Neural Networks and Deep Learning

Charu C. Aggarwal

Chapter 8:
Convolutional Neural Networks

Presented by:
Aref Moqadam Mehr

Table of content

Table of content

8.1 Introduction

- Grid-Structured Data Representation
  - Example: 2D Images
- Have strong spatial dependencies in local regions.
  - Example: adjacent location in image often have similar colors
- Translation Invariance
  - Example: A banana has the same interpretation whether it's at the top or bottom of an image.

Table of content

A bit of History

- Hubel and Wiesel studied cat’s visual cortex (1959)
- LeNet-5 for recognizing hand-written digits on checks (1998)
- ImageNet contests played an important role in increasing the prominence of CNNs
- since 2012, CNNs are consistent winner of this challenge
- In 2012 AlexNet succeeded in this challenge by a large margin

Table of content

Computer Vision

- Computer vision is a rapidly growing field thanks to deep learning methods.
- Problems includes:
  - Image Classification
  - Object Detection
  - Neural Style Transfer

64x64

Cat? (0/1)

Computer Vision

- Computer vision is a rapidly growing field thanks to deep learning methods.
- Problems includes:
  - Image Classification
  - Object Detection
  - Neural Style Transfer

Computer Vision

- Computer vision is a rapidly growing field thanks to deep learning methods.
- Problems includes:
  - Image Classification
  - Object Detection
  - Neural Style Transfer

Computer Vision

- For example, a 1000x1000 image will represent 3 million feature/input to the full connected neural network. If the following hidden layer contains 1000, then we will want to learn weights of the shape [1000, 3 million] which is 3 billion parameter only in the first layer and thats so computationally expensive!

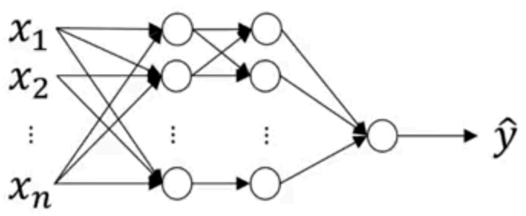


Table of content

- 8.2 Basic Structures
    - Edge Detection
    - Padding
    - Stride
    - Convolution Over Volumes
    - One Layer of Convolution Network
    - Typical Setting
    - Simple CNN
    - Pooling
    - Fully Connected Network
    - The Interleaving between layers
    - Why Convolution Neural Networks?

8.2 Basic Structures

- Image is made of set pixels.
- each pixels contains the intensity of the specified location.
- Image is usually represented as a Metric with three dimensions: *width, height,* and *color channel*

Table of content

Edge Detection

$$
\begin{array}{|c|c|c|c|c|c|}
\hline
3 & 0 & 1 & 2 & 7 & 4 \\ \hline
1 & 5 & 8 & 9 & 3 & 1 \\ \hline
2 & 7 & 2 & 5 & 1 & 3 \\ \hline
0 & 1 & 3 & 1 & 7 & 8 \\ \hline
4 & 2 & 1 & 6 & 2 & 8 \\ \hline
2 & 4 & 5 & 2 & 3 & 9 \\ \hline
\end{array}
\quad * \quad
\begin{array}{|c|c|c|}
\hline
1 & 0 & -1 \\ \hline
1 & 0 & -1 \\ \hline
1 & 0 & -1 \\ \hline
\end{array}
\quad = \quad
\begin{array}{|c|c|c|c|}
\hline
 & & & \\ \hline
 & & & \\ \hline
 & & & \\ \hline
 & & & \\ \hline
\end{array}
$$

Edge Detection

$$\begin{bmatrix} \boxed{\begin{matrix} 3 & 0 & 1 \\ 1 & 5 & 8 \\ 2 & 7 & 2 \end{matrix}} & \begin{matrix} 2 & 7 & 4 \\ 9 & 3 & 1 \\ 5 & 1 & 3 \end{matrix} \\ \begin{matrix} 0 & 1 & 3 \\ 4 & 2 & 1 \\ 2 & 4 & 5 \end{matrix} & \begin{matrix} 1 & 7 & 8 \\ 6 & 2 & 8 \\ 2 & 3 & 9 \end{matrix} \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -5 & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

- 3*1 + 0*0 + 1*(-1) + 1*1 + 5*0 + 8*(-1) + 2*1 + 7*0 + 2*(-1) = -5

Edge Detection

$$\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -5 & -4 & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

Edge Detection

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

$*$

1	0	-1
1	0	-1
1	0	-1

$=$

-5	-4	0	

Edge Detection

$$
\begin{bmatrix}
3 & 0 & 1 & 2 & 7 & 4 \\
1 & 5 & 8 & 9 & 3 & 1 \\
2 & 7 & 2 & 5 & 1 & 3 \\
0 & 1 & 3 & 1 & 7 & 8 \\
4 & 2 & 1 & 6 & 2 & 8 \\
2 & 4 & 5 & 2 & 3 & 9
\end{bmatrix}
*
\begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1
\end{bmatrix}
=
\begin{bmatrix}
-5 & -4 & 0 & 8 \\
 & & & \\
 & & & \\
 & & &
\end{bmatrix}
$$

Edge Detection

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

*

1	0	-1
1	0	-1
1	0	-1

=

-5	-4	0	8
-10			

Edge Detection

$$\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & \boxed{1} & \boxed{7} & \boxed{8} \\ 4 & 2 & 1 & \boxed{6} & \boxed{2} & \boxed{8} \\ 2 & 4 & 5 & \boxed{2} & \boxed{3} & \boxed{9} \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{bmatrix}$$

Edge Detection

- Formal Definition:

$$h_{ijp}^{(q+1)} = \sum_{r=1}^{F_q} \sum_{s=1}^{F_q} \sum_{k=1}^{d_q} w_{rsk}^{(p,q)} h_{i+r-1,j+s-1,k}^{(q)} \quad \forall i \in \{1 \ldots, L_q - F_q + 1\}$$
$$\forall j \in \{1 \ldots B_q - F_q + 1\}$$
$$\forall p \in \{1 \ldots d_{q+1}\}$$

- i,j,k indicate the position along *height, width,* and *depth*
- *q* corresponds to the *q-th* layer of network.
- h^q represent the value of the *q-th* layer.

Edge Detection

- Vertical Edge Detection

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

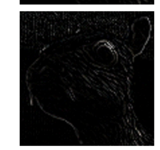
Edge Detection

- We can achieve different results from different kernels

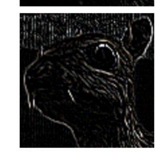
Edge detection

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

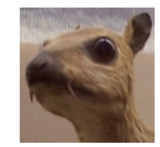


Edge Detection

- We can achieve different results from different kernels

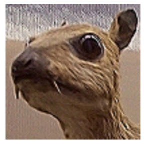
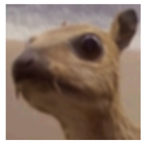
Identity

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$



Edge Detection

- We can achieve different results from different kernels

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Edge Detection

- The challenge is to find right *weights*

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

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Understanding Convolutions

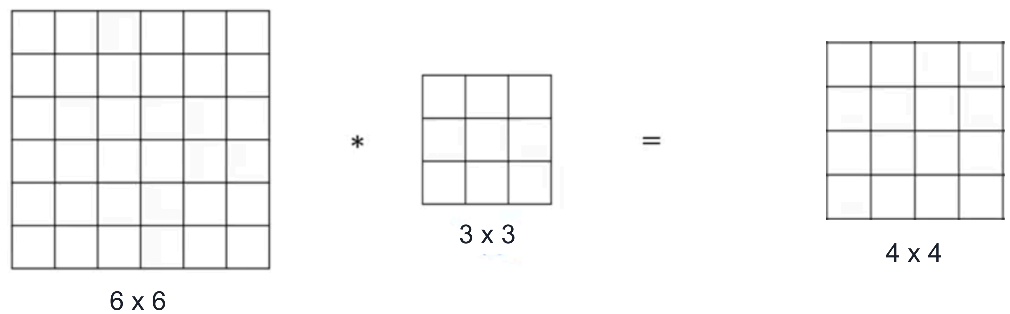
- Sparse connectivity because we are creating a feature from a region in the input volume of the size of the filter.
  - Trying to explore smaller regions of the image to find shapes.
- Shared weights because we use the same filter across entire spatial volume.
  - Interpret a shape in various parts of the image in the same way.

