Ceng 783 – Deep Learning

Week 4 – Convolutional Neural Networks

Fall 2017

Emre Akbas
• Convolutional Neural Networks = ConvNets or CNN


• “CNNs are simply artificial neural networks that use convolution in place of general matrix multiplication in at least one of their layers.” [Goodfellow et al. (2016)].

• Used for data that has a grid-like topology. E.g. images (2-D grid), videos (3-D grid), time series data (1-D grid).
Regular ANN vs CNN?

• ANN → fully connected.
  - Uses matrix multiplication to compute the next layer.
• CNN → sparse connections.
  - Uses convolution to compute the next layer.
• Everything else stays almost the same
  - Activation functions
  - Cost functions
  - Training (back-propagation)
  - ...
ConvNets learn both:

• Hierarchical representations of the data, and
• Supervised decision boundary on these representations

at the same time.
Deep Learning = Learning Hierarchical Representations

Traditional Pattern Recognition: Fixed/Handcrafted Feature Extractor

Mainstream Modern Pattern Recognition: Unsupervised mid-level features

Deep Learning: Representations are hierarchical and trained

[Slide by Yann LeCun (http://www.slideshare.net/embeddedvision/01-am-keynotelecun)]
Deep Learning = Learning Hierarchical Representations

It's deep if it has more than one stage of non-linear feature transformation

Low-Level Feature → Mid-Level Feature → High-Level Feature → Trainable Classifier

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Convolution
Closely related to correlation (a.k.a. cross-correlation).

We use these operations to extract information from a signal.

\[ s(t) = (x \star w)(t) = \int x(a)w(a+t) \, da \]

\[ s[t] = (x \star w)[t] = \sum_{a=-\infty}^{a=\infty} x[a]w[a+t] \]
Naming convention in computer vision and CNNs

\[ s[t] = (x \star w)[t] = \sum_{a=-\infty}^{a=\infty} x[a]w[a+t] \]

Computes **similarity** of two signals. Can be used to find patterns (template matching with normalized cross-correlation).
Convolution and cross-correlation

Both are linear, shift-invariant operations.

Cross-correlation:

\[ s[t] = (x \ast w)[t] = \sum_{a=-\infty}^{a=\infty} x[a]w[a+t] \]

Convolution:

\[ s[t] = (x \ast w)[t] = \sum_{a=-\infty}^{a=\infty} x[a]w[t-a] \]

Identical operations except that the kernel is flipped in convolution. If the kernel is symmetric, then they are identical.
In 2D

\[ S(i, j) = (I * K)(i, j) = \sum_{m} \sum_{n} I(i + m, j + n)K(m, n) \]

This is the formula for cross-correlation in 2D.

Many machine learning libraries implement cross-correlation but call it convolution.
Strictly speaking, this is a cross-correlation, not convolution.
Another example

Applying the Sobel operator for edge detection.

Input x is a grayscale image, i.e. a 2D matrix
Convolution example

```
3  53  50  49  51  51  51  52  52  53  81  126  1
3  53  54  53  54  54  55  52  60  102  152  179  1
6  58  58  58  59  58  55  68  126  165  179  186  1
0  59  60  62  59  58  75  136  172  183  192  196  2
8  59  59  59  58  72  137  170  182  193  198  201  2
6  57  57  57  62  128  171  181  188  197  203  205  2
7  58  57  55  104  166  178  186  198  200  200  211  2
4  56  54  79  153  174  180  193  197  203  217  226  2
9  51  61  129  173  174  190  192  203  217  221  223  2
```
Convolution example

Kernel is a 3x3 matrix

\[ y = x \ast \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \]

\( y \) is the resulting feature map. In this case, a map that shows the magnitude (i.e. contrast) of vertical edges.
Convolution example

\[
\begin{bmatrix}
-1 & 0 & +1 \\
-2 & 0 & +2 \\
-1 & 0 & +1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
-1 & -2 & +1 \\
0 & 0 & 0 \\
+1 & +2 & +1 \\
\end{bmatrix}
\]

[Figures from https://en.wikipedia.org/wiki/Sobel_operator]
Motivation behind ConvNets
1) Sparse interactions
2) Parameter sharing
3) Equivariant representations
4) Ability to process inputs of variable sizes
1) Sparse interactions

In a usual ANN, nodes are fully-connected

In CNN, sparse connections:

Figure 9.2 from Goodfellow et al. (2016).
Sparse interactions

1\textsuperscript{st} (input) layer: 4x4 image

2\textsuperscript{nd} layer

2x2 filter
Sparse interactions

1\textsuperscript{st} (input) layer: 4x4 image

Node in the 2\textsuperscript{nd} layer is not fully-connected to the nodes in the 1\textsuperscript{st} layer.
Sparse interactions

1\textsuperscript{st} (input) layer: 4x4 image

2\textsuperscript{nd} layer

\[\bullet\text{: computed}\]
Sparse interactions

1\textsuperscript{st} (input) layer: 4x4 image

2\textsuperscript{nd} layer

\[\textcolor{gray}{\textbullet} \text{: computed}\]
Sparse interactions

1\textsuperscript{st} (input) layer: 4x4 image

\[ \text{2\textsuperscript{nd} layer} \]

\[ \text{\textbullet: computed} \]
Sparse interactions

1\textsuperscript{st} (input) layer: 4x4 image

2\textsuperscript{nd} layer

\textbullet: computed
Sparse interactions

1\textsuperscript{st} (input) layer: 4x4 image

2\textsuperscript{nd} layer

\(\text{\textbullet}:\) computed
Sparse interactions

Complexity of fully-connected vs sparse:

\[ m: \# \text{ of nodes in the 1}^{\text{st}} \text{ layer} \]
\[ n: \# \text{ of nodes in the 2}^{\text{nd}} \text{ layer} \]
\[ k: \# \text{ of elements in the filter} \]

Fully-connected: \( O(mn) \)
Sparse: \( O(nk) \) where, typically, \( k \ll m \)
Fully Connected Layer

Example: 200x200 image
40K hidden units
~2B parameters!!!

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

[Slide by Marc'Aurelio Ranzato from his Deep Learning Tutorial at CVPR 2014 link]
Locally Connected Layer

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

[Slide by Marc'Aurelio Ranzato from his Deep Learning Tutorial at CVPR 2014 link]
2) Parameter Sharing

$1^{\text{st}}$ (input) layer: 4x4 image

$2^{\text{nd}}$ layer

Same kernel/filter (the red window) is applied everywhere on a layer.

# of total parameters to be learned and storage requirements dramatically reduced.

Note $m$ and $n$ are roughly the same, but $k$ is much less than $m$. 
Convolutional Layer

Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

[Slide by Marc’Aurelio Ranzato from his Deep Learning Tutorial at CVPR 2014 link]
3) Equivariance

This is a direct consequence of parameter sharing.

A function \( f \) is equivariant to function \( g \) if \( f(g(x)) = g(f(x)) \).

Convolution on images creates a 2D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output.

Useful when detecting structures that are common in the input. E.g. edges in an image. Equivariance in early layers is good.
4) Ability to process arbitrary sized inputs

Fully-connected networks accept fixed-size input vector.

How can we classify images of arbitrary sizes?

In ConvNets, we can use the “pooling” technique to summarize the input into a fixed-size vector/matrix.
After convolution...

the next operations: nonlinearity and pooling.

We have already seen many non-linear activation functions.

ReLU is the most widely used one.
Pooling

A pooling function takes the output of the previous layer at a certain location $L$ and computes a “summary” of the neighborhood around $L$.

E.g. max-pooling [Zhou and Chellappa (1988)]
Biological motivation for max-pooling

V1 simple cell

V1 complex cell implements max-pooling.
Max-pooling introduces invariance.

Input layer has shifted to the right 1-pixel.

But only half of the values in the output layer have changed.

Figure 9.8 from Goodfellow et al. (2016).
Pooling

“Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is.” [Goodfellow et al. (2016)]
Pooling Layer

Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?

[Slide by Marc’Aurelio Ranzato from his Deep Learning Tutorial at CVPR 2014 link]
Pooling Layer

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

[Slide by Marc'Aurelio Ranzato from his Deep Learning Tutorial at CVPR 2014 link]
Spatial pooling produces invariance to translation. Pooling over channels produces other invariances. E.g. Maxout networks by Goodfellow et al. (2013).

Figure 9.9 from Goodfellow et al. (2016).
Pooling summarizes.

We can make a sparse summary by using a stride larger than 1.

This reduces the computational complexity and memory requirements.

Figure 9.10 from Goodfellow et al. (2016).
Another use of pooling

You might want to classify images of arbitrary sizes.

This is a problem b/c (at least) the input layer to the classification layer must be of fixed size.

