Learning Transferable Architectures for Scalable Image Recognition

Google Brain

Presented by Vardaan Pahuja
Motivation

- Classical Machine Learning vs Deep Learning
- Classical Machine Learning -> Feature Selection
Motivation

- Classical Machine Learning vs Deep Learning
- Classical Machine Learning -> **Feature Selection**
- Deep Learning -> **Model engineering**
Motivation

- Neural Architecture Search
- Automatically searching for new architectures
Question

- How do researchers come up with new neural architectures?
Question

- How do researchers come up with new neural architectures?

Challenges

- Many hyper-parameters
- Combinatorial choices in terms of hyper-parameters
  - No. of hidden units
  - Activation function
  - LayerNorm/BatchNorm etc.
- Which and how many layers to stack?
- Huge time investment to train and evaluate new architectures
This Work

- Automatically search for neural architectures
Why is NAS promising?

- Many neural architectures consist of repetitions of a set of basic modules (aka "cells").
- Can we automate the process of selection of these cells?
Why is NAS promising?

- Many neural architectures consist of repetitions of a set of basic modules (aka "cells").
- Can we automate the process of selection of these cells?
The controller (RNN) has two steps to generate an architecture

- **Sample an architecture**
  - At each timestep of RNN, the softmax gives prob. over all possible values of a particular hyper-parameter.
  - Product of these probs gives the likelihood of choosing that architecture

![Diagram](image)

**Figure 1.** Overview of Neural Architecture Search [70]. A controller RNN predicts architecture $A$ from a search space with probability $p$. A child network with architecture $A$ is trained to convergence achieving accuracy $R$. Scale the gradients of $p$ by $R$ to update the RNN controller.
Controller

The controller (RNN) has two steps to generate an architecture

- **Sample an architecture**
  - At each timestep of RNN, the softmax gives prob. over all possible values of a particular hyper-parameter.
  - Product of these probs gives the likelihood of choosing that architecture

- **Train the “child” network**
  - Using a Reinforcement learning technique – Proximal policy Optimization.

Figure 1. Overview of Neural Architecture Search [70]. A controller RNN predicts architecture $A$ from a search space with probability $p$. A child network with architecture $A$ is trained to convergence achieving accuracy $R$. Scale the gradients of $p$ by $R$ to update the RNN controller.
Controller

- Iteratively choose each hyperparameter in a “block”.

Figure 3. Controller model architecture for recursively constructing one block of a convolutional cell. Each block requires selecting 5 discrete parameters, each of which corresponds to the output of a softmax layer. Example constructed block shown on right. A convolutional cell contains $B$ blocks, hence the controller contains $5B$ softmax layers for predicting the architecture of a convolutional cell. In our experiments, the number of blocks $B$ is 5.
Controller Mechanism

- Each cell receives as input two initial hidden states which are the outputs of two cells in previous two lower layers or the input image.
- The controller RNN recursively predicts the rest of the structure of the convolutional cell, given these two initial hidden states.
- The controller has 5B predictions for each cell and 2*5B predictions in total.
- The same cell is replicated throughout the CNN.
- Stopping criterion: A predetermined no. of architectures have been sampled.
Controller Mechanism Summary

- **Step 1:** Select a hidden state from $h_i$; $h_{i-1}$ or from the set of hidden states created in previous blocks.
Controller Mechanism Summary

- **Step 1**: Select a hidden state from $h_i; h_{i-1}$ or from the set of hidden states created in previous blocks.
- **Step 2**: Select a second hidden state from the same options as in Step 1.
Controller Mechanism Summary

- **Step 1**: Select a hidden state from $h_i$; $h_{i-1}$ or from the set of hidden states created in previous blocks.

- **Step 2**: Select a second hidden state from the same options as in Step 1.