Table of content

Padding

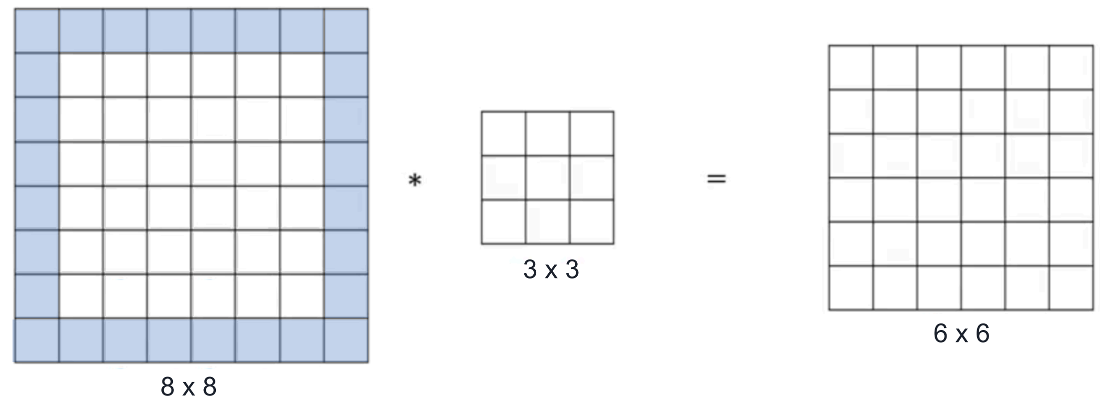
- The convolution operation reduces the size of the *(q + 1)th* layer in comparison with the size of the *q-th* layer.
    - This type of reduction in size is not desirable in general, because it tends to lose some information along the borders of the image (or of the feature map, in the case of hidden layers).
- This problem can be resolved by using padding.

Padding



- Input size: *n*
- Kernel size: *f*
- Output size: $o = n - f + 1$

Padding



- Output size: o = n + 2p - f + 1

Padding

- Terminology for two types of convolution layers:
  - “Valid” Convolution layers: No padding ($p = 0$)
  - “Same” Convolution layers: Input size equals output size ($p = \frac{f-1}{2}$)

Table of content

Stride

$$
\begin{bmatrix} 2 & 3 & 7 & 4 & 6 & 2 & 9 \\ 6 & 6 & 9 & 8 & 7 & 4 & 3 \\ 3 & 4 & 8 & 3 & 8 & 9 & 7 \\ 7 & 8 & 3 & 6 & 6 & 3 & 4 \\ 4 & 2 & 1 & 8 & 3 & 4 & 6 \\ 3 & 2 & 4 & 1 & 9 & 8 & 3 \\ 0 & 1 & 3 & 9 & 2 & 1 & 4 \end{bmatrix} * \begin{bmatrix} 3 & 4 & 4 \\ 1 & 0 & 2 \\ -1 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 91 & & \\ & & \\ & & \end{bmatrix}
$$

Stride

$$\begin{array}{|c|c|c|c|c|c|c|}\hline 2 & 3 & 7 & 4 & 6 & 2 & 9 \\ \hline 6 & 6 & 9 & 8 & 7 & 4 & 3 \\ \hline 3 & 4 & 8 & 3 & 8 & 9 & 7 \\ \hline 7 & 8 & 3 & 6 & 6 & 3 & 4 \\ \hline 4 & 2 & 1 & 8 & 3 & 4 & 6 \\ \hline 3 & 2 & 4 & 1 & 9 & 8 & 3 \\ \hline 0 & 1 & 3 & 9 & 2 & 1 & 4 \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline 3 & 4 & 4 \\ \hline 1 & 0 & 2 \\ \hline -1 & 0 & 3 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 91 & 100 & \\ \hline & & \\ \hline & & \\ \hline \end{array}$$

Stride

$$\begin{bmatrix} 2 & 3 & 7 & 4 & 6 & 2 & 9 \\ 6 & 6 & 9 & 8 & 7 & 4 & 3 \\ 3 & 4 & 8 & 3 & 8 & 9 & 7 \\ 7 & 8 & 3 & 6 & 6 & 3 & 4 \\ 4 & 2 & 1 & 8 & 3 & 4 & 6 \\ 3 & 2 & 4 & 1 & 9 & 8 & 3 \\ 0 & 1 & 3 & 9 & 2 & 1 & 4 \end{bmatrix} * \begin{bmatrix} 3 & 4 & 4 \\ 1 & 0 & 2 \\ -1 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 91 & 100 & 83 \\ & & \\ & & \end{bmatrix}$$

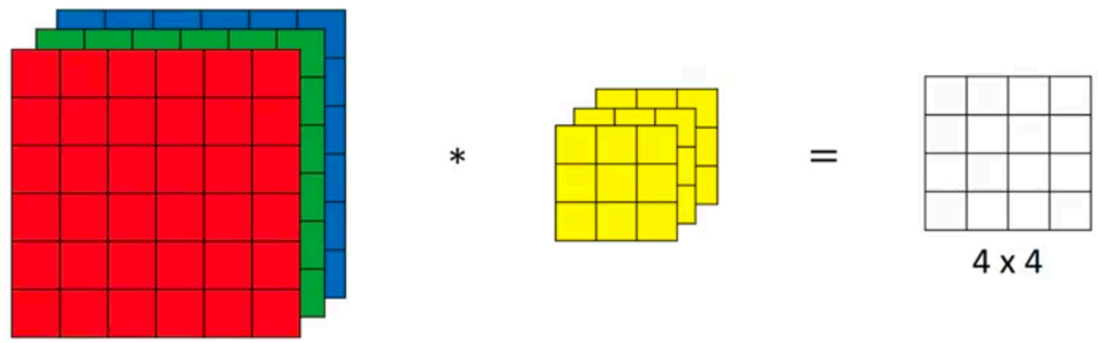
Stride

$$
\begin{array}{|c|c|c|c|c|c|c|}
\hline
2 & 3 & 7 & 4 & 6 & 2 & 9 \\ \hline
6 & 6 & 9 & 8 & 7 & 4 & 3 \\ \hline
3 & 4 & 8 & 3 & 8 & 9 & 7 \\ \hline
7 & 8 & 3 & 6 & 6 & 3 & 4 \\ \hline
4 & 2 & 1 & 8 & 3 & 4 & 6 \\ \hline
3 & 2 & 4 & 1 & 9 & 8 & 3 \\ \hline
0 & 1 & 3 & 9 & 2 & 1 & 4 \\ \hline
\end{array}
\quad * \quad
\begin{array}{|c|c|c|}
\hline
3 & 4 & 4 \\ \hline
1 & 0 & 2 \\ \hline
-1 & 0 & 3 \\ \hline
\end{array}
\quad = \quad
\begin{array}{|c|c|c|}
\hline
91 & 100 & 83 \\ \hline
 & & \\ \hline
 & & \\ \hline
\end{array}
$$

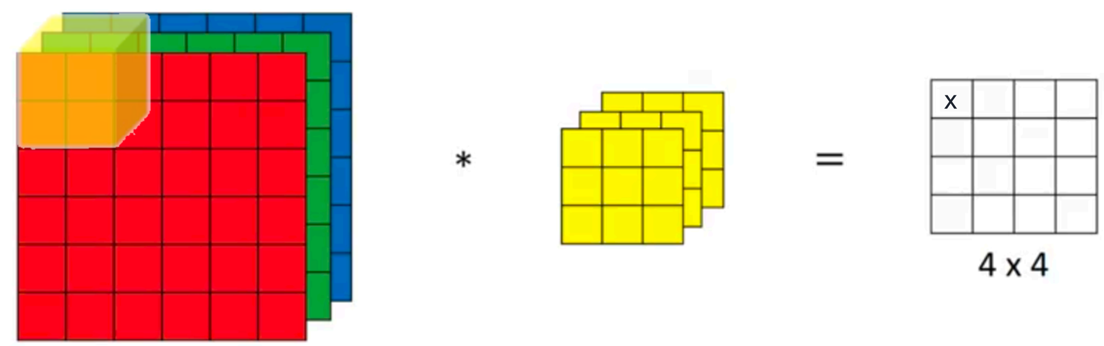
Output size: o = $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$

