Learn a method in 5 minutes:
Extreme Learning Machine

**Hypothesis:**
Randomly initialized layers with non-linearities are a pool of useful non-linear functions from which regression & classification can be performed with a linear layer.

**Method:**
1. Create a set of layers with random weights.
2. **Freeze those weights.**
3. Add only a linear layer on top.
4. Learn the parameters of only the top layer.
Disadvantages of MLPs:

Curse of Dimensionality

- Number of required samples for obtaining small error increases exponentially with input dimensions
- Too many many parameters

Disadvantages of MLPs: **Equivariance**

Vectorizing an image breaks patterns in consecutive pixels.
- Shifting one pixel means a whole new vector
- Makes learning more difficult
- Requires more data to generalize
Equivariance vs. Invariance

• Equivariant problem: image segmentation.
  • $f(g(x)) = g(f(x))$

• Invariant problem: object recognition.
  • $f(g(x)) = f(x)$

• Pooling provides invariance, convolution provides equivariance.

https://www.mathworks.com/discovery/image-segmentation.html
An Alternative to MLPs

Solution (inspiration):

- Hubel & Wiesel: Brain neurons are not fully connected. They have local receptive fields.

\[ \text{Neural response (spikes/sec)} \]

\[ \text{Stimulus orientation (deg)} \]

Model of Striate Module in Cats

http://fourier.eng.hmc.edu/e180/lectures/retina/node1.html
An Alternative to MLPs

Solution: Neocognitron (Fukushima, 1979):

A neural network model unaffected by shift in position, applied to Japanese handwritten character recognition.

- **S** (simple) cells: local feature extraction.
- **C** (complex) cells: provide tolerance to deformation, e.g. shift.
- Self-organized learning method.

Figure: Fukushima (2019), Recent advances in the deep CNN neocognitron.
An Alternative to MLPs

Solution: Convolutional Neural Networks (Lecun, 1998)

- Gradient descent
- Weights shared
- Document recognition

Lecun, 1998
CNNs vs. MLPs: Curse of Dimensionality

- A fully-connected network has too many parameters
  - On CIFAR-10:
    - Images have size 32x32x3 \(\Rightarrow\) one neuron in hidden layer has 3072 weights!
    - With images of size 1024x1024x3 \(\Rightarrow\) one neuron in hidden layer has 3,145,728 weights!
    - This explodes quickly if you increase the number of neurons & layers.
  - Alternative: enforce local connectivity!

CNNs vs. MLPs: **Equivariance**

- Equivariant to translation
  - The output will be the same, just translated, since the weights are shared.

- Not equivariant to scale or rotation.

Figure: https://towardsdatascience.com/translational-invariance-vs-translational-equivariance-f9bc8fca63a
Unit Sample Response

- Now suppose the system is LTI, and define the unit sample response $h[n]$

$$\delta[n] \rightarrow h[n]$$

From Time-Invariance:

$$\delta[n - k] \rightarrow h[n - k]$$

From Linearity:

$$x[n] = \sum_{k=-\infty}^{\infty} x[k] \delta[n - k] \rightarrow y[n] = \sum_{k=-\infty}^{\infty} x[k] h[n - k] = x[n] * h[n]$$

Alan V. Oppenheim and Alan S. Willsky
The output of any DT LTI System is a convolution of the input signal with the unit-sample response, i.e.

\[
\text{Any DT LTI} \quad \iff \quad y[n] = x[n] * h[n] = \sum_{k=-\infty}^{+\infty} x[k] h[n-k]
\]

As a result, any DT LTI Systems are completely characterized by its unit sample response.

Alan V. Oppenheim and Alan S. Willsky
Example multi-dimensional convolution (kernel: finite impulse response)

Input

\[
\begin{array}{cccc}
  a & b & c & d \\
  e & f & g & h \\
  i & j & k & l \\
\end{array}
\]

Kernel

\[
\begin{array}{cc}
  w & x \\
  y & z \\
\end{array}
\]

Output

\[
\begin{array}{cccc}
  aw + bx + ey + fz \\
  bw + cx + fy + gz \\
  ew + fx + iy + jz \\
  fw + gx + jy + kz \\
  cw + dx + gy + hz \\
  gw + hx + ky + lz \\
\end{array}
\]

https://github.com/vdumoulin/conv_arithmetic


Spring 2021
CNN layers

- Stages of CNN:
  - Convolution (in parallel) to produce pre-synaptic activations
  - Detector: Non-linear function
  - Pooling: A summary of a neighborhood

- Pooling of a rectangular region:
  - Max
  - Average
  - L2 norm
  - Weighted average acc. to the distance to the center
  - ...

Previously on CENG501!

Regular ANN

CNN

http://cs231n.github.io/convolutional-networks/
Connectivity in CNN

Local: The behavior of a neuron does not change other than being restricted to a subspace of the input.

