Introduction to Deep Learning

72% Indoor Booth

Slides are adapted from Bill Freeman, Antonio Torralba, Phillip Isola. 6.819 / 6.869
Announcement for the final exam

Final exam date: 07 May 2021 (Friday) 09:30 - 11:30 am:
Online final exam: self-arranged invigilation
Review background on signal processing, convolution, — this is the technology that underlies convnets!
This Week: Into Deep Learning

- Scene classification: A live demo
- A brief overview of Convolutional Neural Networks (CNNs)
- Standard building blocks of CNNs
- Some important networks & their tricks

- You are highly suggested to attend this week’s TA Tutorial on learning PyTorch and how to use deep networks
Image Classification

Try this!

http://places2.csail.mit.edu/demo.html
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Demos:
http://yann.lecun.com/exdb/lenet/index.html
Fig. 13. Examples of unusual, distorted, and noisy characters correctly recognized by LeNet-5. The grey-level of the output label represents the penalty (lighter for higher penalties).
Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

“Alexnet”
ImageNet Classification 2012

- Krizhevsky et al. -- 16.4% error (top-5)
- Next best (non-convnet) – 26.2% error
Krizhevsky, Sutskever, and Hinton, NeurIPS 2012
[“Mask RCNN”, He et al. 2017]
#edges2cats [Chris Hesse]

[“pix2pix”, Isola et al. 2017]
Ivy Tasi @ivymyt

Vitaly Vidmirov @vvid

[“pix2pix”, Isola et al. 2017]
Image recognition

Feature extractors:
- Edges
- Texture
- Colors

Classifier:
- Segments
- Parts

“Indoor booth”
Image recognition

Feature extractors

Edges
Texture
Colors

Segments
Parts

Learned

“Indoor booth”

\[
f_\theta(x) = \sum_{k=1}^{K} \theta_k \phi_k(x)
\]
Image recognition

"Indoor booth"
Image recognition

Neural net

Learned

"Indoor booth"
Image recognition

Learned

Deep neural net

“Indoor booth”
Deep learning

\[ \mathbf{x}_i \]

"Indoor booth"

\[ \mathbf{y}_i \]

\[ \theta_1 \quad \theta_2 \quad \theta_3 \quad \theta_4 \quad \theta_5 \quad \theta_6 \]

\[ \text{Learned} \]

\[ \text{Loss} \]

\[ \mathcal{L}(f_{\theta}(\mathbf{x}_i), \mathbf{y}_i) \]

\[ \theta^* = \arg \min_{\theta} \sum_{i=1}^{N} \mathcal{L}(f_{\theta}(\mathbf{x}_i), \mathbf{y}_i) \]
Gradient descent

\[ \theta^* = \arg \min_{\theta} \sum_{i=1}^{N} \mathcal{L}(\theta(x_i), y_i) \]

\[ J(\theta) \]
Gradient descent

\[ J(\theta) \]

\[ \theta^* = \arg\min_{\theta} J(\theta) \]
Gradient descent

\[ \theta^* = \arg\min_{\theta} \sum_{i=1}^{N} L(f_{\theta}(x_i), y_i) \]

\[ J(\theta) \]

One iteration of gradient descent:

\[ \theta^{t+1} = \theta^t - \eta_t \left. \frac{\partial J(\theta)}{\partial \theta} \right|_{\theta=\theta^t} \]

learning rate
What’s the knowledge we have about $J$?

- We can evaluate $J(\theta)$
- We can evaluate $J(\theta)$ and $\nabla_\theta J(\theta)$
- We can evaluate $J(\theta)$, $\nabla_\theta J(\theta)$, and $H_\theta(J(\theta))$
Comparison of gradient descent variants

[http://ruder.io/optimizing-gradient-descent/]
Computation in a neural net

Input representation → Output representation
Computation in a neural net

**Linear layer**

\[ y_j = \sum_i w_{ij} x_i \]
Computation in a neural net

\[
    y_j = \sum_i w_{ij} x_i + b_j
\]

\[y_j\] represents the output of the linear layer, \[x_i\] are the input representations, \[w_{ij}\] are the weights, and \[b_j\] is the bias term.
Computation in a neural net