The pooling layer could summarize the image by pooling over its four quadrants.
Channel in a ConvNet

Suppose the first (input) layer takes a color image of size 200x200

Then, the input is a 3x200x200 matrix. 3 is the number of channels.

Suppose the second layer is 32x5x5. What do these numbers refer to?

What are the dimensions of the kernel in between?
Typical notation of a convolutional layer in a ConvNet

\[ Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n} \]

\( V \): input layer, \( Z \): output layer, \( K \): kernel
Connectivity types

So far, we have seen

fully-connected layers in MLPs and

sparse connections in ConvNets.

Other types of connectivities?
Local connection: like convolution, but no sharing

Convolution

Fully connected

Figure 9.14 from Goodfellow et al. (2016).
Zero-padding controls size

Without zero padding

With zero padding

Compare with the use of “valid”, “same” and “full” in numpy’s convolve or MATLAB’s conv2.
Now let’s see a real ConvNet from literature
AlexNet [Krizhevsky et al. NIPS 2012]

Processing in different GPUs are connected at only some layers.
**AlexNet** [Krizhevsky et al. NIPS 2012]

5 convolutional layers 3 full-connected layers
AlexNet [Krizhevsky et al. NIPS 2012]

Each convolutional layer consists of:
convolution + ReLU + normalization + max-pooling
Local response normalization

\[
 b^i_{x,y} = a^i_{x,y} / \left( k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}
\]

\[ a^i_{x,y} \text{ ReLU'd output of kernel } i \text{ at location } x,y \]

\[ b^i_{x,y} \text{ Normalized output of the same neuron } \]

The sum runs over \( n \) “adjacent” kernels/channels at the same spatial location.

“\textbf{The constants } k, n, \alpha, \text{ and } \beta \text{ are hyper-parameters whose values are determined using a validation set; we used } k = 2, n = 5, \alpha = 10^{-4}, \text{ and } \beta = 0.75.\”

Normally you wouldn’t need normalization with ReLU’d neurons but they discovered that it improves the results.
Architecture for Classification

Total nr. params: 60M

4M
LINEAR

16M
FULLY CONNECTED

37M
FULLY CONNECTED

MAX POOLING

442K
CONV

1.3M
CONV

884K
CONV

MAX POOLING

307K
LOCAL CONTRAST NORM

35K
LOCAL CONTRAST NORM

Total nr. flops: 832M

4M

16M

37M

74M

224M

149M

223M

105M

input

Krizhevsky et al. “ImageNet Classification with deep CNNs” NIPS 2012
[Slide by Marc’Aurelio Ranzato from his Deep Learning Tutorial at CVPR 2014 link]
Reducing overfitting in AlexNet

In the ImageNet challenge, there are 1.2 million training images. AlexNet has 60 million parameters.

Without the following, AlexNet suffers from substantial overfitting.

- Data augmentation
- Dropout
Data Augmentation 1

• Artificially enlarging the dataset using label preserving transformations.
  – Translations (extract random 224x224x3 images from the original 256x256x3)
  – Horizontal reflection

• Increases the size of the training set 2048 times.

• Test time: “the network makes a prediction by extracting five 224 × 224 patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network’s softmax layer on the ten patches.”
Data Augmentation 2

- Alter the intensities of the RGB channels in training images by adding a combination of the following:
  - Multiples of the principal components of the RGB distribution of training images.

- “This scheme approximately captures an important property of natural images, namely, that object identity is invariant to changes in the intensity and color of the illumination. This scheme reduces the top-1 error rate by over 1%.”
Dropout

“Combining the predictions of many different models is a very successful way to reduce test errors, but it is too expensive for big neural networks that already take several days to train.

Dropout:

- Set to zero the output of each hidden neuron with probability 0.5. The neurons which are “dropped out” in this way do not contribute to the forward pass and do not participate in backpropagation.
- We use dropout in the first two fully-connected layers.
- Dropout roughly doubles the number of iterations required to converge.”

Krizhevsky et al. (2012)
Figure 1: Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

[Figure from Srivastava et al. (2014)]
Dropout

“Every time an input is presented, the neural network samples a different architecture, but all these architectures share weights.

This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.”

Krizhevsky et al. (2012)
Dropout

“At test time, we use all the neurons but multiply their outputs by 0.5, which is a reasonable approximation to taking the geometric mean of the predictive distributions produced by the exponentially-many dropout networks.

We use dropout in the first two fully-connected layers.

Dropout roughly doubles the number of iterations required to converge”

Krizhevsky et al. (2012)
References


