Convolutional Neural Networks
Module Outline

- Foundational models
  - Convolutional Neural Networks
  - Deep Autoencoders and RBM
  - Gated Recurrent Networks (LSTM, GRU, ...)
  - Coding lectures: Keras/TF and Pytorch

- Advanced models
  - Advanced recurrent models (multiscale memories, ...)
  - Advanced sequential models (seq-to-seq, ...)
  - Attention and memory (Transformers, Neural Turing machines, ...)

More advanced topics in the generative DL module and in the final module
Reference Books (all freely available online)

Your choice between one of the two below:


Simon J.D. Prince, Understanding Deep Learning, MIT Press (2023)

Previously was:

Ian Goodfellow and Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press
Module’s Prerequisites

○ Formal model of neuron
○ Neural network
  ● Feed-forward
  ● Recurrent

○ Cost function optimization
  ● Backpropagation/SGD
  ● Regularization

○ Neural network hyper-parameters and model selection
Lecture Outline

○ Introduction and historical perspective
○ Dissecting the **components** of a CNN
  ● Convolution, stride, pooling
○ CNN **architectures** for machine vision
  ● Putting components back together
  ● From LeNet to ResNet
○ Advanced topics
  ● Interpreting convolutions
  ● Advanced **models and applications**

Split in two lectures
CNN Lecture – Part I
Introduction

Convolutional Neural Networks
Introduction

Convolutional Neural Networks

Destroying Machine Vision research since 2012
Neocognitron

- Hubel-Wiesel (‘59) model of brain visual processing
  - Simple cells responding to localized features
  - Complex cells pooling responses of simple cells for invariance
- Fukushima (‘80) built the first hierarchical image processing architecture exploiting this model

Trained by unsupervised learning
CNN for Sequences

- Apply a bank of 16 convolution kernels to sequences (windows of 15 elements)
- Trained by backpropagation with parameter sharing
- Guess who introduced it?
  ...yeah, HIM!

Time delay neural network (Waibel & Hinton, 1987)
CNN for Images

First convolutional neural network for images dates back to 1989 (LeCun)
Dense Vector Multiplication

Processing images: the dense way

32x32x3 image

Reshape it into a vector

$x$ 3072

An input-sized weight vector for each hidden neuron

$W$ 100x3072

Each element contains the activation of 1 neuron

$Wx^T$ 100
Convolution (Refresher)

Matrix input preserving spatial structure

**32x32**

**filter 5x5**

*sum 25 multiplications + bias*
Adaptive Convolution

$w_1$, $w_2$, $w_3$

$w_4$, $w_5$, $w_6$

$w_7$, $w_8$, $w_9$

$w^T x_{2,2}$

$w^T x_{9,7}$

$c_1 = w_1 + w_3 + 2w_4 + 3w_5 + 4w_6 + w_7 + w_9$

$c_2 = w_1 + w_3 + 2w_5 + w_7 + w_9$

Convolutional filter (kernel) with (adaptive) weights $w_i$
Convolutional Features

Slide the filter on the image computing elementwise products and summing up
Multi-Channel Convolution

Convolution filter has a number of slices equal to the number of image channels
Multi-Channel Convolution

All channels are typically *convolved together*
- They are summed-up in the convolution
- The convolution map stays bi-dimensional
Stride

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1
Stride

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1

stride = 1
Stride

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1

\[ \text{stride} = 1 \]
Stride

- Basic convolution **slides the filter** on the image one pixel at a time
  - Stride = 1
Stride

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1
- Can define a different stride
  - Hyperparameter

\( \text{stride} = 2 \)
Stride

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1
- Can define a different stride
  - Hyperparameter

stride = 2
Stride

○ Basic convolution *slides the filter* on the image one pixel at a time
  ● Stride = 1

○ Can define a different stride
  ● Hyperparameter

strand = 2
Stride

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1
- Can define a different stride
  - Hyperparameter
Stride

