Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
Network is a stack of components

GoogleNet, 22 layers
(ILSVRC 2014)
Components of a Convolutional Net

- **Input Image**: 32x32
- **Convolutions**: C1: feature maps 6@28x28, C3: f. maps 16@10x10
- **Subsampling**: S2: f. maps 6@14x14, S4: f. maps 16@5x5
- **Full Connection**: C5: layer 120, F6: layer 84
- **Gaussian Connections**: Output 10
- **Weights**
- **Loss**
two more layers to go: POOL/FC
[ConvNetJS demo: training on CIFAR-10]

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html
Fully Connected layer

\[ B_j = \sum_i (W_{ij} \ast A_i) + b_j \]
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation maps

consider a second, green filter

8 : COS429 : L19 : 29.11.16 : Andras Ferencz
In practice: Common to zero pad the border

e.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

7x7 output!
in general, common to see CONV layers with
stride 1, filters of size FxF, and zero-padding with
(F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
    F = 5 => zero pad with 2
    F = 7 => zero pad with 3
We call the layer convolutional because it is related to convolution of two signals:

\[ f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1,y-n_2] \]

elementwise multiplication and sum of a filter and the signal (image)
(btw, 1x1 convolution layers make perfect sense)

1x1 CONV with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)
Activation Layer

Sigmoid

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

\[ \tanh \text{ tanh}(x) \]

ReLU \quad \text{max}(0, x)

Leaky ReLU

\[ \text{max}(0.1x, x) \]

Maxout

\[ \text{max}(w_1^T x + b_1, w_2^T x + b_2) \]

ELU

\[ f(x) = \begin{cases} 
  x & \text{if } x > 0 \\
  \alpha (\exp(x) - 1) & \text{if } x \leq 0 
\end{cases} \]
TLDR: In practice:

- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU
- Try out tanh but don’t expect much
- Don’t use sigmoid
Pooling layer
- makes the representations smaller and more manageable
- operates over each activation map independently:

![Diagram of pooling layer](image)
Single depth slice

MAX POOLING

max pool with 2x2 filters and stride 2

1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4

6 8
3 4
### Multiclass Hinge loss:

Suppose: 3 training examples, 3 classes, and their scores

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>3.2</th>
<th>1.3</th>
<th>2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

Given an example \((x_i, y_i)\)

where \(x_i\) is the image and where \(y_i\) is the (integer) label,

and using the shorthand for the scores vector:

\[ s = f(x_i, W) \]

the SVM loss has the form:

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \]
Softmax Loss (Multinomial Logistic)

\[ L_i = - \log \left( \frac{e^{s_y_i}}{\sum_j e^{s_j}} \right) \]

<table>
<thead>
<tr>
<th></th>
<th>Unnormalized log probabilities</th>
<th>Unnormalized probabilities</th>
<th>Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>24.5</td>
<td>0.13</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>164.0</td>
<td>0.87</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>0.18</td>
<td>0.00</td>
</tr>
</tbody>
</table>

\[ L_i = - \log(0.13) = 0.89 \]
Softmax vs. SVM

\[ L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \]

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \]

assume scores:

[10, -2, 3]
[10, 9, 9]
[10, -100, -100]

and

\[ y_i = 0 \]

Q: Suppose I take a datapoint and I jiggle a bit (changing its score slightly). What happens to the loss in both cases?
**Hinge Loss (SVM)**

\[
\max(0, -2.85 - 0.28 + 1) + \\
\max(0, 0.86 - 0.28 + 1)
\]

\[
= 1.58
\]

**Cross-Entropy Loss (Softmax)**

\[
\begin{align*}
-2.85 & \quad 0.058 & \quad 0.016 \\
0.86 & \quad 2.36 & \quad -\log(0.353) \\
0.28 & \quad 1.32 & \quad 0.452
\end{align*}
\]

\[
\text{normalize (to sum to one)} \\
\text{0.631} \quad 0.353
\]

\[
\text{0.353}
\]

\[
= 0.452
\]
How to optimize?

```python
# Vanilla Gradient Descent

while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights ← - step_size * weights_grad # perform parameter update
```
The decision boundary perspective...

Initial random weights
The decision boundary perspective…

Present a training instance / adjust the weights
The decision boundary perspective...