Table of content

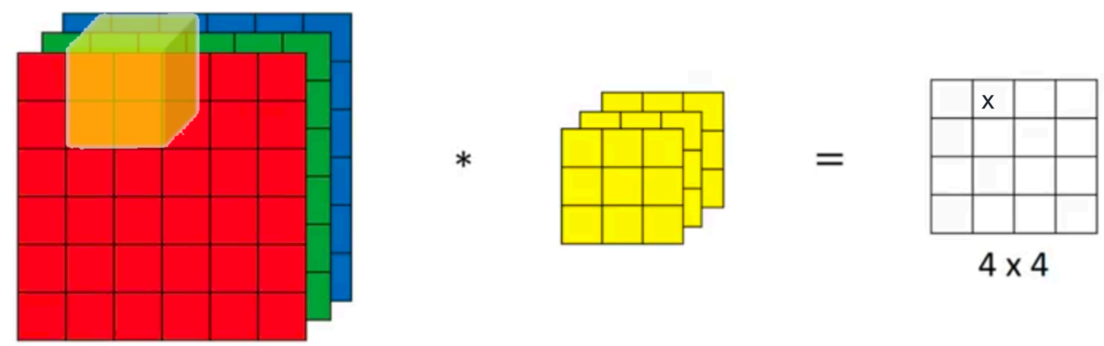
Convolution over Volumes



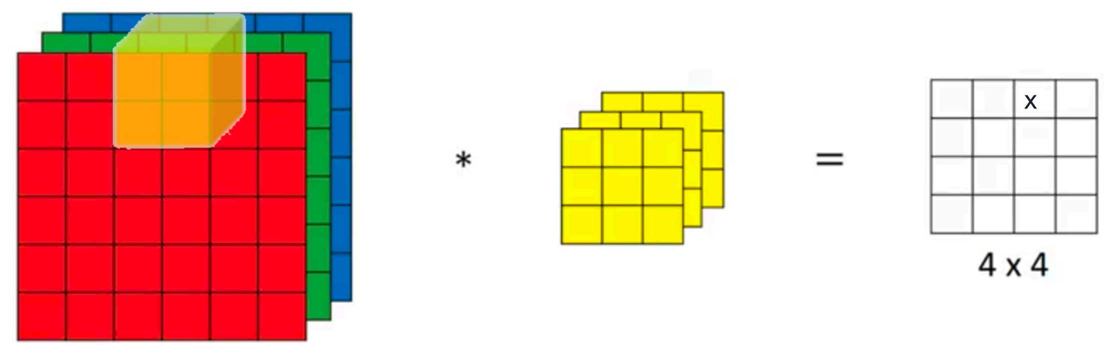
Convolution over Volumes



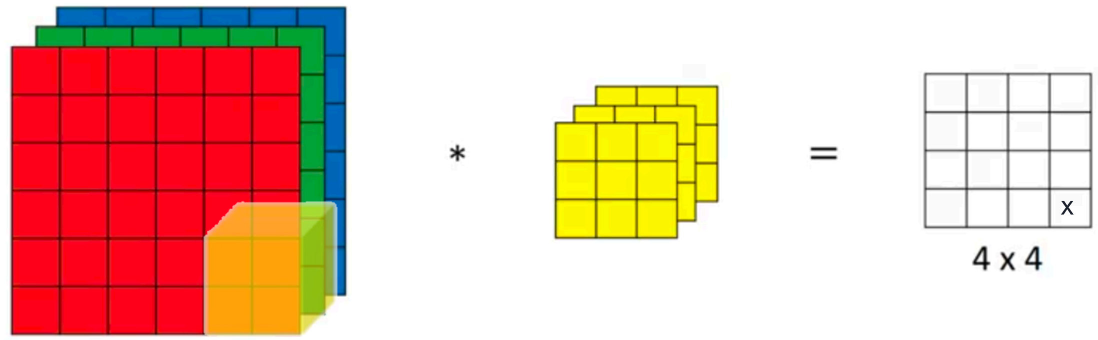
Convolution over Volumes



Convolution over Volumes



Convolution over Volumes



Convolution over Volumes

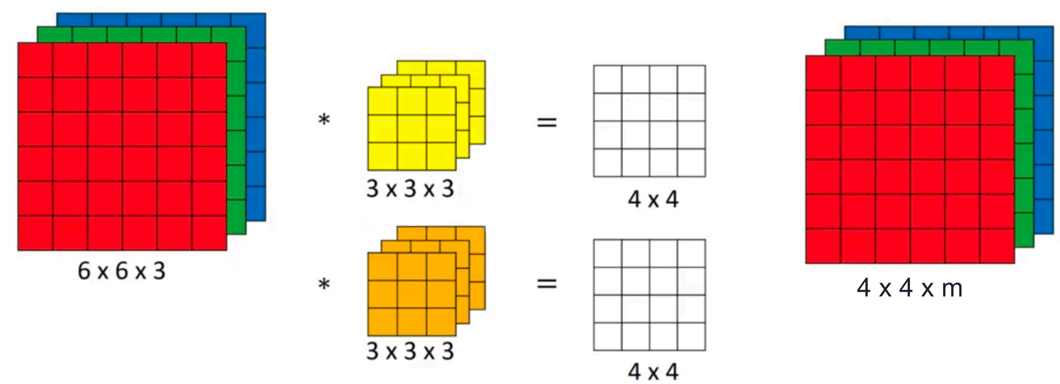


Table of content

One Layer of Convolution Network

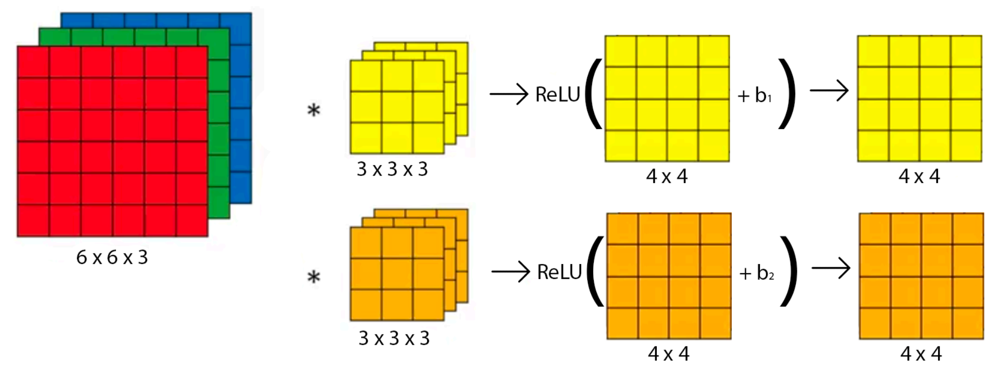


Table of content

Typical Settings

- It is common to use *Stride=1*
- It is common to use square images ($L_q = B_q$) because it is easier to work with.
- Usually F_q = 3 or 5. In general smaller filter often delivers better results.

Table of content

Simple Convolutional Neural Network

- *width* and *height* reduce from left to right.
- *depth* increases from left to right.
- Thus features are more abstract and apply on larger region of the initial image.

