Speeding up Deep Neural Networks with Adaptive Computation and Efficient Multi-Scale Architectures

Rogerio Schmidt Feris
Principal Research Scientist and Manager
IBM Research & MIT-IBM Watson AI Lab
I thought I would die without seeing...
... these results!

ImageNet Top-5 Error
Better Results $\Rightarrow$ More Complexity

ImageNet Top-5 Error

<table>
<thead>
<tr>
<th>Year</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>0.28</td>
</tr>
<tr>
<td>2011</td>
<td>0.26</td>
</tr>
<tr>
<td>2012</td>
<td>0.16</td>
</tr>
<tr>
<td>2013</td>
<td>0.12</td>
</tr>
<tr>
<td>2014</td>
<td>0.07</td>
</tr>
<tr>
<td>2015</td>
<td>0.036</td>
</tr>
<tr>
<td>2016</td>
<td>0.03</td>
</tr>
<tr>
<td>2017</td>
<td>0.023</td>
</tr>
</tbody>
</table>

Number of Network Layers
- Shallow
- 10
- 20
- 100+
Many applications require real-time inferencing
This talk: Speeding up Deep Neural Networks

- Adaptive Computation

- Efficient Multi-Scale Architectures
Feed-Forward Convolutional Neural Networks

Adapted from Veit et al
Feed-Forward Convolutional Neural Networks

What happens when we delete a step?

Adapted from Veit et al
Feed-Forward Convolutional Neural Networks

Adapted from Veit et al
What happens if we delete a layer at test time?

Adapted from Veit et al
What happens if we delete a layer at test time?

Adapted from Veit et al
Why does this happen?
Why does this happen?

The unraveled view is equivalent and showcases the many paths in ResNet.
Deletion of a Layer

Adapted from Veit et al
Deletion of a Layer

All paths are affected

Only half of the paths are affected

Adapted from Veit et al
Can we delete a sequence of layers without performance drop?

This experiment [Veit et al, 2016]:
- Layers were dropped randomly
- Global dropping strategy for all images

Performance varies smoothly when deleting several layers.
BlockDrop: Dynamic Inference Paths in Residual Networks

Zuxuan Wu*, Tushar Nagarajan*, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, Rogerio Feris

CVPR 2018

* Authors contributed equally
Do we really need to run 100+ layers / residual blocks of a neural network if we have an “easy” input image?

[Wu & Nagarajan et al, CVPR 2018]
“Dropping some blocks during testing doesn’t hurt performance much”
(Veit et al., NIPS 16)
How to determine which blocks to drop depending on the input image?

[Wu & Nagarajan et al, CVPR 2018]
Our Idea: BlockDrop

“Predict which blocks to drop conditioned on the input image, in one shot, without compromising accuracy”
BlockDrop: Dynamic Inference Paths in Residual Networks

[Wu & Nagarajan et al, CVPR 2018]
BlockDrop: Dynamic Inference Paths in Residual Networks [CVPR 2018]

Policy Network Training through Reinforcement Learning

[Wu & Nagarajan et al, CVPR 2018]
BlockDrop: Dynamic Inference Paths in Residual Networks

- Reward function takes into account both accuracy and block usage

\[
R(u) = \begin{cases} 
1 - \left( \frac{|u|_0}{K} \right)^2 & \text{if correct} \\
-\gamma & \text{otherwise.}
\end{cases}
\]

\[R(u) = 1 - \left( \frac{8}{16} \right)^2 = 0.75\]

[Wu & Nagarajan et al, CVPR 2018]
BlockDrop: Dynamic Inference Paths in Residual Networks

\[
R(u) = \begin{cases} 
1 - \left( \frac{|u|_0}{K} \right)^2 & \text{if correct} \\
-\gamma & \text{otherwise.}
\end{cases}
\]

\[
R(u) = 1 - \left( \frac{8}{16} \right)^2 = 0.75
\]

\[
R(u) = -10
\]

[Wu & Nagarajan et al, CVPR 2018]
BlockDrop: Dynamic Inference Paths in Residual Networks

Results on ImageNet:

20% - 36% computational savings (FLOPs)

[Wu & Nagarajan et al, CVPR 2018]
BlockDrop: Dynamic Inference Paths in Residual Networks

[Wu & Nagarajan et al, CVPR 2018]
Block usage in neural networks agrees with our perception of **difficulty**

[Wu & Nagarajan et al, CVPR 2018]
Extension of BlockDrop: Adaptive Computation for Transfer Learning
Data Efficiency: Transfer Learning

- Fine-tuning is arguably the most widely used approach for transfer learning.

