a property called equivariance to translation.

To say a function is equivariant means that if the input changes, the output changes in the same way, i.e. The function $f(x)$ is equivariant to a function g if

$$f(g(x)) = g(f(x))$$

Pooling

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs (e.g. a rectangle neighborhood).

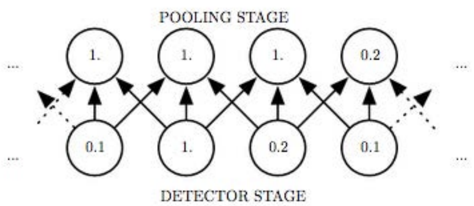
1. Max pooling
2. Average
3. Weighted average (e.g. based on the distance from the central voxel)
4. L2 Norm

Pooling - advantages

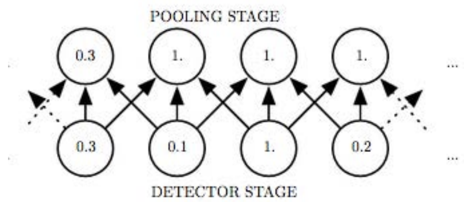
Invariance to small translations of the input (strong prior).

Computational efficiency: fewer pooling units than detector units

Handling inputs of varying size

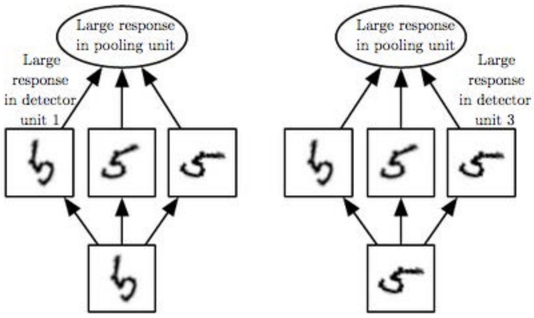


Max pooling

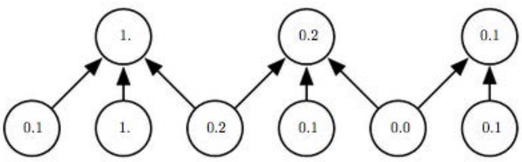


-> shift by one pixel

Max pooling – an example (Invariance to rotation)



Max pooling – an example (reduction of size)



Reduction by a factor of 2

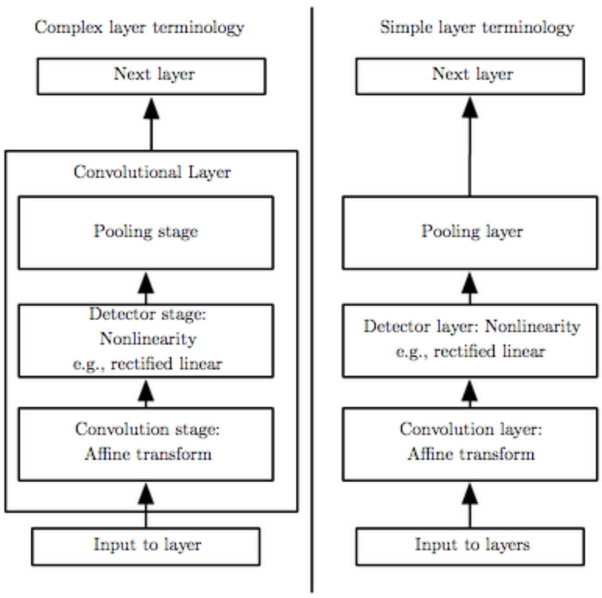
CNN Architecture

A typical layer of a convolutional network consists of three stages:

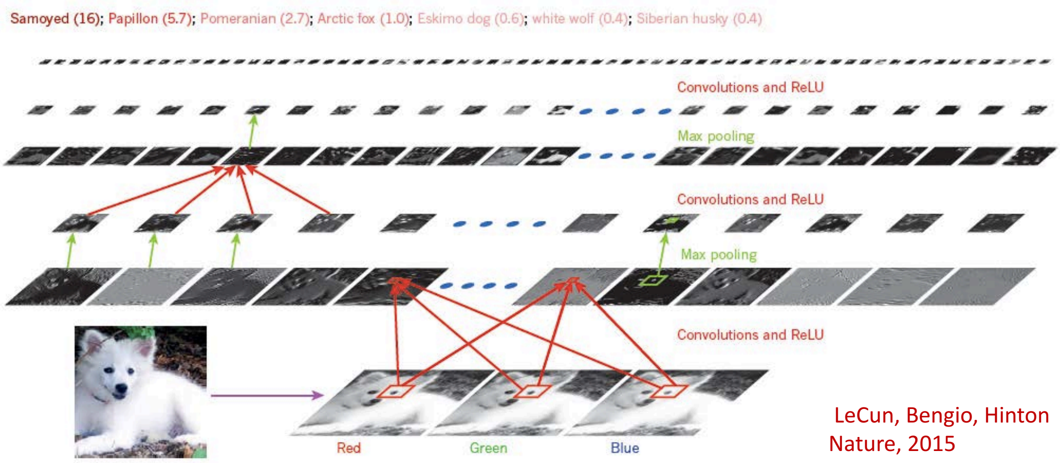
first stage: performs several convolutions in parallel to produce a set of linear activations.