Present a training instance / adjust the weights
The decision boundary perspective...

Present a training instance / adjust the weights
Present a training instance / adjust the weights
The decision boundary perspective...

Eventually ....
**Stochastic Gradient Descent**

A dataset

<table>
<thead>
<tr>
<th>Fields</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4  2.7  1.9</td>
<td>0</td>
</tr>
<tr>
<td>3.8  3.4  3.2</td>
<td>0</td>
</tr>
<tr>
<td>6.4  2.8  1.7</td>
<td>1</td>
</tr>
<tr>
<td>4.1  0.1  0.2</td>
<td>0</td>
</tr>
<tr>
<td>etc ...</td>
<td></td>
</tr>
</tbody>
</table>

Diagram of a neural network.
Stochastic Gradient Descent

**Fields**

| 1.4 | 2.7 | 1.9  | 0 |
| 3.8 | 3.4 | 3.2  | 0 |
| 6.4 | 2.8 | 1.7  | 1 |
| 4.1 | 0.1 | 0.2  | 0 |
| etc  |    |      |   |

**class**

![Diagram of a neural network](image-url)
Stochastic Gradient Descent

*Training data*

<table>
<thead>
<tr>
<th>Fields</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td>2.7</td>
</tr>
<tr>
<td>3.8</td>
<td>3.4</td>
</tr>
<tr>
<td>6.4</td>
<td>2.8</td>
</tr>
<tr>
<td>4.1</td>
<td>0.1</td>
</tr>
<tr>
<td>etc ...</td>
<td></td>
</tr>
</tbody>
</table>
Stochastic Gradient Descent

Training data

Fields               class
1.4  2.7   1.9   0
3.8  3.4   3.2   0
6.4  2.8   1.7   1
4.1  0.1   0.2   0
etc ...
### Stochastic Gradient Descent

**Training data**

<table>
<thead>
<tr>
<th>Fields</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td>2.7</td>
</tr>
<tr>
<td>3.8</td>
<td>3.4</td>
</tr>
<tr>
<td>6.4</td>
<td>2.8</td>
</tr>
<tr>
<td>4.1</td>
<td>0.1</td>
</tr>
<tr>
<td>etc</td>
<td></td>
</tr>
</tbody>
</table>

Feed it through to get output.
Stochastic Gradient Descent

Training data

<table>
<thead>
<tr>
<th>Fields</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4 2.7 1.9</td>
<td>0</td>
</tr>
<tr>
<td>3.8 3.4 3.2</td>
<td>0</td>
</tr>
<tr>
<td>6.4 2.8 1.7</td>
<td>1</td>
</tr>
<tr>
<td>4.1 0.1 0.2</td>
<td>0</td>
</tr>
<tr>
<td>etc ...</td>
<td></td>
</tr>
</tbody>
</table>

Compare with target output

Error 0.8
Stochastic Gradient Descent

**Training data**

<table>
<thead>
<tr>
<th>Fields</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td>2.7</td>
</tr>
<tr>
<td>3.8</td>
<td>3.4</td>
</tr>
<tr>
<td>6.4</td>
<td>2.8</td>
</tr>
<tr>
<td>4.1</td>
<td>0.1</td>
</tr>
<tr>
<td>etc</td>
<td></td>
</tr>
</tbody>
</table>

Adjust weights based on error:

- 1.4
- 2.7
- 1.9

**Error**: 0.8
## Stochastic Gradient Descent

**Training data**

<table>
<thead>
<tr>
<th>Fields</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td>0</td>
</tr>
<tr>
<td>3.8</td>
<td>0</td>
</tr>
<tr>
<td>6.4</td>
<td>1</td>
</tr>
<tr>
<td>4.1</td>
<td>0</td>
</tr>
</tbody>
</table>
e.tc ...
Stochastic Gradient Descent

Training data

Fields | class
---|---
1.4  2.7  1.9 | 0
3.8  3.4  3.2 | 0
**6.4  2.8  1.7** | **1**
4.1  0.1  0.2 | 0
etc ...

