Deep Learning

Lecture 8: Deep Learning Architectures (Pt II)

University of Agder,
Kristiansand, Norway

Prepared by: Hadi Ghauch*, Hossein S. Ghadikolaei†

* Telecom ParisTech

† Royal Institute of Technology, KTH

https://sites.google.com/view/fundl/home

April 2019

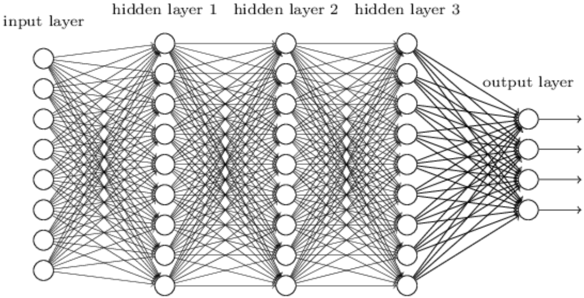
Outline

1. Convolutional Neural Networks

Recap: Deep Neural Networks (DNNs)

Multi-layer Percepteron:

- Output from some perceptrons are used in the inputs to other perceptrons



$\boldsymbol{W}_1$ $\boldsymbol{W}_2$ $\boldsymbol{W}_3$ Output

Feedforward Deep Neural Network (DNN):

- Multi-layer Percepteron with 'many' hidden layers
- Only forward connections: from input to hidden layer or output

Recap: Deep Neural Networks (DNNs)

DNN with J-layers: cascade of J filters + activation

$$\boldsymbol{z}_0 \rightarrow \underbrace{\boxed{\boxed{\boldsymbol{W}_1, \boldsymbol{b}_1} \rightarrow \boxed{\boldsymbol{\sigma}_1}}}_{\text{Layer 1}} \rightarrow \boldsymbol{z}_1 \rightarrow \underbrace{\boxed{\boxed{\boldsymbol{W}_2, \boldsymbol{b}_2} \rightarrow \boxed{\boldsymbol{\sigma}_2}}}_{\text{Layer 2}} \rightarrow \boldsymbol{z}_2 \rightarrow \cdots \rightarrow \boldsymbol{z}_J$$

In compact form: $\boldsymbol{z}_j = \boldsymbol{\sigma}_j(\boldsymbol{W}_j \boldsymbol{z}_{j-1} + \boldsymbol{b}_j)$

$\boldsymbol{z}_j \in \mathbb{R}^{d_j}$: output to layer j

$\boldsymbol{W}_j \in \mathbb{R}^{d_j \times d_{j-1}}$: **Weights** for layer j (drop bias for simplicity)

$\boldsymbol{\sigma}_j : \mathbb{R}^{d_j} \mapsto \mathbb{R}^{d_j}$: **Activation func** for layer j

Outline

Definition and Preliminaries

Recall: $\boldsymbol{w} \in \mathbb{R}^{d_1}$ is vector, $\boldsymbol{W} \in \mathbb{R}^{d_1 \times d_2}$ is matrix. Tensors generalize this definition

third-order tensor: $\overline{\boldsymbol{W}} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$
- vectors/matrices are special case of a third-order tensor
- concatenation of d_3 matrices each of size $d_1 \times d_2$

Color RGB image with $d_1 \times d_2$ pixels: modeled as $d_1 \times d_2 \times 3$ tensor
- $d_1 \times d_2$ matrix for each channel/color, with 3 channel

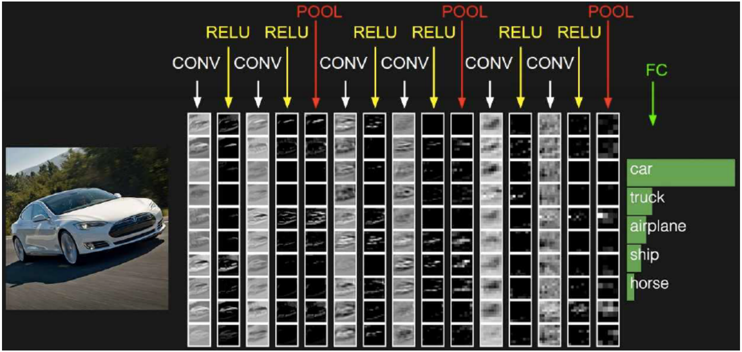
CNNs: input to each layer and action of layer modeled by tensor
- tensors of higher order also used in CNNs (skipped here)