- **Step 3**: Select an operation to apply to the hidden state selected in Step 1.
Controller Mechanism Summary

- **Step 1:** Select a hidden state from $h_i; h_{i-1}$ or from the set of hidden states created in previous blocks.
- **Step 2:** Select a second hidden state from the same options as in Step 1.
- **Step 3:** Select an operation to apply to the hidden state selected in Step 1.
- **Step 4:** Select an operation to apply to the hidden state selected in Step 2.
Controller Mechanism Summary

- **Step 1**: Select a hidden state from $h_i; h_{i-1}$ or from the set of hidden states created in previous blocks.
- **Step 2**: Select a second hidden state from the same options as in Step 1.
- **Step 3**: Select an operation to apply to the hidden state selected in Step 1.
- **Step 4**: Select an operation to apply to the hidden state selected in Step 2.
- **Step 5**: Select a method to combine the outputs of Step 3 and 4 to create a new hidden state.
Best Architectures found

• **Normal Cell:** convolutional cells that return a feature map of the same dimension

• **Reduction Cell:** convolutional cells that return a feature map where the feature map height and width is reduced by a factor of two.

• Three best architectures – NASNet-A, B, C.
Distributed Training

- Child worker pool size = 450,
- There are 450 networks being trained on 450 GPUs concurrently at any time.
- Upon receiving enough child model training results, the controller RNN will perform a gradient update and then sample another batch of architectures that go into the global work queue.
- Training takes 4 days
Datasets

- Train NASNet on CIFAR-10.
- Evaluate on ImageNet
  - Only difference is the no. of cells increases due to different image size
  - Avoids huge computational cost of training on ImageNet.
Results on CIFAR-10 dataset

<table>
<thead>
<tr>
<th>model</th>
<th>depth</th>
<th># params</th>
<th>error rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DenseNet ($L = 40, k = 12$) [26]</td>
<td>40</td>
<td>1.0M</td>
<td>5.24</td>
</tr>
<tr>
<td>DenseNet ($L = 100, k = 12$) [26]</td>
<td>100</td>
<td>7.0M</td>
<td>4.10</td>
</tr>
<tr>
<td>DenseNet ($L = 100, k = 24$) [26]</td>
<td>100</td>
<td>27.2M</td>
<td>3.74</td>
</tr>
<tr>
<td>DenseNet-BC ($L = 100, k = 40$) [26]</td>
<td>190</td>
<td>25.6M</td>
<td>3.46</td>
</tr>
<tr>
<td>Shake-Shake 26 2x32d [18]</td>
<td>26</td>
<td>2.9M</td>
<td>3.55</td>
</tr>
<tr>
<td>Shake-Shake 26 2x96d [18]</td>
<td>26</td>
<td>26.2M</td>
<td>2.86</td>
</tr>
<tr>
<td>Shake-Shake 26 2x96d + cutout [12]</td>
<td>26</td>
<td>26.2M</td>
<td>2.56</td>
</tr>
<tr>
<td>NAS v3 [70]</td>
<td>39</td>
<td>7.1M</td>
<td>4.47</td>
</tr>
<tr>
<td>NAS v3 [70]</td>
<td>39</td>
<td>37.4M</td>
<td>3.65</td>
</tr>
<tr>
<td>NASNet-A (6 @ 768)</td>
<td>-</td>
<td>3.3M</td>
<td>3.41</td>
</tr>
<tr>
<td>NASNet-A (6 @ 768) + cutout</td>
<td>-</td>
<td>3.3M</td>
<td>2.65</td>
</tr>
<tr>
<td>NASNet-A (7 @ 2304)</td>
<td>-</td>
<td>27.6M</td>
<td>2.97</td>
</tr>
<tr>
<td>NASNet-A (7 @ 2304) + cutout</td>
<td>-</td>
<td>27.6M</td>
<td>2.40</td>
</tr>
<tr>
<td>NASNet-B (4 @ 1152)</td>
<td>-</td>
<td>2.6M</td>
<td>3.73</td>
</tr>
<tr>
<td>NASNet-C (4 @ 640)</td>
<td>-</td>
<td>3.1M</td>
<td>3.59</td>
</tr>
</tbody>
</table>
Results on ImageNet dataset

