Introduction to Neural Networks and Deep Learning

Introduction to the Convolutional Network

Andres Mendez-Vazquez

March 28, 2021

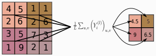
Outline

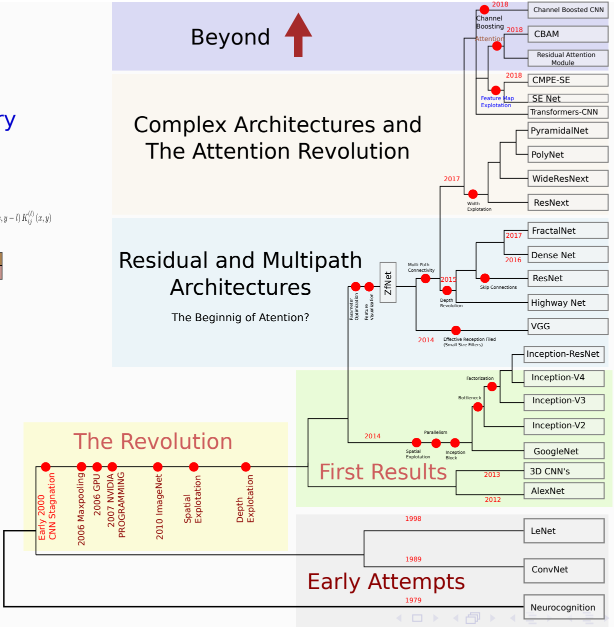
Outline

The Long Path [1]

A Small History of a Revolution

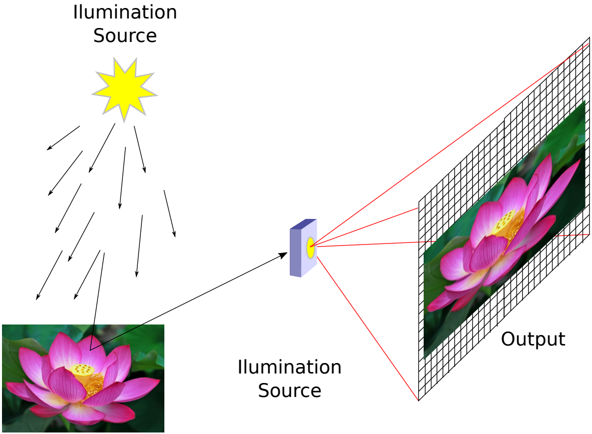
$$Y_i^{(l)}(x,y) = B_i^{(l)}(x,y) + \sum_{j=1}^{m_1^{(l-1)}} \sum_{s=-ks}^{ks} \sum_{l=-ks}^{ks} Y_j^{(l-1)}(x-s,y-l)\, K_{ij}^{(l)}(x,y)$$





Outline

Digital Images as pixels in a digitized matrix [2]



Further [2]

Pixel values typically represent

- Gray levels, colors, heights, opacities etc

Something Notable

- Remember digitization implies that a digital image is an approximation of a real scene

Further [2]

Pixel values typically represent

- Gray levels, colors, heights, opacities etc

Something Notable

- Remember digitization implies that a digital image is an approximation of a real scene

Images

Common image formats include

- On sample/pixel per point (B&W or Grayscale)
- Three samples/pixel per point (Red, Green, and Blue)
- Four samples/pixel per point (Red, Green, Blue, and “Alpha”)

Therefore, we have the following process

Low Level Process

Imagen

Noise Removal

Sharpening

Example

Edge Detection

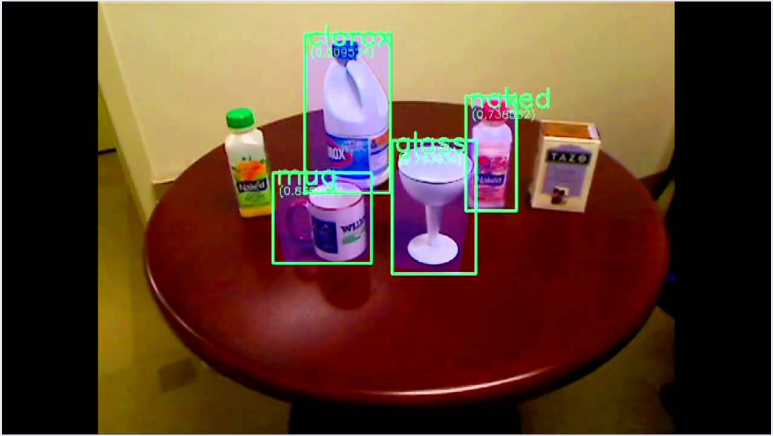
Then

Mid Level Process

Input	Processes	Output
Image	Object Recognition	Attributes
	Segmentation	

Example

Object Recognition



Therefore

It would be nice to automatize all these processes

- We would solve a lot of headaches when setting up such process

Why not to use the data sets

- By using a Neural Networks that replicates the process.

Therefore

It would be nice to automatize all these processes

- We would solve a lot of headaches when setting up such process

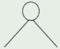
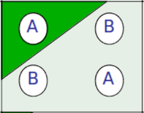
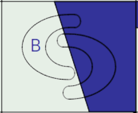
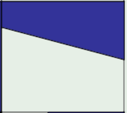
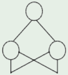
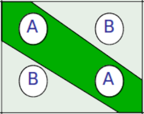
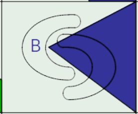
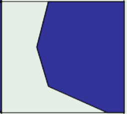
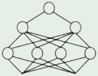
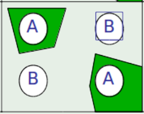
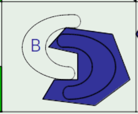
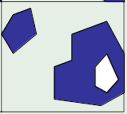
Why not to use the data sets

- By using a Neural Networks that replicates the process.

Outline

Multilayer Neural Network Classification

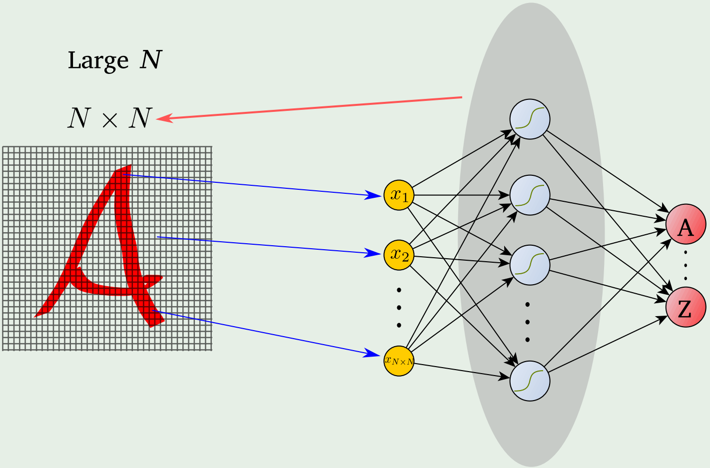
We have the following classification [3]

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer 	Half Plane Bounded By Hyper plane			
Two-Layer 	Convex Open Or Closed Regions			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

Outline

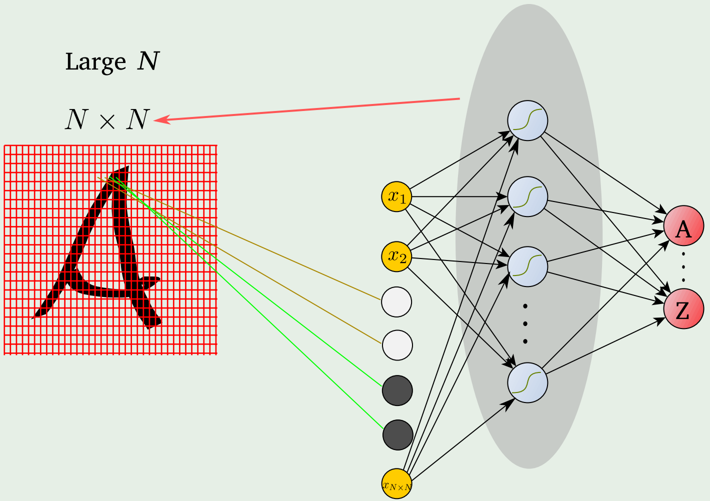
Drawbacks of previous neural networks

The number of trainable parameters becomes extremely large



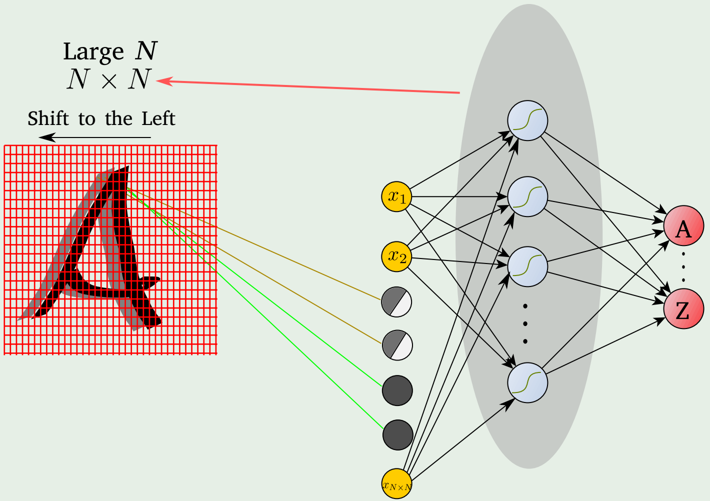
Drawbacks of previous neural networks

In addition, little or no invariance to shifting, scaling, and other forms of distortion



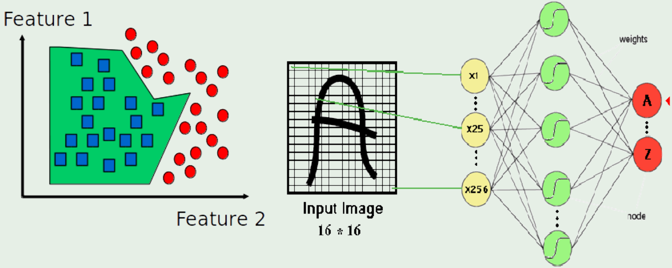
Drawbacks of previous neural networks

In addition, little or no invariance to shifting, scaling, and other forms of distortion



Drawbacks of previous neural networks

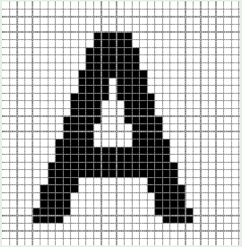
The topology of the input data is completely ignored



For Example

We have

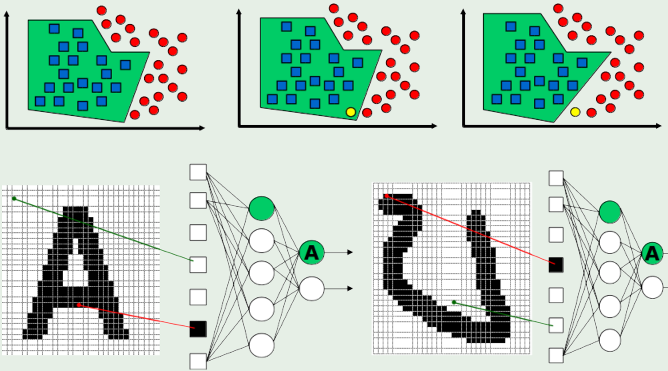
- Black and white patterns: $2^{32\times 32} = 2^{1024}$
- Gray scale patterns: $256^{32\times 32} = 256^{1024}$



32 * 32 input image

For Example

If we have an element that the network has never seen



Possible Solution

We can minimize this drawbacks by getting

- Fully connected network of sufficient size can produce outputs that are invariant with respect to such variations.

Problem!!!

- Training time
- Network size
- Free parameters

Possible Solution

We can minimize this drawbacks by getting

- Fully connected network of sufficient size can produce outputs that are invariant with respect to such variations.

Problem!!!

- Training time
- Network size
- Free parameters

Outline

Hubel/Wiesel Architecture

Something Notable [4]

- D. Hubel and T. Wiesel (1959, 1962, Nobel Prize 1981)

They commented

- The visual cortex consists of a hierarchy of simple, complex, and hyper-complex cells

Hubel/Wiesel Architecture

Something Notable [4]

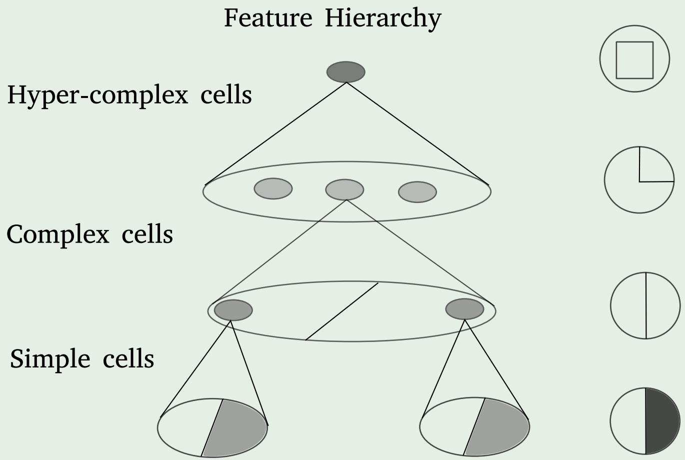
- D. Hubel and T. Wiesel (1959, 1962, Nobel Prize 1981)

They commented

- The visual cortex consists of a hierarchy of simple, complex, and hyper-complex cells

Something Like

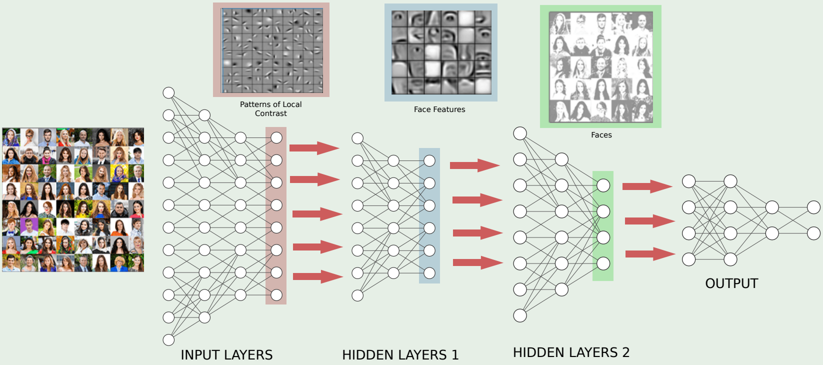
We have



History

Convolutional Neural Networks (CNN) were invented by [5]

In 1989, Yann LeCun and Yoshua Bengio introduced the concept of Convolutional Neural networks.



About CNN's

Something Notable

CNN's Were neurobiologically motivated by the findings of locally sensitive and orientation-selective nerve cells in the visual cortex.

In addition

They designed a network structure that implicitly extracts relevant features.

Properties

Convolutional Neural Networks are a special kind of multi-layer neural networks.

About CNN's

Something Notable

CNN's Were neurobiologically motivated by the findings of locally sensitive and orientation-selective nerve cells in the visual cortex.

In addition

They designed a network structure that implicitly extracts relevant features.

Properties

Convolutional Neural Networks are a special kind of multi-layer neural networks.

About CNN's

Something Notable

CNN's Were neurobiologically motivated by the findings of locally sensitive and orientation-selective nerve cells in the visual cortex.

In addition

They designed a network structure that implicitly extracts relevant features.

Properties

Convolutional Neural Networks are a special kind of multi-layer neural networks.