Vectorization: vec$(\boldsymbol{W})$ stacks columns of $\boldsymbol{W}$ one-by-one to form $d_1 d_2$ column vector
- vec$(-)$ is one-to-one mapping (due to preset order of cols)
- vec(vec$(\overline{\boldsymbol{W}})$) gives a $d_1 d_2 d_3$ column vector
- tensors, matrices and vectors are equivalent via vec$(-)$

Convolutional Neural Networks

Convolutional Neural Networks (CNNs)

- variants of DNNs where at least one convolution layer used
- layer = **convolution**, **non-linear pooling**, or **fully connected**



Source: Trivedi, Kodor, 2017

Convolutional Neural Networks (CNNs)

Task: Classify a color image into C-classes

$$\overline{\boldsymbol{Z}}_1 \to \underset{\text{Layer 1}}{\boxed{\overline{\boldsymbol{W}}_1}} \to \overline{\boldsymbol{Z}}_2 \to \underset{\text{Layer 2}}{\boxed{\overline{\boldsymbol{W}}_2}} \to \overline{\boldsymbol{Z}}_3 \to \cdots \to \boldsymbol{z}_J$$

$\overline{\boldsymbol{Z}}_j \in \mathbb{R}^{d_1^j \times d_2^j \times d_3^j}$: input tensor of layer j
- **third order tensor** of dimension $d_1^j \times d_2^j \times d_3^j$
- each element indexed by (m^j, n^j, p^j):
  - $m^j \in \{0, ..., d_1^j\}$, $n^j \in \{0, ..., d_2^j\}$, $p^j \in \{0, ..., d_3^j\}$, $\forall j$
- e.g., **color image**: (m^j, n^j) pixel position, p^j channel/color

$\overline{\boldsymbol{Z}}_1$: **input tensor to CNN** (colored image to classify)

$\boldsymbol{z}_J \in \mathbb{R}^C$: **CNN output**, a probability mass function (PMF).
- $[\boldsymbol{z}_J]_c$ probability that input belongs to class $c \in C$

$\overline{\boldsymbol{W}}_j$: **tensor models effect of layer** $j \in [J]$
- each layer can be: **convolution**, **non-linear** or **pooling** layer
- math expression depends on type of layer (next)

Convolution with a kernel

Ex: **Convolution** of a matrix (2nd order tensor), $\boldsymbol{A}$, with a kernel, $\boldsymbol{K}$

$$\begin{bmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 1 \end{bmatrix} \star \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} (1/4) = \begin{bmatrix} 3 & 4 & 11/4 \\ 6 & 7 & 17/4 \end{bmatrix} := \boldsymbol{B} \tag{1}$$

check board for calculations

Partition $\boldsymbol{A}$ into **all possible** 2×2 **blocks**

Each block is **mapped into one entry** in $\boldsymbol{B}$
the average of that block in that example (reduce noise)