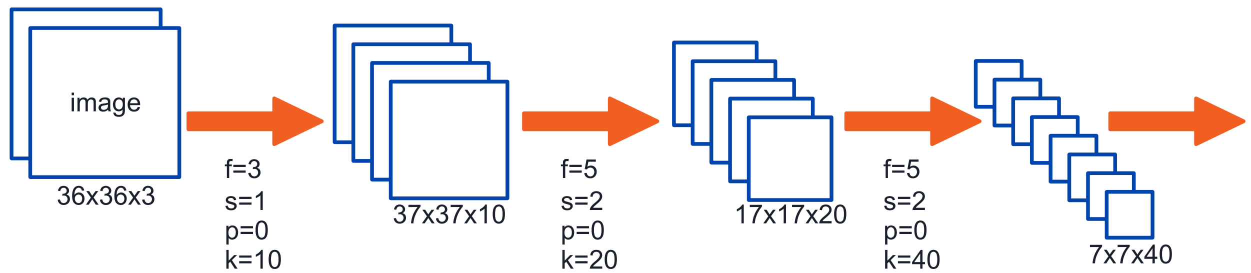
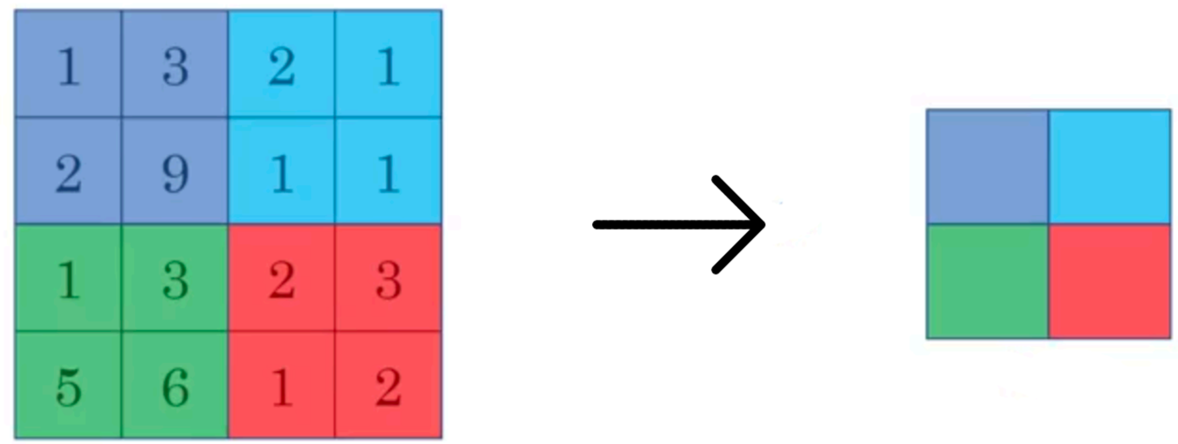


Table of content

Pooling

- The pooling operation works on small grid regions of size $P_q \times P_q$ in each layer, and produces another layer with the same depth.
- For each square region of size $P_q \times P_q$ in each of the d_q activation maps, the maximum of these values is returned.
- It is common to use a stride $S_q > 1$ in pooling (often we have $P_q = S_q$).
- Pooling drastically reduces the spatial dimensions of each activation map.

Pooling



Pooling

- ***Max*** or ***Average*** Pooling
- Have same d_q in input and output.

- No Parameters to Learn!

Table of content

Fully Connected Network

- Each feature in the final spatial layer is connected to each hidden state in the first fully connected layer.
- This layer functions in exactly the same way as a traditional feed-forward network.
- In most cases, one might use more than one fully connected layer to increase the power of the computations towards the end.
- The connections among these layers are exactly structured like a traditional feed-forward network.
- The vast majority of parameters lie in the fully connected layers.

Table of content

The Interleaving between layers

- The convolution, pooling, and ReLU layers are typically interleaved in order to increase expressive power.
- The ReLU layers often follow the convolutional layers, just as a nonlinear activation function typically follows the linear dot product in traditional neural networks.
- After two or three sets of convolutional-ReLU combinations, one might have a max-pooling layer.

Table of content

Why Convolutional Neural Networks?

- Two main advantages of CNNs are:
  - Parameter sharing.
    - A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.
  - sparsity of connections
    - In each layer, each output value depends only on a small number of inputs which makes it translation invariance.

Table of content

8.3 Training a Convolutional Neural Network

- There are three operations: ***convolutions***, ***max-pooling***, and ***ReLU***.
- The ReLU backpropagation is the same as any other network.
  - Passes gradient to a previous layer only if the original input value was positive.
- The max-pooling passes the gradient flow through the largest cell in the input volume.

- Main complexity is in backpropagation through convolutions.

Table of content

Pooling layers

- Max-pooling - the error is just assigned to where it comes from - the “winning unit” because other units in the previous layer’s pooling blocks did not contribute to it hence all the other assigned values of zero
- Average pooling - the error is multiplied by $\frac{1}{N \times N}$ and assigned to the whole pooling block (all units get this same value).

Table of content

Backpropagating through Convolutions

- We are looking for $\frac{\partial E}{\partial w^{l}_{m',n'}}$ (*m, n* are Kernel iterators)
- Convolution between the input feature map of dimension *HxW* and the weight kernel of dimension k1 × k2 produces an output feature map of size (*H− k₁ +1*) by (*W−k₂+1*). The gradient component for the individual weights can be obtained by applying the chain rule in the following way:

$$\frac{\partial E}{\partial w^{l}_{m',n'}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \frac{\partial E}{\partial x^{l}_{i,j}} \frac{\partial x^{l}_{i,j}}{\partial w^{l}_{m',n'}}$$

$$= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^{l}_{i,j} \frac{\partial x^{l}_{i,j}}{\partial w^{l}_{m',n'}}$$

Read more: 📚

Backpropagating through Convolutions

- $x^l_{i,j}$ is equivalent to $\sum_m \sum_n w^l_{m,n} o^{l-1}_{i+m,j+n} + b_l$ and expanding this part of the equation gives us:

$$\frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} = \frac{\partial}{\partial w^l_{m',n'}} \left(\sum_m \sum_n w^l_{m,n} o^{l-1}_{i+m,j+n} + b^l \right)$$

Backpropagating through Convolutions

- Further expanding the summations and taking the partial derivatives for all the components results in zero values for all except the components where *m=m'* and
- *n=n'* in $w^l_{m,n} o^{l-1}_{i+m,j+n}$ as follows

$$\frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} = \frac{\partial}{\partial w^l_{m',n'}} \left(w^l_{0,0} o^{l-1}_{i+0,j+0} + \cdots + w^l_{m',n'} o^{l-1}_{i+m',j+n'} + \cdots + b^l \right)$$

$$= \frac{\partial}{\partial w^l_{m',n'}} \left(w^l_{m',n'} o^{l-1}_{i+m',j+n'} \right)$$

$$= o^{l-1}_{i+m',j+n'}$$

Backpropagating through Convolutions

- Substituting previous equation with the one on page 51 gives us the following results:

$$\frac{\partial E}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^l_{i,j} o^{l-1}_{i+m',j+n'}$$

Backpropagating through Convolutions

- Using chain rule and introducing sums give us the following equation:

$$\frac{\partial E}{\partial x^{l}_{i',j'}} = \sum_{i,j \in Q} \frac{\partial E}{\partial x^{l+1}_{Q}} \frac{\partial x^{l+1}_{Q}}{\partial x^{l}_{i',j'}}$$
$$= \sum_{i,j \in Q} \delta^{l+1}_{Q} \frac{\partial x^{l+1}_{Q}}{\partial x^{l}_{i',j'}}$$

- *Q* is the output region after applying padding and stride

Backpropagating through Convolutions

- A bit more formal way would be:

$$\frac{\partial E}{\partial x^{l}_{i',j'}} = \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \frac{\partial E}{\partial x^{l+1}_{i'-m,j'-n}} \frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^{l}_{i',j'}}$$

$$= \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta^{l+1}_{i'-m,j'-n} \frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^{l}_{i',j'}}$$

Backpropagating through Convolutions

- We know $x^{l+1}_{i'-m,j'-n}$ is equals to $\sum_{m'} \sum_{n'} w^{l+1}_{m',n'} o^{l}_{i'-m+m',j'-n+n'} + b^{l+1}$
- So we have:

$$\frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^{l}_{i',j'}} = \frac{\partial}{\partial x^{l}_{i',j'}} \left(\sum_{m'} \sum_{n'} w^{l+1}_{m',n'} o^{l}_{i'-m+m',j'-n+n'} + b^{l+1} \right)$$

$$= \frac{\partial}{\partial x^{l}_{i',j'}} \left(\sum_{m'} \sum_{n'} w^{l+1}_{m',n'} f\left(x^{l}_{i'-m+m',j'-n+n'} \right) + b^{l+1} \right)$$

