ECE 417 Fall 2018 Lecture 18: ConvNets

Mark Hasegawa-Johnson

University of Illinois

October 25, 2018

Outline

1. Matched Filters
2. The Feature Design Problem in Computer Vision
3. Convolutional Neural Networks
4. Training a Convolutional Neural Network using Pooled Back-Propagation

The Matched Filter Problem

Suppose

- $v[n] \sim \mathcal{N}(0,1)$ is Gaussian white noise.
- There are two possible hypotheses:
  - $H_0 : x[n] = v[n]$, or
  - $H_1 : x[n] = s[n] + v[n]$, for some known signal $s[n]$.
- Your task: find out if H_1 or H_0 is true.

Solution of the Matched Filter Problem

$$h[n] = s[-n], \quad \text{the ``matched filter''}$$

$$y[n] = h[n] * x[n] = \sum_{m=-\infty}^{\infty} x[m]s[m-n] = r_{xs}[n]$$

...then it can be shown that...

$$y[0] = \begin{cases} \|s\|^2 + v & H_1 \\ v & H_0 \end{cases}$$

where $v \sim \mathcal{N}(0, \|s\|^2)$ and $\|s\|^2 = \sum_n s^2[n]$.

Solution of the Matched Filter Problem

So the Bayes-optimal classifier chooses some threshold (maybe $\|s\|^2/2$), and does this:

$$x[n] \rightarrow \boxed{h[n] = s[-n]} \rightarrow y[n] \rightarrow \begin{cases} y[0] > \text{threshold}: & H_1 \\ y[0] < \text{threshold}: & H_0 \end{cases}$$

Why it works:

- Convolving with $s[-n]$ is just like correlating with $s[n]$.
- The signal that correlates most strongly with $s[n]$ is $s[n]$.

Outline

The Feature Design Problem in Computer Vision

PROBLEM: Is there a bicycle in this image?

The Feature Design Problem in Computer Vision

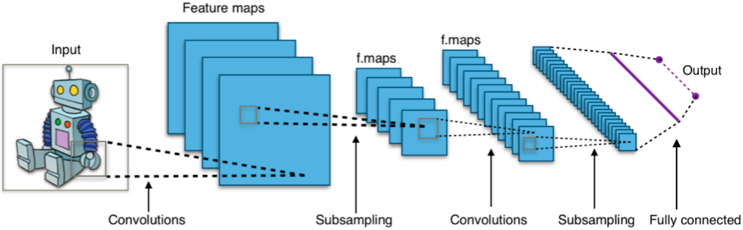
SOLUTION as of 2001 (Burl, Weber and Perona): (1) Use matched filters to find recognizable parts, e.g., handlebars, wheels, (2) If they occur in plausible geometry, call it a bicycle.

The Feature Design Problem in Computer Vision

WHY THE 2001 SOLUTION FAILS TO SCALE: How can you design matched filters for all of the parts of every type of object that you want to recognize?

Outline

ConvNets: Key Idea

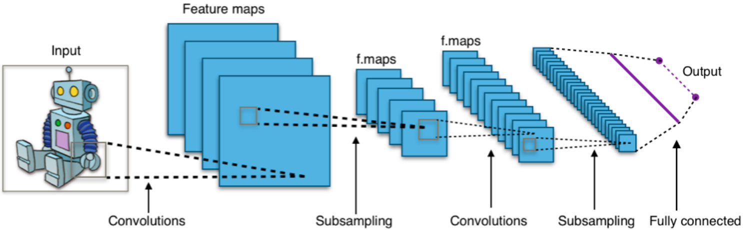


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org

KEY IDEA:

- Convolutional layers learn the features,
- Output layer learns a linear classifier.

