Two (very) Deep Networks

Deep Residual Learning for Image Recognition
K. He, X. Zhang, S. Ren, J. Sun (Microsoft Research, Asia)

Rethinking the Inception Architecture
C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna (Google, UCL)

VGG Reading Group, 4 Feb 2016
Ankush Gupta
Deep Residual Learning
Revolution of Depth

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

Why can’t we just stack layers?

Training error (%) (dashed on right)

Shouldn’t Deep $\geq$ Shallow?

- 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 Mapping

• Instead of relying on the solver to find the identity mapping, explicitly add it in.

• Number of layers range up to 152

• Performance continues to improve with depth

Suppose, we wanted the network to learn a mapping $H(x)$:

Decompose as: $H(x) = F(x) + x$

$=> F(x) = H(x) - x$

$=> F(x)$ is the “residual” mapping

Residual Mapping (2)

- If identity were optimal, easy to set weights as 0
- If optimal mapping is closer to identity, easier to find small fluctuations

Mathematical expression:

\[ H(x) = F(x) + x \]

Network Design

• Inspired by VGG:
  • 3x3 filters
  • Spatial-size/2 => #filters x2

• No max-pooling – use stride-2 convolution instead
• No dropout – use Batch Normalization instead
• No hidden fc – drastically reduces the number of parameters (by 90%)

• 19% of VGG FLOP/forward-pass:
  3.6 billion (ResNet 34 layers) vs. 19.6 billion (VGG-19)
ImageNet Experiments

ImageNet plain nets

solid: test/val
dashed: train

34-layer

18-layer

Deep ResNets can be trained without difficulties
Deeper ResNets have lower training error, and also lower test error

**ImageNet Experiments**

- Deeper ResNets have lower error

<table>
<thead>
<tr>
<th>Model</th>
<th>10-crop Testing, Top-5 Val Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-152</td>
<td>5.7</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>6.1</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>6.7</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>7.4</td>
</tr>
</tbody>
</table>

Rethinking the Inception Architecture
“Inception” inspiration...

Network in network

We need to go deeper
Four Design Principles
Design Principle #1

Avoid representational bottlenecks early on.

Representation size should gently decrease from input to output.
Use higher dimensional representations

Disentangled features => faster training
Reduce the representation dimension before convolution

This is also used in Residual nets: see pg. 6, Figure 5, section “Deeper Bottleneck Architectures”
Design Principle #4

Balance width (#filters) and depth (#layers)

Increase them together
Making Inception Efficient
The Inception Module

- Capture **spatial correlations at multiple scales**
- 1x1 convolutions to reduce dimensions (efficiency)

Recommendations
Factorizing into Smaller Conv.

- Reduce parameters by factorizing 5x5 into sequence of two 3x3 convolutions: same receptive field ( = (9+9)/25 = 28% savings)
- Use ReLU in b/w
Factorizing into Asymmetric Conv.

- Reduce parameters by approximating $nxn$ conv. using two $nx1$ and $1xn$ convolutions: $2n/n^2$ savings. (33% cheaper for $n=3$)

- Does not work in early layers. Good in later layers.
Auxiliary Classifiers

- Originally introduced to strengthen the back-propagation signal
- Does not work for early layers
- Helps convergence towards the end
Other Optimizations

• Reducing spatial size of features: use **stride-2** convolutions and max-pooling

• Do label smoothing:
  – encourage model to be less confident (reduce over-fitting and smoother gradients)
  – improves top-1 and top-5 error by 0.2% (absolute)
That’s It!