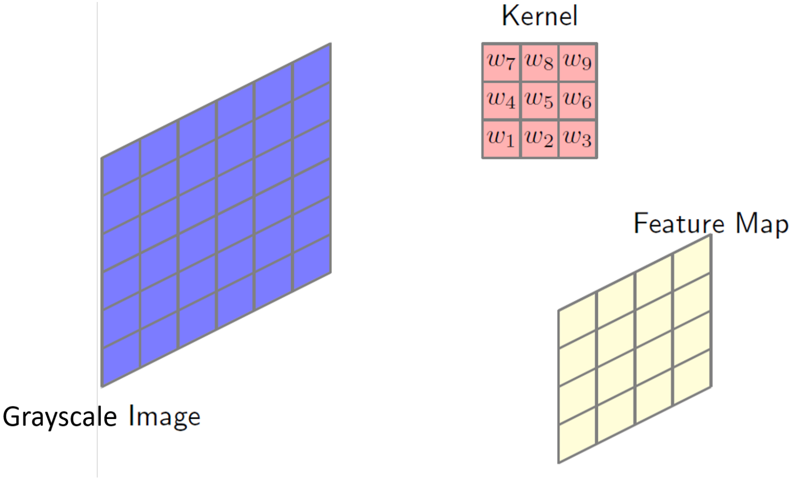
In general, kernels defined via convolutions for a given task:
denoising, edge detection, etc.

In this example convolution done with **2nd order tensor** (matrix) as kernel
e.g. convolution of grayscale image and one kernel matrix

in practice convolution **color images** with **multiple kernel matrices**
convolution among two 3rd order tensors (potentially more)
(1) may be extended to convolutions of 3rd order tensors

Example

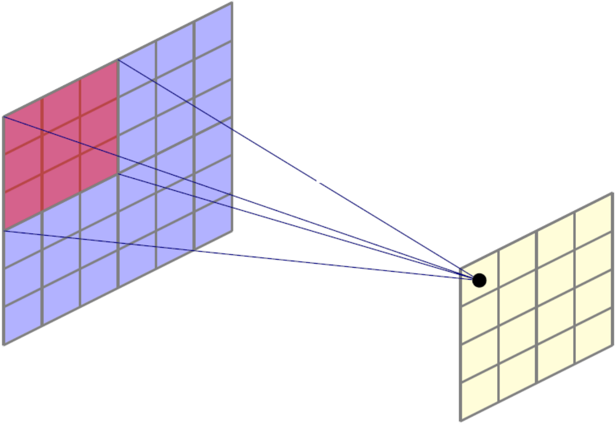
Convolution of grayscale image with Kernel matrix



Source: Trivedi, Kodor, 2017

Example

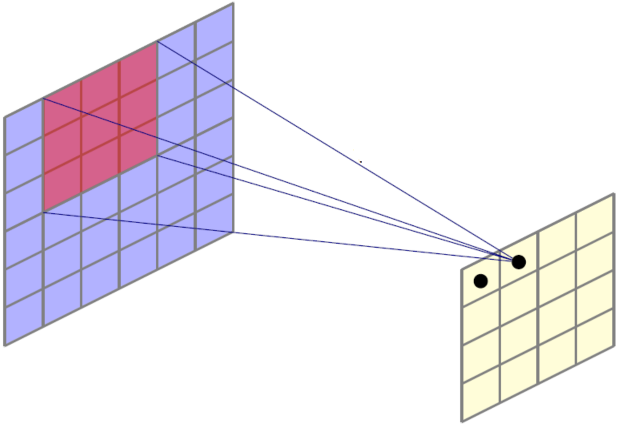
Convolution of grayscale image with Kernel matrix



Source: Trivedi, Kodor, 2017

Example

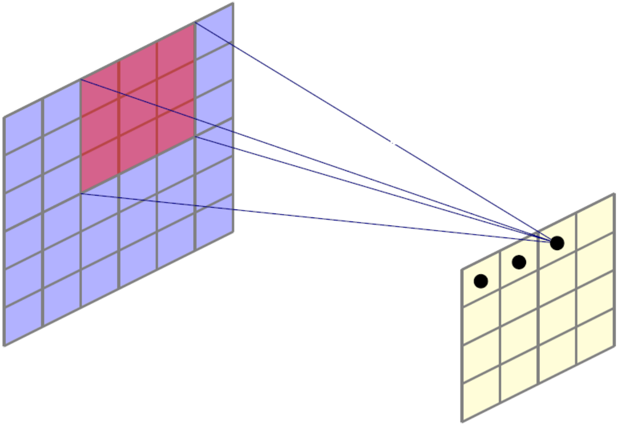
Convolution of grayscale image with Kernel matrix