About CNN's

In addition

- CNN is a feed-forward network that can extract topological properties from an image.
- Like almost every other neural networks they are trained with a version of the back-propagation algorithm.
- Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing.
- They can recognize patterns with extreme variability.

About CNN's

In addition

- CNN is a feed-forward network that can extract topological properties from an image.
- Like almost every other neural networks they are trained with a version of the back-propagation algorithm.
- Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing.
- They can recognize patterns with extreme variability.

About CNN's

In addition

- CNN is a feed-forward network that can extract topological properties from an image.
- Like almost every other neural networks they are trained with a version of the back-propagation algorithm.
- Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing.
- They can recognize patterns with extreme variability.

About CNN's

In addition

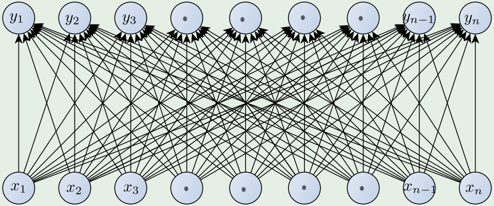
- CNN is a feed-forward network that can extract topological properties from an image.
- Like almost every other neural networks they are trained with a version of the back-propagation algorithm.
- Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing.
- They can recognize patterns with extreme variability.

Outline

Local Connectivity

We have the following idea [6]

- Instead of using a full connectivity...



Input Image

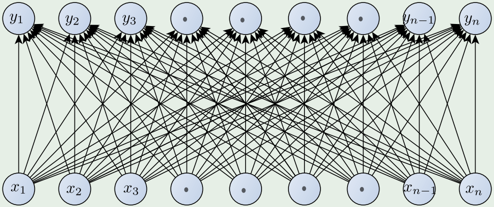
We would have something like this

$$y_i = f\left(\sum_{i=1}^{n} w_i x_i\right) \quad (1)$$

Local Connectivity

We have the following idea [6]

- Instead of using a full connectivity...



We would have something like this

$$y_i = f\left(\sum_{i=1}^{n} w_i x_i\right) \tag{1}$$

Local Connectivity

We decide only to connect the neurons in a local way

- Each hidden unit is connected only to a subregion (patch) of the input image.
- It is connected to all channels:

Local Connectivity

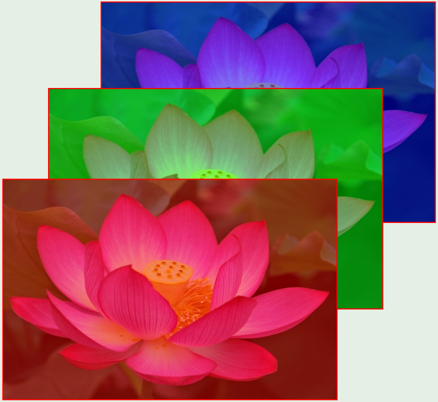
We decide only to connect the neurons in a local way

- Each hidden unit is connected only to a subregion (patch) of the input image.
- It is connected to all channels:

Local Connectivity

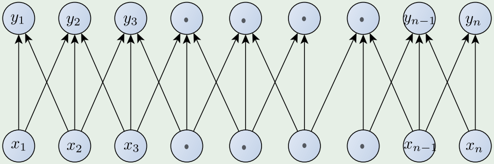
We decide only to connect the neurons in a local way

- Each hidden unit is connected only to a subregion (patch) of the input image.
- It is connected to all channels:



Example

For gray scale, we get something like this



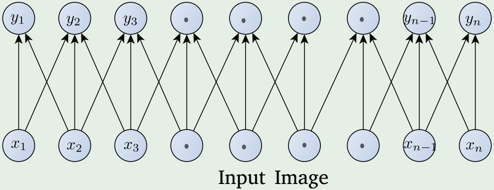
Input Image

Then, our formula changes

$$y_i = f\left(\sum_{i \in L_p} w_i x_i\right) \quad (2)$$

Example

For gray scale, we get something like this

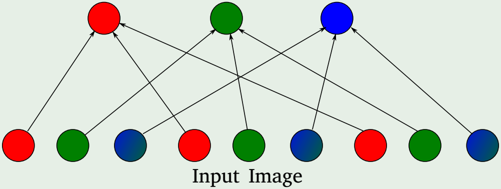


Then, our formula changes

$$y_i = f\left(\sum_{i \in L_p} w_i x_i\right) \tag{2}$$

Example

In the case of the 3 channels

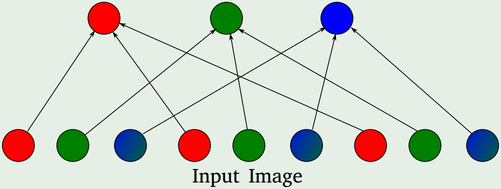


Thus

$$y_i = f\left(\sum_{i \in L_{p,c}} w_i x_i^c\right) \quad (3)$$

Example

In the case of the 3 channels



Thus

$$y_i = f\left(\sum_{i \in L_p, c} w_i x_i^c\right) \tag{3}$$

Solving the following problems...

First

- Fully connected hidden layer would have an unmanageable number of parameters

Second

- Computing the linear activation of the hidden units would have been quite expensive

Solving the following problems...

First

- Fully connected hidden layer would have an unmanageable number of parameters

Second

- Computing the linear activation of the hidden units would have been quite expensive

How this looks in the image...

We have

Receptive Field

Outline

Parameter Sharing

Second Idea

Share matrix of parameters across certain units.

These units are organized into

- The same feature "map"
  - Where the units share same parameters (For example, the same mask)

Parameter Sharing

Second Idea

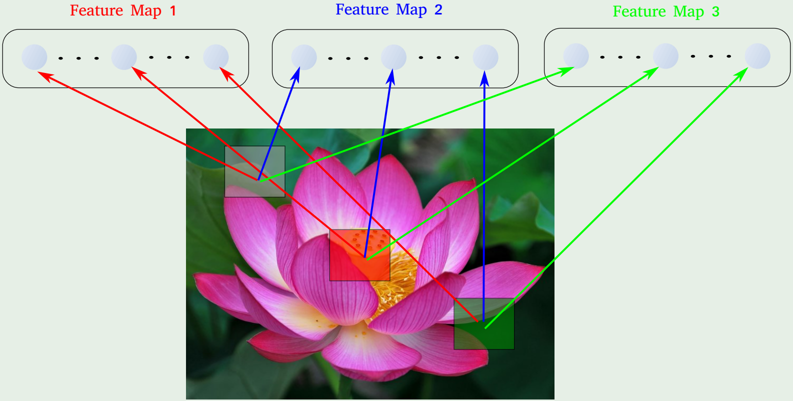
Share matrix of parameters across certain units.

These units are organized into

- The same feature “map”
  - Where the units share same parameters (For example, the same mask)

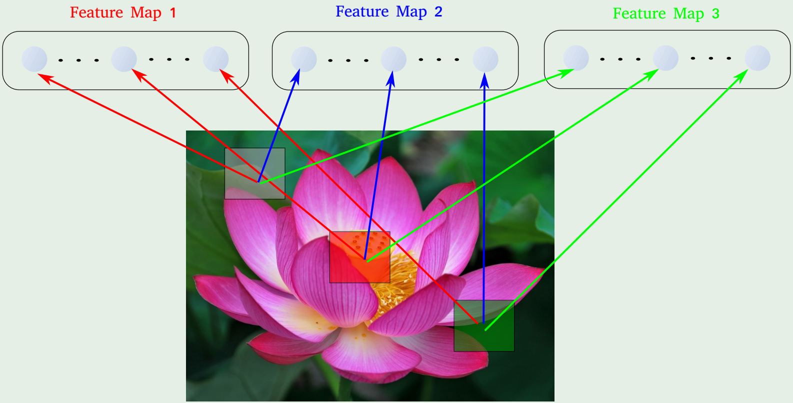
Example

We have something like this



Example

We have something like this



Now, in our notation

We have a collection of matrices representing this connectivity

- W_{ij} is the connection matrix the ith input channel with the jth feature map.
- In each cell of these matrices is the weight to be multiplied with the local input to the local neuron.

Now, in our notation

We have a collection of matrices representing this connectivity

- W_{ij} is the connection matrix the ith input channel with the jth feature map.
- In each cell of these matrices is the weight to be multiplied with the local input to the local neuron.

An now why the name of convolution

Yes!!! The definition is coming now.

Outline

Digital Images

In computer vision [2, 7]

We usually operate on digital (discrete) images:

- Sample the 2D space on a regular grid.
- Quantize each sample (round to nearest integer).

Digital Images

In computer vision [2, 7]

We usually operate on digital (discrete) images:

- Sample the 2D space on a regular grid.
- Quantize each sample (round to nearest integer).

The image can now be represented as a matrix of integer values, $I : [a, b] \times [c, d] \to [0..255]$

$$\begin{array}{cc} & j\longrightarrow \\ i\downarrow & \begin{bmatrix} 79 & 5 & 6 & 90 & 12 & 34 & 2 & 1 \\ 8 & 90 & 12 & 34 & 26 & 78 & 34 & 5 \\ 8 & 1 & 3 & 90 & 12 & 34 & 11 & 61 \\ 77 & 90 & 12 & 34 & 200 & 2 & 9 & 45 \\ 1 & 3 & 90 & 12 & 20 & 1 & 6 & 23 \end{bmatrix} \end{array}$$

Digital Images

In computer vision [2, 7]

We usually operate on digital (discrete) images:

- Sample the 2D space on a regular grid.
- Quantize each sample (round to nearest integer).

The image can now be represented as a matrix of integer values, $I : [a, b] \times [c, d] \to [0..255]$

$$
\begin{array}{cc}
 & j \longrightarrow \\
i \downarrow & \begin{bmatrix}
79 & 5 & 6 & 90 & 12 & 34 & 2 & 1 \\
8 & 90 & 12 & 34 & 26 & 78 & 34 & 5 \\
8 & 1 & 3 & 90 & 12 & 34 & 11 & 61 \\
77 & 90 & 12 & 34 & 200 & 2 & 9 & 45 \\
1 & 3 & 90 & 12 & 20 & 1 & 6 & 23
\end{bmatrix}
\end{array}
$$

Digital Images

In computer vision [2, 7]

We usually operate on digital (discrete) images:

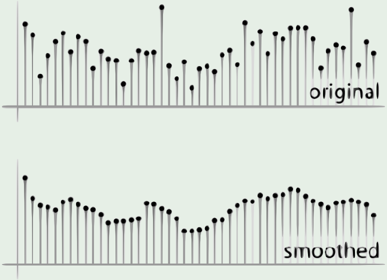
- Sample the 2D space on a regular grid.
- Quantize each sample (round to nearest integer).

The image can now be represented as a matrix of integer values, $I : [a, b] \times [c, d] \to [0..255]$

$$\begin{array}{cc} & j \longrightarrow \\ i \downarrow & \begin{bmatrix} 79 & 5 & 6 & 90 & 12 & 34 & 2 & 1 \\ 8 & 90 & 12 & 34 & 26 & 78 & 34 & 5 \\ 8 & 1 & 3 & 90 & 12 & 34 & 11 & 61 \\ 77 & 90 & 12 & 34 & 200 & 2 & 9 & 45 \\ 1 & 3 & 90 & 12 & 20 & 1 & 6 & 23 \end{bmatrix} \end{array}$$

Many times we want to eliminate noise in a image

For example a moving average



This is defined as

This last moving average can be seen as

$$(I * k)(i) = \sum_{j=-n}^{n} I(i-j) \times K(j) = \frac{1}{N} \sum_{j=m}^{-m} I(i-j) \quad (4)$$

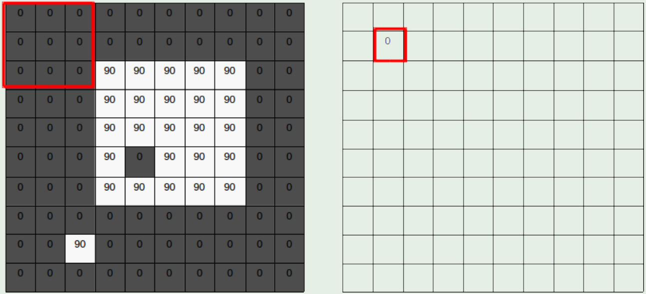
With $I(j)$ representing the value of the pixel at position j,

$$K(j) = \begin{cases} \frac{1}{N} & \text{if } j \in \{-m, -m+1, ..., 1, 0, 1, ..., m-1, m\} \\ 0 & \text{else} \end{cases}$$

with $0 < m < n$.

This can be generalized into the 2D images

Left I and Right $I * K$



This can be generalized into the 2D images

Left I and Right $I * K$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10							

This can be generalized into the 2D images

Left I and Right $I * K$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20						

This can be generalized into the 2D images

Left I and Right $I * K$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Moving average in 2D

Basically in 2D

- We can define different types of filter using the idea of weighted average

$$(I * K)(i, j) = \sum_{s=m}^{-m} \sum_{l=-m}^{m} I(i - s, j - l) \times K(s, l) \tag{5}$$

For example, the Box Filter

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{"The Box Filter"} \tag{6}$$

Moving average in 2D

Basically in 2D

- We can define different types of filter using the idea of weighted average

$$(I * K)(i,j) = \sum_{s=m}^{-m} \sum_{l=-m}^{m} I(i-s, j-l) \times K(s,l) \tag{5}$$

For example, the Box Filter

$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \text{ "The Box Filter"} \tag{6}$$

Another Example

The Gaussian Filter

$$K = \begin{bmatrix} 0 & 1 & 2 & 1 & 0 \\ 1 & 3 & 4 & 3 & 1 \\ 2 & 5 & 9 & 5 & 2 \\ 1 & 3 & 5 & 3 & 1 \\ 0 & 1 & 2 & 1 & 0 \end{bmatrix}$$

Thus, we can define the concept of convolution

- Yes, using the previous ideas

Another Example

The Gaussian Filter

$$K = \begin{bmatrix} 0 & 1 & 2 & 1 & 0 \\ 1 & 3 & 4 & 3 & 1 \\ 2 & 5 & 9 & 5 & 2 \\ 1 & 3 & 5 & 3 & 1 \\ 0 & 1 & 2 & 1 & 0 \end{bmatrix}$$

Thus, we can define the concept of convolution

- Yes, using the previous ideas

Convolution

Definition

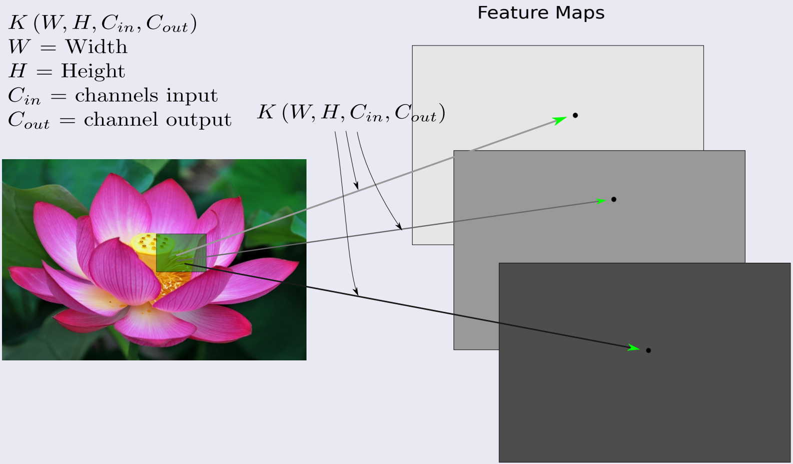
- Let $I : [a,b] \times [c,d] \to [0..255]$ be the image and $K : [e,f] \times [h,i] \to \mathbb{R}$ be the kernel. The output of Convolving I with K, denoted $I * K$ is

$$(I * K)[x,y] = \sum_{s=-n}^{n} \sum_{l=-n}^{n} I(x-s, y-l) \times K(s,l)$$

Now, why not to expand this idea

Imagine that a three channel image is splitted into a three feature map

$K\left(W, H, C_{in}, C_{out}\right)$
W = Width
H = Height
C_{in} = channels input
C_{out} = channel output