- Existing methods are ad-hoc in terms of determining where to fine-tune in a deep neural network (e.g., fine-tuning last k layers).

- We propose SpotTune, a method that automatically decides, per training example, which layers of a pre-trained model should have their parameters frozen (shared with the source domain) or fine-tuned (adapted to the target domain).
Source Task: Transfer pre-trained parameters to new task

Target Task: Which layers to freeze and which layers to fine-tune? (per instance)

Training Example:
- Freeze
- Freeze
- Fine-tune
- Fine-tune

Training Example:
- Fine-tune
- Freeze
- Fine-tune
- Freeze

[Guo et al, CVPR 2019]
SpotTune: Transfer Learning through Adaptive Fine-Tuning

[Guo et al, CVPR 2019]
SpotTune: Transfer Learning through Adaptive Fine-Tuning

Fine-Tuning Policy Visualization

SpotTune automatically identifies the right fine-tuning policy for each dataset, for each training example.

[Guo et al, CVPR 2019]
SpotTune: Transfer Learning through Adaptive Fine-Tuning

<table>
<thead>
<tr>
<th>Model</th>
<th>#par</th>
<th>ImNet</th>
<th>Airc.</th>
<th>C100</th>
<th>DPed</th>
<th>DTD</th>
<th>GTSR</th>
<th>Flwr</th>
<th>OGl</th>
<th>SVHN</th>
<th>UCF</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scratch</td>
<td>10x</td>
<td>59.87</td>
<td>57.10</td>
<td>75.73</td>
<td>91.20</td>
<td>37.77</td>
<td>96.55</td>
<td>56.30</td>
<td>88.74</td>
<td>96.63</td>
<td>43.27</td>
<td>1625</td>
</tr>
<tr>
<td>Scratch+ [37]</td>
<td>11x</td>
<td>59.67</td>
<td>59.59</td>
<td>76.08</td>
<td>92.45</td>
<td>39.63</td>
<td>96.90</td>
<td>56.66</td>
<td>88.74</td>
<td>96.78</td>
<td>44.17</td>
<td>1826</td>
</tr>
<tr>
<td>Feature Extractor</td>
<td>1x</td>
<td>59.67</td>
<td>23.31</td>
<td>63.11</td>
<td>80.33</td>
<td>55.53</td>
<td>68.18</td>
<td>73.69</td>
<td>58.79</td>
<td>43.54</td>
<td>26.80</td>
<td>544</td>
</tr>
<tr>
<td>Fine-tuning [38]</td>
<td>10x</td>
<td>60.32</td>
<td>61.87</td>
<td>82.12</td>
<td>92.82</td>
<td>55.53</td>
<td>99.42</td>
<td>81.41</td>
<td>89.12</td>
<td>96.55</td>
<td>51.20</td>
<td>3096</td>
</tr>
<tr>
<td>BN Adapt. [5]</td>
<td>1x</td>
<td>59.87</td>
<td>43.05</td>
<td>78.62</td>
<td>92.07</td>
<td>51.60</td>
<td>95.82</td>
<td>74.14</td>
<td>84.83</td>
<td>94.10</td>
<td>43.51</td>
<td>1353</td>
</tr>
<tr>
<td>LwF [26]</td>
<td>10x</td>
<td>59.87</td>
<td>61.15</td>
<td>82.23</td>
<td>92.34</td>
<td>58.83</td>
<td>97.57</td>
<td>83.05</td>
<td>88.08</td>
<td>96.10</td>
<td>50.04</td>
<td>2515</td>
</tr>
<tr>
<td>Series Res. adapt. [37]</td>
<td>2x</td>
<td>60.32</td>
<td>61.87</td>
<td>81.22</td>
<td>93.88</td>
<td>57.13</td>
<td>99.27</td>
<td>81.67</td>
<td>89.62</td>
<td>96.57</td>
<td>50.12</td>
<td>3159</td>
</tr>
<tr>
<td>Parallel Res. adapt. [38]</td>
<td>2x</td>
<td>60.32</td>
<td>64.21</td>
<td>81.92</td>
<td>94.73</td>
<td>58.38</td>
<td>99.38</td>
<td>84.68</td>
<td>89.21</td>
<td>96.54</td>
<td>50.94</td>
<td>3412</td>
</tr>
<tr>
<td>Res. adapt. (large) [37]</td>
<td>12x</td>
<td>67.00</td>
<td>67.69</td>
<td>84.69</td>
<td>94.28</td>
<td>59.41</td>
<td>97.43</td>
<td>84.86</td>
<td>89.92</td>
<td>96.59</td>
<td>52.39</td>
<td>3131</td>
</tr>
<tr>
<td>Res. adapt. decay [37]</td>
<td>2x</td>
<td>59.67</td>
<td>61.87</td>
<td>81.20</td>
<td>93.88</td>
<td>57.13</td>
<td>97.57</td>
<td>81.67</td>
<td>89.62</td>
<td>96.13</td>
<td>50.12</td>
<td>2621</td>
</tr>
<tr>
<td>Res. adapt. finetune all [37]</td>
<td>2x</td>
<td>59.23</td>
<td>63.73</td>
<td>81.31</td>
<td>93.30</td>
<td>57.02</td>
<td>97.47</td>
<td>83.43</td>
<td>89.82</td>
<td>96.17</td>
<td>50.28</td>
<td>2643</td>
</tr>
<tr>
<td>DAN [39]</td>
<td>2x</td>
<td>57.74</td>
<td>64.12</td>
<td>80.07</td>
<td>91.30</td>
<td>56.54</td>
<td>98.46</td>
<td>86.05</td>
<td>89.67</td>
<td>96.77</td>
<td>49.48</td>
<td>2851</td>
</tr>
<tr>
<td>PiggyBack [31]</td>
<td>1.28x</td>
<td>57.69</td>
<td>65.29</td>
<td>79.87</td>
<td>96.99</td>
<td>57.45</td>
<td>97.27</td>
<td>79.09</td>
<td>87.63</td>
<td>97.24</td>
<td>47.48</td>
<td>2838</td>
</tr>
<tr>
<td>SpotTune</td>
<td>11x</td>
<td>60.32</td>
<td>63.91</td>
<td>80.48</td>
<td>96.49</td>
<td>57.13</td>
<td>99.52</td>
<td>85.22</td>
<td>88.84</td>
<td>96.72</td>
<td>52.34</td>
<td>3612</td>
</tr>
</tbody>
</table>

