Imagine processing this input image composed of 12,192,768 pixels in fully connected neural networks.
Overview

• Convolutional networks
• Regularization
• Reference
  – Deep Learning (2017), Ch. 9, Ch. 7
  Ian Goodfellow, Yoshua Bengio, and Aaron Courville, The MIT Press

http://www.deeplearningbook.org/
Convolutional networks [LeCun 1989] are neural networks for processing data shaped in grid-like topology that use convolution in place of matrix multiplication in some layers.
Convolution

Convolution is an operator between two functions $f$ and $g$ of a real-valued argument $t$, defining an integral of piece-wise multiplication of $f$ and $g$ as one of the functions, $g$, is shifted over the other function $f$

e.g., $f$ is noisy sensor reading; $g$ is a weighting function, $s(t) = f * g(t)$ is weighted average of sensor reading—higher weights for recent readings

$$s(t) = (f * g)(t) = \int_i f(i)g(t - i)di$$

shift over values of $t$
Convolution

Convolution is an operator between two functions $f$ and $g$ of a real-valued argument $t$, defining a sum of piece-wise multiplication of $f$ and $g$ as one of the functions, $g$, is shifted over the other function $f$.

e.g., $f$ is noisy sensor reading; $g$ is a weighting function, $s(t) = f * g(t)$ is weighted average of sensor reading—higher weights for recent readings.

$$s(t) = (f * g)(t) = \sum_{i=-\infty}^{\infty} f(i)g(t-i)$$
Convolution in CNNs

Convolution is an operator between input (data) tensor $I$ and kernel (parameters) tensor $K$ of multiple real-valued arguments, defining a sum of piece-wise multiplication of $I$ and $K$ as the kernel $K$ is shifted over input data $I$.

e.g., 2-D image input data

$$S(i, j) = (I * K)(i, j) = \sum_{m} \sum_{n} I(m, n)K(i - m, j - n)$$

shift over values of $i$ and $j$
Convolution is commutative

Convolution is an operator between input (data) tensor $I$ and kernel (parameters) tensor $K$ of multiple real-valued arguments, defining a sum of piece-wise multiplication of $I$ and $K$ as the kernel $K$ is shifted over input data $I$.

e.g., 2-D image input data

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

Kernel is flipped
Cross-correlation of convolution

Commonly also referred to as “convolution”

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

Kernel is not flipped
Example of 2-D convolution [Fig 9.2]

Piece-wise multiplication
Properties of convolution in neural networks

• Sparse interactions
• Parameter sharing
• Equivariant representations
Sparse interactions

- **m inputs**
- **n outputs**

Matrix multiplication: $O(m \times n)$

Convolution: $O(k \times n)$

Output nodes affected by input node $x_3$ [Fig. 9.2]
Sparse interactions

Matrix multiplication $O(m \times n)$

Input notes affecting output node $s_3$ [Fig. 9.3]
Sparse interactions

\[ m = 12,192,768 \]

Convolution
\[ O(k \times n) \]
\[ k = \sim 100s \]

Matrix multiplication
\[ O(m \times n) \]
Parameter sharing

“Tied” weights
= Used once
Parameter sharing

Learn weights once
= Used for every location

Reduce memory requirement for storing parameters

“Tied” weights
= Used once
Equivariant to translation

Function $f(x)$ is equivariant to function $g$ if:

$$f(g(x)) = g(f(x))$$

Convolution is equivariant to translation, i.e., if the input changes, the output changes the same way.
Equivariant to translation

Convolution $f$ is equivariant to translation function $g$ if:

$$f(g(x)) = g(f(x))$$

Convolution $g(x,y) = \text{input}(x-1,y)$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>70</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>80</td>
<td>10</td>
</tr>
<tr>
<td>60</td>
<td>50</td>
<td>80</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>70</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>80</td>
</tr>
<tr>
<td>0</td>
<td>60</td>
<td>50</td>
</tr>
</tbody>
</table>
Equivariant to translation