second stage: each linear activation is run through a nonlinear activation function, called the detector stage.

third stage: pooling function is used to modify the output of the layer further



CNN example



LeCun, Bengio, Hinton
Nature, 2015

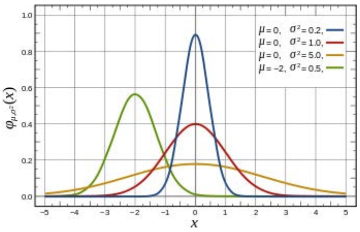
Convolution and Pooling as an Infinitely Strong Prior

Prior probability distribution - a probability distribution over the parameters of a model that encodes our beliefs about what models are reasonable, before we have seen any data.

Priors can be considered weak or strong.

Weak prior – high entropy – e.g. high variance Gaussian dist.

Strong prior – low entropy – e.g. low variance Gaussian dist.



Convolution and Pooling as an Infinitely Strong Prior

CNN = fully connected NN with strong priors

Identical weights – shifted in space

Zero weights

Convolution and Pooling as an Infinitely Strong Prior

Key insights:

1. Convolution and pooling are only useful when the assumptions made by the prior are reasonably accurate.

If not – may cause underfitting

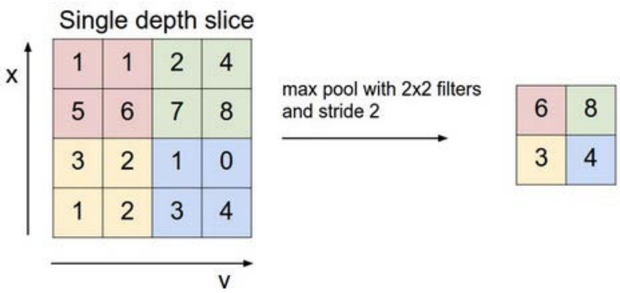
2. We should only compare convolutional models to other convolutional models in benchmarks of statistical learning performance.

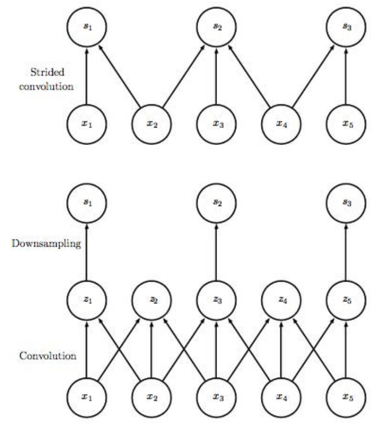
Fully connected NN is permutation invariant.

Variants of the Basic Convolution Function

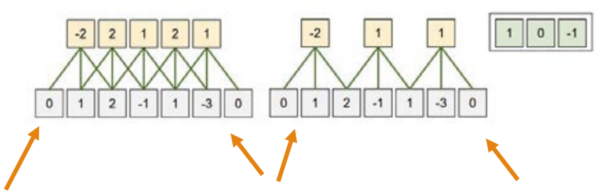
1. Strided convolution
2. Zero padding
3. Unshared convolution
4. Tiled CNN