SpotTune sets the new state of the art on the Visual Decathlon Challenge

[Guo et al, CVPR 2019]
This talk: Speeding up Deep Neural Networks

- Adaptive Computation

- Efficient Multi-Scale Architectures
Multi-Scale Feature Representations

Wavelets [Daubechies/Mallat/etc. 90s]

SIFT Features [Lowe, 1996]

MS-CNN [Cai et al, 2016]

Feature Pyramid Networks [Lin et al, 2017]

MSDNet [Huang et al, 2018]

Many more!
Problem

- Image processing at multiple resolutions usually leads to additional computational time

→ How to design an efficient multi-scale network architecture?

- Goal: Speed up inferencing while maintaining accuracy
Big-Little Net

- A multi-branch network that:

  1) has different computation complexities for each branch/scale

  2) fuses different scales at multiple levels of the network

  in order to achieve the best accuracy-efficiency trade-off

[Chen et al, ICLR 2019]
Big-Little Net

Big-Branch: *expensive* network on low-res

Little-Branch: *efficient* network on high-res

I: input; M: merge operator
S: original resolution of input, S/2: half resolution of input

[Chen et al, ICLR 2019]
Two parameters control the complexity of the Little Branch: \( \alpha \) (network width) and \( \beta \) (network depth)
Experimental Results

- **Image Classification:**
  - Dataset: ImageNet-1K
  - Backbone network: ResNet or ResNeXt