Backpropagating through Convolutions

- By expanding previous equation we would have:

$$
\begin{aligned}
\frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^{l}_{i',j'}} &= \frac{\partial}{\partial x^{l}_{i',j'}}\left(w^{l+1}_{m',n'} f\left(x^{l}_{0-m+m',0-n+n'}\right) + \cdots + w^{l+1}_{m,n} f\left(x^{l}_{i',j'}\right) + \cdots + b^{l+1}\right) \\
&= \frac{\partial}{\partial x^{l}_{i',j'}}\left(w^{l+1}_{m,n} f\left(x^{l}_{i',j'}\right)\right) \\
&= w^{l+1}_{m,n} \frac{\partial}{\partial x^{l}_{i',j'}}\left(f\left(x^{l}_{i',j'}\right)\right) \\
&= w^{l+1}_{m,n} f'\left(x^{l}_{i',j'}\right)
\end{aligned}
$$

Backpropagating through Convolutions

- and finally:

$$\frac{\partial E}{\partial x^l_{i',j'}} = \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta^{l+1}_{i'-m,j'-n} w^{l+1}_{m,n} f'\left(x^l_{i',j'}\right)$$

Backpropagating through Convolutions

- However, this computation assumes that all weights are distinct, whereas the weights in the filter are shared across the entire spatial extent of the layer. Therefore, one has to be careful to account for shared weights, and **sum up the partial derivatives** of all copies of a shared weight.
- In other words, we first pretend that the filter used in each position is distinct in order to compute the partial derivative with respect to each copy of the shared weight, and then add up the partial derivatives of the loss with respect to all copies of a particular weight.

Table of content

Backprop. with Inverted / Transposed Filter

- For simplicity the depth of output and input convolution layers was considered 1.
- In such a case, the convolution filter is inverted both horizontally and vertically for backpropagation.
- The reason for this inversion is that the filter is “moved around” to perform dot product. Whereas the backpropagation derivatives are with respect to the input volume whose relative movement with respect to the filter is the opposite of the filter movement during convolutions.

a	b	c
d	e	f
g	h	i

FILTER DURING CONVOLUTION

→

i	h	g
f	e	d
c	b	a

FILTER DURING BACKPROPAGATION

For example: note that the entry in the extreme upper-left of the convolution filter might not even contribute to the extreme upper-left entry in the output volume (because of the padding), but it will almost always contribute to the extreme lower-right entry of the output volume.

Table of content

Matrix Multiplication

- Assume we have input with size of $A_I = L_q \times L_q \times 1$ ($d_q = 1$)
- and out put with size of $A_O = (L_q - F_q + 1) \times (L_q - F_q + 1) \times 1$
- The process is as below:
  - Flatten the input, A_I into a A_I-*dimensional* column vector
  - Consider the output will be A_O-*dimensional* column vector
  - Create a sparse matrix C from the Filter (a matrix with size of $A_I \times A_o$)
  - The value of each entry in the row corresponds to one of the AI positions in the input matrix. The value is 0, if that input position is not involved in the convolution for that row.
  - Otherwise, the value is set to the corresponding value of the filter.

Matrix Multiplication

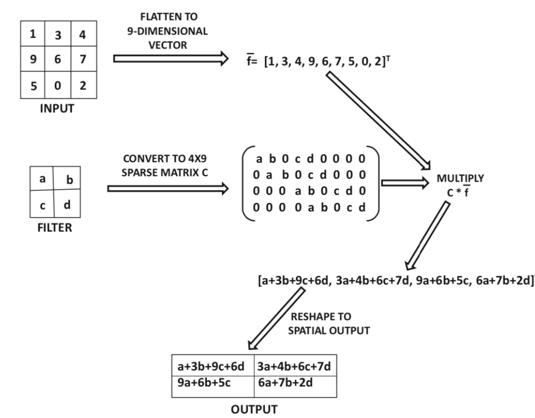
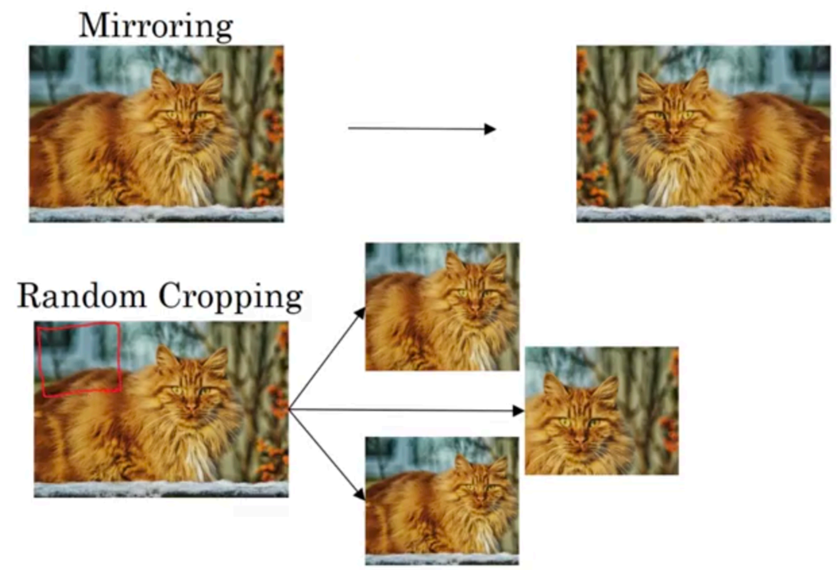


Table of content

Data Augmentation

- If data is increased, your deep NN will perform better. Data augmentation is one of the techniques that deep learning uses to increase the performance of deep NN.
- Some data augmentation methods that are used for computer vision tasks includes:
  - Mirroring.
  - Random cropping.
    - The issue with this technique is that you might take a wrong crop.
    - The solution is to make your crops big enough.
  - Rotation.
  - Shearing.
  - Local warping.
  - Color shifting.
    - For example, we add to R, G, and B some distortions that will make the image identified as the same for the human but is different for the computer.

Data Augmentation



Data Augmentation

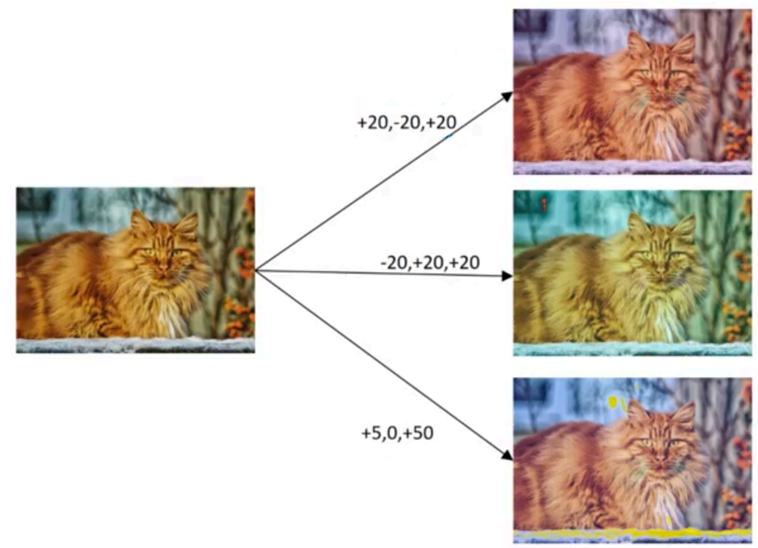


Table of content

8.4 Case Studies

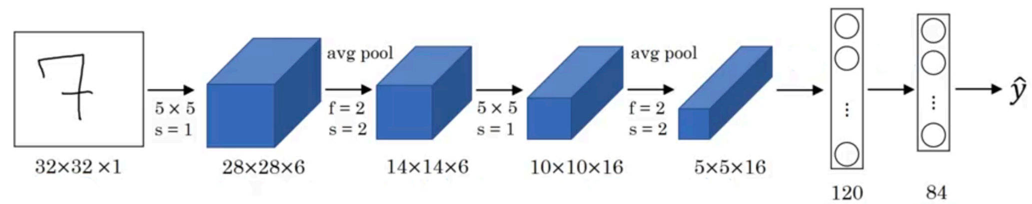
- Classic Networks
  - LeNet-5
  - AlexNet
  - VGG
- Deeper network
  - ResNet (152 layers!!!)
- Inceptions Neural Networks

The last few years in CNN research was about the way to improve the performance of these networks. So it's nice to take look of these studies.