Mathematically, we have the following

Map i

$$(I * k)\left[x, y, o\right] = \sum_{c=1}^{3} \sum_{l=-n}^{n} \sum_{s=-n}^{n} I\left(x-l, y-s, c\right) \times k\left(l, s, c, o\right)$$

Therefore

- The convolution works as a
  - Filter
  - Encoder
  - Decoder
  - etc

Mathematically, we have the following

Map i

$$(I * k)[x, y, o] = \sum_{c=1}^{3} \sum_{l=-n}^{n} \sum_{s=-n}^{n} I(x-l, y-s, c) \times k(l, s, c, o)$$

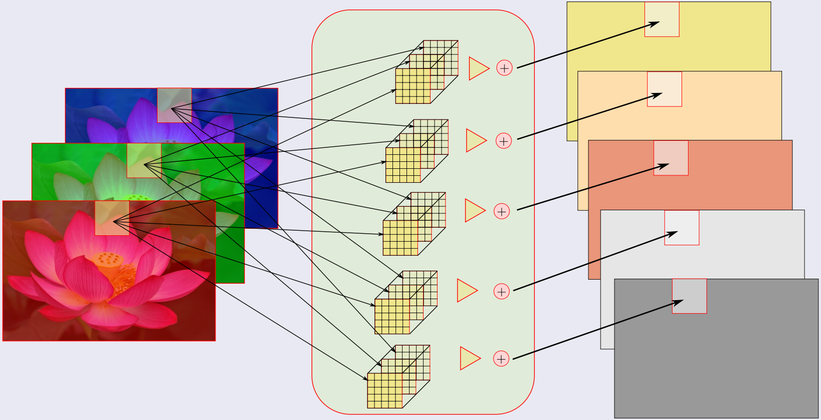
Therefore

- The convolution works as a
  - Filter
  - Encoder
  - Decoder
  - etc

For Example, Encoder

We have the following situation

$$\sum_{c=1}^{3}\sum_{l=-n}^{n}\sum_{s=-n}^{n} I\left(x-l, y-s, c\right)\times k\left(l, s, c, o\right)$$



Notation

We have the following

- $Y_j^{(l)}$ is a matrix representing the l layer and j^{th} feature map.
- $K_{ij}^{(l)}$ is the kernel filter with i^{th} kernel for layer j^{th}.

Therefore

- We can see the Convolutional as a fusion of information from different feature maps.

$$\sum_{j=1}^{m_1^{(l-1)}} Y_j^{(l-1)} * K_{ij}^{(l)}$$

Notation

We have the following

- $Y_j^{(l)}$ is a matrix representing the l layer and j^{th} feature map.
- $K_{ij}^{(l)}$ is the kernel filter with i^{th} kernel for layer j^{th}.

Therefore

- We can see the Convolutional as a fusion of information from different feature maps.

$$\sum_{j=1}^{m_1^{(l-1)}} Y_j^{(l-1)} * K_{ij}^{(l)}$$

Thus, we have

Given a specific layer l, we have that i^{th} feature map in such layer equal to

$$Y_i^{(l)}(x,y) = B_i^{(l)}(x,y) + \sum_{j=1}^{m_1^{(l-1)}} \sum_{s=-ks}^{ks} \sum_{l=-ks}^{ks} Y_j^{(l-1)}(x-s, y-l) K_{ij}^{(l)}(x,y)$$

Thus, we have

Given a specific layer l, we have that i^{th} feature map in such layer equal to

$$Y_i^{(l)}(x,y) = B_i^{(l)}(x,y) + \sum_{j=1}^{m_1^{(l-1)}} \sum_{s=-ks}^{ks} \sum_{l=-ks}^{ks} Y_j^{(l-1)}(x-s, y-l)\, K_{ij}^{(l)}(x,y)$$

Where

- $Y_i^{(l)}$ is the i^{th} feature map in layer l.
- $B_i^{(l)}$ is the bias matrix for output j.
- $K_{ij}^{(l)}$ is the filter of size $\left[2h_1^{(l)}+1\right] \times \left[2h_2^{(l)}+1\right]$.

Thus, we have

Given a specific layer l, we have that i^{th} feature map in such layer equal to

$$Y_i^{(l)}(x,y) = B_i^{(l)}(x,y) + \sum_{j=1}^{m_1^{(l-1)}} \sum_{s=-ks}^{ks} \sum_{l=-ks}^{ks} Y_j^{(l-1)}(x-s, y-l)\, K_{ij}^{(l)}(x,y)$$

Where

- $Y_i^{(l)}$ is the i^{th} feature map in layer l.
- $B_i^{(l)}$ is the bias matrix for output j.
- $K_{ij}^{(l)}$ is the filter of size $\left[2h_1^{(l)}+1\right] \times \left[2h_2^{(l)}+1\right]$.

Thus

- The input of layer l comprises $m_1^{(l-1)}$ feature maps from the previous layer, each of size $m_2^{(l-1)} \times m_3^{(l-1)}$

Thus, we have

Given a specific layer l, we have that i^{th} feature map in such layer equal to

$$Y_i^{(l)}(x,y) = B_i^{(l)}(x,y) + \sum_{j=1}^{m_1^{(l-1)}} \sum_{s=-ks}^{ks} \sum_{l=-ks}^{ks} Y_j^{(l-1)}(x-s, y-l)\, K_{ij}^{(l)}(x,y)$$

Where

- $Y_i^{(l)}$ is the i^{th} feature map in layer l.
- $B_i^{(l)}$ is the bias matrix for output j.
- $K_{ij}^{(l)}$ is the filter of size $\left[2h_1^{(l)}+1\right] \times \left[2h_2^{(l)}+1\right]$.

Thus

- The input of layer l comprises $m_1^{(l-1)}$ feature maps from the previous layer, each of size $m_2^{(l-1)} \times m_3^{(l-1)}$

Thus, we have

Given a specific layer l, we have that i^{th} feature map in such layer equal to

$$Y_i^{(l)}(x,y) = B_i^{(l)}(x,y) + \sum_{j=1}^{m_1^{(l-1)}} \sum_{s=-ks}^{ks} \sum_{l=-ks}^{ks} Y_j^{(l-1)}(x-s, y-l)\, K_{ij}^{(l)}(x,y)$$

Where

- $Y_i^{(l)}$ is the i^{th} feature map in layer l.
- $B_i^{(l)}$ is the bias matrix for output j.
- $K_{ij}^{(l)}$ is the filter of size $\left[2h_1^{(l)}+1\right] \times \left[2h_2^{(l)}+1\right]$.

Thus

- The input of layer l comprises $m_1^{(l-1)}$ feature maps from the previous layer, each of size $m_2^{(l-1)} \times m_3^{(l-1)}$

Therefore

Thew output of layer l

- It consists $m_1^{(l)}$ feature maps of size $m_2^{(l)} \times m_3^{(l)}$

Something Notable

- $m_2^{(l)}$ and $m_3^{(l)}$ are influenced by border effects.
- Therefore, the output feature maps when the Convolutional sum is defined properly have size

$$m_2^{(l)} = m_2^{(l-1)} - 2h_1^{(l)}$$
$$m_3^{(l)} = m_3^{(l-1)} - 2h_2^{(l)}$$

Therefore

Thew output of layer l

- It consists $m_1^{(l)}$ feature maps of size $m_2^{(l)} \times m_3^{(l)}$

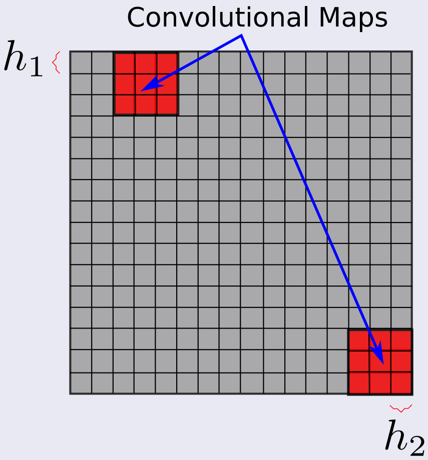
Something Notable

- $m_2^{(l)}$ and $m_3^{(l)}$ are influenced by border effects.
- Therefore, the output feature maps when the Convolutional sum is defined properly have size

$$m_2^{(l)} = m_2^{(l-1)} - 2h_1^{(l)}$$
$$m_3^{(l)} = m_3^{(l-1)} - 2h_2^{(l)}$$

Why? The Border

Example



Special Case

When $l = 1$

The input is a single image I consisting of one or more channels.

Thus

We have

Each feature map $Y_i^{(l)}$ in layer l consists of $m_1^{(l)} \cdot m_2^{(l)}$ units arranged in a two dimensional array.

Thus, the unit at position (x, y) computes

$$\left(Y_i^{(l)}\right)_{x,y} = \left(B_i^{(l)}\right)_{x,y} + \sum_{j=1}^{m_1^{(l-1)}} \left(K_{ij}^{(l)} * Y_j^{(l-1)}\right)_{x,y}$$

$$= \left(B_i^{(l)}\right)_{x,y} + \sum_{j=1}^{m_1^{(l-1)}} \sum_{k=-h_1^{(l)}}^{h_1^{(l)}} \sum_{t=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{ij}^{(l)}\right)_{k,t} \left(Y_j^{(l-1)}\right)_{x-k,x-t}$$

Thus

We have

Each feature map $Y_i^{(l)}$ in layer l consists of $m_1^{(l)} \cdot m_2^{(l)}$ units arranged in a two dimensional array.

Thus, the unit at position (x, y) computes

$$\begin{aligned}\left(Y_i^{(l)}\right)_{x,y} &= \left(B_i^{(l)}\right)_{x,y} + \sum_{j=1}^{m_1^{(l-1)}} \left(K_{ij}^{(l)} * Y_j^{(l-1)}\right)_{x,y} \\ &= \left(B_i^{(l)}\right)_{x,y} + \sum_{j=1}^{m_1^{(l-1)}} \sum_{k=-h_1^{(l)}}^{h_1^{(l)}} \sum_{t=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{ij}^{(l)}\right)_{k,t} \left(Y_j^{(l-1)}\right)_{x-k,x-t}\end{aligned}$$

Here, an interesting case

Only a Historical Note

- The foundations for deconvolution came from Norbert Wiener of the Massachusetts Institute of Technology in his book "Extrapolation, Interpolation, and Smoothing of Stationary Time Series" (1949)

Basically, it tries to solve the following equation with $Y^{(l)}$ unknown layer that we want to recover

$$Y_i^{(l)} * K_{ij}^{(l)} = Y_j^{(l-1)}$$

Here, an interesting case

Only a Historical Note

- The foundations for deconvolution came from Norbert Wiener of the Massachusetts Institute of Technology in his book "Extrapolation, Interpolation, and Smoothing of Stationary Time Series" (1949)

Basically, it tries to solve the following equation with $Y^{(l)}$ unknown layer that we want to recover

$$Y_i^{(l)} * K_{ij}^{(l)} = Y_j^{(l-1)}$$

In [8]

They proposed a sparcity idea to start the implementation as

$$C\left(Y^{(l-1)}\right)=\sum_{i=1}^{m_1^{(l-1)}}\left\|\sum_{j=1}^{m_1^{(l)}} Y_j^{(l)} * K_{ij}^{(l)} - Y_i^{(l-1)}\right\|_2^2+\sum_{j=1}^{m_1^{(l)}}\left|Y_j^{(l)}\right|^p$$

- Typically, $p=1$, although other values are possible.

They look for the arguments to minimize a cost of function over a set of images $y=\left\{y^1,\ldots,y^l\right\}$

$$\arg\min_{Y_j^{(l)} * K_{ij}^{(l)}} C\left(y\right)$$

In [8]

They proposed a sparcity idea to start the implementation as

$$C\left(Y^{(l-1)}\right)=\sum_{i=1}^{m_1^{(l-1)}}\left\|\sum_{j=1}^{m_1^{(l)}} Y_j^{(l)} * K_{ij}^{(l)} - Y_i^{(l-1)}\right\|_2^2+\sum_{j=1}^{m_1^{(l)}}\left|Y_j^{(l)}\right|^p$$

- Typically, $p=1$, although other values are possible.

They look for the arguments to minimize a cost of function over a set of images $y=\left\{y^1,...,y^I\right\}$

$$\arg \min_{Y_j^{(l)} * K_{ij}^{(l)}} C\left(y\right)$$

Here

Then, we can generalize such cost function for that total set of images (Minbatch)

$$C_l\left(y\right)=\frac{\lambda}{2}\sum_{k=1}^{I}\sum_{i=1}^{m_1^{(l-1)}}\left\|\sum_{j=1}^{m_1^{(l)}}g_{ij}^{(l)}\left(Y_j^{(l,k)}*K_{ij}^{(l)}\right)-Y_i^{(l-1,k)}\right\|_2^2+\sum_{j=1}^{m_1^{(l)}}\left|Y_j^{(l,k)}\right|^p$$

Here, we have

- $Y_i^{(l-1,k)}$ are the feature maps from the previous layer
- $g_{ij}^{(l)}$ is a fixed binary matrix that determines the connectivity between feature maps at different layers
    - If $Y_j^{(l,k)}$ is connected to certain $Y_i^{(l-1,k)}$ elments

Here

Then, we can generalize such cost function for that total set of images (Minbatch)

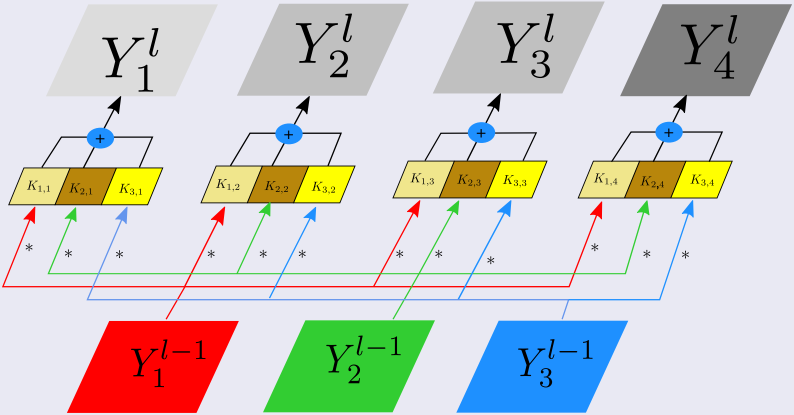
$$C_l\left(y\right) = \frac{\lambda}{2}\sum_{k=1}^{I}\sum_{i=1}^{m_1^{(l-1)}}\left\|\sum_{j=1}^{m_1^{(l)}} g_{ij}^{(l)}\left(Y_j^{(l,k)} * K_{ij}^{(l)}\right) - Y_i^{(l-1,k)}\right\|_2^2 + \sum_{j=1}^{m_1^{(l)}}\left|Y_j^{(l,k)}\right|^p$$

Here, we have

- $Y_i^{(l-1,k)}$ are the feature maps from the previous layer
- $g_{ij}^{(l)}$ is a fixed binary matrix that determines the connectivity between feature maps at different layers
    - If $Y_j^{(l,k)}$ is connected to certain $Y_i^{(l-1,k)}$ elments

This can be sen as

We have the following layer



They noticed some drawbacks

Using the following optimizations

- Direct Gradient Descent
- Iterative Reweighted Least Squares
- Stochastic Gradient Descent

All of they presented problems!!!

