Deep Residual Network and Its Variations

Diyu Yang

(Originally prepared by Kaiming He from Microsoft Research)
Advantages of Depth

Revolution of Depth

- **ILSVRC'15 ResNet**: 3.57%
- **ILSVRC'14 GoogleNet**: 6.7%
- **ILSVRC'14 VGG**: 7.3%
- **ILSVRC'13**: 11.7%
- **ILSVRC'12 AlexNet**: 16.4%
- **ILSVRC'11**: 25.8%
- **ILSVRC'10**: 28.2%

ImageNet Classification top-5 error (%)
Degradation Problem

![Graphs showing training and test error over iterations for 56-layer and 20-layer models.](image)
Possible Causes?

- Vanishing/Exploding Gradients.
- Overfitting
Vanishing/Exploding Gradients
Vanishing Gradients

\[ \frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4} \]

\[ \frac{\partial C}{\partial b_1} = \sigma'(z_1) w_2 \sigma'(z_2) w_3 \sigma'(z_3) w_4 \sigma'(z_4) \frac{\partial C}{\partial a_4} \]

\[ |w_k| < 1 \quad |\sigma'w_k| < \frac{1}{4} \]

Gradients in the first layer become very small

\[ \frac{\partial C}{\partial b_3} = \sigma'(z_3) w_4 \sigma'(z_4) \frac{\partial C}{\partial a_4} \]

http://neuralnetworksanddeeplearning.com/
Exploding Gradients

\[
\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}
\]

\[|w_k| = 100 \quad \frac{\partial C}{\partial b_1} \approx 25\]

Gradients in the first layer become very large

http://neuralnetworksanddeeplearning.com/
Batch Normalization (1)

- Addresses the problem of vanishing/exploding gradients.
- Increases learning speed and solves many other problems.
- Each activation in every iteration each layer is normalized to have zero mean and variance 1 over a minibatch.
- Integrated into back-propagation algorithm.
Batch Normalization

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_1, \ldots, x_m\}$;  
Parameters to be learned: $\gamma, \beta$

**Output:** $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

- $\mu_\mathcal{B} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \quad \text{// mini-batch mean}$
- $\sigma^2_\mathcal{B} \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \quad \text{// mini-batch variance}$
- $\hat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma^2_\mathcal{B} + \epsilon}} \quad \text{// normalize}$
- $y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad \text{// scale and shift}$

BN solves vanishing Gradients(1)

\[
\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}
\]

\[z_i = \sigma(BN(Wz_{i-1} + b_i))\]

Now \(u\) is \(z\):

\[BN(Wu) = BN((aW)u)\]

\[
\frac{\partial BN((aW)u)}{\partial u} = \frac{\partial BN(Wu)}{\partial u}
\]

\[
\frac{\partial BN((aW)u)}{\partial (aW)} = \frac{1}{a} \cdot \frac{\partial BN(Wu)}{\partial W}
\]

Motivation

- A shallower model (18 layers)
- A deeper counterpart (34 layers)

- Richer solution space
- A deeper model should not have higher training error
  - A solution by construction:
    - Original layers: copied from a learned shallower model
    - Extra layers: set as identity
    - At least the same training error
- Optimization difficulties: solvers cannot find the solution when going deeper...
Residual Learning

- 2 weight layers fit $F(X) = H(x) - X$ instead of $F(X)$.
- Networks might have “difficulties” fitting the unity mapping.
Very Deep Networks

- Two Similar approaches.
- ResNet.
y = H(x, W_H) \cdot T(x, W_T) + x \cdot C(x, W_C).

C(x, W_C) = 1 - T(x, W_T)

Highway Net vs. ResNet

- The gates C and T are data dependent and have parameters.
- When a gated shortcut is “closed” the layers in highway networks represent non-residual functions.
- High-2 way networks have not demonstrated accuracy gains with depth of over 100 layers.
ResNet Design Rules

- 3 X3 Filters.
- Same number of filters for same feature map size.

- Feature Map Size halved
- Number of Filters doubled

- Almost no hidden max-pooling.
- Not Hidden FC-Layers.
- No Dropout.
ResNet Building Blocks

- Building Blocks are stacked to create full networks.
ResNet and Plain Nets
ResNets Vs. PlainNets (1)

• Deeper ResNets has lower train and test errors!
ResNets Vs. PlainNets(2)

- Deeper ResNets has lower train and test errors!
ResNets Vs. PlainNets

- For 34 layers ResNet has a lower error.
- For 18 layers error is roughly the same but convergence is faster.