Table of content

LeNet-5

- The goal was recognize hand-written digits.
- 60K Parameters



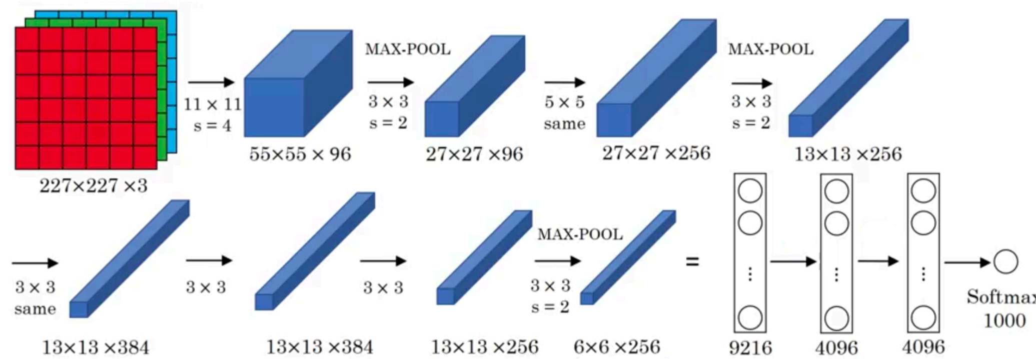
LeCun et al., 1998. Gradient-based learning applied to document recognition

The network has no padding (in those time people usually used valid convolution)
In those time avg pooling was more popular
as you go deeper and deeper the width and height of the volumes tends to get lower
and the depth of the volumes grow bigger.
Some thing you can see here that is also practiced today is pooling layer every couple of conv layers. Then followed by couple of FC layer at the end.

Table of content

AlexNet

- 60M parameters



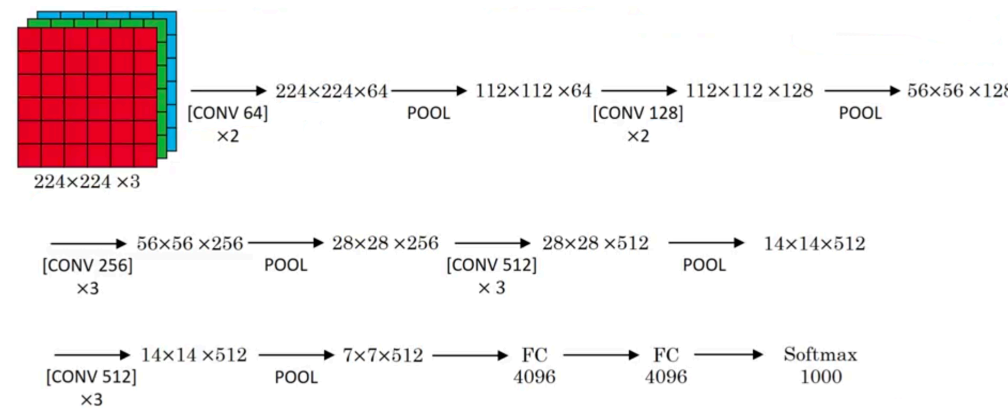
Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks

the author name was alex!
similar to LeNet but much bigger (60M parameters)
in LeNet sigmoid and tanh was used
LRN: look across all the channel and normalize the response (normalize across the depth)
First use of ReLU and Local Response Normalization in CNNs

Table of content

VGG-16

- 138M Parameters



Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition

The basic idea was instead of having so may hyper parameter, use more pooling layers.
simple architectures made it quite appealing.
We can see the weights get doubled
Just having 3x3 conv filters and Stride=1 and “same” padding
It has 16 layers that have weights.
CONV = 3x3 filter, s=1, same
MAX-POOLING = 2x2, s=2

Table of content

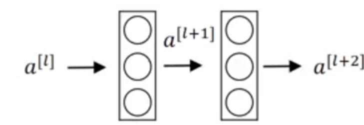
ZFNet

- Based on Alexnet but with minor changes
- Winner of ImageNet in 2013

	AlexNet	*ZFNet*
Volume: Operations:	$224 \times 224 \times 3$ Conv 11×11 (stride 4)	$224 \times 224 \times 3$ Conv 7×7 (stride 2), MP
Volume: Operations:	$55 \times 55 \times 96$ Conv 5×5, MP	$55 \times 55 \times 96$ Conv 5×5 (stride 2), MP
Volume: Operations:	$27 \times 27 \times 256$ Conv 3×3, MP	$13 \times 13 \times 256$ Conv 3×3
Volume: Operations:	$13 \times 13 \times 384$ Conv 3×3	$13 \times 13 \times 512$ Conv 3×3
Volume: Operations:	$13 \times 13 \times 384$ Conv 3×3	$13 \times 13 \times 1024$ Conv 3×3
Volume: Operations:	$13 \times 13 \times 256$ MP, Fully connect	$13 \times 13 \times 512$ MP, Fully connect
FC6: Operations:	4096 Fully connect	4096 Fully connect
FC7: Operations:	4096 Fully connect	4096 Fully connect
FC8: Operations:	1000 Softmax	1000 Softmax

Table of content

Residual Networks (ResNets)



"Shortcut"

$$a^{[l]} \rightarrow \text{Linear} \rightarrow \text{ReLU} \rightarrow a^{[l+1]} \rightarrow \text{Linear} \rightarrow \text{ReLU} \rightarrow \oplus \rightarrow a^{[l+2]}$$

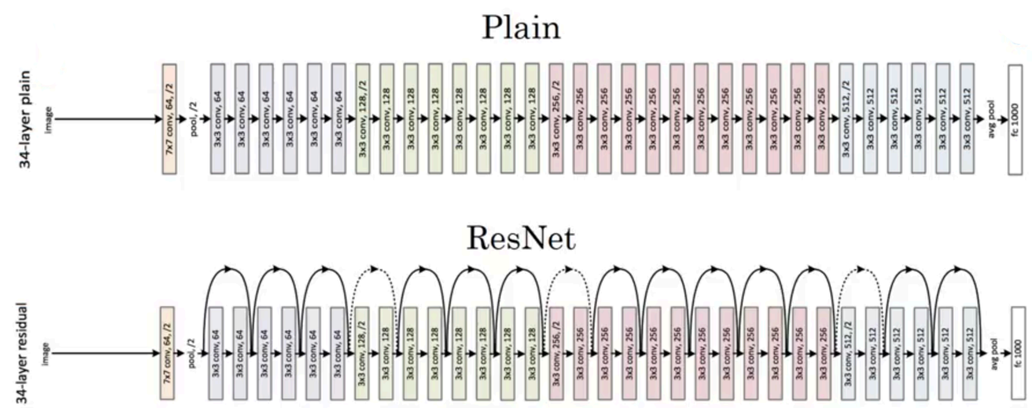
$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]}$ $\quad a^{[l+1]} = g(z^{[l+1]})$ $\quad z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$ $\quad a^{[l+2]} = g(z^{[l+2]})$

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

He et al., 2015. Deep residual networks for image recognition

allows you to create deeper networks even over 100 layers
fast-forward the input (a[l]) and just add it (before the last ReLU applied) to the output
The reason is this short cut helps to vanishing/exploding gradient problem.
Also, these shortcuts can be interpreted as a way that, it reduces to simpler network if the corresponding layer decrease the performance. (in other word, it's easier for this layers to learn identity function, thus it's guaranteed to not heart the network performance.)

Residual Networks (ResNets)



He et al., 2015. Deep residual networks for image recognition

most of these are 3x3 same padding conv layers.

Table of content

Network in Network and 1×1 convolutions

- Convolution by 1x1 filter is just multiplying the image in that filter (value)
- But it does more than that!