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1

- Can define a different stride
  - Hyperparameter

stride = 2
Works in both directions!
Stride

- Basic convolution *slides the filter* on the image one pixel at a time
  - Stride = 1
- Can define a different stride
  - Hyperparameter
- Stride reduces the *number of multiplications*
  - Subsamples the image

\[ \text{stride} = 3 \]
Stride

- Basic convolution slides the filter on the image one pixel at a time
  - Stride = 1
- Can define a different stride
  - Hyperparameter
- Stride reduces the number of multiplications
  - Subsamples the image

Stride = 3
Stride

○ Basic convolution slides the filter on the image one pixel at a time
  ● Stride = 1

○ Can define a different stride
  ● Hyperparameter

○ Stride reduces the number of multiplications
  ● Subsamples the image

stride = 3
Activation Map Size

What is the size of the image after application of a filter with a given size and stride?

Take a 3x3 filter with stride 1

K=3, S=1

Output image is: 5x5
What is the size of the image after application of a filter with a given size and stride?

Take a 3x3 filter with stride 2

Output image is: 3x3
Activation Map Size

What is the size of the image after application of a filter with a given size and stride?

General rule

\[ W' = \frac{W - K}{S} + 1 \]

\[ H' = \frac{H - K}{S} + 1 \]
Activation Map Size

What is the size of the image after application of a filter with a given size and stride?

Take a 3x3 filter with stride 3

\[ K=3, \ S=3 \]

Output image is: not really an image!
Zero Padding

Add *columns and rows of zeros* to the border of the image

```
0 0 0 0 0 0 0 0 0
0
0
0
0
0
0
0
0
```

H=7
W=7
**Zero Padding**

Add **columns and rows of zeros** to the border of the image

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**H=7 (P=1)**

**W=7 (P=1)**

\[ W' = \frac{W - K + 2P}{S} + 1 \]

\[ W' = \frac{7 - 3 + 2 \times 1}{1} + 1 \]

Output image is **7x7**

K=3, S=1
Zero Padding

Add columns and rows of zeros to the border of the image

Zero padding serves to retain the original size of image

\[ P = \frac{K - 1}{2} \]

Pad as necessary to perform convolutions with a given stride S
Feature Map Transformation

- Convolution is a **linear operator**
- Apply an element-wise nonlinearity to obtain a transformed **feature map**
Pooling

- Operates on the feature map to make the representation
  - Smaller (subsampling)
  - Robust to (some) transformations

Max pooling
- 2x2 filters
- stride = 2
- featured map
- pooled map
- W=4
- H=4
- W'=2
- H'=2
Pooling Facts

- Max pooling is the one used more frequently, but other forms are possible
  - Average pooling
  - L2-norm pooling
  - Random pooling
- It is uncommon to use zero padding with pooling

\[ W' = \frac{W - K}{S} + 1 \]
The Convolutional Architecture

- An architecture made by a hierarchical composition of the basic elements
- **Convolution layer** is an abstraction for the composition of the 3 basic operations
- **Network parameters** are in the convolutional component
A Bigger Picture

- **Input**
- **CL 1**
- **CL 2**
- **CL 3**
- **CL 4**
- **FCL 1**
- **FCL 2**

**Contains several convolutional filters with different size and stride**

- **Sparse connectivity**
- **Dense connectivity**

**CL -> Convolutional Layer**

**FCL -> Fully Connected Layer**
Convolutional Filter Banks

Convolutional filters of size $K \times K$

Feature map + nonlinearity

Pooling

Number of model parameters due to this convolution element (add $D_K$ bias terms)

Pooling is often (not always) applied independently on the $D_K$ convolutions
Specifying CNN in Code (Keras)