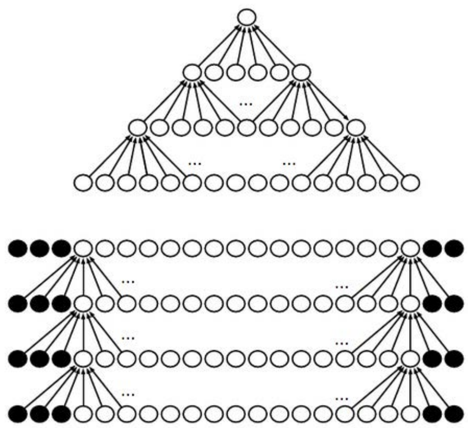
Strided Convolution





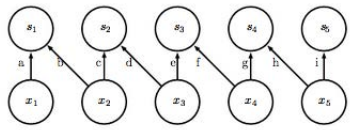
Zero padding



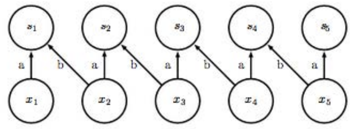


Locally connected, CNN and Fully connected

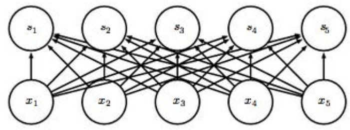
Locally connected



Standard CNN

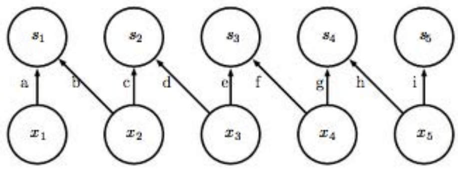


Fully connected

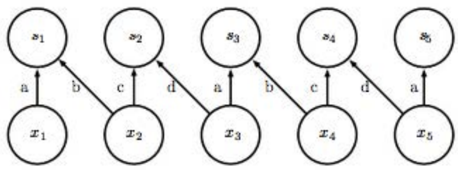


Tiled Convolution

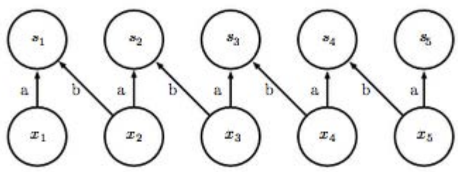
Locally connected
(unshared convolution



Tiled CNN



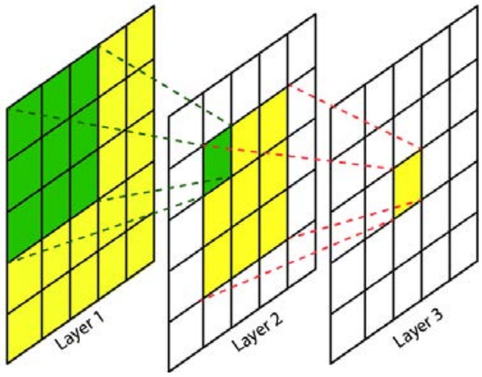
Standard CNN



Receptive field

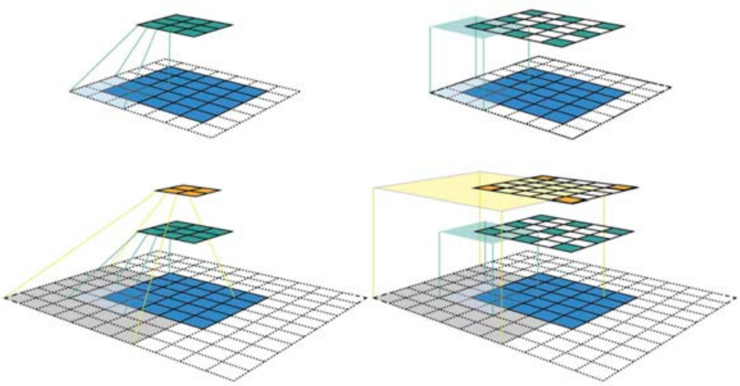
The receptive field is the region in the input space
That a particular CNN's feature is looking at (i.e. affected by).
A receptive field of a feature can be described by its
center location and its **size**.

What is the receptive field of a feature in the 4th layers ?
What is the receptive field of a feature in the Nth layers ?



https://medium.com/hitchhikers-guide-to-deep-learning/10-introduction-to-deep-learning-with-computer-vision-types-of-convolutions-atrous-convolutions-3cf142f77bc0

Math of the Receptive Fields



https://rubikscode.net/2020/05/18/receptive-field-arithmetic-for-convolutional-neural-networks/

Math of the Receptive Fields

Size of the **output** feature map for each layer:

number of the features in layer *i-1* | padding | kernel

$$n_i = \frac{n_{i-1} + 2p - k}{s} + 1$$

number of the features in layer *i* | stride

Cumulative stride $$j_i = j_{i-1} * s$$

https://rubikscode.net/2020/05/18/receptive-field-arithmetic-for-convolutional-neural-networks/

Math of the Receptive Fields

size of the receptive field $$r_i = r_{i-1} + (k-1)j_{i-1}$$

size of the receptive field
General formula
$$r_i = r_{i-1} + (k-1)\prod_{k=1}^{i-1} s_k$$

receptive field's center $$start_i = start_{i-1} + (\frac{k-1}{2} - p)j_{i-1}$$

https://rubikscode.net/2020/05/18/receptive-field-arithmetic-for-convolutional-neural-networks/

Math of the Receptive Fields

Initial values:

n = image size

r = 1

j = 1

start = 0.5

Receptive field calculator

https://fomoro.com/research/article/receptive-field-calculator#

Math of the Receptive Fields

n : number of features
r : receptive field size
j : jump (distance between two consecutive features)
$start$: center coordinate of the first feature

