Neural Networks

COMP 4107

Convolutional Neural Networks
Overview

Goal: Understand what Convolutional Neural Networks (ConvNets) are & intuition behind it.

1. Brief Motivation for Deep Learning
2. What are ConvNets?
3. ConvNets for Object Detection
First of all what is Deep Learning?

- Composition of non-linear transformation of the data.
- Goal: Learn useful representations, aka features, directly from data.

- Many varieties, can be unsupervised or supervised.
- Today is about ConvNets, which is a supervised deep learning method.
Supervised Learning: Examples

Classification
- Image of a puppy
- Label: "dog"

Denoising
- Noisy image of a pepper
- Denoised image of a pepper

OCR
- Image of handwritten numbers "2345"
- Recognized numbers "2345"

Slide: M. Ranzato
Supervised Deep Learning

**Classification**

![Image of a puppy](image1.png)

**Denoising**

![Image of a noisy image](image2.png)

**OCR**

![Image of handwritten digits](image3.png)
So deep learning is about learning feature representation in a compositional manner.

But wait, why learn features?
The Black Box in a Traditional Recognition Approach

Preprocessing

Feature Extraction (HOG, SIFT, etc)

Post-processing (Feature selection, MKL etc)

Classifier (SVM, boosting, etc)
The Black Box in a Traditional Recognition Approach

Preprocessing
(Hand Engineered)

Feature Extraction
(HOG, SIFT, etc)

Post-processing
(Feature selection, MKL etc)

Classifier
(SVM, boosting, etc)

“dog”
Most critical for accuracy
Most time-consuming in development
What is the best feature???
What is next?? Keep on crafting better features?
⇒ Let’s learn feature representation directly from data.
Learn features and classifier together

⇒ Learn an end-to-end recognition system. A non-linear map that takes raw pixels directly to labels.

**Q:** How can we build such a highly non-linear system?

**A:** By combining simple building blocks we can make more and more complex systems.
Building a complicated function

Simple Functions

\[ \sin(x), \log(x), \cos(x), \exp(x), x^3 \]

One Example of Complicated Function

\[ \log(\cos(\exp(\sin^3(x)))) \]
Composition is at the core of deep learning methods
Each “simple function” will have parameters subject to learning
Intuition behind Deep Neural Nets

"CAR"

Slide: M. Ranzato
Intuition behind Deep Neural Nets

NOTE: Each black box can have trainable parameters. Their composition makes a highly non-linear system.
Intuition behind Deep Neural Nets

“CAR”

Layer 1 → Layer 2 → Layer 3 → Layer 4

NOTE: Each black box can have trainable parameters. Their composition makes a highly non-linear system.

The final layer outputs a probability distribution of categories.

Slide: M. Ranzato
Why use hierarchical multi-layered models?

Argument 1: visual scenes are hierarchically organised

object
  ↓
object parts
  ↓
primitive features
  ↓
input image

trees
  ↓
bark, leaves, etc.
  ↓
oriented edges
  ↓
forest image

\[ z_1, z_2, \ldots, z_K \]
Why use hierarchical multi-layered models?

Argument 2: biological vision is hierarchically organised

- **Input Image**
  - **Primitive Features**
    - **Object Parts**
      - **Object**
        - **Trees**
          - **Inferotemporal Cortex**
            - **V4: Different Textures**
        - **Forest Image**
          - **V1: Simple and Complex Cells**
        - **Oriented Edges**
          - **Bark, Leaves, etc.**
            - **Photo-receptors Retina**
Why use hierarchical multi-layered models?

Argument 3: shallow architectures are inefficient at representing deep functions

single layer neural network implements: $x = f_\theta(z)$

shallow networks can be computationally inefficient

networks we met last lecture with large enough single hidden layer can implement any function 'universal approximator'

however, if the function is 'deep' a very large hidden layer may be required
What’s wrong with standard neural networks?