Number of convolution filters $D_k$

```python
model = Sequential()
model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1),
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(64, (5, 5)))
model.add(Activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(1000, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

Define input size (only first hidden layer)

Does for you all the calculations to determine the final size to the dense layer
A (Final?) Note on Convolution

○ We know that discrete convolution between an image $I$ and a filter/kernel $K$ is

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

and it is commutative.

○ In practice, convolution implementation in DL libraries does not flip the kernel

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

Which is cross-correlation and it is not commutative.
CNN as a Sparse Neural Network

Let us take a 1-D input (sequence) to ease graphics

Convolution amounts to **sparse connectivity** (reduce parameters) with **parameter sharing** (enforces invariance)
Dense Network

The dense counterpart would look like this

![Dense Network Diagram]
Strided Convolution

Make connectivity sparser

Strided convolution
Max-Pooling and Spatial Invariance

A feature is detected even if it is spatially translated

![Feature map diagram](image-url)
Cross Channel Pooling and Spatial Invariance
Hierarchical Feature Organization

The deeper the larger the receptive field of a unit
Zero-Padding Effect

Assuming no pooling
CNN Lecture – Part II
CNN Training

Variants of the standard backpropagation that account for the fact that connections share weights (convolution parameters)

The gradient $\Delta w_i$ is obtained by summing the contributions from all connections sharing the weight

Backpropagating gradients from convolutional layer $N$ to $N-1$ is not as simple as transposing the weight matrix (need deconvolution with zero padding)
Backpropagating on Convolution

Convolution

K=3, S=1

Input is a 4x4 image
Output is a 2x2 image

Backpropagation step requires going back from the 2x2 to the 4x4 representation

Can write convolution as dense multiplication with shared weights

\[
\begin{pmatrix}
  w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\
  0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\
  0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\
  0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0
\end{pmatrix}
\]

Backpropagation is performed by multiplying the 4x1 representation to the transpose of this matrix
Deconvolution (Transposed Convolution)

We can obtain the transposed convolution using the same logic of the forward convolution.

If you had no padding in the forward convolution, you need to pad much when performing transposed convolution.
Deconvolution (Transposed Convolution)

If you have striding, you need to fill in the convolution map with zeroes to obtain a correctly sized deconvolution.

K=3, S=2, P=1

https://github.com/vdumoulin/conv_arithmetic
LeNet-5 (1989)

- Grayscale images
- Filters are 5x5 with stride 1 (sigmoid nonlinearity)
- Pooling is 2x2 with stride 2
- No zero padding
AlexNet (2012) - Architecture

- RGB images $227 \times 227 \times 3$
- 5 convolutional layers + 3 fully connected layers
- Split into two parts (top/bottom) each on 1 GPU
AlexNet - Innovations

- Use heavy data augmentation (rotations, random crops, etc.)
- Introduced the use of ReLu
- Dense layers regularized by dropout
ReLU Nonlinearity

- ReLU help counteract gradient vanish
  - Sigmoid first derivative vanishes as we increase or decrease $z$
  - ReLU first derivative is 1 when unit is active and 0 elsewhere
  - ReLU second derivative is 0 (no second order effects)
- Easy to compute (zero thresholding)
- Favors sparsity
AlexNet - Parameters

- 62.3 millions of parameters (6% in convolutions)
- 5-6 days to train on two GTX 580 GPUs (95% time in convolutions)
VGGNet – VGG16 (2014)

- Standardized convolutional layer
  - 3x3 convolutions with stride 1
  - 2x2 max pooling with stride 2 (not after every convolution)

- Various configuration analysed, but best has
  - 16 Convolutional + 3 Fully Connected layers
  - About 140 millions parameters (85% in FC)

ImageNet Top-5: 7.3%
GoogLeNet (2015)

ImageNet Top-5 : 6.7%

Why 1x1 convolutions?

- Kernels of different size to capture details at varied scale
- Aggregated before sending to next layer
- Average pooling
- No fully connected layers
By placing 1x1 convolutions before larger kernels in the Inception module, the number of input channels is reduced, saving computations and parameters.
Back on GoogLeNet

- Only 5 millions of parameters
- 12X less parameters than AlexNet
- Followed by v2, v3 and v4 of the Inception module
  - More filter factorization
  - Introduce heavy use of Batch Normalization

Auxiliary outputs to inject gradients at deeper layers
Batch Normalization

- Very deep neural network are subject to **internal covariate shift**
  - Distribution of *inputs to a layer N might vary* (shift) with different minibatches (due to adjustments of layer N-1)
  - Layer N can get confused by this
  - Solution is to *normalize for mean and variance* in each minibatch (bit more articulated than this actually)

\[
\begin{align*}
\mu_b &= \frac{1}{N_b} \sum_{i=1}^{N_b} x_i \\
\sigma_b^2 &= \frac{1}{N_b} \sum_{i=1}^{N_b} (x_i - \mu_b)^2
\end{align*}
\]

\[
\hat{x}_i = \frac{x_i - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}}
\]