ConvNets: Input

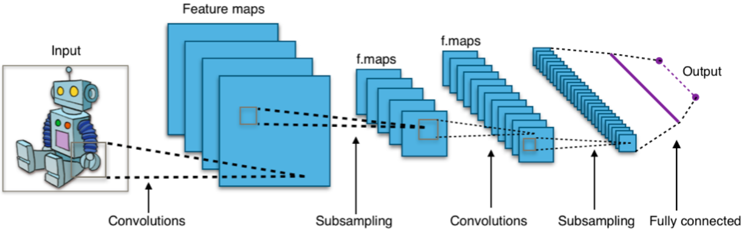


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org

Input to the ConvNet

$x[n_1, n_2, j]$ = image pixel in row n_1, column n_2, color j.

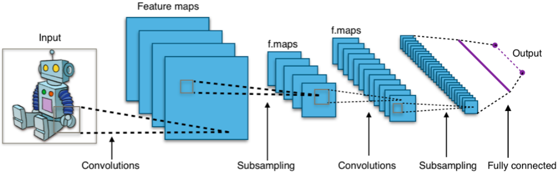
ConvNets: Hidden Layer



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org

Hidden Layers in a ConvNet

- CONVOLUTIONAL LAYER: $a[n_1, n_2, k]$ = conv layer, pixel n_1, n_2, channel k.
- POOLING LAYER: $y[n_1, n_2, k]$ = pooling layer, pixel n_1, n_2, channel k.



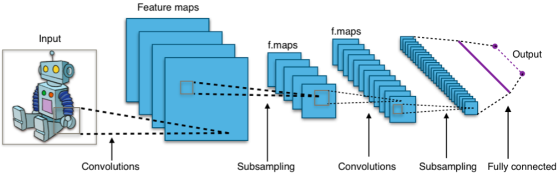
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org

ConvNets: Convolutional Layer

$$a[n_1, n_2, k] = u[n_1, n_2, j, k] * x[n_1, n_2, j]$$

- $u[:, :, :, k]$ is the k^{th} filter
- $a[:, :, k]$ is the k^{th} channel
- The per-channel 2D convolution is defined as:

$$u[n_1, n_2, j, k] * x[n_1, n_2, j] \equiv \sum_j \sum_{m_1} \sum_{m_2} u[n_1 - m_1, n_2 - m_2, j, k] x[m_1, m_2, j]$$



ConvNets: Max-Pooling Layer

$$y[n_1, n_2, k] = \max_{(m_1,m_2)\in\mathcal{A}(n_1,n_2)} \max(0, a[m_1, m_2, k])$$

- M is the max-pooling stride
- Finds the "maximum activation" of the k^{th} filter within the $(n_1, n_2)^{\text{th}}$ receptive field, which is defined as:

$$\mathcal{A}(n_1, n_2) = \left\{ \begin{array}{l} (m_1, m_2) : \\ n_1 M \le m_1 < (n_1 + 1)M, \\ n_2 M \le m_2 < (n_2 + 1)M \end{array} \right\}$$

ConvNets: Output Layer

We can "vectorize" $y[n_1, n_2, k]$ by just re-shaping it into a vector $\vec{y}$. For example, if the size of the image is $N_1 \times N_2 \times K$, then we could define $\vec{y}$ as

$$y_{kN_1N_2+n_1N_2+n_2} = y[n_1, n_2, k]$$

Then the output layer is the classifier:

$$\vec{z} = \text{softmax}(\vec{b}) = \text{softmax}(V\vec{y})$$

...and then we define error just as in any other neural net, e.g., cross-entropy, or mean-squared error.

Outline

Training the Output Layer

The output layer is trained just like in a regular neural net. For training token $\vec{x}_i$, you first find $\vec{a}_i$, then $\vec{y}_i$, then $\vec{b}_i$, then $\vec{z}_i$, then E_i, then

$$\epsilon_{\ell i} = \frac{\partial E_i}{\partial b_{\ell i}}$$

$$\frac{\partial E_i}{\partial V} = \vec{\epsilon}_i \vec{y}_i^T$$

$$V \leftarrow V - \frac{\eta}{n} \sum_{i=1}^{n} \vec{\epsilon}_i \vec{y}_i^T$$

Training the Convolutional Layer

In order to train $u[m_1, m_2, j, k]$, we need to back-propagate the error from the output layer ($\epsilon_{\ell i}$) to the convolutional layer:

$$\frac{\partial E_i}{\partial u[m_1, m_2, j, k]}$$

Training the Convolutional Layer: Chain Rule

Chain rule:

$$\frac{\partial E_i}{\partial u[m_1, m_2, j, k]} = \sum_{n_1} \sum_{n_2} \left(\frac{\partial E_i}{\partial a_i[n_1, n_2, k]} \right) \left(\frac{\partial a_i[n_1, n_2, k]}{\partial u[m_1, m_2, j, k]} \right)$$