Feed it through to get output

```
6.4 2.8 1.7
```

0.9
Stochastic Gradient Descent

**Training data**

<table>
<thead>
<tr>
<th>Fields</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td>2.7</td>
</tr>
<tr>
<td>3.8</td>
<td>3.4</td>
</tr>
<tr>
<td><strong>6.4</strong></td>
<td><strong>2.8</strong></td>
</tr>
<tr>
<td>4.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

etc ...

Compare with target output

\[
\begin{align*}
6.4 & \\
2.8 & \\
1.7 & \text{error -0.1}
\end{align*}
\]

\[
\begin{align*}
& 0.9 \\
& 1
\end{align*}
\]
Stochastic Gradient Descent

Training data

<table>
<thead>
<tr>
<th>Fields</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4 2.7 1.9</td>
<td>0</td>
</tr>
<tr>
<td>3.8 3.4 3.2</td>
<td>0</td>
</tr>
<tr>
<td>6.4 2.8 1.7</td>
<td>1</td>
</tr>
<tr>
<td>4.1 0.1 0.2</td>
<td>0</td>
</tr>
</tbody>
</table>

etc …

Adjust weights based on error
### Stochastic Gradient Descent

#### Training data

<table>
<thead>
<tr>
<th>Fields</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4 2.7 1.9</td>
<td>0</td>
</tr>
<tr>
<td>3.8 3.4 3.2</td>
<td>0</td>
</tr>
<tr>
<td>6.4 2.8 1.7</td>
<td>1</td>
</tr>
<tr>
<td>4.1 0.1 0.2</td>
<td>0</td>
</tr>
</tbody>
</table>

And so on....

![Diagram of a neural network with labels and error values.](attachment:image.png)

- 6.4
- 2.8
- 1.7
- error -0.1
- 0.9
- 1
Mini-batch SGD

Loop:
1. **Sample** a batch of data
2. **Forward** prop it through the graph, get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient
Follow the slope

In 1-dimension, the derivative of a function:

\[
\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}
\]

In multiple dimensions, the \textbf{gradient} is the vector of (partial derivatives).
This is silly. The loss is just a function of $W$:

\[
L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2
\]

\[
L_i = \sum_{j\neq y_i} \max(0, s_j - s_{y_i} + 1)
\]

\[
s = f(x; W) = Wx
\]

\[
\nabla_W L = \ldots
\]
\[ f = WX \]

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \]
\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, y = 5, z = -4 \)
\[ f(x, y, z) = (x + y)z \]

E.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want:
\[
\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}
\]
\[ f(x, y, z) = (x + y)z \]

\[ \text{e.g. } x = -2, \ y = 5, \ z = -4 \]

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \ \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \ \frac{\partial f}{\partial z} = q \]

Want: \[ \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \]
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \[ \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \]
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \)

Chain rule:

\[ \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y} \]
\[ f(x, y, z) = (x + y)z \]

\[ \text{e.g. } x = -2, \ y = 5, \ z = -4 \]

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \ \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \ \frac{\partial f}{\partial z} = q \]

Want: \[ \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \]
\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want:\[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \]

Chain rule:

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} \]
activations
activations

\[ \frac{\partial z}{\partial x} \]

\[ \frac{\partial z}{\partial y} \]

"local gradient"
activations

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}
\]

"local gradient"

gradients
activations

\[ \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} \]

\[ \frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y} \]

"local gradient"
activations

\[ \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} \]

\[ \frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y} \]

“local gradient”

gradients
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

**First layer** (CONV1): 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size? Hint: \((227-11)/4+1 = 55\)
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

**First layer** (CONV1): 96 11x11 filters applied at stride 4

=>

Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

**First layer** (CONV1): 96 11x11 filters applied at stride 4

=>

Output volume **[55x55x96]**

Parameters: \((11*11*3)*96 = 35K\)
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: \((55-3)/2+1 = 27\)
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

**Second layer** (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...
Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

- [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)
Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