\[
y = \gamma \hat{x}_i + \beta
\]

Scale and shift

Trainable linear transform potentially allowing to cancel unwanted zero-centering effects (e.g. sigmoid)

Need to backpropagate through this!

Begin of the Ultra-Deep Network Era (152 Layers)

ImageNet Top-5 : 3.57%

Why wasn’t this working before?

Gradient vanishes when backpropagating too deep!
ResNet Trick

The input to the block $X$ bypasses the convolution and is then combined with its residual $F(X)$ resulting from the convolutions.

When backpropagating the gradient flows in full through these bypass connections.
ResNet & Batch Norm

When connecting several Residual Blocks in series, one need to be careful about amplification/compounding of variance due to the residual connectivity

• Batch norm can alleviate this effect
MobileNets

Making CNNs efficient to run on mobile devices by depthwise separable convolutions

Basically run channel-independent convolutions followed by 1x1 convolutions for cross-channel mixing

arxiv.org/pdf/1704.04861.pdf
CNN Architecture Evolution
Understanding CNN Embedding

tSNE projection of AlexNet last hidden dense layer

https://cs.stanford.edu/people/karpathy/cnnembed/
Interpreting Intermediate Levels

○ What about the information captured in convolutional layers?
○ Visualize kernel weights (filters)
  ● Naïve approach
  ● Works only for early convolutional layers
○ Map the activation of the convolutional kernel back in pixel space
  ● Requires to reverse convolution
  ● Deconvolution

Zeiler & Fergus, Visualizing and Understanding Convolutional Networks, ICML 2013
Deconvolutional Network (DeConvNet)

- Attach a DeConvNet to a target layer
- Plug an input and forward propagate activations until layer
- Zero activations of target neuron
- Backpropagate on the DeConvNet and see what parts of the reconstructed image are affected
Inspect Deconvolution Layers
Filters & Patches – Layer 1

Reconstructed filters in pixel space

Corresponding top-9 image patches

Zeiler & Fergus, Visualizing and Understanding Convolutional Networks, ICML 2013
Filters & Patches – Layer 2

Zeiler&Fergus, Visualizing and Understanding Convolutional Networks, ICML 2013
Filters & Patches – Layer 3

Zeiler & Fergus, Visualizing and Understanding Convolutional Networks, ICML 2013
Filters & Patches – Layer 4

Zeiler & Fergus, Visualizing and Understanding Convolutional Networks, ICML 2013
Filters & Patches – Layer 5

Zeiler & Fergus, Visualizing and Understanding Convolutional Networks, ICML 2013
Occlusions

- Measure what happens to feature maps and object classification if we occlude part of the image
- Slide a grey mask on the image and project back the response of the best filters using deconvolution
Occlusions

Zeiler & Fergus, Visualizing and Understanding Convolutional Networks, ICML 2013
Dense CNN

- **batch normalization** + ReLU + 3x3 conv

**Transition layers**
- batch normalization + 1x1 convolutional + 2x2 average pooling layer
- batch normalization + ReLU + 3x3 conv

- Gradient flows well in bypass connections
- Each layer in the dense block has access to all information from previous layers

Huang et al, Densely Connected Convolutional Networks, CVPR 2017
Causal Convolutions

Preventing a convolution from allowing to see into the future...

Problem is the context size grows slow with depth
Causal & Dilated Convolutions

\[(I * K)(i, j) = \sum_m \sum_n I(i - lm, i - ln)K(m, n)\]

Similar to striding, but size is preserved

Semantic Segmentation

Traditional CNN cannot be used for this task due to the downsampling of the striding and pooling operations
Fully Convolutional Networks (FCN)

- Convolutional part to extract interesting features at various scales
- Fuse information from feature maps of different scale
- Learn an upsampling function of the fused map to generate the semantic segmentation map

Shelhamer et al, Fully Convolutional Networks for Semantic Segmentation, PAMI 2016
Maxpooling indices transferred to decoder to improve the segmentation resolution.
SegNet Segmentation

Demo here: [http://mi.eng.cam.ac.uk/projects/segnet/](http://mi.eng.cam.ac.uk/projects/segnet/)
Use Dilated Convolutions

Always perform 3x3 convolutions with no pooling at each level

Context increases without
- Pooling (changes map size)
- Increasing computational complexity

Yu et al, Multi-Scale Context Aggregation by Dilated Convolutions, ICLR 2016
Segmentation by Dilated CNN

Yu et al, Multi-Scale Context Aggregation by Dilated Convolutions, ICLR 2016
1D convolutions throughout the input sequence
- Trained to respond to task-specific motifs
- Applied to small sequence regions
DeepBind

- 927 CNN models predicting a binding score for transcription factors and RNA-binding proteins
  - Score new sequences
  - Assess mutations that deplete/increase binding score
- Use convolution visualization to interpret results of CNN training

DeepSea

The feature detectors in the deeper layers are shared between the predictive tasks

Software

○ CNN are supported by any deep learning framework (Keras-TF, Pytorch, MS Cognitive TK, Intel OpenVino, ...)

○ Caffe was one of the initiators and basically built around CNN
  ● Introduced protobuffer network specification
  ● ModelZoo of pretrained models (LeNet, AlexNet, ...)
  ● Support for GPU

○ Caffe2 is Facebook’s extensions to Caffe
  ● Less CNN oriented
  ● Support from large scale to mobile nets
  ● More production oriented than other frameworks
Caffe Protobuffer

