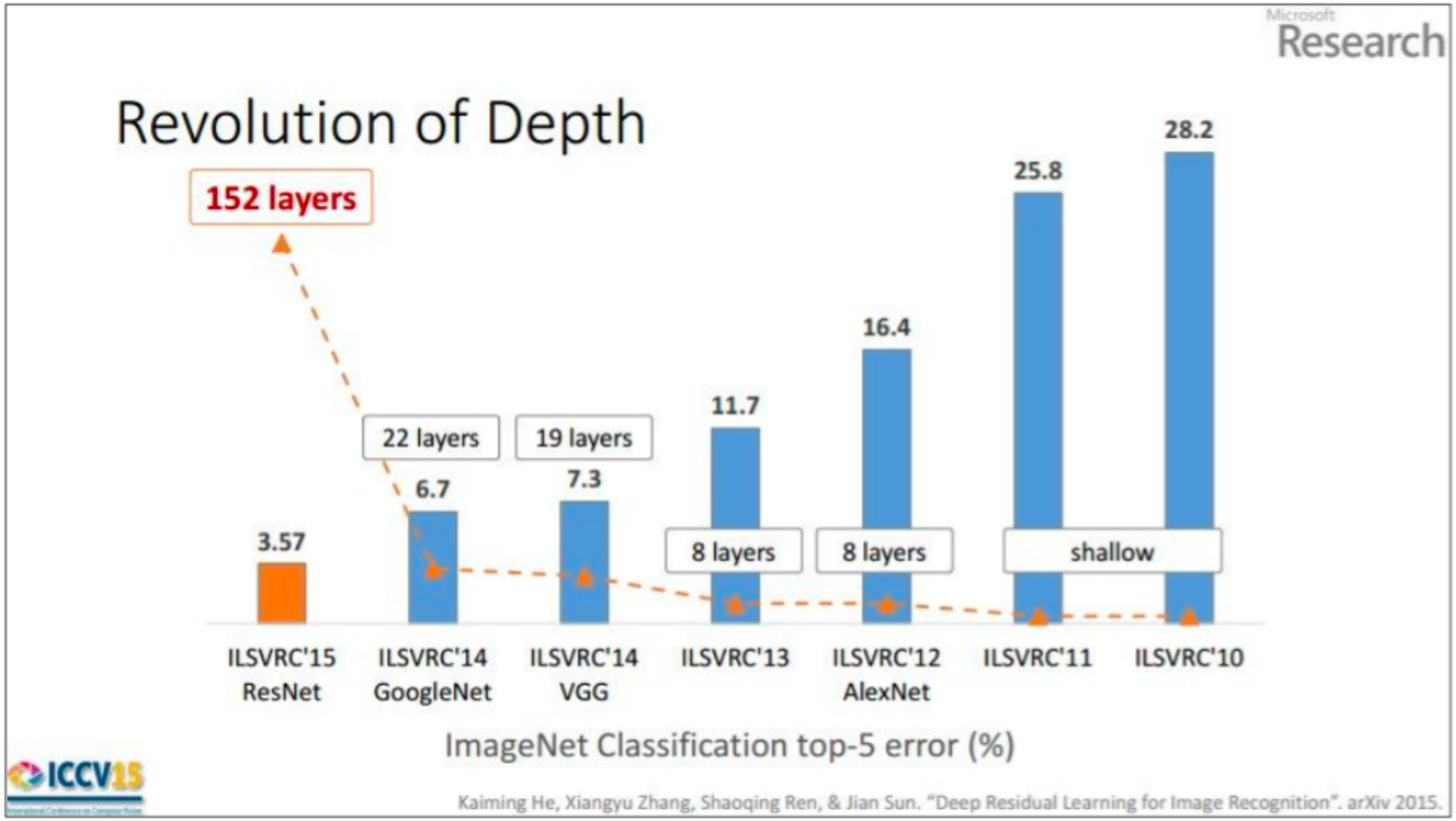
TTIC 31230, Fundamentals of Deep Learning

David McAllester, April 2017

Convolutional Neural Networks — CNNs

Imagenet Classification

1000 kinds of objects.