- They solved it using a new cost function

They noticed some drawbacks

Using the following optimizations

- Direct Gradient Descent
- Iterative Reweighted Least Squares
- Stochastic Gradient Descent

All of they presented problems!!!

- They solved it using a new cost function

We have that

An interesting use of an auxiliar variable/layer $X_i^{(l,k)}$

$$C_l\left(y\right)=\frac{\lambda}{2}\sum_{k=1}^{I}\sum_{i=1}^{m_1^{(l-1)}}\left\|\sum_{j=1}^{m_1^{(l)}}g_{ij}^{(l)}\left(Y_j^{(l,k)}*K_{ij}^{(l)}\right)-Y_i^{(l-1,k)}\right\|_2^2+\ldots$$

$$\frac{\beta}{2}\sum_{k=1}^{I}\sum_{j=1}^{m_1^{(l)}}\left\|Y_j^{(l,k)}-X_i^{(l,k)}\right\|_2^2+\sum_{k=1}^{I}\sum_{j=1}^{m_1^{(l)}}\left|Y_j^{(l,k)}\right|^p$$

This can be solved using

- Alternating minimization...

We have that

An interesting use of an auxiliar variable/layer $X_i^{(l,k)}$

$$C_l\left(y\right) = \frac{\lambda}{2}\sum_{k=1}^{I}\sum_{i=1}^{m_1^{(l-1)}}\left\|\sum_{j=1}^{m_1^{(l)}} g_{ij}^{(l)}\left(Y_j^{(l,k)} * K_{ij}^{(l)}\right) - Y_i^{(l-1,k)}\right\|_2^2 + \dots$$

$$\frac{\beta}{2}\sum_{k=1}^{I}\sum_{j=1}^{m_1^{(l)}}\left\|Y_j^{(l,k)} - X_i^{(l,k)}\right\|_2^2 + \sum_{k=1}^{I}\sum_{j=1}^{m_1^{(l)}}\left|Y_j^{(l,k)}\right|^p$$

This can be solved using

- Alternating minimization...

This is based on

Fixing the values of $Y_j^{(l,k)}$ and $X_i^{(l,k)}$

- They call these two stages the Y and X sub-problems...

Therefore, they noticed

- These terms introduce the sparsity constraint and gives numerical stability [9, 10]

This is based on

Fixing the values of $Y_j^{(l,k)}$ and $X_i^{(l,k)}$

- They call these two stages the Y and X sub-problems...

Therefore, they noticed

- These terms introduce the sparsity constraint and gives numerical stability [9, 10]

Y sub-problem

Taking the derivative of $Y_j^{(l,k)}$

$$\frac{\partial C_l\left(y\right)}{\partial Y_j^{(l,k)}} = \lambda \sum_{i=1}^{m_1^{(l-1)}} F_{ij}^{(l)T} \left[\sum_{t=1}^{m_1^{(l)}} F_{tj}^{(l)} Y_j^{(l,k)} - Y_j^{(l-1,k)} \right] + \beta \left[Y_j^{(l,k)} - X_j^{(l,k)} \right] = 0$$

Where

$$F_{ij}^{(l)} = \begin{cases} \text{It is a sparse convolution matrix} & \text{if } g_{ij}^{(l)} = 1 \\ 0 & \text{if } g_{ij}^{(l)} = 0 \end{cases}$$

Y sub-problem

Taking the derivative of $Y_j^{(l,k)}$

$$\frac{\partial C_l\left(y\right)}{\partial Y_j^{(l,k)}} = \lambda \sum_{i=1}^{m_1^{(l-1)}} F_{ij}^{(l)T} \left[\sum_{t=1}^{m_1^{(l)}} F_{tj}^{(l)} Y_j^{(l,k)} - Y_j^{(l-1,k)} \right] + \beta \left[Y_j^{(l,k)} - X_j^{(l,k)} \right] = 0$$

Where

$$F_{ij}^{(l)} = \begin{cases} \text{It is a sparse convolution matrix} & \text{if } g_{ij}^{(l)} = 1 \\ 0 & \text{if } g_{ij}^{(l)} = 0 \end{cases}$$

Therefore

$F_{ij}^{(l)}$ as a sparse convolution matrix

- Equivalent to convolve with $K_{ij}^{(l)}$

Actually if you fix i, you finish with a linear system $Ax = 0$

- Please take a look at the paper... it is interesting
  - Actually this seems to be the implementation at the Tensorflow framework

Therefore

$F_{ij}^{(l)}$ as a sparse convolution matrix

- Equivalent to convolve with $K_{ij}^{(l)}$

Actually if you fix i, you finish with a linear system $Ax = 0$

- Please take a look at the paper... it is interesting
  - Actually this seems to be the implementation at the Tensorflow framework

Outline

As in a Multilayer Perceptron

We use a non-linearity

- However, there is a drawback when using Back-Propagation under a sigmoid function

$$s(x) = \frac{1}{1 + e^{-x}}$$

Because if we imagine a Convolutional Network as a series of layer functions f_i

$$y(A) = f_t \circ f_{t-1} \circ \cdots \circ f_2 \circ f_1(A)$$

With f_t is the last layer.

Therefore, we finish with a sequence of derivatives

$$\frac{\partial y(A)}{\partial w_{1i}} = \frac{\partial f_t(f_{t-1})}{\partial f_{t-1}} \cdot \frac{\partial f_{t-1}(f_{t-2})}{\partial f_{t-2}} \cdot \cdots \cdot \frac{\partial f_2(f_1)}{\partial f_2} \cdot \frac{\partial f_1(A)}{\partial w_{1i}}$$

As in a Multilayer Perceptron

We use a non-linearity

- However, there is a drawback when using Back-Propagation under a sigmoid function

$$s(x) = \frac{1}{1 + e^{-x}}$$

Because if we imagine a Convolutional Network as a series of layer functions f_i

$$y(A) = f_t \circ f_{t-1} \circ \cdots \circ f_2 \circ f_1(A)$$

With f_t is the last layer.

Therefore, we finish with a sequence of derivatives

$$\frac{\partial y(A)}{\partial w_{1i}} = \frac{\partial f_t(f_{t-1})}{\partial f_{t-1}} \cdot \frac{\partial f_{t-1}(f_{t-2})}{\partial f_{t-2}} \cdot \cdots \cdot \frac{\partial f_2(f_1)}{\partial f_2} \cdot \frac{\partial f_1(A)}{\partial w_{1i}}$$

As in a Multilayer Perceptron

We use a non-linearity

- However, there is a drawback when using Back-Propagation under a sigmoid function

$$s(x) = \frac{1}{1+e^{-x}}$$

Because if we imagine a Convolutional Network as a series of layer functions f_i

$$y(A) = f_t \circ f_{t-1} \circ \cdots \circ f_2 \circ f_1(A)$$

With f_t is the last layer.

Therefore, we finish with a sequence of derivatives

$$\frac{\partial y(A)}{\partial w_{1i}} = \frac{\partial f_t(f_{t-1})}{\partial f_{t-1}} \cdot \frac{\partial f_{t-1}(f_{t-2})}{\partial f_{t-2}} \cdot \cdots \cdot \frac{\partial f_2(f_1)}{\partial f_2} \cdot \frac{\partial f_1(A)}{\partial w_{1i}}$$

Therefore

Given the commutativity of the product

- You could put together the derivative of the sigmoid's

$$f\left(x\right)=\frac{ds\left(x\right)}{dx}=\frac{e^{-x}}{\left(1+e^{-x}\right)^{2}}$$

Therefore, deriving again

$$\frac{df\left(x\right)}{dx}=-\frac{e^{-x}}{\left(1+e^{-x}\right)^{2}}+\frac{2\left(e^{-x}\right)^{2}}{\left(1+e^{-x}\right)^{3}}$$

After making $\frac{df(x)}{dx}=0$

- We have the maximum is at $x=0$

Therefore

Given the commutativity of the product

- You could put together the derivative of the sigmoid's

$$f(x) = \frac{ds(x)}{dx} = \frac{e^{-x}}{\left(1+e^{-x}\right)^2}$$

Therefore, deriving again

$$\frac{df(x)}{dx} = -\frac{e^{-x}}{\left(1+e^{-x}\right)^2} + \frac{2\left(e^{-x}\right)^2}{\left(1+e^{-x}\right)^3}$$

After making $\frac{df(x)}{dx} = 0$

- We have the maximum is at $x = 0$

Therefore

Given the commutativity of the product

- You could put together the derivative of the sigmoid's

$$f(x) = \frac{ds(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2}$$

Therefore, deriving again

$$\frac{df(x)}{dx} = -\frac{e^{-x}}{(1+e^{-x})^2} + \frac{2(e^{-x})^2}{(1+e^{-x})^3}$$

After making $\frac{df(x)}{dx} = 0$

- We have the maximum is at $x = 0$

Therefore

The maximum for the derivative of the sigmoid

- $f(0) = 0.25$

Therefore, Given a Deep Convolutional Network

- We could finish with

$$\lim_{k\to\infty}\left(\frac{ds(x)}{dx}\right)^k = \lim_{k\to\infty}(0.25)^k \to 0$$

A vanishing derivative

- Making quite difficult to do train a deeper network using this activation function

Therefore

The maximum for the derivative of the sigmoid

- $f(0) = 0.25$

Therefore, Given a **Deep** Convolutional Network

- We could finish with

$$\lim_{k \to \infty} \left(\frac{ds(x)}{dx} \right)^k = \lim_{k \to \infty} (0.25)^k \to 0$$

A vanishing derivative

- Making quite difficult to do train a deeper network using this activation function

Therefore

The maximum for the derivative of the sigmoid

- $f(0) = 0.25$

Therefore, Given a **Deep** Convolutional Network

- We could finish with

$$\lim_{k \to \infty} \left(\frac{ds(x)}{dx} \right)^k = \lim_{k \to \infty} (0.25)^k \to 0$$

A vanishing derivative

- Making quite difficult to do train a deeper network using this activation function

Thus

The need to introduce a new function

$$f(x) = x^{+} = \max(0, x)$$

It is called ReLu or Rectifier

With a smooth approximation (Softplus function)

$$f(x) = \frac{\ln\left(1 + e^{kx}\right)}{k}$$

Thus

The need to introduce a new function

$$f(x) = x^{+} = \max(0, x)$$

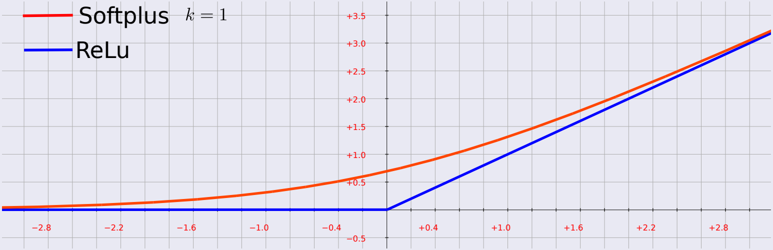
It is called ReLu or Rectifier

With a smooth approximation (Softplus function)

$$f(x) = \frac{\ln\left(1 + e^{kx}\right)}{k}$$

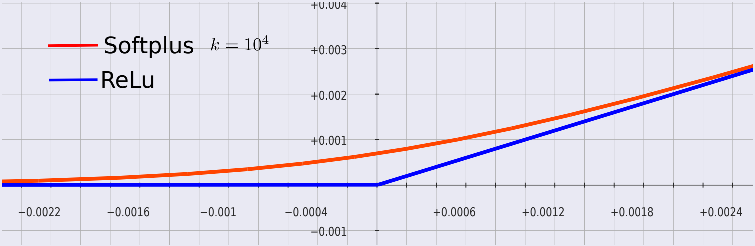
Therefore, we have

When $k = 1$



Increase k

When $k = 10^4$



Non-Linearity Layer

If layer l is a non-linearity layer

Its input is given by $m_1^{(l)}$ feature maps.

What about the output

Its output comprises again $m_1^{(l)} = m_1^{(l-1)}$ feature maps

Each of them of size

$$m_2^{(l-1)} \times m_3^{(l-1)} \quad (7)$$

With $m_2^{(l)} = m_2^{(l-1)}$ and $m_3^{(l)} = m_3^{(l-1)}$.

Non-Linearity Layer

If layer l is a non-linearity layer

Its input is given by $m_1^{(l)}$ feature maps.

What about the output

Its output comprises again $m_1^{(l)} = m_1^{(l-1)}$ feature maps

Each of them of size

$$m_2^{(l-1)} \times m_3^{(l-1)} \quad (7)$$

With $m_2^{(l)} = m_2^{(l-1)}$ and $m_3^{(l)} = m_3^{(l-1)}$.

Non-Linearity Layer

If layer l is a non-linearity layer

Its input is given by $m_1^{(l)}$ feature maps.

What about the output

Its output comprises again $m_1^{(l)} = m_1^{(l-1)}$ feature maps

Each of them of size

$$m_2^{(l-1)} \times m_3^{(l-1)} \tag{7}$$

With $m_2^{(l)} = m_2^{(l-1)}$ and $m_3^{(l)} = m_3^{(l-1)}$.

Thus

With the final output

$$Y_i^{(l)} = f\left(Y_i^{(l-1)}\right) \tag{8}$$

Where

f is the activation function used in layer l and operates point wise.

You can also add a gain to compensate

$$Y_i^{(l)} = g_i f\left(Y_i^{(l-1)}\right) \tag{9}$$

Thus

With the final output

$$Y_i^{(l)} = f\left(Y_i^{(l-1)}\right) \quad (8)$$

Where

f is the activation function used in layer l and operates point wise.

You can also add a gain to compensate

$$Y_i^{(l)} = g_i f\left(Y_i^{(l-1)}\right) \quad (9)$$

Thus

With the final output

$$Y_i^{(l)} = f\left(Y_i^{(l-1)}\right) \quad (8)$$

Where

f is the activation function used in layer l and operates point wise.

You can also add a gain to compensate

$$Y_i^{(l)} = g_i f\left(Y_i^{(l-1)}\right) \quad (9)$$

Outline

Rectification Layer, R_{abs}

Now a rectification layer

Then its input comprises $m_1^{(l)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$.

Then, the absolute value for each component of the feature maps is computed

$$Y_i^{(l)} = \left| Y_i^{(l)} \right| \tag{10}$$

Where the absolute value

It is computed point wise such that the output consists of $m_1^{(l)} = m_1^{(l-1)}$ feature maps unchanged in size.

Rectification Layer, R_{abs}

Now a rectification layer

Then its input comprises $m_1^{(l)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$.

Then, the absolute value for each component of the feature maps is computed

$$Y_i^{(l)} = \left| Y_i^{(l)} \right| \quad (10)$$

Where the absolute value

It is computed point wise such that the output consists of $m_1^{(l)} = m_1^{(l-1)}$ feature maps unchanged in size.

Rectification Layer, R_{abs}

Now a rectification layer

Then its input comprises $m_1^{(l)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$.

Then, the absolute value for each component of the feature maps is computed

$$Y_i^{(l)} = \left| Y_i^{(l)} \right| \quad (10)$$

Where the absolute value

It is computed point wise such that the output consists of $m_1^{(l)} = m_1^{(l-1)}$ feature maps unchanged in size.

Thus

We have that

Experiments show that rectification plays a central role in achieving good performance.

You can find this in

K. Jarrett, K. Kavukcuogl, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In Computer Vision, International Conference on, pages 2146–2153, 2009.