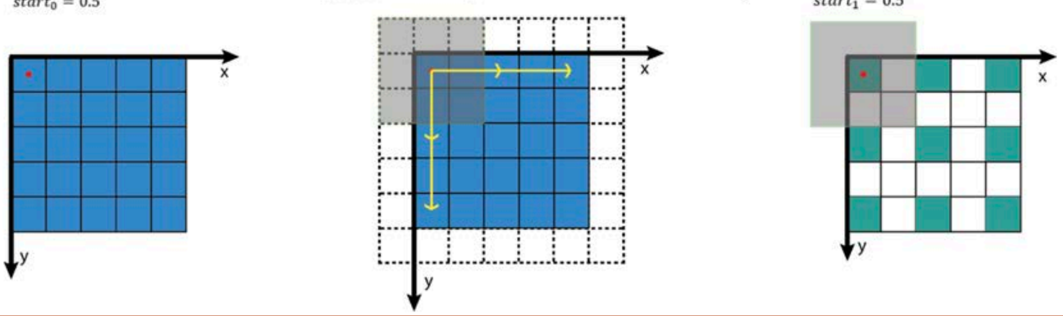
k : convolution kernel size
p : convolution padding size
s : convolution stride size

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$
$$j_{out} = j_{in} * s$$
$$r_{out} = r_{in} + (k - 1) * j_{in}$$
$$start_{out} = start_{in} + \left(\frac{k-1}{2} - p\right) * j_{in}$$

Layer 0: $n_0 = 5;\ r_0 = 1;\ j_0 = 1;\ start_0 = 0.5$

Conv1: $k_1 = 3;\ p_1 = 1;\ s_1 = 2$

Layer 1: $n_1 = 3;\ r_1 = 3;\ j_1 = 2;\ start_1 = 0.5$

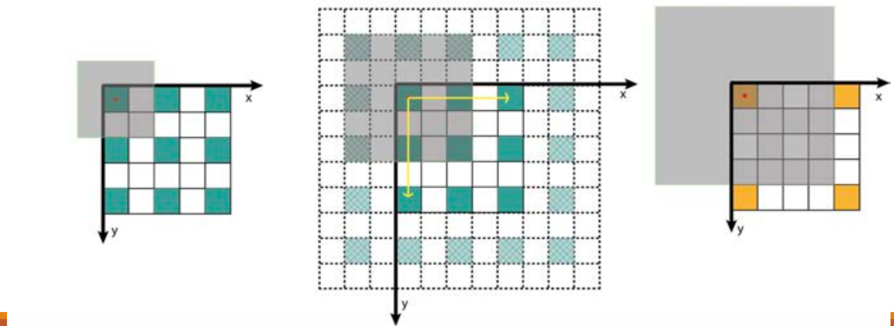


Math of the Receptive Fields

$n:$ number of features
$r:$ receptive field size
$j:$ jump (distance between two consecutive features)
$start:$ center coordinate of the first feature

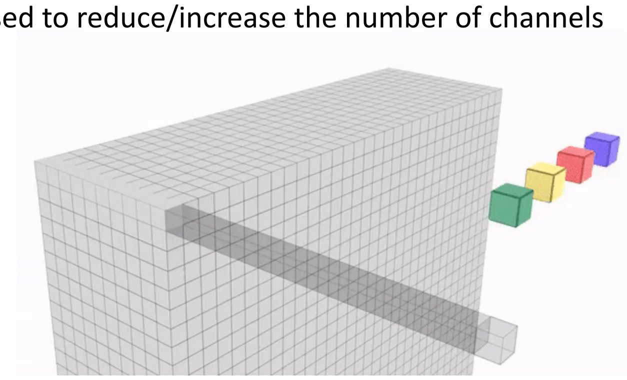
$k:$ convolution kernel size
$p:$ convolution padding size
$s:$ convolution stride size

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$
$$j_{out} = j_{in} * s$$
$$r_{out} = r_{in} + (k-1) * j_{in}$$
$$start_{out} = start_{in} + \left(\frac{k-1}{2} - p\right) * j_{in}$$



Pointwise Convolutions | 1x1 Convolutions:

1×1 convolutional kernels are used to reduce/increase the number of channels



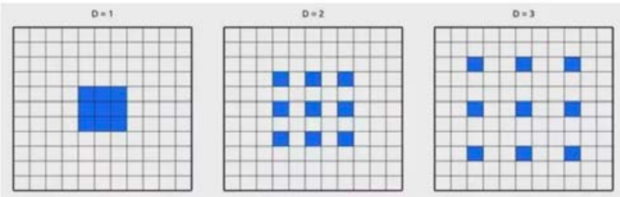
https://medium.com/hitchhikers-guide-to-deep-learning/10-introduction-to-deep-learning-with-computer-vision-types-of-convolutions-atrous-convolutions-3cf142f77bc0

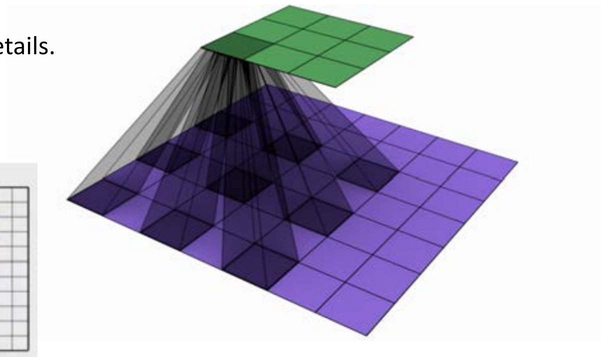
Dilated convolutions | Atrous Convolutions:

Dilated convolution – increase receptive field without increasing the depth & computational cost.