```protobuf
definition:
  name: "LeNet"
  layer {    
    name: "data"
    type: "Input"
    ...
    input_param { shape: { dim: 64 dim: 1 dim: 28 dim: 28 } }
  }
  layer {    
    name: "conv1"
    type: "Convolution"
    bottom: "data"
    ...
    convolution_param { 
      num_output: 20
      kernel_size: 5
      stride: 1
      weight_filler { 
        type: "xavier"
      }
    }
  }
```
Other Software

○ Matlab distributes its Neural Network Toolbox which allows importing pretrained models from Caffe and Keras-TF

○ Matconvnet is an unofficial Matlab library specialized for CNN development (GPU, modelzoo, ...)

○ Want to have a CNN in your browser?
  ● Try ConvNetJS (https://cs.stanford.edu/people/karpathy/convnetjs/)
GUls

Major hardware producers have GUI and toolkits wrapping Caffe, Keras and TF to play with CNNs

NVIDIA Digits

Plus others...
Take Home Messages

- **Key things**
  - **Convolutions** in place of dense multiplications allow sparse connectivity and weight sharing
  - **Pooling** enforces invariance and allows to change resolution but shrinks data size
  - **Full connectivity** compress information from all convolutions but accounts for 90% of model complexity

- **Lessons learned**
  - **ReLU** are efficient and counteract gradient vanish
  - **1x1 convolutions** are useful
  - Need **batch normalization**
  - **Bypass connections** allow to go deeper

- **Dilated (à trous) convolutions**
- **You can use CNN outside** of machine vision
Next Lecture

Deep Autoencoders

- Autoencoders and dimensionality reduction
- Neural autoencoders (sparse, denoising contractive)
- Deep neural autoencoders and pretraining
- Deep generative-based autoencoders
- Visualization and multi-modal data fusion with autoencoders
Next Week Lectures

- Wednesday h. 16-18
- Thursday h. 14-16
- Friday h. 16-18 – Room E
Happy Easter Break!