(slide from Kaiming He's recent presentation)


nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last — a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO

Speech Recognition

Current state of the art speech recognition systems use CNNs (as well as RNNs).

Computer transcription of conversational speech now matches the error rate of professional human transcribers.

(It should be noted that RNNs are also are being used in vision).

Protein Folding

Very Recent work of Jinbo Xu (here at TTIC) has used CNNs to fold proteins.

Construct an image $I_{x,y}$ where x and y are positions in the sequence of protein P and $I_{x,y}$ is the mutual information between variants at position x and variants at position y in the different version of P in different species.

For proteins with known structure, construct a target image $D_{x,y}$ where $D_{x,y}$ is the physical distance between the residue at position x and the residue at position y.

Train a CNN (resnet) to produce output $D_{x,y}$ from input image $I_{x,y}$.

The result is a revolution in predicting three dimensional structure from a protein sequence.

Convolution

In Deep learning we use the following definition of (1D) convolution.

$$(x * f)(t) = \sum_{j=0}^{|f|-1} x[t+j]f[j]$$

This version has the spatial dimensions of the filter reversed relative to the classical definition.

The classical definition yields $f * g = g * f$ which does not hold for the deep learning definition.

2D Convnets

$$\text{1D:} \quad \text{Conv}(s, f)[t] = \sum_j s[t+j] f[j]$$

$$\text{2D:} \quad \text{Conv}(I, f)[x, y] = \sum_{j,k} I[x+j, y+k] f[j, k]$$

We will write equations for 1D convnets.

For 2D one just replaces the time dimension with two spatial dimensions in both the signal and the filter.

Padding

$$\text{Conv}(x, f)[i] = \sum_{j=0}^{|f|-1} x[t+j]f[j]$$

$$|\text{Conv}(f, x)| = |x| - |f| + 1$$

We typically want the filter to slide off the edge of the signal to some extent.

This is done by padding the signal with zeros.

$$\text{Pad}(x, 2) = (0, 0, x[0], x[1], \ldots, x[|x| - 1], 0, 0)$$

Padding in Numpy

```
class Pad
    def __init__(self, x, p):
        ...

    def forward(self):
        x = self.x
        p = self.p
        s = x.size
        self.value = np.zeros(s+2*p)
        self.value[p:p+size] = x.value
```

Incorporating Padding into a Convolution Layer

Typically $|f|$ is odd and we do

$$\text{Conv}(\text{Pad}(x, \lfloor |f|/2 \rfloor), f)$$

For convenience we could define a procedure

$$\text{Convp}(x, f) \equiv \text{Conv}(\text{Pad}(x, \ |f|/2), f)$$

Or, for efficiency, we could implement Convp directly as a class.

Channels

In speech or audition one typically is given a channel vector, such as the Mel-cepstral coefficients, at each time position of the input.

The convolution operation also produces a channel vector at each time position.

In this case the filter has shape (T, C_2, C_1) and we have

$$\text{Conv}(x, f)[t, c] = \sum_{j,c'} x[t + j, c']f[j, c', c]$$

Note that the time dimension is handled as before.

Padding can be generalized straightforwardly to handle channels.

Adding An Activation Function

Each convolution operation is typically followed by an activation function nonlinearity.

$\text{Relu}(\text{Conv}(x, f))$

Note that the activation function is scalar-to-scalar and is applied to each channel at each time (or image) position.

Max Pooling

Pooling merges a segment of length p into a single channel vector by selecting, for each channel, the maximum value of that channel over the segment.

There are two parameters.

p is the size of the region pooled.