Training the Convolutional Layer: Forward-Prop

Chain rule:

$$\frac{\partial E_i}{\partial u[m_1,m_2,j,k]} = \sum_{n_1}\sum_{n_2}\left(\frac{\partial E_i}{\partial a_i[n_1,n_2,k]}\right)\left(\frac{\partial a_i[n_1,n_2,k]}{\partial u[m_1,m_2,j,k]}\right)$$

First, this part:

$$a_i[n_1,n_2,k] = \sum_{m_1}\sum_{m_2} u[m_1,m_2,j,k]x_i[n_1-m_1,n_2-m_2,j]$$

...so...

$$\frac{\partial a_i[n_1,n_2,k]}{\partial u[m_1,m_2,j,k]} = x_i[n_1-m_1,n_2-m_2,j]$$

Training the Convolutional Layer: Back-Prop

Chain rule:

$$\begin{aligned}\frac{\partial E_i}{\partial u[m_1,m_2,j,k]} &= \sum_{n_1}\sum_{n_2}\left(\frac{\partial E_i}{\partial a_i[n_1,n_2,k]}\right)\left(\frac{\partial a_i[n_1,n_2,k]}{\partial u[m_1,m_2,j,k]}\right)\\ &= \sum_{n_1}\sum_{n_2}\delta_i[n_1,n_2,k]x_i[n_1-m_1,n_2-m_2,j]\\ &= \delta_i[m_1,m_2,k]*x_i[m_1,m_2,j]\end{aligned}$$

Where we've now defined the back-prop error term as:

$$\delta_i[n_1,n_2,k]=\frac{\partial E_i}{\partial a_i[n_1,n_2,k]}$$

Training the Convolutional Layer: Back-Prop

$$
\begin{aligned}
\delta_i[n_1,n_2,k] &= \frac{\partial E_i}{\partial a_i[n_1,n_2,k]} \\
&= \sum_\ell \sum_{o_1} \sum_{o_2} \left(\frac{\partial E_i}{\partial b_{\ell i}}\right)\left(\frac{\partial b_{\ell i}}{\partial y_i[o_1,o_2,k]}\right)\left(\frac{\partial y_i[o_1,o_2,k]}{\partial a_i[n_1,n_2,k]}\right) \\
&= \begin{cases} \sum_\ell \epsilon_{\ell i} v_{\ell,kN_1N_2+o_1N_2+o_2} \\ \quad \text{if } (n_1,n_2) = \text{argmax}_{(p_1,p_2)\in\mathcal{A}(o_1,o_2)} a_i[p_1,p_2,k] \\ 0 \\ \quad \text{otherwise} \end{cases}
\end{aligned}
$$

That last condition just says that we back-prop only to the hidden nodes $a[n_1,n_2,k]$ that survive max-pooling, not to any others. And to make the notation easier to remember, we can write

$$
\vec{\delta}_i = V^T \vec{\epsilon}_i|_{(n_1,n_2)}
$$

Training a ConvNet: Putting it all together

$$\frac{\partial E_i}{\partial V} = \vec{\epsilon}_i \vec{y}_i^T$$

$$\frac{\partial E_i}{\partial u[m_1, m_2, j, k]} = \delta_i[m_1, m_2, k] * x_i[m_1, m_2, j]$$

. . . where . . .

$$\epsilon_{\ell i} = \frac{\partial E_i}{\partial b_{\ell i}}$$

$$\delta_i[n_1, n_2, k] = V^T \vec{\epsilon}_i |_{(n_1, n_2)}$$