**Linear layer**

**Input representation**

**Output representation**

\[
y_j = x^T w_j + b_j
\]

\[
\theta = \{W, b\}
\]

parameters of the model
Example: linear regression with a neural net

**Linear layer**

- **Input representation**: $x$
- **Output representation**: $y$

$$f_{w,b}(x) = x^T w + b$$
Computation in a neural net

"Perceptron"

Input representation

Output representation

$g(y) = \begin{cases} 
1, & \text{if } y > 0 \\
0, & \text{otherwise}
\end{cases}$

Pointwise Non-linearity
Example: linear classification with a perceptron

\[ y = \mathbf{x}^T \mathbf{w} + b \]
Example: linear classification with a perceptron

\[ y = \mathbf{x}^T \mathbf{w} + b \]

\[ g(y) = \begin{cases} 
1, & \text{if } y > 0 \\
0, & \text{otherwise}
\end{cases} \]
Example: linear classification with a perceptron

\[ y = \mathbf{x}^T \mathbf{w} + b \]

\[ g(y) = \begin{cases} 
1, & \text{if } y > 0 \\
0, & \text{otherwise}
\end{cases} \]
Example: linear classification with a perceptron

$$g(y) = \begin{cases} 
1, & \text{if } y > 0 \\
0, & \text{otherwise}
\end{cases}$$

$$y = \mathbf{x}^T \mathbf{w} + b$$
Example: linear classification with a perceptron

\[ \hat{y} = \mathbf{x}^T \mathbf{w} + b \]

\[ g(\hat{y}) = \begin{cases} 
  1, & \text{if } \hat{y} > 0 \\
  0, & \text{otherwise}
\end{cases} \]

\[ \mathbf{w}^*, b^* = \arg\min_{\mathbf{w}, b} \mathcal{L}(g(\hat{y}), y_i) \]
Example: linear classification with a perceptron

\[ \hat{y} = x^T w + b \]

\[ g(\hat{y}) = \begin{cases} 
1, & \text{if } \hat{y} > 0 \\
0, & \text{otherwise}
\end{cases} \]

\[ w^*, b^* = \arg \min_{w, b} \mathcal{L}(g(\hat{y}), y_i) \]
Computation in a neural net

\[ g(y) = \begin{cases} 
1, & \text{if } y > 0 \\
0, & \text{otherwise}
\end{cases} \]
Computation in a neural net — nonlinearity

\[ g(y) = \frac{1}{1 + e^{-y}} \]
Computation in a neural net — nonlinearity

- Interpretation as firing rate of neuron
- Bounded between \([0, 1]\)
- Saturation for large +/- inputs
- Gradients go to zero
- Outputs centered at 0.5 (poor conditioning)
- Not used in practice

\[
g(y) = \frac{1}{1 + e^{-y}}
\]
Computation in a neural net — nonlinearity

- Bounded between [-1, +1]
- Saturation for large +/- inputs
- Gradients go to zero
- Outputs centered at 0
- Preferable to sigmoid

$$\text{tanh}(x) = 2 \text{ sigmoid}(2x) - 1$$

**Tanh**

$$g(y) = \frac{e^y - e^{-y}}{e^y + e^{-y}}$$
Compute in a neural net — nonlinearity

- Unbounded output (on positive side)
- Efficient to implement: \( \frac{\partial g}{\partial y} = \begin{cases} 0, & \text{if } y < 0 \\ 1, & \text{if } y \geq 0 \end{cases} \)
- Also seems to help convergence (see 6x speedup vs tanh in [Krizhevsky et al.])
- Drawback: if strongly in negative region, unit is dead forever (no gradient).
- Default choice: widely used in current models.