<table>
<thead>
<tr>
<th>Model</th>
<th>image size</th>
<th># parameters</th>
<th>Mult-Adds</th>
<th>Top 1 Acc. (%)</th>
<th>Top 5 Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception V2 [29]</td>
<td>224×224</td>
<td>11.2 M</td>
<td>1.94 B</td>
<td>74.8</td>
<td>92.2</td>
</tr>
<tr>
<td>NASNet-A (5 @ 1538)</td>
<td>299×299</td>
<td>10.9 M</td>
<td>2.35 B</td>
<td>78.6</td>
<td>94.2</td>
</tr>
<tr>
<td>Inception V3 [59]</td>
<td>299×299</td>
<td>23.8 M</td>
<td>5.72 B</td>
<td>78.0</td>
<td>93.9</td>
</tr>
<tr>
<td>Xception [9]</td>
<td>299×299</td>
<td>22.8 M</td>
<td>8.38 B</td>
<td>79.0</td>
<td>94.5</td>
</tr>
<tr>
<td>Inception ResNet V2 [57]</td>
<td>299×299</td>
<td>55.8 M</td>
<td>13.2 B</td>
<td>80.4</td>
<td>95.3</td>
</tr>
<tr>
<td>NASNet-A (7 @ 1920)</td>
<td>299×299</td>
<td>22.6 M</td>
<td>4.93 B</td>
<td>80.8</td>
<td>95.3</td>
</tr>
<tr>
<td>ResNeXt-101 (64 x 4d) [67]</td>
<td>320×320</td>
<td>83.6 M</td>
<td>31.5 B</td>
<td>80.9</td>
<td>95.6</td>
</tr>
<tr>
<td>PolyNet [68]</td>
<td>331×331</td>
<td>92 M</td>
<td>34.7 B</td>
<td>81.3</td>
<td>95.8</td>
</tr>
<tr>
<td>DPN-131 [8]</td>
<td>320×320</td>
<td>79.5 M</td>
<td>32.0 B</td>
<td>81.5</td>
<td>95.8</td>
</tr>
<tr>
<td>SENet [25]</td>
<td>320×320</td>
<td>145.8 M</td>
<td>42.3 B</td>
<td>82.7</td>
<td>96.2</td>
</tr>
<tr>
<td>NASNet-A (6 @ 4032)</td>
<td>331×331</td>
<td>88.9 M</td>
<td>23.8 B</td>
<td>82.7</td>
<td>96.2</td>
</tr>
</tbody>
</table>
Downstream tasks: Object Detection

- Use NASNet-A with the Faster-RCNN framework for object detection.
- Dataset: MS-COCO

<table>
<thead>
<tr>
<th>Model</th>
<th>resolution</th>
<th>mAP (mini-val)</th>
<th>mAP (test-dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet-224 [24]</td>
<td>600 × 600</td>
<td>19.8%</td>
<td>-</td>
</tr>
<tr>
<td>ShuffleNet (2x) [69]</td>
<td>600 × 600</td>
<td>24.5%(^1)</td>
<td>-</td>
</tr>
<tr>
<td>NASNet-A (4 @ 1056)</td>
<td>600 × 600</td>
<td>29.6%</td>
<td>-</td>
</tr>
<tr>
<td>ResNet-101-FPN [35]</td>
<td>800 (short side)</td>
<td>-</td>
<td>36.2%</td>
</tr>
<tr>
<td>Inception-ResNet-v2 (G-RMI) [28]</td>
<td>600 × 600</td>
<td>35.7%</td>
<td>35.6%</td>
</tr>
<tr>
<td>Inception-ResNet-v2 (TDM) [51]</td>
<td>600 × 1000</td>
<td>37.3%</td>
<td>36.8%</td>
</tr>
<tr>
<td>NASNet-A (6 @ 4032)</td>
<td>800 × 800</td>
<td>41.3%</td>
<td>40.7%</td>
</tr>
<tr>
<td>NASNet-A (6 @ 4032)</td>
<td>1200 × 1200</td>
<td>43.2%</td>
<td>43.1%</td>
</tr>
<tr>
<td>ResNet-101-FPN (RetinaNet) [36]</td>
<td>800 (short side)</td>
<td>-</td>
<td>39.1%</td>
</tr>
</tbody>
</table>
Question