Understand the overall picture rather than finer details.

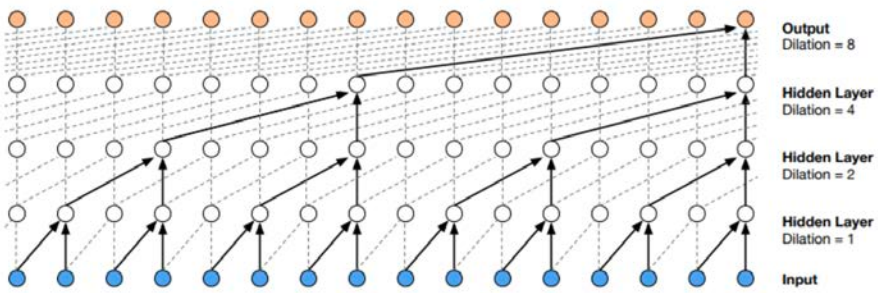
Dilation rates





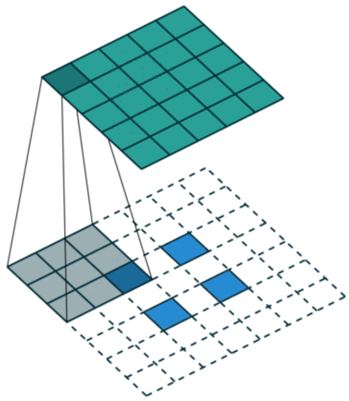
Dilated convolutions | Atrous Convolutions:

Google's Wavenet

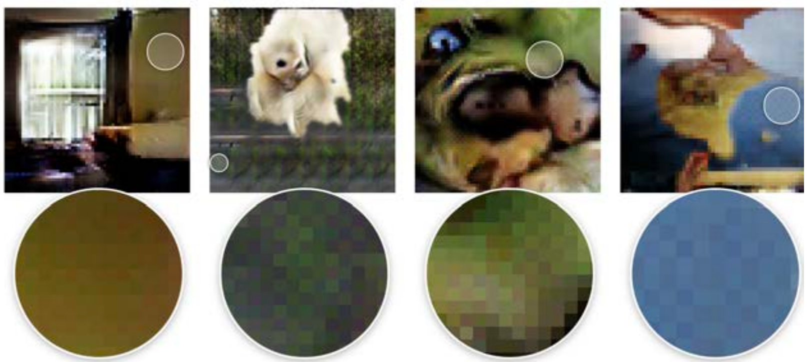


Deconvolution | Transpose convolution | Fractionally Strided Convolution:

- Image data correlate
- Convolution learns these correlations
- Deconvolution “removes” pixel-wise and channel-wise correlations before the data is fed to subsequent layers
- Deconvolution: spread out the pixels/features of the original image/layer and pad the spaces with some values
- May lead to **checkerboard artifacts**



Checkerboard artifacts

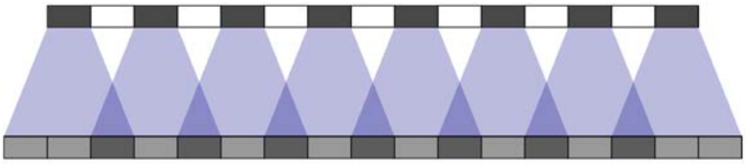


Radford, et al., 2015 [1] Salimans et al., 2016 [2] Donahue, et al., 2016 [3] Dumoulin, et al., 2016 [4]

https://distill.pub/2016/deconv-checkerboard/

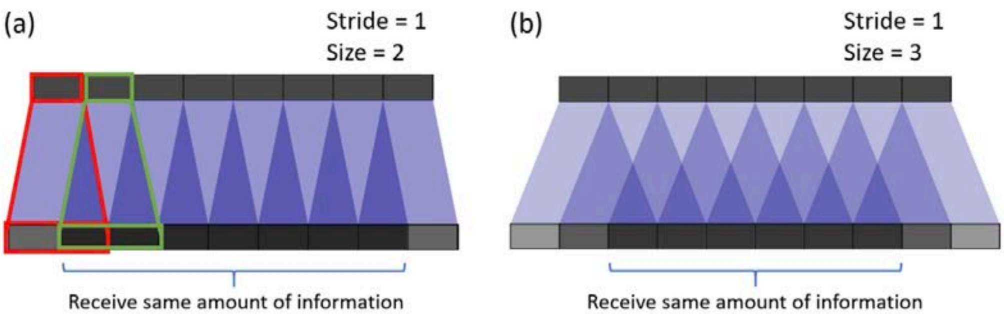
Checkerboard artifacts

- Checkerboard artifacts result from an “uneven overlap” of transposed convolution.
- Such overlap covers some places more than others.
- Deconvolution has uneven overlap when the kernel size is not divisible by the stride.