- Each neuron is connected to slice of the previous layer
- A layer is actually a volume having a certain width x height and depth (or channel)
- A neuron is connected to a subspace of width x height but to all channels (depth)
- Example: CIFAR-10
  - Input: 32 x 32 x 3 (3 for RGB channels)
  - A neuron in the next layer with receptive field size 5x5 has input from a volume of 5x5x3.
Important parameters

- **Depth (number of channels)**
  - We will have more neurons getting input from the same receptive field
  - This is similar to the hidden neurons with connections to the same input
  - These neurons learn to become selective to the presence of different signals in the same receptive field

- **Stride**
  - The amount of space between neighboring receptive fields
  - If it is small, RFs overlap more
  - If it is big, RFs overlap less

- **How to handle the boundaries?**
  1. Option 1: Don’t process the boundaries. Only process pixels on which convolution window can be placed fully.
  2. Option 2: Zero-pad the input so that convolution can be performed at the boundary pixels.

Weights:

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

http://cs231n.github.io/convolutional-networks/
Padding illustration

- Only convolution layers are shown.
- Top: no padding → layers shrink in size.
- Bottom: zero padding → layers keep their size fixed.

Figure 9.11: The effect of zero padding on network size. Consider a convolutional network with a kernel of width six at every layer. In this example, do not use any pooling, so only the convolution operation itself shrinks the network size. Top) In this convolutional network, we do not use any implicit zero padding. This causes the representation to shrink by five pixels at each layer. Starting from an input of sixteen pixels, we are only able to have three convolutional layers, and the last layer does not even move the kernel, so arguably only two of the layers are truly convolutional. The rate of shrinking can be mitigated by using smaller kernels, but smaller kernels are less expressive and some shrinking is inevitable in this kind of architecture. Bottom) By adding five implicit zeroes to each layer, we prevent the representation from shrinking with depth. This allows us to make an arbitrarily deep convolutional network.

Size of the next layer

• Along a dimension:
  • $W$: Size of the input
  • $F$: Size of the receptive field
  • $S$: Stride
  • $P$: Amount of zero-padding

• Then: the number of neurons as the output of a convolution layer:

$$\frac{W - F + 2P}{S} + 1$$

• If this number is not an integer, your strides are incorrect and your neurons cannot tile nicely to cover the input volume

Previously on CENG501!
Today

- Convolutional Neural Networks (CNNs)
  - Types of convolution in CNNs
  - Pooling
  - Non-linearity
  - Normalization
  - FC layer & alternatives
  - A blueprint for CNN architectures
  - Backpropagation
Administrative Issues

• Project paper selection (15 April)
  https://docs.google.com/spreadsheets/d/1u_4cYR5OFpJlwd8IXpH_SVLQuOh9i-Fgm6prz7eRdT/edit?usp=sharing

• Programming Assignment 1 (PA1):
  • Deadline: 17 April.

• Midterm exam:
  • Deadline: 24 April 23:59.
Types of Convolution: Unshared convolution

- In some cases, sharing the weights does not make sense
  - When?

- Different parts of the input might require different types of processing/features

- In such a case, we just have a network with local connectivity

- E.g., a face.
  - Features are not repeated across the space.
Types of Convolution:
Dilated (Atrous) Convolution

**Purpose:** Increase effective receptive field size without increasing parameters.

Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) $F_1$ is produced from $F_0$ by a 1-dilated convolution; each element in $F_1$ has a receptive field of $3 \times 3$. (b) $F_2$ is produced from $F_1$ by a 2-dilated convolution; each element in $F_2$ has a receptive field of $7 \times 7$. (c) $F_3$ is produced from $F_2$ by a 4-dilated convolution; each element in $F_3$ has a receptive field of $15 \times 15$. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

https://github.com/vdumoulin/conv_arithmetic
Types of Convolution: Transposed Convolution

**Purpose:** Increasing layer width + height (upsampling).

The size of the output:
- Regular convolution: \( O = \frac{W - F + 2 \times P}{S} + 1 \)
- Transpose convolution: \( W = (O - 1) \times S + F - 2 \times P \)

Figure: [https://d2l.ai/chapter_computer-vision/transposed-conv.html](https://d2l.ai/chapter_computer-vision/transposed-conv.html)
Types of Convolution: Transposed Convolution

https://github.com/vdumoulin/conv_arithmetic
Types of Convolution: 3D Convolution

Purpose: Work with 3D data, e.g. learn spatial + temporal representations for videos.

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution: 1x1 Convolution

**Purpose:** Reduce number of channels.

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution: Separable Convolution

Purpose: Reduce number of parameters and multiplications.

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

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution:
Depth-wise Separable Convolution

Purpose: Reduce number of parameters and multiplications.