<table>
<thead>
<tr>
<th></th>
<th>plain</th>
<th>ResNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>18 layers</td>
<td>27.94</td>
<td>27.88</td>
</tr>
<tr>
<td>34 layers</td>
<td>28.54</td>
<td>25.03</td>
</tr>
</tbody>
</table>
Layer Responses Magnitude

- Responses of 3X3 layers before non-linearity and after BN.
- Residual functions are closer to zero than non-residual.
Exercise 1

• Implement shortcut connections in Keras.

```
Input
32, 3x3 CNN
32, 3x3 CNN
32, 3x3 CNN
32, 3x3 CNN
32, 3x3 CNN
Dense 1
Dense 2
Output

Input
32, 3x3 CNN
32, 3x3 CNN
32, 3x3 CNN
32, 3x3 CNN
32, 3x3 CNN
Dense 1
Dense 2
Output

32, 3x3 CNN
32, 3x3 CNN
32, 3x3 CNN
32, 3x3 CNN
32, 3x3 CNN
Dense 1
Dense 2
Output
```
Dimensionality Change

- Shortcut connections assume dimension equality between input $X$ and output $F(x)$.
- If dimensions do not match:

$$y = \mathcal{F}(x, \{W_i\}) + W_s x.$$
Exploring Different Shortcuts Types

- **Three Options:**
  - **A** - Zero padding for increasing dimensions.
  - **B** – Projection shortcuts for increasing dimensions; others are identity.
  - **C** – All shortcuts are projections.
## Overall Results

<table>
<thead>
<tr>
<th>model</th>
<th>top-1 err.</th>
<th>top-5 err.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16 [41]</td>
<td>28.07</td>
<td>9.33</td>
</tr>
<tr>
<td>GoogLeNet [44]</td>
<td>-</td>
<td>9.15</td>
</tr>
<tr>
<td>plain-34</td>
<td>28.54</td>
<td>10.02</td>
</tr>
<tr>
<td>ResNet-34 A</td>
<td>25.03</td>
<td>7.76</td>
</tr>
<tr>
<td>ResNet-34 B</td>
<td>24.52</td>
<td>7.46</td>
</tr>
<tr>
<td>ResNet-34 C</td>
<td>24.19</td>
<td>7.40</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>22.85</td>
<td>6.71</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>21.75</td>
<td>6.05</td>
</tr>
<tr>
<td>ResNet-152</td>
<td><strong>21.43</strong></td>
<td><strong>5.71</strong></td>
</tr>
</tbody>
</table>
Exercise 2

- Implement projected shortcut connections using CNN.

![Diagram showing two neural networks with shortcut connections using CNN layers.](image)
Uses For Image Detection and localization

• Based on Faster RCNN architecture.
• ResNet-101 architecture is used.
• Obtained best results on MS-COCO, imageNet localization and imageNet Detection datasets.
Conclusions

• Degradation problem is addressed for very deep NN.
• No additional parameter complexity.
• Faster convergence.
• Good for different types of tasks.
• Can be easily trained with existing solvers (Caffe, MatConvNet, etc...).
• Sepp Hochreiter, presumably described the phenomena in 1991.
Bottleneck Architectures

- Bottleneck building block with shortcut identity mapping.
- Similar time complexity to non-bottleneck block.
- 50-layer, 101-layer and 152-layer ResNets were constructed.
## ResNet Implementation Details

<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>7×7, 64, stride 2</td>
<td>3×3 max pool, stride 2</td>
<td>3×3 max pool, stride 2</td>
<td>3×3 max pool, stride 2</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>conv3_x</td>
<td>28×28</td>
<td>[3×3, 128] ×2</td>
<td>[3×3, 128] ×4</td>
<td>[1×1, 128] ×4</td>
<td>[1×1, 128] ×4</td>
<td>[1×1, 128] ×4</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>1×1</td>
<td>average pool, 1000-d fc, softmax</td>
<td>1×1, 2048</td>
<td>1×1, 2048</td>
<td>1×1, 2048</td>
<td>1×1, 2048</td>
</tr>
</tbody>
</table>

FLOPs

1.8×10^9 | 3.6×10^9 | 3.8×10^9 | 7.6×10^9 | 11.3×10^9
Architectures For CIFAR-10

- First layer is 3X3 convolutional.
- 6n layer stacked, 2n layers for each feature map.
- Feature map size halved => number of filters doubled.
- Only identity shortcuts are used.
- Each pair of layers has a shortcut connection.