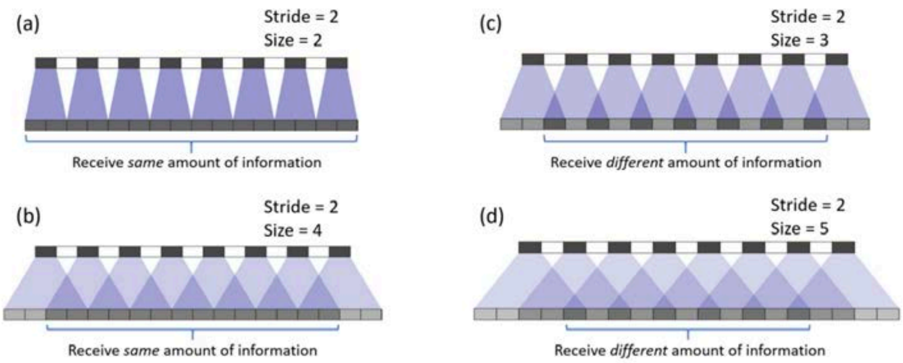
https://distill.pub/2016/deconv-checkerboard/

Checkerboard artifacts



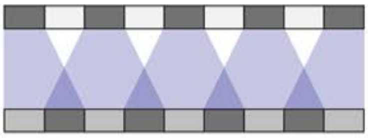
https://distill.pub/2016/deconv-checkerboard/

Checkerboard artifacts

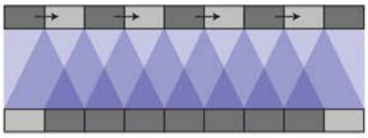


https://distill.pub/2016/deconv-checkerboard/

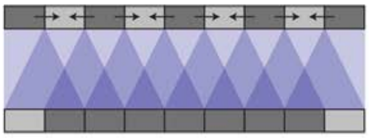
Checkboard Artifacts –possible solutions



Deconvolution



NN-Resize Convolution

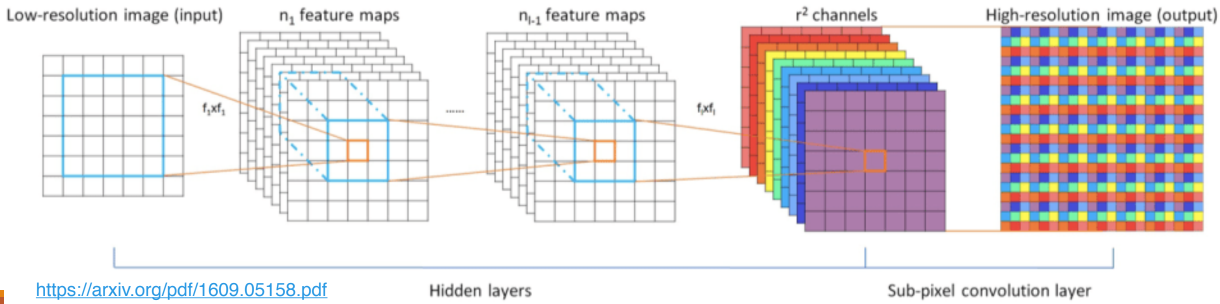


Bilinear-Resize Convolution

Pixel Shuffle convolution for high resolution

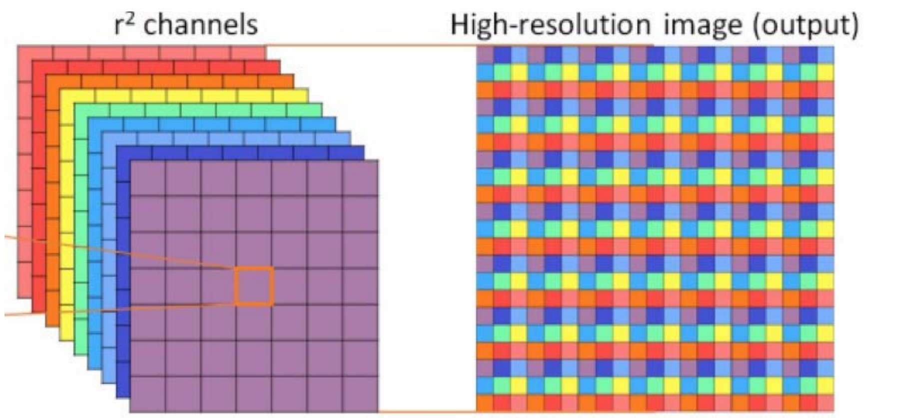
Take the input image(MxNxC), run over a 1x1 kernel to extract the RxR number of channels.