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution: Group Convolution

AlexNet (Krizhevsky et al.)
Types of Convolution:

Group Convolution

**Purpose:** Reduce number of parameters and multiplications.

Normal Convolution

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution: Group Convolution

• Benefits:
  • Efficiency in training (distribute groups to different GPUs)
  • Decrease in # of parameters as the # of groups increases
  • Better performance?

Figure: https://blog.yani.ai/filter-group-tutorial/

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
Types of Convolution: Deformable Convolution


Purpose: Flexible receptive field.

Figure 1: Illustration of the sampling locations in $3 \times 3$ standard and deformable convolutions. (a) regular sampling grid (green points) of standard convolution. (b) deformed sampling locations (dark blue points) with augmented offsets (light blue arrows) in deformable convolution. (c)(d) are special cases of (b), showing that the deformable convolution generalizes various transformations for scale, (anisotropic) aspect ratio and rotation.

Figure 2: Illustration of $3 \times 3$ deformable convolution.
Types of Convolution:
Deformable Convolution


Bilinear interpolation for $x(p)$.

$$\mathcal{R} = \{(−1,−1),(−1,0),\ldots,(0,1),(1,1)\}$$
defines a $3 \times 3$ kernel with dilation 1.

For each location $p_0$ on the output feature map $y$, we have

$$y(p_0) = \sum_{p_n \in \mathcal{R}} w(p_n) \cdot x(p_0 + p_n), \quad (1)$$

where $p_n$ enumerates the locations in $\mathcal{R}$.

In deformable convolution, the regular grid $\mathcal{R}$ is augmented with offsets $\{\Delta p_n|n = 1, \ldots, N\}$, where $N = |\mathcal{R}|$. Eq. (1) becomes

$$y(p_0) = \sum_{p_n \in \mathcal{R}} w(p_n) \cdot x(p_0 + p_n + \Delta p_n). \quad (2)$$

Now, the sampling is on the irregular and offset locations $p_n + \Delta p_n$. As the offset $\Delta p_n$ is typically fractional, Eq. (2) is implemented via bilinear interpolation as

$$x(p) = \sum_{q} G(q, p) \cdot x(q), \quad (3)$$

where $p$ denotes an arbitrary (fractional) location $(p = p_0 + p_n + \Delta p_n$ for Eq. (2)), $q$ enumerates all integral spatial locations in the feature map $x$, and $G(\cdot, \cdot)$ is the bilinear interpolation kernel. Note that $G$ is two dimensional. It is separated into two one dimensional kernels as

$$G(q, p) = g(q_x, p_x) \cdot g(q_y, p_y), \quad (4)$$

where $g(a,b) = \max(0, 1 - |a - b|)$. Eq. (3) is fast to compute as $G(q, p)$ is non-zero only for a few $q$s.

In the deformable convolution Eq. (2), the gradient w.r.t. the offset $\Delta p_n$ is computed as

$$\frac{\partial y(p_0)}{\partial \Delta p_n} = \sum_{p_n \in \mathcal{R}} w(p_n) \cdot \frac{\partial x(p_0 + p_n + \Delta p_n)}{\partial \Delta p_n}$$

$$= \sum_{p_n \in \mathcal{R}} \left[w(p_n) \cdot \sum_{q} \frac{\partial G(q,p_0 + p_n + \Delta p_n)}{\partial \Delta p_n} x(q)\right], \quad (7)$$

where the term $\frac{\partial G(q,p_0 + p_n + \Delta p_n)}{\partial \Delta p_n}$ can be derived from Eq. (4). Note that the offset $\Delta p_n$ is 2D and we use $\partial \Delta p_n$ to denote $\partial \Delta p_n^0$ and $\partial \Delta p_n^0$ for simplicity.
Types of Convolution: Position-sensitive convolution

- Learn to use position information when necessary
Convolution demos & tutorials

• https://github.com/vdumoulin/conv_arithmetic

• http://cs231n.github.io/assets/conv-demo/index.html

• https://ezyang.github.io/convolution-visualizer/index.html

• https://ikhlestov.github.io/pages/machine-learning/convolutions-types/

• https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
OPERATIONS IN A CNN:
Pooling
Pooling

• Apply an operation on the “detector” results to combine or to summarize the answers of a set of units.
  • Applied to each channel (depth slice) independently
  • The operation has to be differentiable of course.

• Alternatives:
  • Maximum
  • Sum
  • Average
  • Weighted average with distance from the value of the center pixel
  • L2 norm
  • Second-order statistics?
  • …

• Different problems may perform better with different pooling methods

• Pooling can be overlapping or non-overlapping

Remember the motivation for CNNs:
S (simple) cells: local feature extraction.
C (complex) cells: provide tolerance to deformation, e.g. shift.


http://cs231n.github.io/convolutional-networks/
Pooling

- Example
  - Pooling layer with filters of size 2x2
  - With stride = 2
  - Discards 75% of the activations
  - Depth dimension remains unchanged

- Max pooling with F=3, S=2 or F=2, S=2 are quite common.
  - Pooling with bigger receptive field sizes can be destructive

- Avg pooling is an obsolete choice. Max pooling is shown to work better in practice.
Pooling

• Pooling provides invariance to small translation.