- How would you design a neural architecture search space for RNN-like architectures?
Deep Speech: Scaling up end-to-end speech recognition

Baidu Research

Presented by Vardaan Pahuja
Motivation

- Traditional Speech recognition systems
  - Heavy feature engineering
  - Hidden Markov Models (HMMs)
Motivation

- Traditional Speech recognition systems
  - Heavy feature engineering
  - Hidden Markov Models (HMMs)
- This work
  - End-to-end speech recognition using deep neural networks
Challenges for DL in Speech

- Need large neural models
- Need large training sets to train these models
  - Aligned speech-text transcripts are scarce
Model

Input features: Spectrograms (similar to Fourier transform at every time-step)

Training loss: Connectionist Temporal Classification loss

\[
h_{t,k}^{(6)} = \hat{y}_{t,k} \equiv \mathbb{P}(c_t = k|x) = \frac{\exp(W_k^{(6)} h_t^{(5)} + b_k^{(6)})}{\sum_j \exp(W_j^{(6)} h_t^{(5)} + b_j^{(6)})}
\]

\[
h_t^{(5)} = g(W^{(5)}(h_t^{(f)} + h_t^{(b)}) + b^{(5)})
\]

\[
h_t^{(f)} = g(W^{(4)} h_t^{(3)} + W_r^{(f)} h_{t-1}^{(f)} + b^{(4)})
\]

\[
h_t^{(b)} = g(W^{(4)} h_t^{(3)} + W_r^{(b)} h_{t+1}^{(b)} + b^{(4)})
\]

\[
h_t^{(l)} = g(W^{(l)} h_t^{(l-1)} + b^{(l)})
\]

Figure 1: Structure of our RNN model and notation.
Regularization

- Randomly jitter inputs by translations or reflections
- Feed each jittered version through the network
- Vote or average the results.
Language Model

- Errors made by the RNN in this case tend to be phonetically plausible renderings of English words.

<table>
<thead>
<tr>
<th>RNN output</th>
<th>Decoded Transcription</th>
</tr>
</thead>
<tbody>
<tr>
<td>what is the weather like in bostin right now</td>
<td>what is the weather like in boston right now</td>
</tr>
<tr>
<td>prime miniter nerenr modi</td>
<td>prime minister narendra modi</td>
</tr>
<tr>
<td>arther n tickets for the game</td>
<td>are there any tickets for the game</td>
</tr>
</tbody>
</table>

Table 1: Examples of transcriptions directly from the RNN (left) with errors that are fixed by addition of a language model (right).

- Use N-gram language model to include language constraints in output.

Given the output $P(c|x)$ of our RNN we perform a search to find the sequence of characters $c_1, c_2, \ldots$ that is most probable according to both the RNN output and the language model (where the language model interprets the string of characters as words). Specifically, we aim to find a sequence $c$ that maximizes the combined objective:

$$Q(c) = \log(P(c|x)) + \alpha \log(P_{lm}(c)) + \beta \text{ word\_count}(c)$$
Optimization for efficient training

- 5 Billion parameters
- Data Parallelism
  - Distribute the batch across several GPUs
  - Similar length examples bundled together in a minibatch
Optimization for efficient training

- 5 Billion parameters
- Data Parallelism
  - Distribute the batch across several GPUs
  - Similar length examples bundled together in a minibatch
- Model Parallelism
  - Layers are divided among several GPUs on the time dimension.
  - For Recurrent layers, the first GPU begins computing the forward activations while the second begins computing the backward activations.
  - At the mid-point \((t = T/2)\), the two GPUs exchange the intermediate activations and swap roles
Question

- Is the strategy adopted for model parallelism of recurrent layers more efficient?
- If so, why?
Striding