How many parameters does this neural network have?

\[ |\theta| = 3D^2 + D \]

For a small 32 by 32 image:

\[ |\theta| = 3 \times 32^4 + 32^2 \approx 3 \times 10^6 \]

Hard to train
  over-fitting and local optima

Need to initialise carefully
  layer wise training
  unsupervised schemes

Convolutional nets reduce the number of parameters
The Story So Far

• Neural networks can recognize simple images; e.g., handwritten digits

\[504192\]

• Used fully connected, multi-layer FF NN
The Solution

• Each pixel represented in input. So, for 28x28 image = 784 neurons

• Can do well; e.g., 98% accuracy on MNIST handwritten digit data

• Fully connected layers don’t take into account spatial structure of images

• Pixels far apart and close together treated equally; ignores fact that features depend upon pixels close together
When the input data is an image..

Example: 1000x1000 image
1000 hidden units
1B parameters!!!

- Spatial correlation is local
- Better to put resources elsewhere!
Convolutional Neural Networks

• CNNs don’t ignore spatial relationships
• Well suited for image analysis
• An example of a deep neural network; i.e., one having many layers
• Based upon 3 ideas:
  – Local receptive fields
  – Shared weights
  – Pooling
The key ideas behind convolutional neural networks

- **image statistics are translation invariant** (objects and viewpoint translates)
  - build this translation invariance into the model (rather than learning it)
  - tie lots of the weights together in the network
  - reduces number of parameters

- **expect learned low-level features to be local** (e.g. edge detector)
  - build this into the model by allowing only local connectivity
  - reduces the numbers of parameters further

- **expect high-level features learned to be coarser** (c.f. biology)
  - build this into the model by subsampling more and more up the hierarchy
  - reduces the number of parameters again
Reduce connection to local regions

Example: 1000x1000 image
1000 hidden units
Filter size: 10x10
100000 parameters

Significantly smaller!

100000 = 1000 * 10 * 10
Local Receptive Fields

• In fully connected layers, inputs shown as a vertical line of neurons
• In CNN, think of inputs as (say) 28x28 square of neurons
• Don’t connect all input neurons to hidden layer, just small, localized regions
Local Receptive Fields

- Region in input image is called **local receptive field** for hidden neuron
- Small window on input pixels
- Each connection learns a weight.
- Hidden neuron learns a bias too
Local Receptive Field

- Slide local receptive field across entire input image.
- For each local receptive field, there is a different hidden neuron in the first hidden layer.
Local Receptive Field

- Slide local receptive field right one pixel.
- Connect to second neuron in the first hidden layer
Local Receptive Fields

• So, if we have 28x28 image, and 5x5 local receptive fields, we will have 24x24 neurons in first hidden layer
• Local receptive field is also called a filter
• Distance moved (here 1 pixel) is called the stride length
• May have to pad the image if the local receptive field and stride length don’t fit nicely (see the video for a discussion on this)
Reuse the same kernel everywhere

Because interesting features (edges) can happen at anywhere in the image.

Share the same parameters across different locations:
Convolutions with learned kernels

ONLY ONE SET OF WEIGHTS!
Shared Weights and Biases

- Each hidden neuron has a bias and (here) 5x5 weights (which is small compared to fully connected FF NN)
- So, for each hidden neuron, output is:

\[ \sigma \left( b + \sum_{l=0}^{4} \sum_{m=0}^{4} w_{l,m} a_{j+l,k+m} \right). \]

- So, all neurons in first layer detect the same feature but at different locations
Feature Map

• Map from input layer to hidden layer is called a feature map.
• Weights defining feature maps are shared weights.
• Shared weights and bias are often called a kernel or filter.
• CNNs are well adapted to the spatial invariance of images
  – A cat is still a cat wherever it appears in an image
Multiple Features

• Will need many features, so have several feature maps (filters).
• Here, 3 are shown; may need many more!
Convolutional Neural Nets

Filters are FEATURES.
For example a line at an angle.

E.g.: 1000x1000 image
100 Filters
Filter size: 10x10
10K parameters

Significantly smaller!

10,000 = 100 * 10 * 10

LeCun et al. “Gradient-based learning applied to document recognition” IEEE 1998
Detail

If the input has 3 channels (R,G,B), 3 separate k by k filter is applied to each channel.

Output of convolving 1 feature is called a feature map.

This is just sliding window, ex. the output of one part filter of DPM is a feature map.
Example

• LeNet-5 used 6 feature maps and a 5x5 local receptive field to recognize MNIST digits
• White means small weight (more –ve), black means +ve
• Represents type of feature in input image
Feature Maps?