Source: Trivedi, Kodor, 2017

Example

Convolution of grayscale image with Kernel matrix



Source: Trivedi, Kodor, 2017

Convolutional Layer

Layer j is **convolution** layer. Input-output relation :

$$\overline{\boldsymbol{Z}}_{j+1} = \overline{\boldsymbol{W}}_j \ \star \ \overline{\boldsymbol{Z}}_j$$

apply **more than one kernel matrix** $\Rightarrow$ modeled as 3rd order tensor

$\overline{\boldsymbol{W}}_j \in \mathbb{R}^{D_1^j \times D_2^j \times D_3^j}$: set of **all kernels applied at layer** j

at layer j: **apply** D_3^j **kernels, each kernel is a** $D_1^j \times D_2^j$ **matrix**

$\overline{\boldsymbol{Z}}_j \in \mathbb{R}^{d_1^j \times d_2^j \times d_3^j}$: input tensor at layer j

$\overline{\boldsymbol{Z}}_{j+1} \in \mathbb{R}^{d_1^{j+1} \times d_2^{j+1} \times d_3^{j+1}}$: output tensor at layer j

convolution $\star$ is among 3rd order tensors

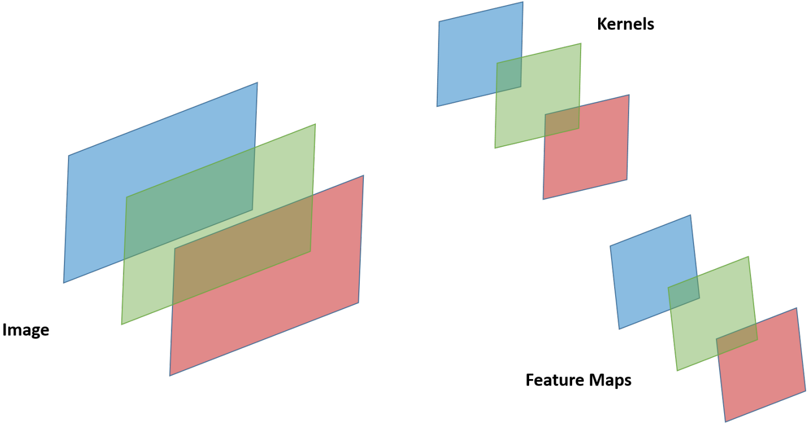
expression generalization of 2D case in (1)

$\star$ rewritten as matrix products

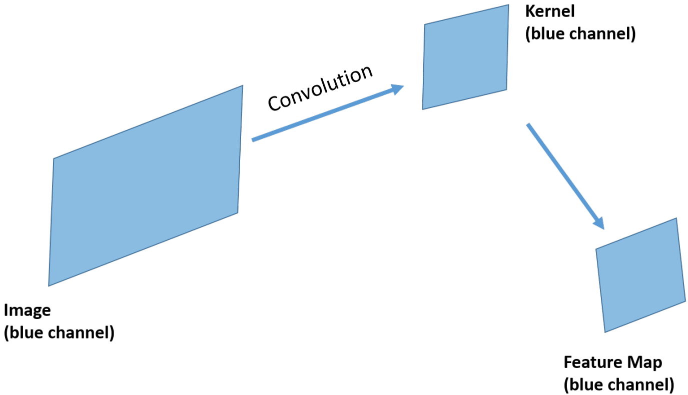
dimensions of $\overline{\boldsymbol{Z}}_{j+1}$ depends on that of $\overline{\boldsymbol{Z}}_{j+1}$ and $\overline{\boldsymbol{W}}_j$:

dimensions of $\overline{\boldsymbol{Z}}_{j+1}$ smaller if kernel larger 1×1

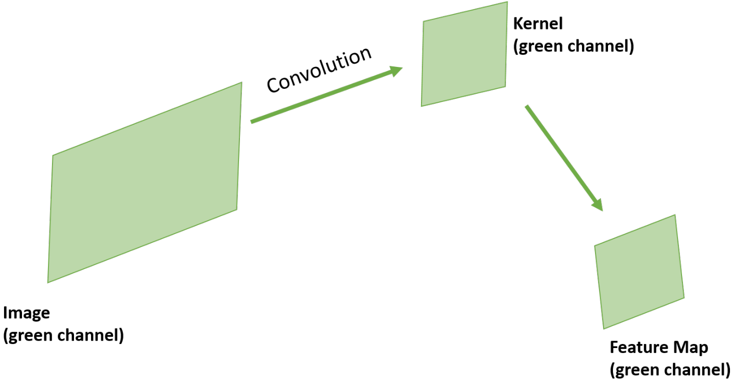
Convolutional Layer: Example



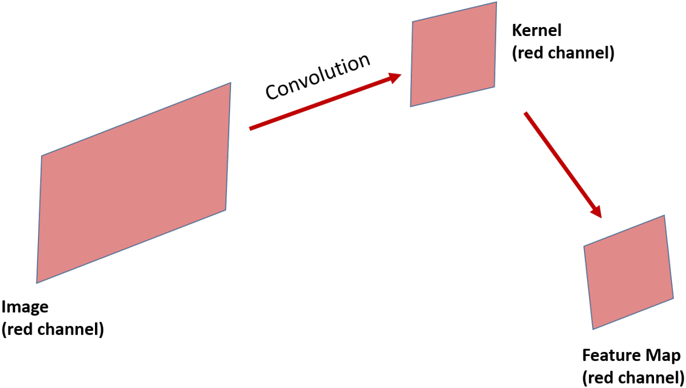
Convolutional Layer: Example



Convolutional Layer: Example



Convolutional Layer: Example



Nonlinear layer

Model for non-linear layers: $\overline{\boldsymbol{Z}}_{j+1} = \overline{\boldsymbol{\Sigma}}_j(\ \overline{\boldsymbol{Z}}_j)$

$\overline{\boldsymbol{\Sigma}}_j$: **non-linear element-by-element activation** for layer j

$\overline{\boldsymbol{\Sigma}}_j : \mathbb{R}^{d_1^j \times d_2^j \times d_3^j} \mapsto \mathbb{R}^{d_1^{j+1} \times d_2^{j+1} \times d_3^{j+1}}$

$\overline{\boldsymbol{\Sigma}}_j()$ element-by-element func: $d_1^{j+1} = d_1^j, d_2^{j+1} = d_2^j, d_3^{j+1} = d_3^j$

ReLU: commonly used for hidden layers

$$[\overline{\boldsymbol{Z}}_{j+1}]_{(m^j,n^j,p^j)} = \max\left(0, [\overline{\boldsymbol{Z}}_j]_{(m^j,n^j,p^j)}\right)$$

$\forall m^j \in \{0, ..., d_1^j\}, \forall n^j \in \{0, ..., d_2^j\}, \forall p^j \in \{0, ..., d_3^j\}$

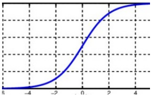
non-linearity increases approximation power of model

Intuition: positive value implies presence of cat in image

negative value implies absence of cat in image

output of ReLU:

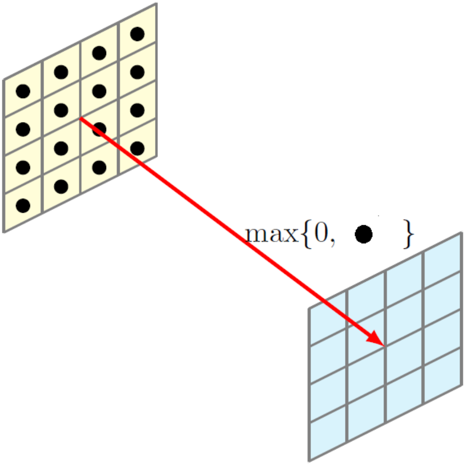
- if cat absent (negative value), set output to 0
- if cat present (positive value), pass to next layer