Remark

- Rectification could be included in the non-linearity layer.
- But also it can be seen as an independent layer.

Thus

We have that

Experiments show that rectification plays a central role in achieving good performance.

You can find this in

K. Jarrett, K. Kavukcuogl, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In Computer Vision, International Conference on, pages 2146–2153, 2009.

Remark

- Rectification could be included in the non-linearity layer.
- But also it can be seen as an independent layer.

Thus

We have that

Experiments show that rectification plays a central role in achieving good performance.

You can find this in

K. Jarrett, K. Kavukcuogl, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In Computer Vision, International Conference on, pages 2146–2153, 2009.

Remark

- Rectification could be included in the non-linearity layer.
- But also it can be seen as an independent layer.

Given that we are using Backpropagation

We need a soft approximation to $f(x) = |x|$

For this, we have

$$\frac{\partial f}{\partial x} = \text{sgn}(x)$$

- When $x \neq 0$. Why?

We can use the following approximation

$$\text{sgn}(x) = 2\left(\frac{\exp\{kx\}}{1+\exp\{kx\}}\right) - 1$$

Therefore, we have by integration and working the C

$$f(x) = \frac{2}{k}\ln(1+\exp\{kx\}) - x - \frac{2}{k}\ln(2)$$

Given that we are using Backpropagation

We need a soft approximation to $f(x) = |x|$

For this, we have

$$\frac{\partial f}{\partial x} = \mathsf{sgn}(x)$$

- When $x \neq 0$. Why?

We can use the following approximation

$$\mathsf{sgn}(x) = 2\left(\frac{\exp\{kx\}}{1+\exp\{kx\}}\right) - 1$$

Therefore, we have by integration and working the C

$$f(x) = \frac{2}{k}\ln\left(1+\exp\{kx\}\right) - x - \frac{2}{k}\ln(2)$$

Given that we are using Backpropagation

We need a soft approximation to $f(x) = |x|$

For this, we have

$$\frac{\partial f}{\partial x} = \mathsf{sgn}(x)$$

- When $x \neq 0$. Why?

We can use the following approximation

$$\mathsf{sgn}(x) = 2\left(\frac{\exp\{kx\}}{1+\exp\{kx\}}\right) - 1$$

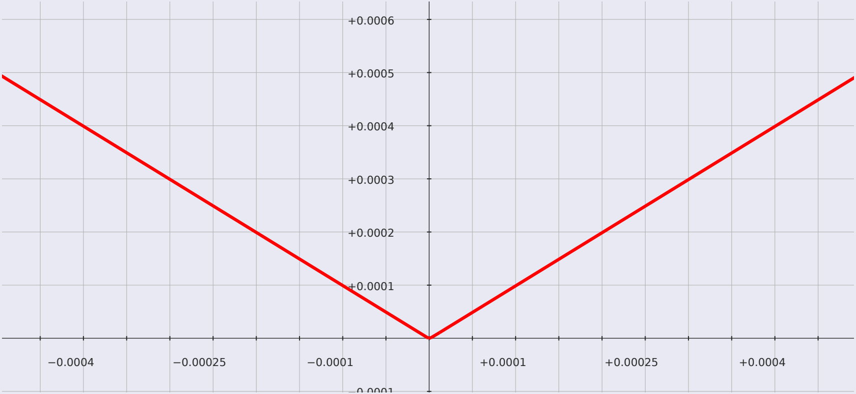
Therefore, we have by integration and working the C

$$f(x) = \frac{2}{k}\ln\left(1+\exp\{kx\}\right) - x - \frac{2}{k}\ln(2)$$

We get the following situation

Something Notable

$$f(x) = \frac{2}{k}\ln\left(1+\exp\left\{kx\right\}\right) - x - \frac{2}{k}\ln(2)$$



Outline

Normalizing

Contrast normalization layer

The task of a local contrast normalization layer:

- To enforce local competitiveness between adjacent units within a feature map.
- To enforce competitiveness units at the same spatial location.

Normalizing

Contrast normalization layer

The task of a local contrast normalization layer:

- To enforce local competitiveness between adjacent units within a feature map.
- To enforce competitiveness units at the same spatial location.

We have two types of operations

- Subtractive Normalization.
- Brightness Normalization.

Normalizing

Contrast normalization layer

The task of a local contrast normalization layer:

- To enforce local competitiveness between adjacent units within a feature map.
- To enforce competitiveness units at the same spatial location.

We have two types of operations

- Subtractive Normalization.
- Brightness Normalization.

Normalizing

Contrast normalization layer

The task of a local contrast normalization layer:

- To enforce local competitiveness between adjacent units within a feature map.
- To enforce competitiveness units at the same spatial location.

We have two types of operations

- Subtractive Normalization.
- Brightness Normalization.

Normalizing

Contrast normalization layer

The task of a local contrast normalization layer:

- To enforce local competitiveness between adjacent units within a feature map.
- To enforce competitiveness units at the same spatial location.

We have two types of operations

- Subtractive Normalization.
- Brightness Normalization.

Subtractive Normalization

Given $m_1^{(l-1)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$

The output of layer l comprises $m_1^{(l)} = m_1^{(l-1)}$ feature maps unchanged in size.

With the operation

$$Y_i^{(l)} = Y_i^{(l-1)} - \sum_{j=1}^{m_1^{(l-1)}} K_{G(\sigma)} * Y_j^{(l-1)} \tag{11}$$

With

$$\left(K_{G(\sigma)}\right)_{x,y} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{\frac{x^2+y^2}{2\sigma^2}\right\} \tag{12}$$

Subtractive Normalization

Given $m_1^{(l-1)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$

The output of layer l comprises $m_1^{(l)} = m_1^{(l-1)}$ feature maps unchanged in size.

With the operation

$$Y_i^{(l)} = Y_i^{(l-1)} - \sum_{j=1}^{m_1^{(l-1)}} K_{G(\sigma)} * Y_j^{(l-1)} \tag{11}$$

With

$$\left(K_{G(\sigma)}\right)_{x,y} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{\frac{x^2+y^2}{2\sigma^2}\right\} \tag{12}$$

Subtractive Normalization

Given $m_1^{(l-1)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$

The output of layer l comprises $m_1^{(l)} = m_1^{(l-1)}$ feature maps unchanged in size.

With the operation

$$Y_i^{(l)} = Y_i^{(l-1)} - \sum_{j=1}^{m_1^{(l-1)}} K_{G(\sigma)} * Y_j^{(l-1)} \tag{11}$$

With

$$\left(K_{G(\sigma)}\right)_{x,y} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{\frac{x^2+y^2}{2\sigma^2}\right\} \tag{12}$$

Brightness Normalization

An alternative is to normalize the brightness in combination with the rectified linear units

$$\left(Y_i^{(l)}\right)_{x,y} = \frac{\left(Y_i^{(l-1)}\right)_{x,y}}{\left(\kappa + \lambda \sum_{j=1}^{m_1^{(l-1)}} \left(Y_j^{(l-1)}\right)_{x,y}^2\right)^{\mu}} \tag{13}$$

Where

- κ, μ and λ are hyperparameters which can be set using a

$$f(x) = \frac{\ln\left(1 + e^{kx}\right)}{k}$$

validation set.

Brightness Normalization

An alternative is to normalize the brightness in combination with the **rectified linear units**

$$\left(Y_i^{(l)}\right)_{x,y} = \frac{\left(Y_i^{(l-1)}\right)_{x,y}}{\left(\kappa + \lambda \sum_{j=1}^{m_1^{(l-1)}} \left(Y_j^{(l-1)}\right)_{x,y}^2\right)^{\mu}} \tag{13}$$

Where

- κ, μ and λ are hyperparameters which can be set using a

$$f(x) = \frac{\ln\left(1 + e^{kx}\right)}{k}$$

validation set.

Outline

Sub-sampling Layer

Motivation

The motivation of subsampling the feature maps obtained by previous layers is robustness to noise and distortions.

How?

- Normally, in traditional Convolutional Networks subsampling this is done by applying skipping factors!!!
- However, it is possible to combine subsampling with pooling and do it in a separate layer

Sub-sampling Layer

Motivation

The motivation of subsampling the feature maps obtained by previous layers is robustness to noise and distortions.

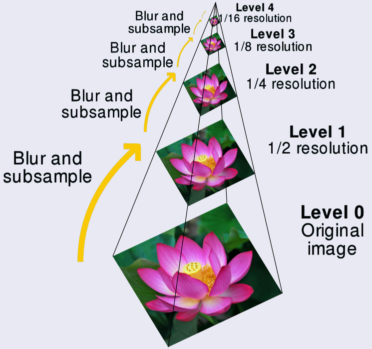
How?

- Normally, in traditional Convolutional Networks subsampling this is done by applying skipping factors!!!
- However, it is possible to combine subsampling with pooling and do it in a separate layer

Sub-sampling

The subsampling layer

- It seems to be acting as the well know sub-sampling pyramid



How is sub-sampling implemented?

We know that Image Pyramids

They were designed to find:

1. filter-based representations to decompose images into information at multiple scales,
2. To extract features/structures of interest,
3. To attenuate noise.

How is sub-sampling implemented?

We know that Image Pyramids

They were designed to find:

1. filter-based representations to decompose images into information at multiple scales,
2. To extract features/structures of interest,
3. To attenuate noise.

Example of usage of this filters

- The SURF and SIFT filters

How is sub-sampling implemented?

We know that Image Pyramids

They were designed to find:

1. filter-based representations to decompose images into information at multiple scales,
2. To extract features/structures of interest,
3. To attenuate noise.

Example of usage of this filters

- The SURF and SIFT filters

How is sub-sampling implemented?

We know that Image Pyramids

They were designed to find:

1. filter-based representations to decompose images into information at multiple scales,
2. To extract features/structures of interest,
3. To attenuate noise.

Example of usage of this filters

- The SURF and SIFT filters

How is sub-sampling implemented?

We know that Image Pyramids

They were designed to find:

1. filter-based representations to decompose images into information at multiple scales,
2. To extract features/structures of interest,
3. To attenuate noise.

Example of usage of this filters

- The SURF and SIFT filters

There are also other ways of doing this

subsampling can be done using so called skipping factors

$$s_1^{(l)} \text{ and } s_2^{(l)}$$

The basic idea is to skip a fixed number of pixels

Therefore the size of the output feature map is given by

$$m_2^{(l)} = \frac{m_2^{(l-1)} - 2h_1^{(l)}}{s_1^{(l)} + 1} \text{ and } m_3^{(l)} = \frac{m_3^{(l-1)} - 2h_2^{(l)}}{s_2^{(l)} + 1}$$

There are also other ways of doing this

subsampling can be done using so called skipping factors

$$s_1^{(l)} \text{ and } s_2^{(l)}$$

The basic idea is to skip a fixed number of pixels

Therefore the size of the output feature map is given by

$$m_2^{(l)} = \frac{m_2^{(l-1)} - 2h_1^{(l)}}{s_1^{(l)} + 1} \text{ and } m_3^{(l)} = \frac{m_3^{(l-1)} - 2h_2^{(l)}}{s_2^{(l)} + 1}$$

What is Pooling?

Pooling

- **Spatial pooling is way to compute image representation based on encoded local features.**

Pooling

Let l be a pooling layer

- It outputs from $m_i^{(l)} > m_i^{(l-1)}$ feature maps of reduced size.

Pooling Operation

It operates by placing windows at non-overlapping positions in each feature map and keeping one value per window such that the feature maps are sub-sampled.

Pooling

Let l be a pooling layer

- It outputs from $m_i^{(l)} > m_i^{(l-1)}$ feature maps of reduced size.

Pooling Operation

It operates by placing windows at non-overlapping positions in each feature map and keeping one value per window such that the feature maps are sub-sampled.

Thus

In the previous example

- All feature maps are pooled and sub-sampled individually.

Each unit

- In one of the $m_1^{(l)} = 4$ output feature maps represents the average or the maximum within a fixed window of the corresponding feature map in layer $(l-1)$.

Thus

In the previous example

- All feature maps are pooled and sub-sampled individually.

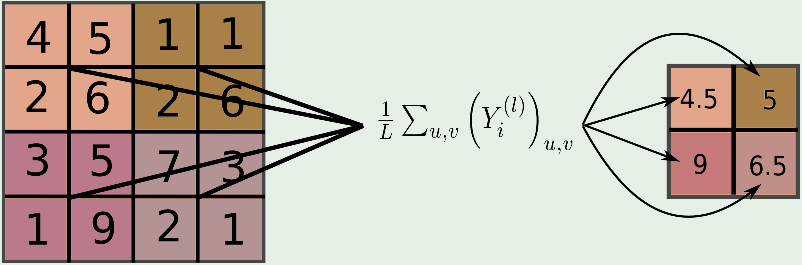
Each unit

- In one of the $m_1^{(l)} = 4$ output feature maps represents the average or the maximum within a fixed window of the corresponding feature map in layer $(l-1)$.

Examples of pooling

Average pooling

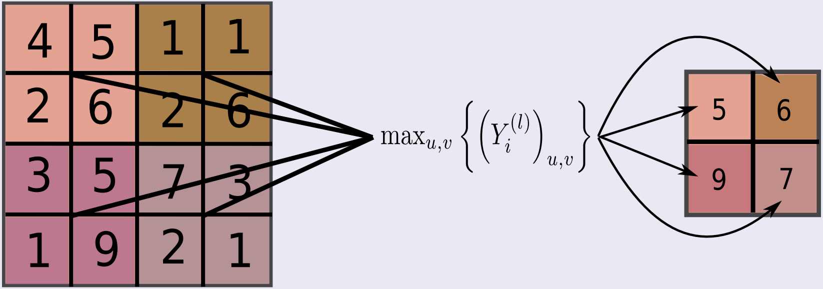
When using a boxcar filter, the operation is called average pooling and the layer denoted by P_A.



Examples of pooling

Max pooling

For max pooling, the maximum value of each window is taken. The layer is denoted by P_M.



An interesting property

Something notable depending in the pooling area

- "In all cases, pooling helps to make the representation become approximately invariant to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change."
  - Page 342, Ian Goodfellow, Introduction to Deep Learning, 2016 [11].

The small amount

- In the case of the previous examples, **1 pixel**

An interesting property

Something notable depending in the pooling area

- "In all cases, pooling helps to make the representation become approximately invariant to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change."
  - Page 342, Ian Goodfellow, Introduction to Deep Learning, 2016 [11].

The small amount

- In the case of the previous examples, **1 pixel**

Other Poolings

There are other types of pooling

- L_2 norm of a rectangular neighborhood
- Weighted average based on the distance from the central pixel

However, we have another way of doing pooling

- Striding!!!

Other Poolings

There are other types of pooling

- L_2 norm of a rectangular neighborhood
- Weighted average based on the distance from the central pixel

However, we have another way of doing pooling

- Striding!!!

Springerberg et al. [12]

They started talking about sustituing maxpooling for something called a Stride on the Convolution

$$\left(Y_i^{(l)}\right)_{x,y} = \left(B_i^{(l)}\right)_{x,y} + \sum_{j=1}^{m_1^{(l-1)}} \sum_{k=-h_1^{(l)}}^{h_1^{(l)}} \sum_{t=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{ij}^{(l)}\right)_{k,t} \left(Y_j^{(l-1)}\right)_{x-k,x-t}$$

This is a Heuristic ...

- Basically you jump around by a factro r and t for the width and height of the layer
  - It was proposed to decrease memory usage...