Rectified linear unit (ReLU)

\[ g(y) = \max(0, y) \]
Computation in a neural net — nonlinearity

- where $\alpha$ is small (e.g. 0.02)
- Efficient to implement: $\frac{\partial g}{\partial y} = \begin{cases} 
-a, & \text{if } y < 0 \\
1, & \text{if } y \geq 0 
\end{cases}$
- Also known as probabilistic ReLU (PReLU)
- Has non-zero gradients everywhere (unlike ReLU)
- $\alpha$ can also be learned (see Kaiming He et al. 2015).

$$g(y) = \begin{cases} 
\max(0, y), & \text{if } y \geq 0 \\
\alpha \min(0, y), & \text{if } y < 0 
\end{cases}$$
Stacking layers

Input representation

Intermediate representation

Output representation

$h = \text{"hidden units"}$
Stacking layers

\[ h = g(W^{(1)}x + b^{(1)}) \quad \text{and} \quad y = W^{(2)}h + b^{(2)} \]

\[ \theta = \{ W^{(1)}, \ldots, W^{(L)}, b^{(1)}, \ldots, b^{(L)} \} \]
Stacking layers

\[ h = g(W^{(1)}x + b^{(1)}) \quad y = W^{(2)}h + b^{(2)} \]

\[ \theta = \{W^{(1)}, \ldots, W^{(L)}, b^{(1)}, \ldots, b^{(L)}\} \]
Stacking layers

\[ h = g(W^{(1)}x + b^{(1)}) \]
\[ y = W^{(2)}h + b^{(2)} \]
\[ \theta = \{W^{(1)}, \ldots, W^{(L)}, b^{(1)}, \ldots, b^{(L)}\} \]
Input representation \( \mathbf{x} \)

Intermediate representation \( \mathbf{h} = g(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)}) \)

Output representation \( \mathbf{y} = \mathbf{W}^{(2)} \mathbf{h} + \mathbf{b}^{(2)} \)

\[ \theta = \{ \mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)} \} \]
Stacking layers

\[ h = g(W^{(1)}x + b^{(1)}) \quad y = W^{(2)}h + b^{(2)} \]

\[ \theta = \{W^{(1)}, \ldots, W^{(L)}, b^{(1)}, \ldots, b^{(L)}\} \]
Connectivity patterns

Input representation \( x \) \( \rightarrow \) Output representation \( y \) through two types of layers:

- **Fully connected layer**
  \[ W^{(1)} \]

- **Locally connected layer (Sparse \( W \))**
2-dimensional input representation

\[ \mathbb{R}^{H \times W \times C^{(l)}} \rightarrow \mathbb{R}^{H \times W \times C^{(l+1)}} \]

2-dimensional output representation

[Figure from Andrea Vedaldi]
Deep Neural Networks

\[ f(x) = f_L(\ldots f_2(f_1(x))) \]

Classify

“clown fish”
Loss function

Network output

dolphin
cat
grizzly bear
angel fish
chameleon
**clown fish**
iguana
elephant
...

Ground truth label

“clown fish”

Loss $\rightarrow$ error
Loss function

Network output

- dolphin
- cat
- grizzly bear
- angel fish
- chameleon
- clown fish
- iguana
- elephant

Ground truth label

- “clown fish”

Loss → small
Loss function

Network output

- dolphin
- cat
- grizzly bear
- angel fish
- chameleon
- clown fish
- iguana
- iguana
- elephant

Ground truth label

“grizzly bear”

Loss → large
Prediction $\hat{y}$

$$f_\theta : X \rightarrow \mathbb{R}^K$$

dolphin

$$\bullet$$

cat

grizzly bear

$$\bullet$$

angel fish

$$\bullet$$

chameleon

$$\bullet$$

clown fish

iguana

elephant

$$\vdots$$

0

1

Ground truth label $y$

dolphin

cat

grizzly bear

$$\bullet$$

angel fish

$$\bullet$$

chameleon

$$\bullet$$

clown fish

iguana

elephant

$$\vdots$$

0

1
Network output  

\[ \hat{y} \]

- dolphin
- cat
- grizzly bear
- angel fish
- chameleon
- clown fish
- iguana
- elephant
- ...

Ground truth label  

\[ y \]

Probability of the observed data under the model

\[ H(y, \hat{y}) = - \sum_{k=1}^{K} y_k \log \hat{y}_k \]
Representational power

• 1 layer? Linear decision surface.