<table>
<thead>
<tr>
<th>output map size</th>
<th>32×32</th>
<th>16×16</th>
<th>8×8</th>
</tr>
</thead>
<tbody>
<tr>
<td># layers</td>
<td>1+2n</td>
<td>2n</td>
<td>2n</td>
</tr>
<tr>
<td># filters</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
</tbody>
</table>
## Localization and Detection Results

### Detection:

<table>
<thead>
<tr>
<th>method</th>
<th>top-5 localization err</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>val</td>
</tr>
<tr>
<td>OverFeat [40] (ILSVRC’13)</td>
<td>30.0</td>
</tr>
<tr>
<td>GoogLeNet [44] (ILSVRC’14)</td>
<td>-</td>
</tr>
<tr>
<td>VGG [41] (ILSVRC’14)</td>
<td>26.9</td>
</tr>
<tr>
<td>ours (ILSVRC’15)</td>
<td>8.9</td>
</tr>
</tbody>
</table>

### Localization:

<table>
<thead>
<tr>
<th>method</th>
<th>val2</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>GoogLeNet [44] (ILSVRC’14)</td>
<td>-</td>
<td>43.9</td>
</tr>
<tr>
<td>our single model (ILSVRC’15)</td>
<td>60.5</td>
<td>58.8</td>
</tr>
<tr>
<td>our ensemble (ILSVRC’15)</td>
<td>63.6</td>
<td>62.1</td>
</tr>
</tbody>
</table>
### Results on detection MS-COCO

<table>
<thead>
<tr>
<th>training data</th>
<th>COCO train</th>
<th>COCO trainval</th>
<th>COCO val</th>
<th>COCO test-dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>test data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mAP</td>
<td>@.5</td>
<td>@[.5, .95]</td>
<td>@.5</td>
<td>@[.5, .95]</td>
</tr>
<tr>
<td>baseline Faster R-CNN (VGG-16)</td>
<td>41.5</td>
<td>21.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>baseline Faster R-CNN (ResNet-101)</td>
<td>48.4</td>
<td>27.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+box refinement</td>
<td>49.9</td>
<td>29.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+context</td>
<td>51.1</td>
<td>30.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+multi-scale testing</td>
<td>53.8</td>
<td>32.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ensemble</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>59.0</strong></td>
<td><strong>37.4</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Overall Results CIFAR-10

<table>
<thead>
<tr>
<th>method</th>
<th>error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maxout [10]</td>
<td>9.38</td>
</tr>
<tr>
<td>NIN [25]</td>
<td>8.81</td>
</tr>
<tr>
<td>DSN [24]</td>
<td>8.22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>method</th>
<th># layers</th>
<th># params</th>
</tr>
</thead>
<tbody>
<tr>
<td>FitNet [35]</td>
<td>19</td>
<td>2.5M</td>
</tr>
<tr>
<td>Highway [42, 43]</td>
<td>19</td>
<td>2.3M</td>
</tr>
<tr>
<td>Highway [42, 43]</td>
<td>32</td>
<td>1.25M</td>
</tr>
<tr>
<td>ResNet</td>
<td>20</td>
<td>0.27M</td>
</tr>
<tr>
<td>ResNet</td>
<td>32</td>
<td>0.46M</td>
</tr>
<tr>
<td>ResNet</td>
<td>44</td>
<td>0.66M</td>
</tr>
<tr>
<td>ResNet</td>
<td>56</td>
<td>0.85M</td>
</tr>
<tr>
<td>ResNet</td>
<td>110</td>
<td>1.7M</td>
</tr>
<tr>
<td>ResNet</td>
<td>1202</td>
<td>19.4M</td>
</tr>
</tbody>
</table>

- ResNet-110 train five times and results are in the format of “best (mean ± std)”.
- n = {3, 5, 7, 9, 18, 200}.
- For 1202-layer net result degradation due to overfit.
### Overall Results

**ImageNet(2)**

<table>
<thead>
<tr>
<th>method</th>
<th>top-5 err. (test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG [41] (ILSVRC’14)</td>
<td>7.32</td>
</tr>
<tr>
<td>GoogLeNet [44] (ILSVRC’14)</td>
<td>6.66</td>
</tr>
<tr>
<td>VGG [41] (v5)</td>
<td>6.8</td>
</tr>
<tr>
<td>PReLU-net [13]</td>
<td>4.94</td>
</tr>
<tr>
<td>BN-inception [16]</td>
<td>4.82</td>
</tr>
<tr>
<td>ResNet (ILSVRC’15)</td>
<td><strong>3.57</strong></td>
</tr>
</tbody>
</table>