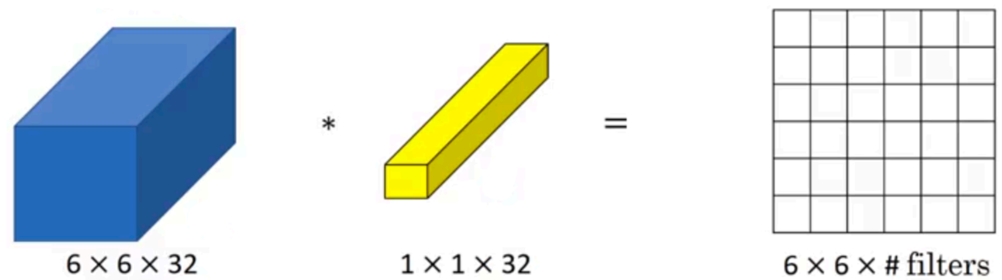
$$\underbrace{\begin{bmatrix} 1 & 2 & 3 & 6 & 5 & 8 \\ 3 & 5 & 5 & 1 & 3 & 4 \\ 2 & 1 & 3 & 4 & 9 & 3 \\ 4 & 7 & 8 & 5 & 7 & 9 \\ 1 & 5 & 3 & 7 & 4 & 8 \\ 5 & 4 & 9 & 8 & 3 & 5 \end{bmatrix}}_{6 \times 6} * \boxed{2} = \begin{bmatrix} 2 & 4 & 6 & \cdots & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \end{bmatrix}$$

Lin et al., 2013. Network in network

convolution

Network in Network and 1×1 convolutions

- It do element-wise product of the volume.
- And then apply ReLU



Lin et al., 2013. Network in network

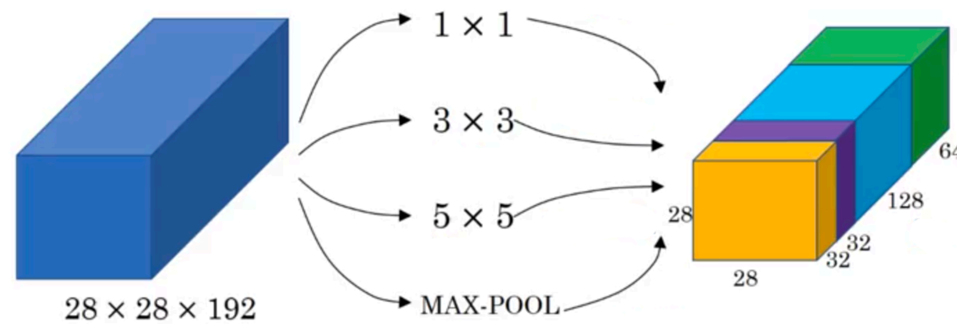
pooling can reduce the width and height
1x1 conv can reduce depth (somehow it's a pooling for depth!)

for instance, on input of (28x28x192) we can apply 32, conv1x1 then we will have 28x28x32 volume.

Table of content

Inception Network

- Design the layers by itself, in other word, have all of the layer architecture inside it.

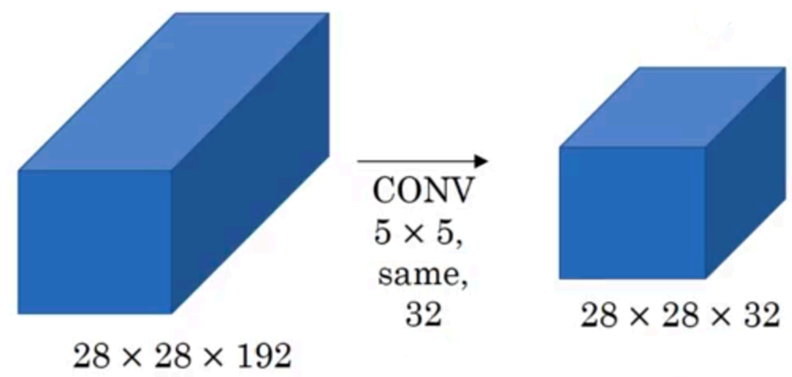


Szegedy et al., 2014. Going deeper with convolutions

the max pooling has padding (same padding + stride=1)
the basic idea is that instead of you needing to pick what you want, just have them all and let the network to have and optimize what is needed and better for solving the problem

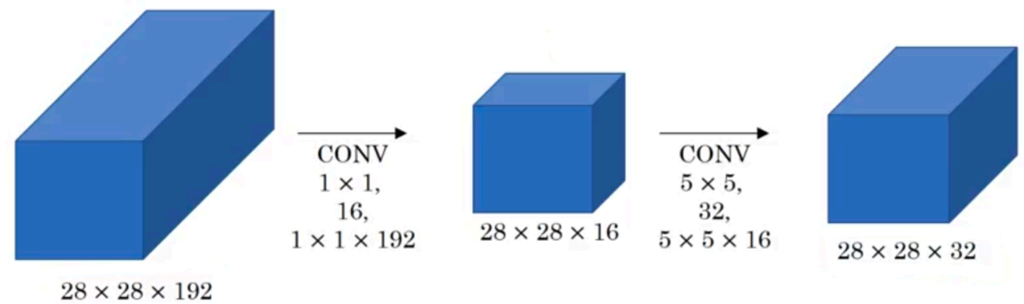
Inception Network

- Instead of using this: (120M parameters)

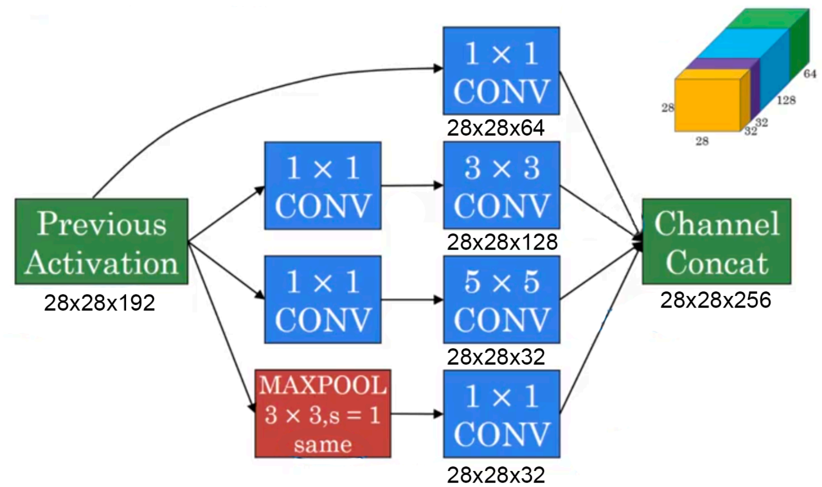


Inception Network

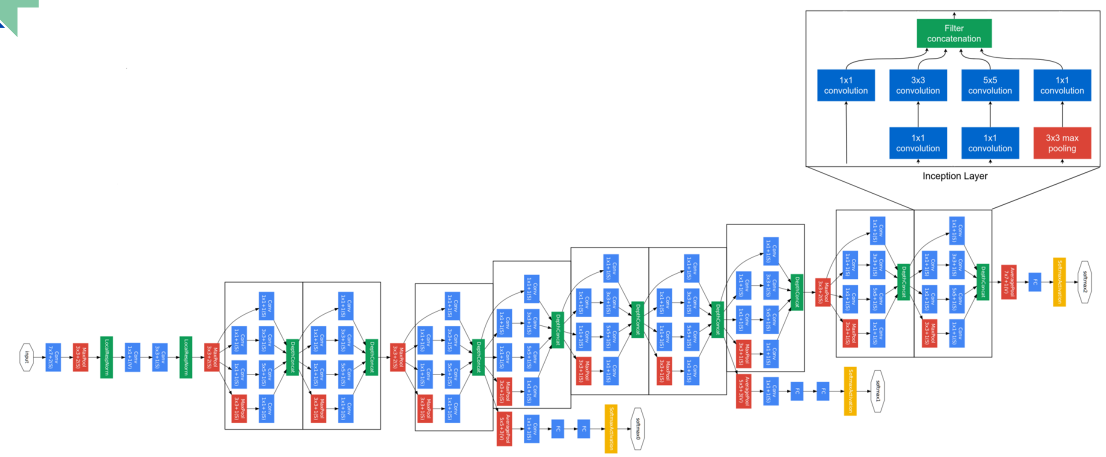
- We use this: (12.4M parameters)



Inception Module



Inception Network (GoogLeNet)



These side branches takes a hidden layer and try to predict the output layer.
The reason is that to prevent the network to overfit.