- **INPUT**
  - [227x227x3] Input

- **CONV1**:
  - [55x55x96] 96 11x11 filters at stride 4, pad 0

- **MAX POOL1**:
  - [27x27x96] 3x3 filters at stride 2

- **NORM1**:
  - Normalization layer

- **CONV2**:
  - [27x27x256] 256 5x5 filters at stride 1, pad 2

- **MAX POOL2**:
  - [13x13x256] 3x3 filters at stride 2

- **NORM2**:
  - Normalization layer

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

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

- **CONV5**:
  - [13x13x256] 256 3x3 filters at stride 1, pad 1

- **MAX POOL3**:
  - [6x6x256] 3x3 filters at stride 2

- **FC6**:
  - [4096] 4096 neurons

- **FC7**:
  - [4096] 4096 neurons

- **FC8**:
  - [1000] 1000 neurons (class scores)

Details/Retrospectives:

- First use of ReLU
- Used Norm layers (not common anymore)
- Heavy data augmentation
- Dropout 0.5
- Batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%
Case Study: ZFNet  

AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 15.4% -> 14.8%
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1 and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error
INPUT: [224x224x3] memory: 224*224*3 = 150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64 = 3.2M params: (3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64 = 3.2M params: (3*3)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64 = 800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128 = 1.6M params: (3*3)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128 = 1.6M params: (3*3)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128 = 400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256 = 800K params: (3*3)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256 = 800K params: (3*3)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256 = 800K params: (3*3)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256 = 200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512 = 400K params: (3*3)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512 = 400K params: (3*3)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512 = 400K params: (3*3)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512 = 100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512 = 100K params: (3*3)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512 = 100K params: (3*3)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512 = 100K params: (3*3)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512 = 25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~ 93MB / image (only forward! ~ *2 for bwd)
TOTAL params: 138M parameters
INPUT: [224x224x3]  memory: 224*224*3 = 150K  params: 0

CONV3-64: [224x224x64]  memory: 224*224*64 = 3.2M  params: (3*3*3)*64 = 1,728

CONV3-64: [224x224x64]  memory: 224*224*64 = 3.2M  params: (3*3*64)*64 = 36,864

POOL2: [112x112x64]  memory: 112*112*64 = 800K  params: 0

CONV3-128: [112x112x128]  memory: 112*112*128 = 1.6M  params: (3*3*64)*128 = 73,728

CONV3-128: [112x112x128]  memory: 112*112*128 = 1.6M  params: (3*3*128)*128 = 147,456

POOL2: [56x56x128]  memory: 56*56*128 = 400K  params: 0

CONV3-256: [56x56x256]  memory: 56*56*256 = 800K  params: (3*3*128)*256 = 294,912

CONV3-256: [56x56x256]  memory: 56*56*256 = 800K  params: (3*3*256)*256 = 589,824

CONV3-256: [56x56x256]  memory: 56*56*256 = 800K  params: (3*3*256)*256 = 589,824

POOL2: [28x28x256]  memory: 28*28*256 = 200K  params: 0

CONV3-512: [28x28x512]  memory: 28*28*512 = 400K  params: (3*3*256)*512 = 1,179,648

CONV3-512: [28x28x512]  memory: 28*28*512 = 400K  params: (3*3*512)*512 = 2,359,296

CONV3-512: [28x28x512]  memory: 28*28*512 = 400K  params: (3*3*512)*512 = 2,359,296

POOL2: [14x14x512]  memory: 14*14*512 = 100K  params: 0

CONV3-512: [14x14x512]  memory: 14*14*512 = 100K  params: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512]  memory: 14*14*512 = 100K  params: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512]  memory: 14*14*512 = 100K  params: (3*3*512)*512 = 2,359,296

POOL2: [7x7x512]  memory: 7*7*512 = 25K  params: 0

FC: [1x1x4096]  memory: 4096  params: 7*7*512*4096 = 102,760,448

FC: [1x1x4096]  memory: 4096  params: 4096*4096 = 16,777,216

FC: [1x1x1000]  memory: 1000  params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)

TOTAL params: 138M parameters

Note:
Most memory is in early CONV
Most params are in late FC
(Not counting biases)
Case Study: GoogLeNet

[Szegedy et al., 2014]

Inception module

ILSVRC 2014 winner (6.7% top 5 error)
Case Study: GoogLeNet

### Fun features:
- Only 5 million params! (Removes FC layers completely)

### Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)
Case Study: ResNet \cite{He2015}

ILSVRC 2015 winner (3.6\% top 5 error)