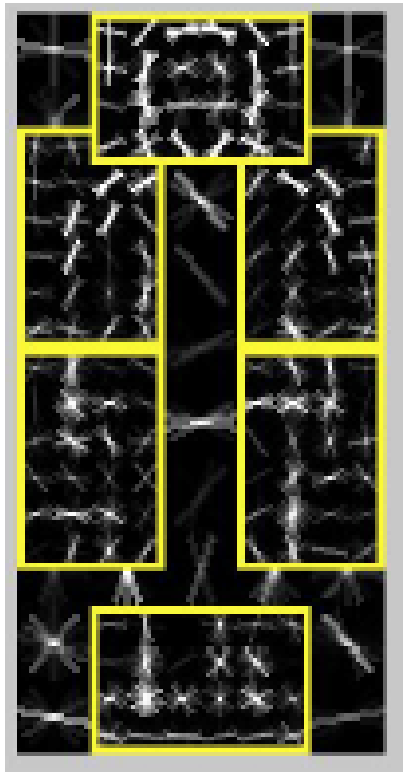
s is the "stride" — the size of the region shift on each iteration.

$$\mathbf{MaxPool}(x, p, s)[t, c] = \max_{j \in \{0,\ldots,p-1\}} x[st + j, c]$$

$$|\mathbf{MaxPool}(x, p, s)| = \lfloor (|x| - p)/s \rfloor + 1$$

MaxPooling Handles "Deformation"

The deformable part model (DPM):



In DPM the part filters are at a higher spatial resolution applied in a region around their nominal position.