• 2+ layers? In theory, can represent any function. Assuming non-trivial non-linearity.
  – Bengio 2009,  
  – Bengio, Courville, Goodfellow book  
    http://www.deeplearningbook.org/contents/mlp.html
  – Simple proof by M. Neilsen  
  – D. Mackay book  
    http://www.inference.phy.cam.ac.uk/mackay/itprnn/ps/482.491.pdf

• But issue is efficiency: very wide two layers vs narrow deep model? In practice, more layers helps.
Example: linear classification with a perceptron

\[ y = \mathbf{x}^T \mathbf{w} + b \]
Example: nonlinear classification with a deep net net

\[
h = g(W^{(1)}x + b^{(1)})
\]

\[
y = W^{(2)}h + b^{(2)}
\]
[http://playground.tensorflow.org]
Example: nonlinear classification with a deep net

What class is ?

Answer: Underfitting

Answer: Appropriate model

Answer: Overfitting
Deep learning

\[ y_1 \text{ “clown fish”} \]

\[ x_1 \]

\[ \theta_1 \quad \theta_2 \quad \theta_3 \quad \theta_4 \quad \theta_5 \quad \theta_6 \]

\[ \text{Loss} \quad \mathcal{L}(f_{\theta}(x_1), y_1) \]

\[ \theta^* = \arg \min_{\theta} \sum_{i=1}^{N} \mathcal{L}(f_{\theta}(x_i), y_i) \]
Deep learning

\[ \theta^* = \arg \min_{\theta} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i) \]

\[ \mathcal{L}(f_\theta(x_2), y_2) \]

"grizzly bear"

\[ x_2 \]

\[ y_2 \]
Deep learning

\( y_i \)  
"chameleon"

\( x_i \)

\[ \theta^* = \arg \min_{\theta} \sum_{i=1}^{N} \mathcal{L}(f_{\theta}(x_i), y_i) \]

\[ \text{Loss} \] 
\[ \mathcal{L}(f_{\theta}(x_i), y_i) \]
Batch (parallel) processing

Images

Features

Loss

\[ \sum \]
Tensors
(multi-dimensional arrays)

Each layer is a representation of the data
Tensors
(multi-dimensional arrays)

\[ h^{(1)} \in \mathbb{R}^{N_{\text{batch}} \times C^{(1)}} \]

- # neurons
- # features
- # units
- # “channels”
Tensors
(multi-dimensional arrays)

$$h^{(1)} \in \mathbb{R}^{N_{\text{batch}} \times C^{(1)}}$$

- # neurons
- # features
- # units
- # “channels”
“Tensor flow”

\[ \mathbf{h}^{(1)} \in \mathbb{R}^{N_{\text{batch}} \times C^{(1)}} \]

\[ \mathbf{h}^{(2)} \in \mathbb{R}^{N_{\text{batch}} \times C^{(2)}} \]
Layer 1 representation

Layer 6 representation

[DeCAF, Donahue, Jia, et al. 2013]
[Visualization technique: t-sne, van der Maaten & Hinton, 2008]
“Tensor flow”

\[ \mathbf{h}^{(1)} \in \mathbb{R}^{N_{\text{batch}} \times H^{(1)} \times W^{(1)} \times C^{(1)}} \]

\[ \mathbf{h}^{(2)} \in \mathbb{R}^{N_{\text{batch}} \times H^{(2)} \times W^{(2)} \times C^{(2)}} \]
Convolutional Neural Networks
Bird Classifier
Problem:
What happens to objects that are bigger?
What if an object crosses multiple cells?
“Cell”-based approach is limited.

What can we do instead?
What’s the object class of the center pixel?

- “Bird”
- “Bird”
- “Sky”
- “Sky”
What’s the object class of the center pixel?

Training data

\[
x \quad y
\]

\[
\{ \text{“Bird”} \}, \quad \{ \text{“Bird”} \}, \quad \{ \text{“Sky”} \}, \quad \ldots
\]
This problem is called **semantic segmentation**
What’s the object class of the center pixel?

An equivariant mapping:

\[ f(\text{translate}(x)) = \text{translate}(f(x)) \]
\( W \) computes a weighted sum of all pixels in the patch.

\( W \) is a convolutional kernel applied to the full image!
Convolution

The image shows a convolution process. On the left, there is an input image, a clownfish. In the middle, there is a filter, and on the right, the output of the convolution process, which appears to highlight the edges of the clownfish.
Fully-connected network

Fully-connected (fc) layer

\[
W \quad x \quad b \quad y \quad g(y)
\]
Often, we assume output is a **local** function of input.