- Shorten the recurrent layers by taking strides of size 2 in the original input
- The unrolled RNN has half as many steps, thus reducing computational time.
Training data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Type</th>
<th>Hours</th>
<th>Speakers</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSJ</td>
<td>read</td>
<td>80</td>
<td>280</td>
</tr>
<tr>
<td>Switchboard</td>
<td>conversational</td>
<td>300</td>
<td>4000</td>
</tr>
<tr>
<td>Fisher</td>
<td>conversational</td>
<td>2000</td>
<td>23000</td>
</tr>
<tr>
<td>Baidu</td>
<td>read</td>
<td>5000</td>
<td>9600</td>
</tr>
</tbody>
</table>

Training data generation

- Synthesis by superposition
  - Combine clean and noisy audio signals to improve performance in noisy environments
- Capture Lombard effect
  - Speakers actively change the pitch or inflections of their voice to overcome noise around them
  - Induce the Lombard effect intentionally by playing loud background noise through headphones worn by the subject.
Benchmark Datasets

- Hub5'00 (LDC2002S23)
- Splits
  - “easy” (Switchboard)
  - “hard” (CallHome)
### Results

<table>
<thead>
<tr>
<th>Model</th>
<th>SWB</th>
<th>CH</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vesely et al. (GMM-HMM BMMI) [44]</td>
<td>18.6</td>
<td>33.0</td>
<td>25.8</td>
</tr>
<tr>
<td>Vesely et al. (DNN-HMM sMBR) [44]</td>
<td>12.6</td>
<td>24.1</td>
<td>18.4</td>
</tr>
<tr>
<td>Maas et al. (DNN-HMM SWB) [28]</td>
<td>14.6</td>
<td>26.3</td>
<td>20.5</td>
</tr>
<tr>
<td>Maas et al. (DNN-HMM FSH) [28]</td>
<td>16.0</td>
<td>23.7</td>
<td>19.9</td>
</tr>
<tr>
<td>Seide et al. (CD-DNN) [39]</td>
<td>16.1</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Kingsbury et al. (DNN-HMM sMBR HF) [22]</td>
<td>13.3</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Sainath et al. (CNN-HMM) [36]</td>
<td>11.5</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Soltau et al. (MLP/CNN+I-Vector) [40]</td>
<td><strong>10.4</strong></td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td><strong>Deep Speech SWB</strong></td>
<td>20.0</td>
<td>31.8</td>
<td>25.9</td>
</tr>
<tr>
<td><strong>Deep Speech SWB + FSH</strong></td>
<td>12.6</td>
<td><strong>19.3</strong></td>
<td><strong>16.0</strong></td>
</tr>
</tbody>
</table>

Table 3: Published error rates (%WER) on Switchboard dataset splits. The columns labeled “SWB” and “CH” are respectively the easy and hard subsets of Hub5’00.
Evaluation on Noisy Speech

- Evaluation of commercial speech systems on noisy speech samples
- Train on full data and evaluate on small subset of clean and noisy samples.

<table>
<thead>
<tr>
<th>System</th>
<th>Clean (94)</th>
<th>Noisy (82)</th>
<th>Combined (176)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple Dictation</td>
<td>14.24</td>
<td>43.76</td>
<td>26.73</td>
</tr>
<tr>
<td>Bing Speech</td>
<td>11.73</td>
<td>36.12</td>
<td>22.05</td>
</tr>
<tr>
<td>Google API</td>
<td>6.64</td>
<td>30.47</td>
<td>16.72</td>
</tr>
<tr>
<td>wit.ai</td>
<td>7.94</td>
<td>35.06</td>
<td>19.41</td>
</tr>
<tr>
<td><strong>Deep Speech</strong></td>
<td><strong>6.56</strong></td>
<td><strong>19.06</strong></td>
<td><strong>11.85</strong></td>
</tr>
</tbody>
</table>

Table 4: Results (%WER) for 5 systems evaluated on the original audio. Scores are reported only for utterances with predictions given by all systems. The number in parentheses next to each dataset, e.g. Clean (94), is the number of utterances scored.
Thank You