• If you pool over different convolution operators, you can gain invariance to different transformations.

Pooling can downsample

• Especially needed when to produce an output with fixed-length on varying length input.

- If you want to use the network on images of varying size, you can arrange this with pooling (with the help of convolutional layers)

CNNs without pooling

• “Striving for Simplicity: The All Convolutional Net proposes to discard the pooling layer in favor of architecture that only consists of repeated CONV layers. To reduce the size of the representation they suggest using larger stride in CONV layer once in a while.”

http://cs231n.github.io/convolutional-networks/

<table>
<thead>
<tr>
<th>Model</th>
<th>Error (%)</th>
<th># parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>without data augmentation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model A</td>
<td>12.47%</td>
<td>≈ 0.9 M</td>
</tr>
<tr>
<td>Strided-CNN-A</td>
<td>13.46%</td>
<td>≈ 0.9 M</td>
</tr>
<tr>
<td>ConvPool-CNN-A</td>
<td>10.21%</td>
<td>≈ 1.28 M</td>
</tr>
<tr>
<td>ALL-CNN-A</td>
<td>10.30%</td>
<td>≈ 1.28 M</td>
</tr>
<tr>
<td>Model B</td>
<td>10.20%</td>
<td>≈ 1 M</td>
</tr>
<tr>
<td>Strided-CNN-B</td>
<td>10.98%</td>
<td>≈ 1 M</td>
</tr>
<tr>
<td>ConvPool-CNN-B</td>
<td>9.33%</td>
<td>≈ 1.35 M</td>
</tr>
<tr>
<td>ALL-CNN-B</td>
<td>9.10%</td>
<td>≈ 1.35 M</td>
</tr>
<tr>
<td>Model C</td>
<td>9.74%</td>
<td>≈ 1.3 M</td>
</tr>
<tr>
<td>Strided-CNN-C</td>
<td>10.19%</td>
<td>≈ 1.3 M</td>
</tr>
<tr>
<td>ConvPool-CNN-C</td>
<td>9.31%</td>
<td>≈ 1.4 M</td>
</tr>
<tr>
<td>ALL-CNN-C</td>
<td>9.08%</td>
<td>≈ 1.4 M</td>
</tr>
</tbody>
</table>

(ALL-CNN: No pooling)

Summary: Convolution & pooling

• Provide strong bias on the model and the solution
• They directly affect the overall performance of the system
OPERATIONS IN A CNN:
nonlinearity
Non-linearity

- Sigmoid
- Tanh
- ReLU and its variants
  - The common choice
  - Faster
  - Easier (in backpropagation etc.)
  - Avoids saturation issues
- ...
OPERATIONS IN A CNN:

normalization
• From Krizhevsky et al. (2012):

generalization. Denoting by $a_{x,y}^i$ the activity of a neuron computed by applying kernel $i$ at position $(x, y)$ and then applying the ReLU nonlinearity, the response-normalized activity $b_{x,y}^i$ is given by the expression

$$b_{x,y}^i = a_{x,y}^i / \left( k + \alpha \sum_{j=\max(0, i-n/2)}^{\min(N-1, i+n/2)} (a_{x,y}^j)^2 \right)^{\beta}$$

where the sum runs over $n$ “adjacent” kernel maps at the same spatial position, and $N$ is the total number of kernels in the layer. The ordering of the kernel maps is of course arbitrary and determined before training begins. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels. The constants $k$, $n$, $\alpha$, and $\beta$ are hyper-parameters whose values are determined using a validation set; we used $k = 2$, $n = 5$, $\alpha = 10^{-4}$, and $\beta = 0.75$. We
Normalization

For each channel independently.

https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffe7
OPERATIONS IN A CNN:
fully connected layer
Fully-connected layer

- At the top of the network for mapping the feature responses to output labels
- Full connectivity
- Can be many layers
- Various activation functions can be used
Alternative to FC: Global Average Pooling

Alternative to FC: Global Average Pooling

• We have $n$ feature maps:
  $f_1, \ldots, f_n$.

• Global average pooling is then:
  $$\bar{f}_i = \sum_{x,y} f_i(x, y)$$

• Classification scores are obtained by:
  $$S_c = \sum_i w_i^c \bar{f}_i$$


• Advantages:
  – No parameters, hence significant improvement in terms of overfitting problem.
  – Forces the feature maps to capture confidence maps.
  – It is more suitable to the nature of CNNs.
  – Provides invariance to spatial transformations.