If we use the same weights (**weight sharing**) to compute each local function, we get a convolutional neural network.
Convolutional neural network

Often, we assume output is a **local** function of input.

If we use the same weights (**weight sharing**) to compute each local function, we get a convolutional neural network.
Often, we assume output is a \textbf{local} function of input.

If we use the same weights (\textbf{weight sharing}) to compute each local function, we get a convolutional neural network.
Toeplitz matrix

\[
\begin{pmatrix}
a & b & c & d & e \\
f & a & b & c & d \\
g & f & a & b & c \\
h & g & f & a & b \\
i & h & g & f & a \\
\end{pmatrix}
\]

\[x^{(l+1)} = \text{Toeplitz matrix} \ast x^{(l)}\]

e.g., pixel image

- Constrained linear layer
- Fewer parameters \(\rightarrow\) easier to learn, less overfitting
Conv layers can be applied to arbitrarily-sized inputs
\[
\begin{bmatrix}
X^{(l+1)} \\
\end{bmatrix}
= 
K 
\ast 
\begin{bmatrix}
X^{(l)} \\
\end{bmatrix}
\]
Five views on convolutional layers

1. Equivariant with translation (stationarity) \( f(\text{translate}(x)) = \text{translate}(f(x)) \)

2. Patch processing (Markov assumption)

3. Image filter

4. Parameter sharing

5. A way to process variable-sized tensors
What if we have color?

(aka multiple input channels?)
Multiple channels

Conv layer

$x^{(0)}$  $x^{(1)}$

$\mathbb{R}^{N \times C} \rightarrow \mathbb{R}^{N \times 1}$
Multiple channels

Conv layer

\[
\mathbb{R}^{N \times C^{(0)}} \rightarrow \mathbb{R}^{N \times C^{(1)}}
\]
Multiple channels

Conv layer

\[ x^{(l)} \xrightarrow{\cdot W_{1}} \ldots \xrightarrow{\cdot W_{C^{(l+1)}}} x^{(l+1)} \]

\[ \mathbb{R}^{N \times C^{(l)}} \rightarrow \mathbb{R}^{N \times C^{(l+1)}} \]
Multiple channels: Example

How many parameters does each filter have?

(a) 9       (b) 27     (c) 96     (d) 864
Multiple channels: Example

How many filters are in the bank?

(a) 3       (b) 27     (c) 96     (d) can’t say
Filter sizes

When mapping from

\[ \mathbf{x}^{(l)} \in \mathbb{R}^{H \times W \times C^{(l)}} \rightarrow \mathbf{x}^{(l+1)} \in \mathbb{R}^{H \times W \times C^{(l+1)}} \]

using an filter of spatial extent \( M \times N \)

Number of parameters per filter: \( M \times N \times C^{(l)} \)

Number of filters: \( C^{(l+1)} \)
Input features \[ \mathbb{R}^{H \times W \times C^{(l)}} \] to \[ \mathbb{R}^{H \times W \times C^{(l+1)}} \] via a bank of 2 filters.
Image classification

image $x$ \rightarrow \text{Classifier} \rightarrow \text{"Indoor booth"}

label $y$
Image classification

image $x$  $\rightarrow$ Conv ReLU Conv ReLU Conv ReLU $\rightarrow$ “Indoor booth”  label $y$
Multiscale representations are great!

How can we use multi-scale modeling in ConvNets?
Pooling

Max pooling
\[ z_k = \max_{j \in \mathcal{N}(j)} g(y_j) \]
Pooling

**Max pooling**

\[ z_k = \max_{j \in \mathcal{N}(j)} g(y_j) \]

**Mean pooling**

\[ z_k = \frac{1}{|\mathcal{N}|} \sum_{j \in \mathcal{N}(j)} g(y_j) \]
Pooling — Why?

Pooling across spatial locations achieves stability w.r.t. small translations:
Pooling — Why?

Pooling across spatial locations achieves stability w.r.t. small translations:

large response regardless of exact position of edge
Pooling — Why?