Springerberg et al. [12]

They started talking about sustituing maxpooling for something called a Stride on the Convolution

$$\left(Y_i^{(l)}\right)_{x,y} = \left(B_i^{(l)}\right)_{x,y} + \sum_{j=1}^{m_1^{(l-1)}} \sum_{k=-h_1^{(l)}}^{h_1^{(l)}} \sum_{t=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{ij}^{(l)}\right)_{k,t} \left(Y_j^{(l-1)}\right)_{x-k,x-t}$$

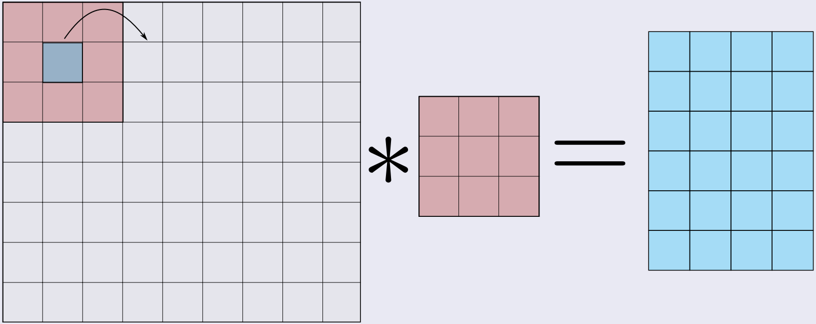
This is a Heuristic ...

- Basically you jump around by a factro r and t for the width and height of the layer
  - It was proposed to decrease memory usage...

Example

Horizontal Stride

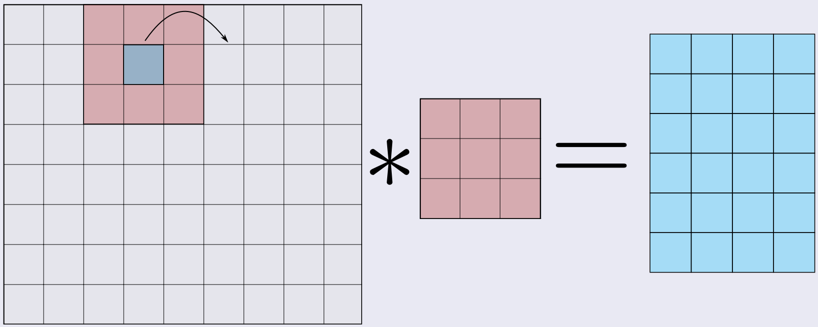
Horizontal Stride $r = 2$



Example

Horizontal Stride

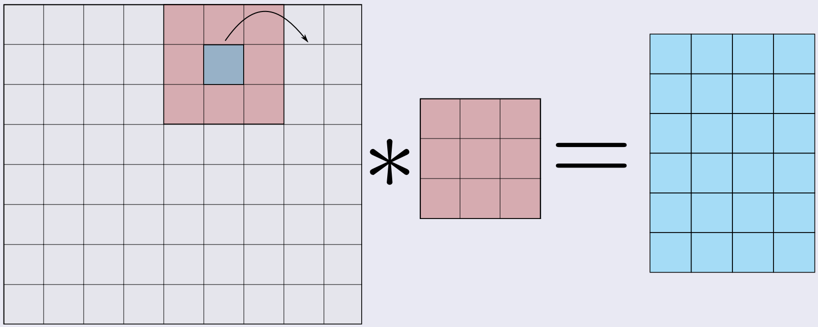
Horizontal Stride $r = 2$



Example

Horizontal Stride

Horizontal Stride $r = 2$



There are attempts to understand its effects

At Convolution Level and using Tensors [13]

- "Take it in your stride: Do we need striding in CNNs?" by Chen Kong, Simon Lucey [14]

Please read Kolda's Paper before you get into the other

- You need a little bit of notation...

There are attempts to understand its effects

At Convolution Level and using Tensors [13]

- "Take it in your stride: Do we need striding in CNNs?" by Chen Kong, Simon Lucey [14]

Please read Kolda's Paper before you get into the other

- You need a little bit of notation...

Outline

Here, the people at Google [15] around 2015

They commented in the "Internal Covariate Shift Phenomena"

- Due to the change in the distribution of each layer's input

They claim

- The min-batch forces to have those changes which impact on the learning capabilities of the network.

In Neural Networks, they define this

- Internal Covariate Shift as the change in the distribution of network activation's due to the change in network parameters during training.

Here, the people at Google [15] around 2015

They commented in the "Internal Covariate Shift Phenomena"

- Due to the change in the distribution of each layer's input

They claim

- The min-batch forces to have those changes which impact on the learning capabilities of the network.

In Neural Networks, they define this

- Internal Covariate Shift as the change in the distribution of network activation's due to the change in network parameters during training.

Here, the people at Google [15] around 2015

They commented in the "Internal Covariate Shift Phenomena"

- Due to the change in the distribution of each layer's input

They claim

- The min-batch forces to have those changes which impact on the learning capabilities of the network.

In Neural Networks, they define this

- Internal Covariate Shift as the change in the distribution of network activation's due to the change in network parameters during training.

They gave the following reasons

Consider a layer with the input u that adds the learned bias b

- Then, it normalizes the result by subtracting the mean of the activation over the training data:

$$\widehat{\boldsymbol{x}} = \boldsymbol{x} - E\left[\boldsymbol{x}\right]$$

  - $\mathcal{X} = \{\boldsymbol{x}, ..., \boldsymbol{x}_N\}$ the data samples and $E\left[\boldsymbol{x}\right] = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i$

Now, if the gradient ignores the dependence of $E\left[x\right]$ on b

- Then $b = b + \Delta b$ where $\Delta b \propto -\frac{\partial l}{\partial \widehat{x}}$

Finally

$$u + (b + \Delta b) - E[u + (b + \Delta b)] = u + b - E[u + b]$$

They gave the following reasons

Consider a layer with the input u that adds the learned bias b

- Then, it normalizes the result by subtracting the mean of the activation over the training data:

$$\widehat{\boldsymbol{x}} = \boldsymbol{x} - E\left[\boldsymbol{x}\right]$$

  - $\mathcal{X} = \{\boldsymbol{x}, ..., \boldsymbol{x}_N\}$ the data samples and $E\left[\boldsymbol{x}\right] = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i$

Now, if the gradient ignores the dependence of $E\left[x\right]$ on b

- Then $b = b + \Delta b$ where $\Delta b \propto -\frac{\partial l}{\partial \widehat{x}}$

Finally

$$u + (b + \Delta b) - E[u + (b + \Delta b)] = u + b - E[u + b]$$

They gave the following reasons

Consider a layer with the input u that adds the learned bias b

- Then, it normalizes the result by subtracting the mean of the activation over the training data:

$$\widehat{\boldsymbol{x}} = \boldsymbol{x} - E\left[\boldsymbol{x}\right]$$

  - $\mathcal{X} = \{\boldsymbol{x}, ..., \boldsymbol{x}_N\}$ the data samples and $E\left[\boldsymbol{x}\right] = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i$

Now, if the gradient ignores the dependence of $E\left[x\right]$ on b

- Then $b = b + \Delta b$ where $\Delta b \propto -\frac{\partial l}{\partial \widehat{x}}$

Finally

$$u + (b + \Delta b) - E[u + (b + \Delta b)] = u + b - E[u + b]$$

Then

The following will happen

- The update to b by Δb leads to **no change** in the output of the layer.

Therefore

- We need to integrate the normalization into the process of training.

Then

The following will happen

- The update to b by Δb leads to **no change** in the output of the layer.

Therefore

- We need to integrate the normalization into the process of training.

Normalization via Mini-Batch Statistic

It is possible to describe the normalization as a transformation layer

$$\widehat{\boldsymbol{x}} = Norm\left(\boldsymbol{x}, \mathcal{X}\right)$$

- Which depends on all the training samples $\mathcal{X}$ which also depends on the layer parameters

For back-propagation, we will need to generate the following terms

$$\frac{\partial Norm\left(\boldsymbol{x}, \mathcal{X}\right)}{\partial \boldsymbol{x}} \text{ and } \frac{\partial Norm\left(\boldsymbol{x}, \mathcal{X}\right)}{\partial \mathcal{X}}$$

Normalization via Mini-Batch Statistic

It is possible to describe the normalization as a transformation layer

$$\widehat{\boldsymbol{x}} = Norm\left(\boldsymbol{x}, \mathcal{X}\right)$$

- Which depends on all the training samples $\mathcal{X}$ which also depends on the layer parameters

For back-propagation, we will need to generate the following terms

$$\frac{\partial Norm\left(\boldsymbol{x}, \mathcal{X}\right)}{\partial \boldsymbol{x}} \text{ and } \frac{\partial Norm\left(\boldsymbol{x}, \mathcal{X}\right)}{\partial \mathcal{X}}$$

Definition of Whitening

Whitening

- Suppose X is a random (column) vector with non-singular covariance matrix Σ and mean 0.

Then

- Then the transformation $Y = WX$ with a whitening matrix W satisfying the condition $W^T W = \Sigma^{-1}$ yields the whitened random vector Y with unit diagonal covariance.

Definition of Whitening

Whitening

- Suppose X is a random (column) vector with non-singular covariance matrix Σ and mean 0.

Then

- Then the transformation $Y = WX$ with a whitening matrix W satisfying the condition $W^T W = \Sigma^{-1}$ yields the whitened random vector Y with unit diagonal covariance.

Such Normalization

It could be used for all layer

- But whitening the layer inputs is expensive, as it requires computing the covariance matrix

$$Cov\left[\boldsymbol{x}\right] = E_{\boldsymbol{x}\in\mathcal{X}}\left[\boldsymbol{x}\boldsymbol{x}^T\right] \text{ and } E\left[\boldsymbol{x}\right]E\left[\boldsymbol{x}\right]^T$$

  - To produce the whitened activations

Therefore

A Better Options, we can normalize each input layer

$$\widehat{\boldsymbol{x}}^{(k)} = \frac{\boldsymbol{x}^{(k)} - \mu}{\sigma}$$

- with $\mu = E\left[\boldsymbol{x}^{(k)}\right]$ and $\sigma^2 = Var\left[\boldsymbol{x}^{(k)}\right]$

This allows to speed up convergence

- Simply normalizing each input of a layer may change what the layer can represent.

So, we need to insert a transformation in the network

- Which can represent the identity transform

Therefore

A Better Options, we can normalize each input layer

$$\widehat{\boldsymbol{x}}^{(k)} = \frac{\boldsymbol{x}^{(k)} - \mu}{\sigma}$$

- with $\mu = E\left[\boldsymbol{x}^{(k)}\right]$ and $\sigma^2 = Var\left[\boldsymbol{x}^{(k)}\right]$

This allows to speed up convergence

- Simply normalizing each input of a layer may change what the layer can represent.

So, we need to insert a transformation in the network

- Which can represent the identity transform

Therefore

A Better Options, we can normalize each input layer

$$\widehat{\boldsymbol{x}}^{(k)} = \frac{\boldsymbol{x}^{(k)} - \mu}{\sigma}$$

- with $\mu = E\left[\boldsymbol{x}^{(k)}\right]$ and $\sigma^2 = Var\left[\boldsymbol{x}^{(k)}\right]$

This allows to speed up convergence

- Simply normalizing each input of a layer may change what the layer can represent.

So, we need to insert a transformation in the network

- Which can represent the identity transform

The Transformation

The Linear transformation

$$\boldsymbol{y}^{(k)} = \gamma^{(k)} \widehat{\boldsymbol{x}}^{(k)} + \beta^{(k)}$$

The parameters $\gamma^{(k)}, \beta^{(k)}$

- This allow to recover the identity by setting $\gamma^{(k)} = \sqrt{Var\left[\boldsymbol{x}^{(k)}\right]}$ and $\beta^{(k)} = E\left[\boldsymbol{x}^{(k)}\right]$ if necessary.

The Transformation

The Linear transformation

$$\boldsymbol{y}^{(k)} = \gamma^{(k)} \widehat{\boldsymbol{x}}^{(k)} + \beta^{(k)}$$

The parameters $\gamma^{(k)}, \beta^{(k)}$

- This allow to recover the identity by setting $\gamma^{(k)} = \sqrt{Var\left[\boldsymbol{x}^{(k)}\right]}$ and $\beta^{(k)} = E\left[\boldsymbol{x}^{(k)}\right]$ if necessary.

Finally

Batch Normalizing Transform

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

Output: $\{y_i = BN_{\gamma,\beta}(\boldsymbol{x}_i)\}$

1. $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$
2. $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$
3. $\widehat{x} = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
4. $y_i = \gamma^{(k)}\widehat{x}_i + \beta = BN_{\gamma,\beta}(x_i)$

Finally

Batch Normalizing Transform

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

Output: $\{y_i = BN_{\gamma,\beta}(\boldsymbol{x}_i)\}$

1. $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i$
2. $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{x}_i - \mu)^2$
3. $\widehat{\boldsymbol{x}} = \frac{\boldsymbol{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
4. $y_i = \gamma^{(k)} \widehat{\boldsymbol{x}}_i + \beta = BN_{\gamma,\beta}(\boldsymbol{x}_i)$

Finally

Batch Normalizing Transform

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

Output: $\{y_i = BN_{\gamma,\beta}(\boldsymbol{x}_i)\}$

1. $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i$
2. $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{x}_i - \mu)^2$
3. $\widehat{x} = \frac{\boldsymbol{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
4. $y_i = \gamma^{(k)}\widehat{\boldsymbol{x}}_i + \beta = BN_{\gamma,\beta}(\boldsymbol{x}_i)$

Finally

Batch Normalizing Transform

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

Output: $\{y_i = BN_{\gamma,\beta}(\boldsymbol{x}_i)\}$

1. $\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}_i$
2. $\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{x}_i - \mu)^2$
3. $\widehat{\boldsymbol{x}} = \frac{\boldsymbol{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
4. $y_i = \gamma^{(k)} \widehat{\boldsymbol{x}}_i + \beta = BN_{\gamma,\beta}(\boldsymbol{x}_i)$

Finally

Batch Normalizing Transform

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

Output: $\{y_i = BN_{\gamma,\beta}(\boldsymbol{x}_i)\}$

1. $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i$
2. $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{x}_i - \mu)^2$
3. $\widehat{\boldsymbol{x}} = \frac{\boldsymbol{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
4. $\boldsymbol{y}_i = \gamma^{(k)}\widehat{\boldsymbol{x}}_i + \beta = BN_{\gamma,\beta}(\boldsymbol{x}_i)$

Backpropagation

We have the following equations by using the loss function l

1. $\frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} = \frac{\partial l}{\partial \boldsymbol{y}_i} \times \gamma$
2. $\frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times (\boldsymbol{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times \left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-\frac{3}{2}}$
3. $\frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}\right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^{m} -2 \times (\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m}$
4. $\frac{\partial l}{\partial \boldsymbol{x}_i} = \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{2 \times (\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m} + \frac{\partial l}{\partial \mu_{\mathcal{B}}} \times \frac{1}{m}$
5. $\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i} \times \widehat{\boldsymbol{x}}_i$
6. $\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i}$

Backpropagation

We have the following equations by using the loss function l

1. $\frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} = \frac{\partial l}{\partial \boldsymbol{y}_i} \times \gamma$
2. $\frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times (\boldsymbol{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times \left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-\frac{3}{2}}$
3. $\frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}\right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^{m} -2 \times (\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m}$
4. $\frac{\partial l}{\partial \boldsymbol{x}_i} = \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{2 \times (\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m} + \frac{\partial l}{\partial \mu_{\mathcal{B}}} \times \frac{1}{m}$
5. $\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i} \times \widehat{\boldsymbol{x}}_i$
6. $\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i}$