**MSRA @ ILSVRC & COCO 2015 Competitions**

- **1st places in all five main tracks**
  - ImageNet Classification: \textit{Ultra-deep} (quote Yann) \textbf{152-layer} nets
  - ImageNet Detection: \textbf{16\%} better than 2nd
  - ImageNet Localization: \textbf{27\%} better than 2nd
  - COCO Detection: \textbf{11\%} better than 2nd
  - COCO Segmentation: \textbf{12\%} better than 2nd

*improvements are relative numbers

Slide from Kaiming He’s recent presentation [https://www.youtube.com/watch?v=1PGLj-uKT1w](https://www.youtube.com/watch?v=1PGLj-uKT1w)
Revolution of Depth

152 layers

ImageNet Classification top-5 error (%)

ILSVRC'15 ResNet
22 layers
6.7
3.57

ILSVRC'14 GoogleNet
8 layers
2.3

ILSVRC'14 VGG
19 layers
7.3

ILSVRC'13
8 layers
11.7

ILSVRC'12 AlexNet
8 layers
16.4

ILSVRC'11
shallow
25.8

ILSVRC'10
28.2

(slide from Kaiming He's recent presentation)
CIFAR-10 experiments

CIFAR-10 plain nets

56-layer
44-layer
32-layer
20-layer

CIFAR-10 ResNets

20-layer
32-layer
44-layer
56-layer
110-layer

solid: test
dashed: train
Case Study: ResNet  [He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)

Revolution of Depth

- AlexNet, 8 layers (ILSVRC 2012)
- VGG, 19 layers (ILSVRC 2014)
- ResNet, 152 layers (ILSVRC 2015)

2-3 weeks of training on 8 GPU machine
at runtime: faster than a VGGNet!
even though it has 8x more layers

(slide from Kaiming He’s recent presentation)
Case Study: ResNet

[He et al., 2015]
Case Study: ResNet \cite{He2015}

- Plain net

\[ H(x) = \text{relu} \left( \text{weight layer} \left( \text{relu} \left( \text{weight layer} (x) \right) \right) \right) \]

- Residual net

\[ F(x) = \text{relu} \left( \text{weight layer} \left( \text{weight layer} (x) \right) \right) \]

\[ H(x) = F(x) + x \]
Case Study: ResNet  [He et al., 2015]

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of $1e^{-5}$
- No dropout used
Case Study: ResNet [He et al., 2015]
Case Study: ResNet [He et al., 2015]

(this trick is also used in GoogLeNet)
## Case Study: ResNet [He et al., 2015]

<table>
<thead>
<tr>
<th>layer name</th>
<th>output size</th>
<th>18-layer</th>
<th>34-layer</th>
<th>50-layer</th>
<th>101-layer</th>
<th>152-layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>112×112</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7×7, 64, stride 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3×3 max pool, stride 2</td>
</tr>
<tr>
<td>conv2_x</td>
<td>56×56</td>
<td>[3×3, 64]×2</td>
<td>[3×3, 64]×3</td>
<td>[1×1, 64]×3</td>
<td>[1×1, 64]×3</td>
<td>[1×1, 64]×3</td>
</tr>
<tr>
<td>conv5_x</td>
<td>7×7</td>
<td>[3×3, 512]×2</td>
<td>[3×3, 512]×3</td>
<td>[1×1, 512]×3</td>
<td>[1×1, 512]×3</td>
<td>[1×1, 512]×3</td>
</tr>
<tr>
<td></td>
<td>FLOPs</td>
<td>1.8×10⁹</td>
<td>3.6×10⁹</td>
<td>3.8×10⁹</td>
<td>7.6×10⁹</td>
<td>11.3×10⁹</td>
</tr>
<tr>
<td></td>
<td>average pool, 1000-d fc, softmax</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case Study Bonus: DeepMind’s AlphaGo
The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a $23 \times 23$ image, then convolves $k$ filters of kernel size $5 \times 5$ with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a $21 \times 21$ image, then convolves $k$ filters of kernel size $3 \times 3$ with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size $1 \times 1$ with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used $k = 192$ filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with $k = 128, 256$ and 384 filters.

**policy network:**

[19x19x48] Input

CONV1: 192 5x5 filters, stride 1, pad 2 => [19x19x192]

CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]

CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] *(probability map of promising moves)*