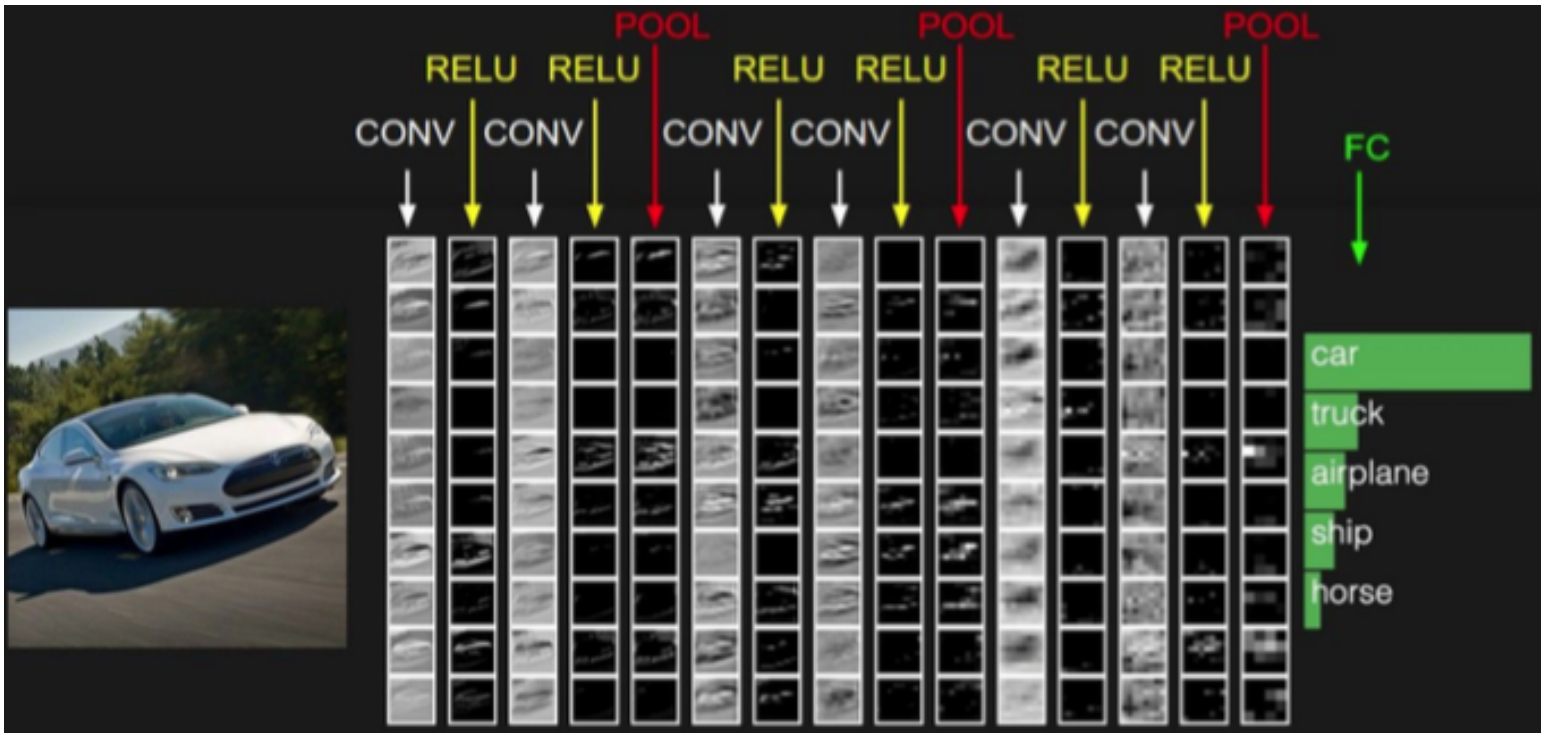
Average Pooling

Average pooling is the same as max pooling but takes an average rather than a max.

$$\mathbf{AvePool}(x, p, s)[t, c] = \frac{1}{p} \sum_{j \in \{0,\ldots,p-1\}} x[st + j, c]$$

$$|\mathbf{MaxPool}(x, p, s)| = \lfloor (|x| - p)/s \rfloor + 1$$

Example



Stanford CS231 Network

Convolution with Strides

Instead of advancing the filter one time value at each iteration, it is common to advance the filter by a stride s.

we can add a stride parameter to the convolution operation.

$$\text{Conv}(x, f, s)[t, c] = \sum_{j,c'} x[st + j, c']f[j, c, c']$$

$$|\textbf{MaxPool}(x, p, s)| = \lfloor (|x| - |f|)/s \rfloor + 1$$

Thinking About the Backward Method

Consider $y = \text{Conv}(x, f, s)$.

$$y.\text{value}[t, c] \texttt{ += } x.\text{value}[st + j, c']f.\text{value}[j, c, c']$$

Each increment can be backpropagated independently.

$$x.\text{grad}[st + j, c'] \texttt{ += } y.\text{grad}[t, c]\ f.\text{value}[j, c, c']$$

$$f.\text{grad}[j, c, c'] \texttt{ += } y.\text{grad}[t, c]\ x.\text{value}[st + j, c']$$

Until someone writes an appropriate compiler, one must still hand code the appropriate Numpy or CUDA vector operations.

The Backward Method with Minibatching

Forward:

$$y.\text{value}[b, t, c] \texttt{ += } x.\text{value}[b, st + j, c']f.\text{value}[j, c, c']$$

Backward:

$$x.grad[b, st + j, c'] \texttt{ += } y.\text{grad}[b, t, c]\ f.\text{value}[j, c, c']$$

$$f.\text{grad}[j, c, c'] \texttt{ += } y.grad[t, c]\ x.\text{value}[b, st + j, c']$$

Note that the backpropagation to f sums over both b and t.

Image to Column (Im2C)

Matrix multiplication is a highly optimized operation. Using more space, convolution can be reduced to matrix multiplication.

$$\begin{aligned}
\mathrm{Conv}(x, f)[t, c] &= \sum_{j,c'} x[t + j, c'] f[j, c', c] \\
&= \sum_{j,c'} X[t, j, c'] f[j, c', c] \\
X[t, j, c'] &= x[t + j, c']
\end{aligned}$$

This uses more space, the same value of x is included multiple times in X. The second line can be computed by a matrix multiplication of reshapings.

END