- Network is learning things related to spatial structure
- NOT learning (e.g. Gabor) filters
- Here, each feature map needs $5\times5+1 = 26$ weights. With 20, we need $26\times20 = 520$
- For a FFNN, with 30 hidden layer neurons, we need $28\times28\times30 + 30 = 23,550$ parameters

This is 40x a CNN
Using multiple filters

Each filter detects features in the output of previous layer.
So to capture different features, learn multiple filters.

**NOTE:** the nr. of output feature maps is usually larger than the nr. of input feature maps
Example of filtering

- Convolutional
  - Translation equivariance
  - Tied filter weights
    (same at each position → few parameters)

Input

Feature Map
Convolutional Neural Nets

Learn multiple filters.

E.g.: 1000x1000 image
100 Filters
Filter size: 10x10
10K parameters

Building Translation Invariance

Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?
Building Translation Invariance via Spatial Pooling

By “pooling” (e.g., max or average) filter responses at different locations we gain robustness to the exact spatial location of features.

Pooling also subsamples the image, allowing the next layer to look at larger spatial regions.
Pooling Layers

• Used immediately after convolutional layers
• They simplify information in output from convolutional layer
• Here 2x2 mapped to 1 neuron. So, go from 24x24 to 12x12. Do this for each map.
Pooling

• Pooling essentially “throws away” exact positional information: condensed information.

• Rationale is that once a feature is found we don’t care about its position.
Single depth slice

\[
\begin{array}{cccc}
1 & 1 & 2 & 4 \\
5 & 6 & 7 & 8 \\
3 & 2 & 1 & 0 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

max pool with 2x2 filters and stride 2

\[
\begin{array}{cc}
6 & 8 \\
3 & 4 \\
\end{array}
\]
Pooling types

• Convolutional layer connects only to its corresponding pooling layer.
• **Max Pooling**: take maximum value in pooled neurons
• **L2 Pooling**: Take square root of the sum of the squares of the activation in the region
Complete Architecture

- Final layer connects every neuron from pooled layer to 10 (# classes) output neurons
- Training is via backpropagation
One more step!

- Add *softmax* layer to provide normalization (common in image classification problems)

\[ P(y = j | x) = \frac{e^{x^T w_j}}{\sum_{k=1}^{K} e^{x^T w_k}} \]
Summary

Building block of a convolutional neural network

- **Pooling stage**:
  \[ x_{i,j} = \max_{|k| < \tau, |l| < \tau} y_{i-k,j-l} \]
  Mean or subsample also used

- **Non-linear stage**:
  \[ y_{i,j} = f(a_{i,j}) \]
  e.g. \[ f(a) = [a]_+ \]
  \[ f(a) = \text{sigmoid}(a) \]

- **Convolutional stage**:
  \[ a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l} \]
  Only parameters

- **Input image**:
  \[ z_{i,j} \]
Full convolutional neural network

connects to several feature maps

layer 2

layer 1

will have different filters

\[ a_{i,j}^{(1,q)} = \sum_{k,l} w_{k,l}^{q} z_{i-k,j-l} \]

'normal' neural network

non-linear stage

convolutional stage

non-linear stage

convolutional stage

non-linear stage
How many parameters does a convolutional network have?

How many parameters does this neural network have?

\[ |\theta| = 3K^2 + 9K^2 + 9K^2 + 3(D/S)^2 \]
\[ = 21K^2 + D \]

For a small 32 by 32 image:

\[ K = 5 \quad S = 2 \]
\[ |\theta| = 21 \times 5^2 + 4^2 \approx 600 \]
Training

• **back-propagation for training**: stochastic gradient ascent

  – like tutorial output interpreted as a class label probability, \( x = p(t = 1 \mid z) \)
  – now \( x \) is a more complex function of the inputs \( z \)
  – can optimise same objective function computed over a mini-batch of datapoints

• **data-augmentation**: always improves performance substantially (include shifted, rotations, mirroring, locally distorted versions of the training data)

• **typical numbers**:

  – 5 convolutional layers, 3 layers in top neural network
  – 500,000 neurons
  – 50,000,000 parameters
  – 1 week to train (GPUs)
Neural Net Training