From these 1,2,3,…, R^2 channels take one pixel at a time from each consecutive layer then shuffle them to get the output.



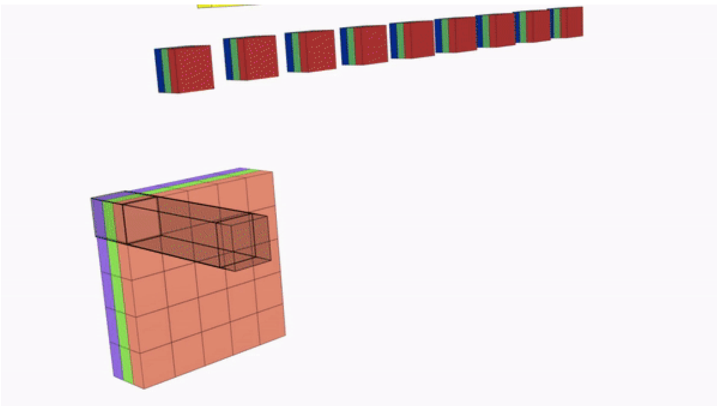
https://arxiv.org/pdf/1609.05158.pdf

Pixel Shuffle



https://medium.com/hitchhikers-guide-to-deep-learning/10-introduction-to-deep-learning-with-computer-vision-types-of-convolutions-atrous-convolutions-3cf142f77bc0

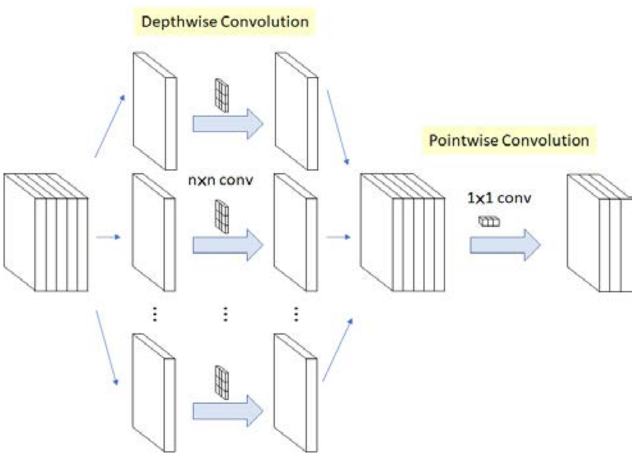
Pixel Shuffle Convolution



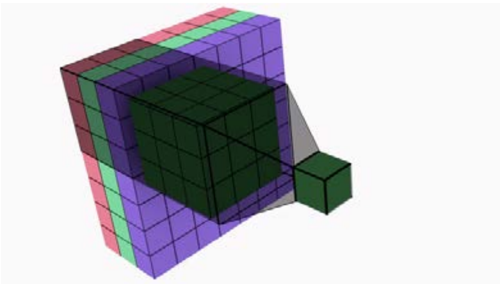
https://medium.com/hitchhikers-guide-to-deep-learning/10-introduction-to-deep-learning-with-computer-vision-types-of-convolutions-atrous-convolutions-3cf142f77bc0

Depthwise Convolution

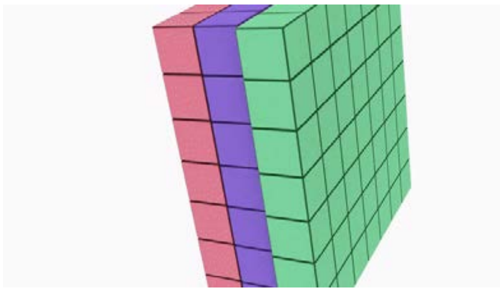
- Split NxMxC input image into C images.
- Convolve to get multiple feature maps.
- Use 1x1xC pointwise convolutions to get the desired number of channels.