Backpropagation

We have the following equations by using the loss function l

1. $\frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} = \frac{\partial l}{\partial \boldsymbol{y}_i} \times \gamma$
2. $\frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times (\boldsymbol{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times \left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-\frac{3}{2}}$
3. $\frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}\right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^{m} -2 \times (\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m}$
4. $\frac{\partial l}{\partial \boldsymbol{x}_i} = \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{2 \times (\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m} + \frac{\partial l}{\partial \mu_{\mathcal{B}}} \times \frac{1}{m}$
5. $\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i} \times \widehat{\boldsymbol{x}}_i$
6. $\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i}$

Backpropagation

We have the following equations by using the loss function l

1. $\frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} = \frac{\partial l}{\partial \boldsymbol{y}_i} \times \gamma$
2. $\frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times (\boldsymbol{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times \left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-\frac{3}{2}}$
3. $\frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}}\right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^{m} -2\times(\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m}$
4. $\frac{\partial l}{\partial \boldsymbol{x}_i} = \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}} + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{2\times(\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m} + \frac{\partial l}{\partial \mu_{\mathcal{B}}} \times \frac{1}{m}$
5. $\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i} \times \widehat{\boldsymbol{x}}_i$
6. $\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i}$

Backpropagation

We have the following equations by using the loss function l

1. $\frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} = \frac{\partial l}{\partial \boldsymbol{y}_i} \times \gamma$
2. $\frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times (\boldsymbol{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times \left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-\frac{3}{2}}$
3. $\frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}}\right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^{m} -2\times(\boldsymbol{x}_i-\mu_{\mathcal{B}})}{m}$
4. $\frac{\partial l}{\partial \boldsymbol{x}_i} = \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}} + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{2\times(\boldsymbol{x}_i-\mu_{\mathcal{B}})}{m} + \frac{\partial l}{\partial \mu_{\mathcal{B}}} \times \frac{1}{m}$
5. $\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i} \times \widehat{\boldsymbol{x}}_i$
6. $\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i}$

Backpropagation

We have the following equations by using the loss function l

1. $\frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} = \frac{\partial l}{\partial \boldsymbol{y}_i} \times \gamma$
2. $\frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times (\boldsymbol{x}_i - \mu_{\mathcal{B}}) \times \left(-\frac{1}{2}\right) \times \left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-\frac{3}{2}}$
3. $\frac{\partial l}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}\right) + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{\sum_{i=1}^{m} -2 \times (\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m}$
4. $\frac{\partial l}{\partial \boldsymbol{x}_i} = \frac{\partial l}{\partial \widehat{\boldsymbol{x}}_i} \times \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial l}{\partial \sigma_{\mathcal{B}}^2} \times \frac{2 \times (\boldsymbol{x}_i - \mu_{\mathcal{B}})}{m} + \frac{\partial l}{\partial \mu_{\mathcal{B}}} \times \frac{1}{m}$
5. $\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i} \times \widehat{\boldsymbol{x}}_i$
6. $\frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial \boldsymbol{y}_i}$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{ \boldsymbol{x}^{(k)} \right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1 \ldots K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(\boldsymbol{x}^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $y^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{ \gamma^{(k)}, \beta^{(k)} \right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1 \ldots K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[\boldsymbol{x}] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[\boldsymbol{x}] = \frac{m}{m-1}{}_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^2\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(\boldsymbol{x})$ with
11. $$y = \frac{\gamma}{\sqrt{Var[\boldsymbol{x}]+\epsilon}} \times \boldsymbol{x} + \left[\beta - \frac{\gamma E[\boldsymbol{x}]}{\sqrt{Var[\boldsymbol{x}]+\epsilon}}\right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{\boldsymbol{x}^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1 \ldots K$ do
3.   Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(\boldsymbol{x}^{(k)}\right)$ to N_{BN}^{tr}
4.   Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $y^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1 \ldots K$ do
8.   Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E\left[\boldsymbol{x}\right] = E_{\mathcal{B}}\left[\mu_{\mathcal{B}}\right]$ and $Var\left[\boldsymbol{x}\right] = \frac{m}{m-1}{}_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^{2}\right]$
10.   In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}\left(\boldsymbol{x}\right)$ with
11. $$\boldsymbol{y} = \frac{\gamma}{\sqrt{Var[\boldsymbol{x}]+\epsilon}} \times \boldsymbol{x} + \left[\beta - \frac{\gamma E[\boldsymbol{x}]}{\sqrt{Var[\boldsymbol{x}]+\epsilon}}\right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{x^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1 \ldots K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(x^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $x^{(k)}$ to take $y^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1 \ldots K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1} E_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^2\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with
11. $$y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{\boldsymbol{x}^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1...K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(\boldsymbol{x}^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $x^{(k)}$ to take $y^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1...K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1}_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^2\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with
11. $$y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{\boldsymbol{x}^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1...K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(\boldsymbol{x}^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $\boldsymbol{y}^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1...K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1}{}_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^{2}\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with
11. $$y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{ \boldsymbol{x}^{(k)} \right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1 \ldots K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)}, \beta^{(k)}} \left(\boldsymbol{x}^{(k)} \right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $\boldsymbol{y}^{(k)}$ instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{ \gamma^{(k)}, \beta^{(k)} \right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1 \ldots K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1} E_{\mathcal{B}} \left[\sigma_{\mathcal{B}}^2 \right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with
11. $y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}} \right]$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{\boldsymbol{x}^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1 \dots K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(\boldsymbol{x}^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $\boldsymbol{y}^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1 \dots K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1} E_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^{2}\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with
11. $y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right]$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{\boldsymbol{x}^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1...K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(\boldsymbol{x}^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $\boldsymbol{y}^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1...K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1}E_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^2\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with
11. $$y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{ \boldsymbol{x}^{(k)} \right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1 \ldots K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}} \left(\boldsymbol{x}^{(k)} \right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $\boldsymbol{y}^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{ \gamma^{(k)}, \beta^{(k)} \right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1 \ldots K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[x] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[x] = \frac{m}{m-1}_{\mathcal{B}} \left[\sigma_{\mathcal{B}}^2 \right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with
11. $$y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}} \right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{\boldsymbol{x}^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1...K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(\boldsymbol{x}^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $\boldsymbol{y}^{(k)}$ instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1...K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E[\boldsymbol{x}] = E_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $Var[\boldsymbol{x}] = \frac{m}{m-1}{}_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^2\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$ with
11. $$y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \times x + \left[\beta - \frac{\gamma E[x]}{\sqrt{Var[x]+\epsilon}}\right]$$

Training Batch Normalization Networks

Input: Network N with trainable parameters Θ; subset of activations $\left\{\boldsymbol{x}^{(k)}\right\}_{k=1}^{K}$

Output: Batch-normalized network for inference N_{BN}^{inf}

1. $N_{BN}^{tr} = N$ // Training BN network
2. for $k = 1...K$ do
3. Add transformation $y^{(k)} = BN_{\gamma^{(k)},\beta^{(k)}}\left(\boldsymbol{x}^{(k)}\right)$ to N_{BN}^{tr}
4. Modify each layer in N_{BN}^{tr} with input $\boldsymbol{x}^{(k)}$ to take $\boldsymbol{y}^{(k)}$instead
5. Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \left\{\gamma^{(k)}, \beta^{(k)}\right\}_{k=1}^{K}$
6. $N_{BN}^{inf} = N_{BN}^{tr}$ // Inference BN network with frozen parameters
7. for $k = 1...K$ do
8. Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them
9. $E\left[\boldsymbol{x}\right] = E_{\mathcal{B}}\left[\mu_{\mathcal{B}}\right]$ and $Var\left[\boldsymbol{x}\right] = \frac{m}{m-1}{}_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^2\right]$
10. In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}\left(\boldsymbol{x}\right)$ with
11. $\boldsymbol{y} = \frac{\gamma}{\sqrt{Var[\boldsymbol{x}]+\epsilon}} \times \boldsymbol{x} + \left[\beta - \frac{\gamma E[\boldsymbol{x}]}{\sqrt{Var[\boldsymbol{x}]+\epsilon}}\right]$

However

Santurkar et al. [16]

- They found thats is not the covariance shift the one affected by it!!!

Santurkar et al. recognize that

- Batch normalization has been arguably one of the most successful architectural innovations in deep learning.

They used a standard Very deep convolutional network

- on CIFAR-10 with and without BatchNorm

However

Santurkar et al. [16]

- They found thats is not the covariance shift the one affected by it!!!

Santurkar et al. recognize that

- Batch normalization has been arguably one of the most successful architectural innovations in deep learning.

They used a standard Very deep convolutional network

- on CIFAR-10 with and without BatchNorm

However

Santurkar et al. [16]

- They found thats is not the covariance shift the one affected by it!!!

Santurkar et al. recognize that

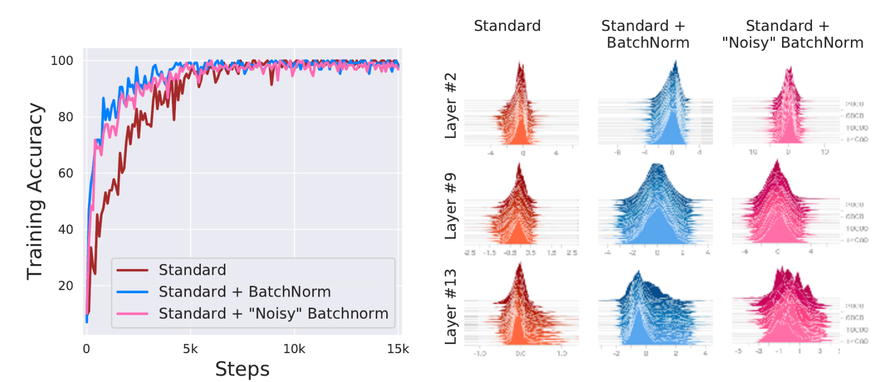
- Batch normalization has been arguably one of the most successful architectural innovations in deep learning.

They used a standard Very deep convolutional network

- on CIFAR-10 with and without BatchNorm

They found something quite interesting

The following facts



Actually Batch Normalization

It does not do anything to the Internal Covariate Shift

- Actually smooth the optimization manifold
  - It is not the only way to achieve it!!!

They suggest that

- "This suggests that the positive impact of BatchNorm on training might be somewhat serendipitous."

Actually Batch Normalization

It does not do anything to the Internal Covariate Shift

- Actually smooth the optimization manifold
  - It is not the only way to achieve it!!!

They suggest that

- "This suggests that the positive impact of BatchNorm on training might be somewhat serendipitous."

They actually have a connected result

To the analysis of gradient clipping!!!

- They are the same group

Theorem (The effect of BatchNorm on the Lipschitzness of the loss)

- For a BatchNorm network with loss $\widehat{\mathcal{L}}$ and an identical non-BN network with (identical) loss $\mathcal{L}$,

$$\left\|\nabla_{y_j}\widehat{\mathcal{L}}\right\|^2 \le \frac{\gamma^2}{\sigma_j^2}\left[\left\|\nabla_{y_j}\mathcal{L}\right\|^2 - \frac{1}{m}\left\langle \mathbf{1}, \nabla_{y_j}\mathcal{L}\right\rangle^2 - \frac{1}{\sqrt{m}}\left\langle \nabla_{y_j}\mathcal{L}, \hat{y}_j\right\rangle^2\right]$$

They actually have a connected result

To the analysis of gradient clipping!!!

- They are the same group

Theorem (The effect of BatchNorm on the Lipschitzness of the loss)

- For a BatchNorm network with loss $\widehat{\mathcal{L}}$ and an identical non-BN network with (identical) loss $\mathcal{L}$,

$$\left\|\nabla_{\boldsymbol{y}_j}\widehat{\mathcal{L}}\right\|^2 \leq \frac{\gamma^2}{\sigma_j^2}\left[\left\|\nabla_{y_j}\mathcal{L}\right\|^2 - \frac{1}{m}\left\langle \mathbf{1}, \nabla_{y_j}\mathcal{L}\right\rangle^2 - \frac{1}{\sqrt{m}}\left\langle \nabla_{y_j}\mathcal{L}, \widehat{\boldsymbol{y}}_j\right\rangle^2\right]$$

Outline

Fully Connected Layer

If a layer l is a fully connected layer

- If layer $(l-1)$ is a fully connected layer, use the equation to compute the output of i^{th} unit at layer l:

$$z_i^{(l)} = \sum_{k=0}^{m^{(l)}} w_{i,k}^{(l)} y_k^{(l)} \text{ thus } y_i^{(l)} = f\left(z_i^{(l)}\right)$$

Otherwise

- Layer l expects $m_1^{(l-1)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$ as input.

Fully Connected Layer

If a layer l is a fully connected layer

- If layer $(l-1)$ is a fully connected layer, use the equation to compute the output of i^{th} unit at layer l:

$$z_i^{(l)} = \sum_{k=0}^{m^{(l)}} w_{i,k}^{(l)} y_k^{(l)} \text{ thus } y_i^{(l)} = f\left(z_i^{(l)}\right)$$

Otherwise

- Layer l expects $m_1^{(l-1)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$ as input.

Then

Thus, the i^{th} unit in layer l computes

$$y_i^{(l)} = f\left(z_i^{(l)}\right)$$

$$z_i^{(l)} = \sum_{j=1}^{m_1^{(l-1)}} \sum_{r=1}^{m_2^{(l-1)}} \sum_{s=1}^{m_3^{(l-1)}} w_{i,j,r,s}^{(l)} \left(Y_j^{(l-1)}\right)_{r,s}$$

Here

Where $w_{i,j,r,s}^{(l)}$

- It denotes the weight connecting the unit at position (r, s) in the j^{th} feature map of layer $(l-1)$ and the i^{th} unit in layer l.

Something Notable

- In practice, Convolutional Layers are used to learn a feature hierarchy and one or more fully connected layers are used for classification purposes based on the computed features.

Here

Where $w^{(l)}_{i,j,r,s}$

- It denotes the weight connecting the unit at position (r, s) in the j^{th} feature map of layer $(l-1)$ and the i^{th} unit in layer l.

Something Notable

- In practice, Convolutional Layers are used to learn a feature hierarchy and one or more fully connected layers are used for classification purposes based on the computed features.

Basically

We can use a loss function at the output of such layer

$$L(\boldsymbol{W}) = \sum_{n=1}^{N} E_n(\boldsymbol{W}) = \sum_{n=1}^{N}\sum_{k=1}^{K}\left(y_{nk}^{(l)} - t_{nk}\right)^2 \text{ (Sum of Squared Error)}$$

$$L(\boldsymbol{W}) = \sum_{n=1}^{N} E_n(\boldsymbol{W}) = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\log\left(y_{nk}^{(l)}\right) \text{ (Cross-Entropy Error)}$$

Assuming W the tensor used to represent all the possible weights

- We can use the Backpropagation idea as long we can apply the corresponding derivatives.

Basically

We can use a loss function at the output of such layer

$$L(\boldsymbol{W}) = \sum_{n=1}^{N} E_n(\boldsymbol{W}) = \sum_{n=1}^{N} \sum_{k=1}^{K} \left(y_{nk}^{(l)} - t_{nk} \right)^2 \text{ (Sum of Squared Error)}$$

$$L(\boldsymbol{W}) = \sum_{n=1}^{N} E_n(\boldsymbol{W}) = \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \log\left(y_{nk}^{(l)} \right) \text{ (Cross-Entropy Error)}$$

Assuming $\boldsymbol{W}$ the tensor used to represent all the possible weights

- We can use the Backpropagation idea as long we can apply the corresponding derivatives.