Pooling across spatial locations achieves stability w.r.t. small translations:
CNNs are stable w.r.t. diffeomorphisms

[“Unreasonable effectiveness of Deep Features as a Perceptual Metric”, Zhang et al. 2018]
Pooling — Why?

Pooling across feature channels (filter outputs) can achieve other kinds of invariances:

large response for any edge, regardless of its orientation

[Derived from slide by Andrea Vedaldi]
Computation in a neural net

\[ f(x) = f_L(\ldots f_2(f_1(x))) \]
Downsampling

Filter

Pool and downsample

\[ x \quad w \quad y \quad g(y) \quad z \]

and downsample
**Downsampling**

\[
\mathbb{R}^{H^{(l)} \times W^{(l)} \times C^{(l)}} \rightarrow \mathbb{R}^{H^{(l+1)} \times W^{(l+1)} \times C^{(l+1)}}
\]
**Strided operations** combine a given operation (convolution or pooling) and downsampling into a single operation.

\[ y \rightarrow g(y) \]
Computation in a neural net

\[ f(x) = f_L(\ldots f_2(f_1(x))) \]
Receptive fields

Pool and
downsampling by 2

3x1 Filter

Pool and
downsampling by 2

RF = RF * 2

RF = RF + \text{floor}(3/2) * 2

RF = RF * 2

kernel size

scale factor
Effective Receptive Field
Contributing input units to a convolutional filter.

@jimmfleming // fomoro.com
Some networks

... and what makes them work
2012: AlexNet
5 conv. layers

[Krizhevsky et al: ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012]
Alexnet — [Krizhevsky et al. NIPS 2012]

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)
What filters are learned?
What filters are learned?
Get to know your units

11x11 convolution kernel
(3 color channels)
Get to know your units
Get to know your units
Get to know your units
Get to know your units
Get to know your units
Get to know your units
Get to know your units

96 Units in conv1
Units at Higher Layers

Zeiler and Fergus, ECCV’14
2014: VGG
16 conv. layers

ImageNet Classification Error (Top 5)

<table>
<thead>
<tr>
<th>Year</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>26.0</td>
</tr>
<tr>
<td>2012</td>
<td>16.4</td>
</tr>
<tr>
<td>2013</td>
<td>11.7</td>
</tr>
<tr>
<td>2014</td>
<td>7.3</td>
</tr>
</tbody>
</table>

Error: 7.3%

[Simonyan & Zisserman: Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015]
VGG-Net [Simonyan & Zisserman, 2015]

Main developments

- Small convolutional kernels: only 3x3

- Increased depth (5 -> 16/19 layers)

Error: 7.3%
Chaining convolutions

3x3 3x3 = 5x5
25 coefficients, but only 18 degrees of freedom

9 coefficients, but only 6 degrees of freedom.
Only separable filters… would this be enough?
ImageNet Classification Error (Top 5)

- 2011 (XRCE): 26.0
- 2012 (AlexNet): 16.4
- 2013 (ZF): 11.7
- 2014 (VGG): 7.3
ImageNet Classification Error (Top 5)

<table>
<thead>
<tr>
<th>Year</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011 (XRCE)</td>
<td>26.0</td>
</tr>
<tr>
<td>2012 (AlexNet)</td>
<td>16.4</td>
</tr>
<tr>
<td>2013 (ZF)</td>
<td>11.7</td>
</tr>
<tr>
<td>2014 (VGG)</td>
<td>7.3</td>
</tr>
<tr>
<td>2014 (GoogLeNet)</td>
<td>6.7</td>
</tr>
<tr>
<td>Human</td>
<td>5.0</td>
</tr>
<tr>
<td>2015 (ResNet)</td>
<td>3.6</td>
</tr>
</tbody>
</table>

Error: 3.6%

[He et al: Deep Residual Learning for Image Recognition, CVPR 2016]
ResNet [He et al, 2016]

Main developments

• Increased depth possible through residual blocks
Residual Blocks

\[ F(x) \]

\[ F(x) + x \]

relu

weight layer

identity

relu

weight layer

x
Residual Blocks

Why do they work?

• Gradients can propagate faster (via the identity mapping)

• Within each block, only small residuals have to be learned