A) Compute loss on small mini-batch
Neural Net Training

A) Compute loss on small mini-batch
Neural Net Training

A) Compute loss on small mini-batch

F-PROP

Layer 1 → Layer 2
Neural Net Training

A) Compute loss on small mini-batch
B) Compute gradient w.r.t. parameters

B-PROP
Neural Net Training

A) Compute loss on small mini-batch
B) Compute gradient w.r.t. parameters

B-PROP

Layer 1  \rightarrow  Layer 3
Neural Net Training

A) Compute loss on small mini-batch
B) Compute gradient w.r.t. parameters

B-PROP
Neural Net Training

A) Compute loss on small mini-batch
B) Compute gradient w.r.t. parameters
C) Use gradient to update parameters $W \leftarrow W - \eta \frac{dL}{dW}$
CIFAR 10 dataset: 50,000 training images, 10,000 test images
http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html
Looking into a convolutional neural network's brain

reconstruction of image patches from that unit (indicates aspect of patches which unit is sensitive to)

top 9 image patches that cause maximal activation in layer 2 unit
Looking into a convolutional neural network’s brain
Looking into a convolutional neural network’s brain
Architecture of Alex Krizhevsky et al.

- 8 layers total.
- Trained on Imagenet Dataset (1000 categories, 1.2M training images, 150k test images)
- 18.2% top-5 error
  - Winner of the ILSVRC-2012 challenge.
Architecture of Alex Krizhevsky et al.

Image cutoff is from the paper – not me!
ConvNets as generic feature extractor

- A well-trained ConvNets is an excellent feature extractor.
- Chop the network at desired layer and use the output as a feature representation to train a SVM on some other vision dataset.

<table>
<thead>
<tr>
<th></th>
<th>Cal-101 (30/class)</th>
<th>Cal-256 (60/class)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM 1</td>
<td>44.8 ± 0.7</td>
<td>24.6 ± 0.4</td>
</tr>
<tr>
<td>SVM 2</td>
<td>66.2 ± 0.5</td>
<td>39.6 ± 0.3</td>
</tr>
<tr>
<td>SVM 3</td>
<td>72.3 ± 0.4</td>
<td>46.0 ± 0.3</td>
</tr>
<tr>
<td>SVM 4</td>
<td>76.6 ± 0.4</td>
<td>51.3 ± 0.1</td>
</tr>
<tr>
<td>SVM 5</td>
<td><strong>86.2 ± 0.8</strong></td>
<td>65.6 ± 0.3</td>
</tr>
<tr>
<td>SVM 7</td>
<td><strong>85.5 ± 0.4</strong></td>
<td><strong>71.7 ± 0.2</strong></td>
</tr>
<tr>
<td>Softmax 5</td>
<td>82.9 ± 0.4</td>
<td>65.7 ± 0.5</td>
</tr>
<tr>
<td>Softmax 7</td>
<td><strong>85.4 ± 0.4</strong></td>
<td><strong>72.6 ± 0.1</strong></td>
</tr>
</tbody>
</table>

- Improve further by taking a pre-trained ConvNet and re-training it on a different dataset. Called *fine-tuning*
One way to do detection with ConvNets

Since ConvNets extract discriminative features, one can crop images at the object bounding box and train a good SVM on each category.

⇒ Extract regions that are likely to have objects, then apply ConvNet + SVM on each and use the confidence to do maximum suppression.
R-CNN: Regions with CNN features

Best performing method on PASCAL 2012
Detection improving previous methods by 30%

Slide: R. Girshick
Summary

• higher level layers encode more **abstract features**

• higher level layers show more **invariance to instantiation parameters**
  – translation
  – rotation
  – lighting changes

• a method for **learning feature detectors**
  – first layer learns edge detectors
  – subsequent layers more complex
  – integrates training of the classifier with training of the featural representation
Finally some cautionary words

- hierarchical modelling is a **very old idea** and not new

- the ‘deep learning’ revolution has come about mainly due to new methods for initialising learning of neural networks

- current methods aim at invariance, but this is far from all there is to computer and biological vision: e.g. **instantiation parameters should also be represented**

- **classification can only go so far**: "tell us a story about what happened in this picture"
ConvNet Libraries

- **Cuda-convnet** (Alex Krizhevsky, Google)
- **Caffe** (Y. Jia, Berkeley, now Google)
  - Pre-trained Krizhevsky model available.
- **Torch7** (Idiap, NYU, NEC)

more around.