Convolution $f$ is equivariant to translation function $g$ if:

$$f(g(x)) = g(f(x))$$
Components of convolutional layer

Convolutional layer

Poolings stage

Detection stage:
Nonlinearity
 e.g., ReLU

Convolution stage:
Affine transformation

Input

Output

Pooling function: to refine the output
Pooling

- Max pooling [Zhou & Chellappa, 1988]: report the maximum output among a rectangular neighborhood
- Average pooling
- $L^2$ norm pooling
- Weighted average pooling
Pooling over spatial regions

Invariance to translation (small shift)

[Fig 9.8]
Pooling over features

Invariance to features, e.g., rotation

[fig 9.9]
Pooling also provides

- Invariance to translation or other transformations
- **Efficiency via downsampling**—i.e., fewer pooling units than detection units
- Support for **variable-size input** by varying the size of offset between pooling regions to meet fixed size output

![Diagram showing pooling units and offsets](Fig 9.10)
Convolution with a stride

[Fig 9.12]
Convolution with a stride

Strided convolution
Convolution with a stride

Stride of size 2
Zero padding

Convolution shrinks the network size

No zero padding, valid convolution

Full zero padding, same convolution
Learning features unsupervised

• Initialize randomly – works very well
• Design by hand, e.g., detect edges at certain orientations
• Learn using unsupervised methods, e.g., k-means [Coates et al., 2011]
Example: Image classification

AlexNet [Krizhevsky et al., 2012]

• ImageNet Large-Scale Visual Recognition Challenge (ILSVRC): 1.2M training, 50K validation, and 150K testing images
• Fixed-size input
• 5 Convolutional layers + 3 fully connected layers
• 2 GPUs
• 1000 classes image classification
• Reducing overfitting by:
  – Data augmentation (image translation, horizontal reflection, intensity alteration using PCA)
  – Dropout (zero out output with some probability, e.g., 0.5)
AlexNet

Input: 224x224x3 (RGB)

96 (48+48) 11x11x3 kernels

Max pooling

Stride of 4

[Fig 3]

[Softmax over 1000 classes]

[Krizhevsky et al., 2012]
Semantic segmentation

Example: pixel-level classification

[Joint work with Xavier Perez, 2017]

The material is based on the work sponsored by the U.S. Air Force Office of Scientific Research under award number FA2386-17-1-4660 using the ISPRS Potsdam data set: www.isprs.org and Cityscapes: https://www.cityscapes-dataset.com/.

Jean Oh, Carnegie Mellon University (jeanoh@cmu.edu)
Application: self-driving cars

[Joint work with Xavier Perez, 2018]
Application: self-driving cars

[Joint work with Xavier Perez, 2018]
Regularization

Any modification we make to a learning algorithm to reduce “generalization error” as opposed to “training error”
Parameter norm penalties

- $L^2$ norm regularization (aka Tikhonov regularization or ridge regression)

$$\Omega(\theta) = \frac{1}{2} \| w \|_2^2$$

- $L^1$ norm regularization

$$\Omega(\theta) = \| w \|_2 = \sum_{i} |w_i|$$
Data augmentation

• Train on more data
• Invariant to various transformations
• Effective in image classification, object recognition
• But not for all problem domains
• Examples
  – Rotation
  – Horizontal, vertical flip
  – Translation
Early stopping

During training, keep track of the parameters that give the lowest validation error, return them at the end.

While training error is decreased, validation error may go up due to overfitting.

[Fig 7.3]
Batch normalization

[ioffe & Szegedy, 2015]

• Internal covariance shift: the distribution of input to each layer changes as the parameters of previous layer changes
• Normalize layer inputs for each mini-batch
• Advantages:
  – higher learning rate, faster training
  – No dropout needed
  – Accepted as part of common architecture
Other methods for regularization

- Noise injection, label smoothing
- Parameter sharing
  - CNN
- Bootstrapping aggregating (aka bagging)
  - ensemble or model averaging
- Dropout [Srivastava et al., 2014]
  - Ignore random output with some probability, meaning not used for learning at all
  - Popular earlier but not so much currently