Depthwise Convolution



Standard Convolution



Depthwise Convolution

https://medium.com/hitchhikers-guide-to-deep-learning/10-introduction-to-deep-learning-with-computer-vision-types-of-convolutions-atrous-convolutions-3cf142f77bc0

Depthwise convolution

Image size $N \times M \times C$

kernel size $K \times K$

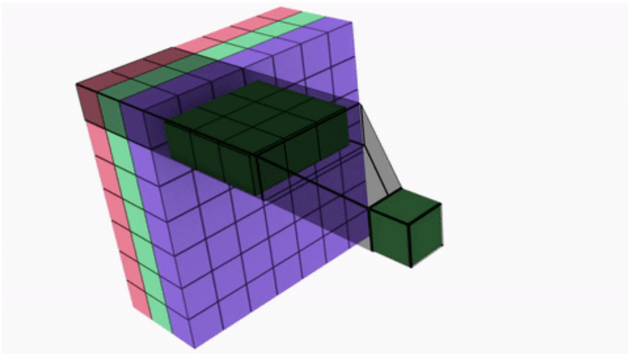
Output size O

$$(N \times M \times C) \times [(K \times K \times C) \times O] \rightarrow (N \times M \times O)$$

$$C \times (N \times M) \times [C \times (K \times K \times 1) + (1 \times 1 \times C) \times O] \rightarrow (N \times M \times O)$$

Used by Architectures such as Inception, Xception and Mobile Net

Spatially Separable Convolution



https://medium.com/hitchhikers-guide-to-deep-learning/10-introduction-to-deep-learning-with-computer-vision-types-of-convolutions-atrous-convolutions-3cf142f77bc0

Data Types

	Single Channel	Multi Channel
1D	Audio waveform	Skeleton animation data
2D	Fourier transform of Audio data	Color image data
3D 4D	Volumetric data – e.g. CT scans Heart scans	Color video data Multi-modal MRI

Batch Normalization

- Introduced by two researchers at Google, Sergey Ioffe and Christian Szegedy in their paper '*Batch Normalization: **Accelerating Deep Network Training by Reducing Internal Covariate Shift***' in 2015.
- They showed that batch normalization improved the top result of ImageNet (2014) by a significant margin using only 7% of the training steps
- Today, Batch Normalization is used in almost all CNN architectures.

Internal Covariate Shift

https://learnopencv.com/batch-normalization-in-deep-networks/

Internal Covariate Shift

https://learnopencv.com/batch-normalization-in-deep-networks/

Internal Covariate Shift

https://learnopencv.com/batch-normalization-in-deep-networks/

Internal Covariate Shift

- **Covariate shift:** when the mini-batches have images that are not uniformly sampled from the entire distribution
- **Solution** for the input layer is to randomize the data before creating mini-batches.

What about the hidden layers?

- In a neural network, each hidden unit's input distribution changes every time there is a parameter update in the previous layer.
- Called **internal covariate shift**
- Makes training slow and requires a very small learning rate and a good parameter initialization
- Solution: Batch Normalization

https://learnopencv.com/batch-normalization-in-deep-networks/

Other benefits to BatchNorm

- Allows higher learning rate without vanishing or exploding gradients.
- Have a regularizing effect such that the network improves its generalization properties
- The network becomes more robust to different initialization schemes and learning rates.

Batch Normalization

- Batch normalization is achieved through a normalization step that fixes the means and variances of each layer's inputs
- Ideally, the normalization would be conducted over the entire training set (obviously impractical with stochastic optimization methods).
- In practice, normalization is restrained to each mini-batch in the training process.

Batch Normalization

CS231n Convolutional Neural Networks for Visual Recognition

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

The two parameters γ, β are learned through the training process.

Batch Normalization -algorithm

Use B to denote a mini-batch of size m of the entire training set.

The empirical mean and variance of *B* could thus be denoted as

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i, \text{ and } \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

For a layer of the network with *d*-dimensional input, $x = (x^{(1)}, \ldots, x^{(d)})$,

each dimension of its input is then normalized (i.e. re-centered and re-scaled) separately,

$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)^2} + \epsilon}}$, where $k \in [1, d]$ and $i \in [1, m]$; $\mu_B^{(k)}$ and $\sigma_B^{(k)^2}$ are the per-dimension mean and variance, respectively.

Batch Normalization –Cont.

To restore the representation power of the network, a transformation step then follows as

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)},$$

where the parameters $\gamma^{(k)}$ and $\beta^{(k)}$ are subsequently learned in the optimization process.

Inference with Batch-Normalized Networks

During the training stage, the normalization steps depend on the mini-batches to ensure efficient and reliable training.

The normalization step in the inference stage is computed with the population statistics such that the output could depend on the input in a deterministic manner.

$$E[x^{(k)}] = E_B[\mu_B^{(k)}], \text{ and } \mathrm{Var}[x^{(k)}] = \frac{m}{m-1} E_B[{\sigma_B^{(k)}}^2]$$

The BN transform in the inference step thus becomes

$$y^{(k)} = BN^{\text{inf}}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)}) = \frac{\gamma^{(k)}}{\sqrt{\mathrm{Var}[x^{(k)}] + \epsilon}} x^{(k)} + \left(\beta^{(k)} - \frac{\gamma^{(k)} E[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}] + \epsilon}}\right)$$

Since the parameters are fixed in this transformation, the batch normalization procedure is essentially applying a linear transform to the activation.