Table of content

Transfer Learning

- If you are using a specific NN architecture that has been trained before, you can use this pre-trained parameters/weights instead of random initialization to solve your problem.
- It can help you boost the performance of the NN.
- The pre-trained models might have trained on a large datasets like ImageNet, Ms COCO, or pascal and took a lot of time to learn those parameters/weights with optimized hyper-parameters. This can save you a lot of time.

Transfer Learning

- For Example
  - Lets say you have a cat classification problem which contains 3 classes *Tigger*, *Misty* and *neither*.
  - You don't have much a lot of data to train a NN on these images.
  - Download a good NN with its weights, remove the softmax activation layer and put your own one and make the network learn only the new layer while other layer weights are fixed/frozen.
  - One of the tricks that can speed up your training, is to run the pre-trained NN without final softmax layer and get an intermediate representation of your images and save them to disk. And then use these representation to a shallow NN network. This can save you the time needed to run an image through all the layers.
    - Its like converting your images into vectors.

Table of content

8.6 Application of Convolution Network

- Convolutional neural networks have several applications in **object detection**, **localization**, **video**, and **text processing**.
- The success of convolutional neural networks remains unmatched by almost any class of neural networks. In recent years, competitive methods have even been proposed for sequence-to-sequence learning, which has traditionally been the domain of recurrent networks.

Read more about sequence-to-sequence learning 📚

Table of content

Content-Based Image Retrieval

- In content-based image retrieval, each image is first engineered into a set of multidimensional features by using a pretrained classifier like AlexNet.
- The multidimensional representations of the images can be used in conjunction with any multidimensional retrieval system to provide results of high quality.
- The reason that this approach works is because the features extracted from AlexNet have semantic significance to the different types of shapes present in the data.

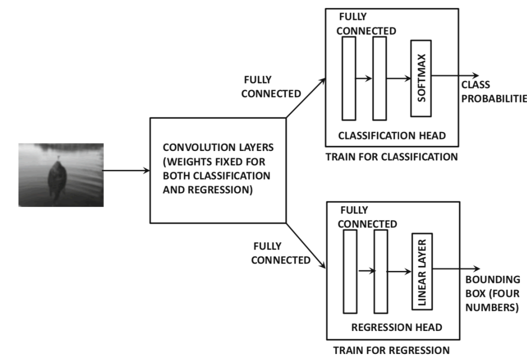


Table of content

Object Detection & Localization

- In object localization, we have a fixed set of objects in an image, and we would like to identify the rectangular regions in the image in which the object occurs.

Table of content

- 8.6 Application of Convolution Network
  - ✓ Content-Based Image Retrieval
  - ✓ Object Detection & Localization
  - Natural Language and Sequence Learning
  - Video Classification

Natural Language and Sequence Learning

- At first sight, convolutional neural networks do not seem like a natural fit for text-mining tasks.
  - Unlike image, in text position of the representing data is quite important
- Instead of 3D boxes with a spatial extent and a depth, the filter for text data are 2D boxes with a window length for sliding the sentence.
- Use of one-hot encoding increases the number of channels, and therefore blows up the number of parameters in the filter in the first layer
  - Instead using pretained embeddings of the words such as Word2Vec or GLoVe are used.

Table of content

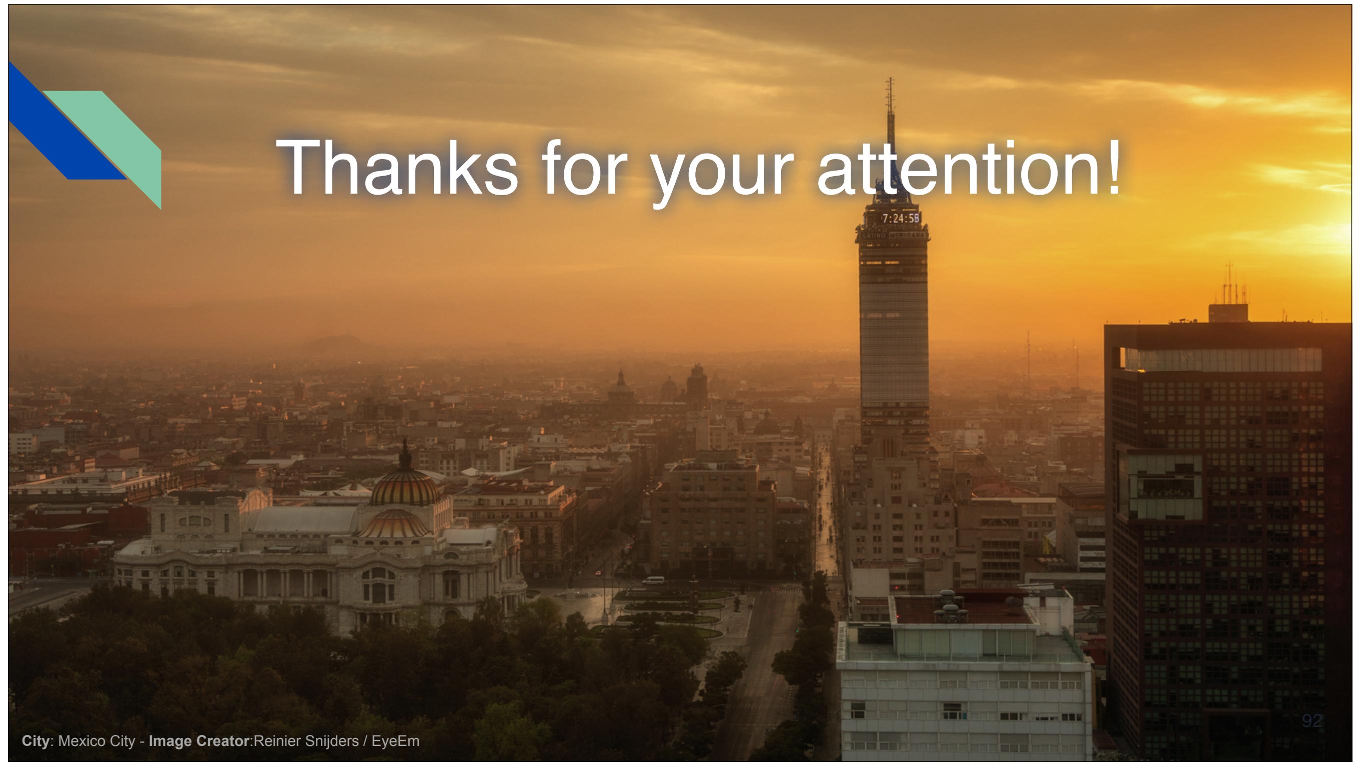
Video Classification

- Videos can be considered generalizations of image data in which a temporal component is inherent to a sequence of images. (spatio-temporal data)
- Instead of 2D (+ depth) filter, a 3D filter (+ depth) is used.
- An interesting observation is that 3-dimensional convolutions add only a limited amount to what one can achieve by averaging the classifications of individual frames by image classifiers
    - A part of the problem is that motion adds only a limited amount to the information that is available in the individual frames for classification purposes.
    - sufficiently large video data sets are hard to come by.
- For the case of longer videos, it makes sense to combine recurrent neural networks (or LSTMs) with convolutional neural networks

Consider a situation in which each image is of size 224 × 224 × 3, and a total of 10 frames are received. Therefore, the size of the video segment is 224 × 224 × 10 × 3.

8.7 Summery

- Primary focus of these networks are in Image Processing and Computer Vision
- These networks are biologically inspired and are among the earliest success stories of Neural Networks.
- CNNs typically learn hierarchical features in different layers, where the earlier layers learn primitive shapes, whereas the later layers learn more complex shapes.
- Recently, convolutional neural networks have also been used for text processing, where they have shown competitive performance with recurrent neural networks.


Thanks for your attention!
City: Mexico City - **Image Creator**:Reinier Snijders / EyeEm
92

Copyright Note:

Some of the content of this presentation is copyrighted under:
Deep Learning Specialization by Andrew Ng - © 2019 Coursera Inc