About this

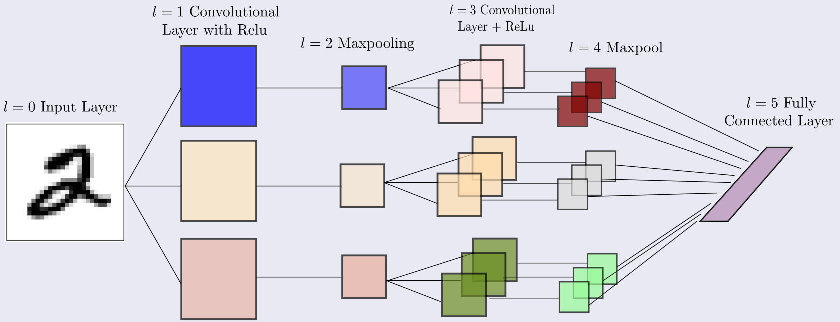
As part of the seminar

- We are preparing a series of slides about Loss Functions...

Outline

We have the following Architecture

Simplified Architecture by Jean LeCun “Backpropagation applied to handwritten zip code recognition”



Therefore, we have

Layer $l = 1$

- This Layer is using a ReLu f with 3 channels

$$\left(Y_1^{(l)}\right)_{x,y} = \left(B_1^{(l)}\right)_{x,y} + \sum_{k=-h_1^{(l)}}^{h_1^{(l)}} \sum_{t=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{11}^{(l)}\right)_{k,t} \left(Y_1^{(l-1)}\right)_{x-k,x-t}$$

$$\left(Y_2^{(l)}\right)_{x,y} = \left(B_2^{(l)}\right)_{x,y} + \sum_{k=-h_1^{(l)}}^{h_1^{(l)}} \sum_{t=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{21}^{(l)}\right)_{k,t} \left(Y_1^{(l-1)}\right)_{x-k,x-t}$$

$$\left(Y_3^{(l)}\right)_{x,y} = \left(B_3^{(l)}\right)_{x,y} + \sum_{k=-h_1^{(l)}}^{h_1^{(l)}} \sum_{t=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{31}^{(l)}\right)_{k,t} \left(Y_1^{(l-1)}\right)_{x-k,x-t}$$

Layer $l = 2$

We have a maxpooling of size 2×2

$$\left(Y_i^{(l)}\right)_{x',y'} = \max\left\{\left(Y_i^{(l-1)}\right)_{x,y}, \left(Y_i^{(l-1)}\right)_{x+1,y} \left(Y_i^{(l-1)}\right)_{x,y+1}, \left(Y_i^{(l-1)}\right)_{x+1,y+1}\right\}$$

Then, you repeat the previous process

Thus we obtain a reduced convoluted version $Y_m^{(3)}$ of the $Y_n^{(4)}$ convolution and maxpooling

- Thus, we use those as inputs for the fully connected layer of input.

The fully connected layer

Now assuming a single $k = 1$ neuron

$$
\begin{aligned}
y_1^{(6)} &= f\left(z_1^{(5)}\right) \\
z_1^{(5)} &= \sum_{k=1}^{9} \sum_{r=1}^{m_2^{(6)}} \sum_{s=1}^{m_3^{(6)}} w_{r,s,k}^{(5)} \left(Y_k^{(4)}\right)_{r,s}
\end{aligned}
$$

We have for simplicity sake

That our final cost function is equal to

$$L = \frac{1}{2}\left(y_1^{(6)} - t_1^{(6)}\right)^2$$

Outline

After collecting all input/output

Therefore

- We have using sum of squared errors (loss function):

$$L = \frac{1}{2}\left(y_1^{(6)} - t_1^{(6)}\right)^2$$

Therefore, we can obtain

$$\frac{\partial L}{\partial w_{r,s,k}^{(5)}} = \frac{1}{2} \times \frac{\partial \left(y_1^{(6)} - t_1^{(6)}\right)^2}{\partial w_{r,s,k}^{(5)}}$$

After collecting all input/output

Therefore

- We have using sum of squared errors (loss function):
$$L = \frac{1}{2}\left(y_1^{(6)} - t_1^{(6)}\right)^2$$

Therefore, we can obtain

$$\frac{\partial L}{\partial w_{r,s,k}^{(5)}} = \frac{1}{2} \times \frac{\partial \left(y_1^{(6)} - t_1^{(6)}\right)^2}{\partial w_{r,s,k}^{(5)}}$$

Therefore

We get in the first part of the equation

$$\frac{\partial\left(y_1^{(6)}-t_1^{(6)}\right)^2}{\partial w_{r,s,k}^{(5)}}=\left(y_1^{(6)}-t_1^{(6)}\right)\frac{\partial y_1^{(6)}}{\partial w_{r,s,k}^{(5)}}$$

With

$$y_1^{(6)}=ReLu\left(z_1^{(5)}\right)$$

Therefore

We get in the first part of the equation

$$\frac{\partial\left(y_1^{(6)}-t_1^{(6)}\right)^2}{\partial w_{r,s,k}^{(5)}}=\left(y_1^{(6)}-t_1^{(6)}\right)\frac{\partial y_1^{(6)}}{\partial w_{r,s,k}^{(5)}}$$

With

$$y_1^{(6)}=ReLu\left(z_1^{(5)}\right)$$

Therefore

We have

$$\frac{\partial y_1^{(6)}}{\partial w_{r,s,k}^{(5)}} = \frac{\partial f\left(z_1^{(5)}\right)}{\partial z_1^{(5)}} \times \frac{\partial z_1^{(5)}}{\partial w_{r,s,k}^{(5)}}$$

Therefore if we use the approximation

$$\frac{\partial f\left(z_1^{(5)}\right)}{\partial z_1^{(5)}} = \frac{e^{kz_1^{(5)}}}{\left(1 + e^{kz_1^{(5)}}\right)}$$

Therefore

We have

$$\frac{\partial y_1^{(6)}}{\partial w_{r,s,k}^{(5)}} = \frac{\partial f\left(z_1^{(5)}\right)}{\partial z_1^{(5)}} \times \frac{\partial z_1^{(5)}}{\partial w_{r,s,k}^{(5)}}$$

Therefore if we use the approximation

$$\frac{\partial f\left(z_1^{(5)}\right)}{\partial z_1^{(5)}} = \frac{e^{kz_1^{(5)}}}{\left(1+e^{kz_1^{(5)}}\right)}$$

Now, we need to derive $\frac{\partial z_1^{(5)}}{\partial w_{r,s,k}^{(5)}}$

We know that

$$z_1^{(5)} = \sum_{k=1}^{9} \sum_{r=1}^{m_2^{(6)}} \sum_{s=1}^{m_3^{(6)}} w_{r,s,k}^{(5)} \left(Y_k^{(4)}\right)_{r,s}$$

Finally

$$\frac{\partial z_1^{(5)}}{\partial w_{r,s,k}^{(5)}} = \left(Y_k^{(4)}\right)_{r,s}$$

Now, we need to derive $\frac{\partial z_1^{(5)}}{\partial w_{r,s,k}^{(5)}}$

We know that

$$z_1^{(5)} = \sum_{k=1}^{9} \sum_{r=1}^{m_2^{(6)}} \sum_{s=1}^{m_3^{(6)}} w_{r,s,k}^{(5)} \left(Y_k^{(4)}\right)_{r,s}$$

Finally

$$\frac{\partial z_1^{(5)}}{\partial w_{r,s,k}^{(5)}} = \left(Y_k^{(4)}\right)_{r,s}$$

Now, we need to derive $\frac{\partial z_1^{(5)}}{\partial w_{r,s,k}^{(5)}}$

We know that

$$z_1^{(5)} = \sum_{k=1}^{9} \sum_{r=1}^{m_2^{(6)}} \sum_{s=1}^{m_3^{(6)}} w_{r,s,k}^{(5)} \left(Y_k^{(4)}\right)_{r,s}$$

Finally

$$\frac{\partial z_1^{(5)}}{\partial w_{r,s,k}^{(5)}} = \left(Y_k^{(4)}\right)_{r,s}$$

Maxpooling

This is not derived after all, but we go directly go for the max term

- Assume you get the max element for $f = 1, 2, ..., 9$ and $j = 1$

$$\left(Y_f^{(3)}\right)_{x,y} = \left(B_f^{(3)}\right)_{x,y} + \sum_{k=-h_1^{(l)}}^{h_1^{(l)}} \sum_{t=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{f1}^{(3)}\right)_{k,t} \left(Y_1^{(2)}\right)_{x-k,x-t}$$

Therefore

We have then

$$\frac{\partial L}{\partial\left(K_{f1}^{(3)}\right)_{k,t}} = \frac{1}{2} \times \frac{\partial\left(y_1^{(6)} - t_1^{(6)}\right)^2}{\partial\left(K_{f1}^{(3)}\right)_{k,t}}$$

We have the following chain of derivations given

$$\left(Y_f^{(4)}\right)_{x,y} = f\left[\left(Y_f^{(3)}\right)_{x,y}\right]$$

$$\frac{\partial L}{\partial\left(K_{f1}^{(3)}\right)_{k,t}} = \left(y_1^{(6)} - t_1^{(6)}\right)\frac{\partial f\left(z_1^{(5)}\right)}{\partial z_1^{(5)}} \times \frac{\partial z_i^{(5)}}{\partial\left(Y_f^{(4)}\right)_{x,y}} \times \frac{\partial f\left[\left(Y_f^{(3)}\right)_{x,y}\right]}{\partial\left(K_{f1}^{(3)}\right)_{k,t}}$$

Therefore

We have then

$$\frac{\partial L}{\partial\left(K_{f1}^{(3)}\right)_{k,t}}=\frac{1}{2}\times\frac{\partial\left(y_1^{(6)}-t_1^{(6)}\right)^2}{\partial\left(K_{f1}^{(3)}\right)_{k,t}}$$

We have the following chain of derivations given $\left(Y_f^{(4)}\right)_{x,y}=f\left[\left(Y_f^{(3)}\right)_{x,y}\right]$

$$\frac{\partial L}{\partial\left(K_{f1}^{(3)}\right)_{k,t}}=\left(y_1^{(6)}-t_1^{(6)}\right)\frac{\partial f\left(z_1^{(5)}\right)}{\partial z_1^{(5)}}\times\frac{\partial z_i^{(5)}}{\partial\left(Y_f^{(4)}\right)_{x,y}}\times\frac{\partial f\left[\left(Y_f^{(3)}\right)_{x,y}\right]}{\partial\left(K_{f1}^{(3)}\right)_{k,t}}$$

Therefore

We have

$$\frac{\partial z_i^{(5)}}{\partial \left(Y_f^{(3)}\right)_{x,y}} = w_{x,y,f}^{(5)}$$

Then assuming that

$$\left(Y_f^{(3)}\right)_{x,y} = \left(B_f^{(3)}\right)_{x,y} + \sum_{k=-h_1^{(l)}}^{h_1^{(l)}} \sum_{l=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{f1}^{(3)}\right)_{k,l} \left(Y_1^{(2)}\right)_{x-k,x-l}$$

Therefore

We have

$$\frac{\partial z_i^{(5)}}{\partial \left(Y_f^{(3)}\right)_{x,y}} = w_{x,y,f}^{(5)}$$

Then assuming that

$$\left(Y_f^{(3)}\right)_{x,y} = \left(B_f^{(3)}\right)_{x,y} + \sum_{k=-h_1^{(l)}}^{h_1^{(l)}} \sum_{t=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{f1}^{(3)}\right)_{k,t} \left(Y_1^{(2)}\right)_{x-k,x-t}$$

Therefore

We have

$$\frac{\partial f\left[\left(Y_f^{(3)}\right)_{x,y}\right]}{\partial\left(K_{f1}^{(3)}\right)_{k,t}} = \frac{\partial f\left[\left(Y_f^{(3)}\right)_{x,y}\right]}{\partial\left(Y_f^{(3)}\right)_{x,y}} \times \frac{\partial\left(Y_f^{(3)}\right)_{x,y}}{\partial\left(K_{f1}^{(3)}\right)_{k,t}}$$

Then

$$\frac{\partial f\left[\left(Y_f^{(3)}\right)_{x,y}\right]}{\partial\left(Y_f^{(3)}\right)_{x,y}} = f'\left[\left(Y_f^{(3)}\right)_{x,y}\right]$$

Therefore

We have

$$\frac{\partial f\left[\left(Y_f^{(3)}\right)_{x,y}\right]}{\partial\left(K_{f1}^{(3)}\right)_{k,t}}=\frac{\partial f\left[\left(Y_f^{(3)}\right)_{x,y}\right]}{\partial\left(Y_f^{(3)}\right)_{x,y}}\times\frac{\partial\left(Y_f^{(3)}\right)_{x,y}}{\partial\left(K_{f1}^{(3)}\right)_{k,t}}$$

Then

$$\frac{\partial f\left[\left(Y_f^{(3)}\right)_{x,y}\right]}{\partial\left(Y_f^{(3)}\right)_{x,y}}=f'\left[\left(Y_f^{(3)}\right)_{x,y}\right]$$

Finally, we have

The equation

$$\frac{\partial\left(Y_f^{(3)}\right)_{x,y}}{\partial\left(K_{f1}^{(3)}\right)_{k,t}}=\left(Y_1^{(2)}\right)_{x-k,x-t}$$

The Other Equations

I will leave you to devise them

- They are a repetitive procedure.

The interesting case the average pooling

- The others are the stride and the deconvolution

The Other Equations

I will leave you to devise them

- They are a repetitive procedure.

The interesting case the average pooling

- The others are the stride and the deconvolution

[1] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, “A survey of the recent architectures of deep convolutional neural networks,” *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5455–5516, 2020.

[2] R. Szeliski, *Computer Vision: Algorithms and Applications*. Berlin, Heidelberg: Springer-Verlag, 1st ed., 2010.

[3] S. Haykin, *Neural Networks and Learning Machines*. No. v. 10 in Neural networks and learning machines, Prentice Hall, 2009.

[4] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat's visual cortex,” *The Journal of physiology*, vol. 160, no. 1, p. 106, 1962.

[5] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] W. Zhang, K. Itoh, J. Tanida, and Y. Ichioka, “Parallel distributed processing model with local space-invariant interconnections and its optical architecture,” *Appl. Opt.*, vol. 29, pp. 4790–4797, Nov 1990.

[7] J. J. Weng, N. Ahuja, and T. S. Huang, “Learning recognition and segmentation of 3-d objects from 2-d images,” in *1993 (4th) International Conference on Computer Vision*, pp. 121–128, IEEE, 1993.

[8] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, “Deconvolutional networks,” in *2010 IEEE Computer Society Conference on computer vision and pattern recognition*, pp. 2528–2535, IEEE, 2010.

[9] D. Krishnan and R. Fergus, “Fast image deconvolution using hyper-laplacian priors,” *Advances in neural information processing systems*, vol. 22, pp. 1033–1041, 2009.

[10] Y. Wang, J. Yang, W. Yin, and Y. Zhang, "A new alternating minimization algorithm for total variation image reconstruction," *SIAM Journal on Imaging Sciences*, vol. 1, no. 3, pp. 248–272, 2008.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[12] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," 2015.

[13] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[14] C. Kong and S. Lucey, "Take it in your stride: Do we need striding in cnns?," *arXiv preprint arXiv:1712.02502*, 2017.

[15] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[16] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How does batch normalization help optimization?,” in *Advances in Neural Information Processing Systems*, pp. 2483–2493, 2018.