- **Speech Recognition:**
  - Dataset: Switchboard
  - Backbone network: ResNet

[Chen et al, ICLR 2019]
## Experimental Results: ImageNet

<table>
<thead>
<tr>
<th>Model (bL-model, $\alpha=2$, $\beta=4$)</th>
<th>Top-1 Error</th>
<th>FLOPs $(10^9)$</th>
<th>Params $(10^6)$</th>
<th>GPU speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-101</td>
<td>21.95%</td>
<td>7.80</td>
<td>44.54</td>
<td>-</td>
</tr>
<tr>
<td>bL-ResNet-101</td>
<td><strong>21.80%</strong></td>
<td><strong>3.89 (2.01×)</strong></td>
<td>41.85</td>
<td><strong>1.33×</strong></td>
</tr>
<tr>
<td>ResNet-152</td>
<td>21.51%</td>
<td>11.51</td>
<td>60.19</td>
<td>-</td>
</tr>
<tr>
<td>bL-ResNet-152</td>
<td><strong>21.16%</strong></td>
<td><strong>5.04 (2.28×)</strong></td>
<td>57.36</td>
<td><strong>1.49×</strong></td>
</tr>
<tr>
<td>ResNeXt-50 (32×4d)</td>
<td>22.20%</td>
<td>4.23</td>
<td>25.03</td>
<td>-</td>
</tr>
<tr>
<td>bL-ResNeXt-50 (32×4d)</td>
<td><strong>21.60%</strong></td>
<td><strong>3.03 (1.40×)</strong></td>
<td>25.03</td>
<td><strong>1.26×</strong></td>
</tr>
<tr>
<td>ResNeXt-101 (32×4d)</td>
<td>21.20%</td>
<td>7.97</td>
<td>44.17</td>
<td>-</td>
</tr>
<tr>
<td>bL-ResNeXt-101 (32×4d)</td>
<td><strong>21.08%</strong></td>
<td><strong>4.08 (1.95×)</strong></td>
<td>41.51</td>
<td><strong>1.59×</strong></td>
</tr>
<tr>
<td>ResNeXt-101 (64×4d)</td>
<td>20.73%</td>
<td>15.46</td>
<td>83.46</td>
<td>-</td>
</tr>
<tr>
<td>bL-ResNeXt-101 (64×4d)</td>
<td><strong>20.48%</strong></td>
<td><strong>7.14 (2.17×)</strong></td>
<td>77.36</td>
<td><strong>1.98×</strong></td>
</tr>
<tr>
<td>SEResNeXt-50 (32×4d)</td>
<td>21.78%</td>
<td>4.23</td>
<td>27.56</td>
<td>-</td>
</tr>
<tr>
<td>bL-SEResNeXt-50 (32×4d)</td>
<td><strong>21.44%</strong></td>
<td><strong>3.03 (1.40×)</strong></td>
<td>28.77</td>
<td><strong>1.33×</strong></td>
</tr>
<tr>
<td>SEResNeXt-101 (32×4d)</td>
<td>21.00%</td>
<td>7.97</td>
<td>48.96</td>
<td>-</td>
</tr>
<tr>
<td>bL-SEResNeXt-101 (32×4d)</td>
<td><strong>20.87%</strong></td>
<td><strong>4.08 (1.95×)</strong></td>
<td>45.88</td>
<td><strong>1.60×</strong></td>
</tr>
</tbody>
</table>

[Chen et al, ICLR 2019]
Experimental Results: Comparison with CNNs based on ResNet and ResNeXt on ImageNet

[Chen et al, ICLR 2019]
Experimental Results: Comparison with SOTA networks in accuracy and GPU runtime on ImageNet
## Experimental Results: Speech Recognition

### Dataset: Switchboard

<table>
<thead>
<tr>
<th>Model</th>
<th>FLOPs ($10^9$)</th>
<th>Params ($10^6$)</th>
<th>WER Avg</th>
<th>Hub5</th>
<th>Hub5 CH</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-22</td>
<td>1.11</td>
<td>3.02</td>
<td>14.67%</td>
<td>11.15%</td>
<td>18.17%</td>
</tr>
<tr>
<td>bL-ResNet-22 ($\alpha=4$, $\beta=1$)</td>
<td>0.68</td>
<td>3.15</td>
<td>14.72%</td>
<td>11.24%</td>
<td>18.18%</td>
</tr>
<tr>
<td>bL-ResNet-22 ($\alpha=4$, $\beta=2$)</td>
<td>0.66</td>
<td>3.11</td>
<td>14.47%</td>
<td><strong>10.95%</strong></td>
<td>17.95%</td>
</tr>
<tr>
<td>bL-ResNet-22 ($\alpha=4$, $\beta=3$)</td>
<td><strong>0.65</strong></td>
<td>3.10</td>
<td>14.66%</td>
<td>11.25%</td>
<td>18.05%</td>
</tr>
<tr>
<td>bL-ResNet-22 ($\alpha=2$, $\beta=3$)</td>
<td>0.77</td>
<td>3.07</td>
<td><strong>14.46%</strong></td>
<td>11.10%</td>
<td><strong>17.80%</strong></td>
</tr>
</tbody>
</table>

[Chen et al, ICLR 2019]
Recent work related to Big-Little Net

Drop an Octave [Chen et al, CVPR 2019]

SlowFast Networks [Feichtenhofer et al, 2019]
Summary

- Adaptive Computation: BlockDrop

What’s Next?

- Big-Little Net with dynamic scale selection
- Neural architecture search: compact multi-task networks using Gumbel-Softmax
- Extension to Video Understanding
Thank you!