Softmax: used at output layer to ensure that $\boldsymbol{z}_J$ is PMF

not an element-by-function

Nonlinear Layer: Example



Source: Trivedi, Kodor, 2017

Pooling layer

Pooling layer: $\overline{\boldsymbol{Z}}_{j+1} = \overline{\boldsymbol{P}}_j(\overline{\boldsymbol{Z}}_j)$, $\overline{\boldsymbol{P}}_j()$ pooling operator for layer j

$\overline{\boldsymbol{P}}_j : \mathbb{R}^{d_1^j \times d_2^j \times d_3^j} \mapsto \mathbb{R}^{d_1^{j+1} \times d_2^{j+1} \times d_3^{j+1}}$

for each channel: **partition image into** $(D_1^j \times D_2^j)$ orthogonal blocks

combine/map each block into one scalar

depends only on (D_1^j, D_2^j) (no weights to learn)

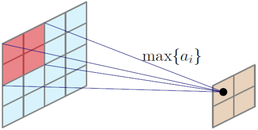
dim of output tensor: $d_1^{j+1} = d_1^j / D_1^j$, $d_2^{j+1} = d_2^j / D_2^j, d_3^{j+1} = d_3^j$

subsampling and downconversion

Max pooling:

for each channel: partition image in orthogonal $(D_1^j \times D_2^j)$ blocks of pixels

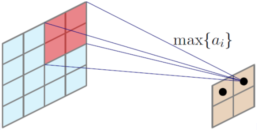
take max of each block



Average pooling:

for each channel: partition image in orthogonal $(D_1^j \times D_2^j)$ blocks of pixels

take average of each block



Training a CNN

Training a CNN for image classification:

Training set: $\{\overline{\boldsymbol{X}}_i, \boldsymbol{y}_i\}_{i=1}^N$
$\overline{\boldsymbol{X}}_i$ 3rd order tensor representing (colored) image i (matrix for grayscale image)
$\boldsymbol{y}_i$ vector has label of image i (cat, dog, tree, etc)

Loss function ?
ℓ_2 loss sometimes used, cross entropy loss for classification

Dersivations for backpropagation (chain rule) extended to 3rd order tensors; see [Wu, 2017] section 6

Architectures and Variants

VGG-Verydeep-16:
pre-trained CNNs with 39 layers developed by Oxford, VGG

LeNet-5:
shallow CNN, few parameters to train (earlier arch)

AlexNet:
state-of-art arch for image classification (developed by A Krizhevsky, 2014)

Deep Scattering Transform [Mallat, 2014]

Deep CNNs use so-called scattering transforms

Convolution for each layer uses preset wavelet coeff. **No training!**

Many proofs showing its approximation power

Conclusions

Several DL architectures

- CNNs: mathematical model using tensors.
  math model for each layer: convolutional, non-linear, pooling
  some variants and practical implementations

Some references

J. Wu, "Introduction to convolutional neural networks ", May 1, 2017, available on arxiv

A. Goodfellow, Y. Bendgio, A Courville, "Deep Learning", MIT Press, Chap. 8, 9

Y. LeCun, L Bottou, G. Orr, K-B Muller, "Efficient Backprop", Neural Networks: Tricks of the trade

S. Trivedi, R. Kondor, CMSC 35246 Deep Learning, Spring 2017

C. M. Bishop, "Pattern Recognition and Machine Learning", Springer-Verlag, 2006

S. Haykin, "Neural Networks and Learning Machines", 3rd, 2009, Chap 4

Check Tutorial by Ruslan Salakhutdinov:
`http://www.cs.cmu.edu/~rsalakhu/`

Deep Learning

Lecture 8: Deep Learning Architectures (Pt II)

University of Agder,
Kristiansand, Norway

Prepared by: Hadi Ghauch*, Hossein S. Ghadikolaei†

* Telecom ParisTech

† Royal Institute of Technology, KTH

https://sites.google.com/view/fundl/home

April 2019