Neural Architecture Search and Beyond

Barret Zoph
Progress in AI

● Generation 1: Good Old Fashioned AI
  ○ Handcraft predictions
  ○ Learn nothing

● Generation 2: Shallow Learning
  ○ Handcraft features
  ○ Learn predictions

● Generation 3: Deep Learning
  ○ Handcraft algorithm (architectures, data processing, ...)
  ○ Learn features and predictions end-to-end

● Generation 4: Learn2Learn (?)
  ○ Handcraft nothing
  ○ Learn algorithm, features and predictions end-to-end
Importance of architectures for Vision

- Designing neural network architectures is hard
- Lots of human efforts go into tuning them
- There is not a lot of intuition into how to design them well
- Can we try and learn good architectures automatically?

Canziani et al, 2017

Two layers from the famous Inception V4 computer vision model.
Szegedy et al, 2017
Convolutional Architectures

Krizhevsky et al, 2012
How does architecture search work?


Uses primitives found in CV Research

Reinforcement Learning or Evolution

Controller

Sample models from search space

Trainer

Accuracy

Reward
How does architecture search work?

Controller: proposes ML models

Train & evaluate models

Iterate to find the most accurate model

20K times
Example: Using reinforcement learning controller (NAS)

Example: Using evolutionary controller

Possible Mutations
- Insert convolution
- Remove convolution
- Insert nonlinearity
- Remove nonlinearity
- Add-skip
- Remove skip
- Alter strides
- Alter number of channels
- Alter horizontal filter size
- Alter vertical filters size
- Alter Learning Rate
- Identity
- Reset weights
ImageNet Neural Architect Search Improvements

![Architecture Search Diagram]

- VGG-16
- Inception
- InceptionV3
- Inception-ResnetV2
- ResNet152
- PolyNet
- DPN131
- NasNetA
- AmoebaNetB
- AmoebaNetC

Google
ImageNet

Architect Search

Old Architectures

MobileNetV3

Tan & Le. EfficientNet: Rethinking Model Scaling for Deep Convolutional Neural Networks, 2019
arxiv.org/abs/1905.11946
Object detection: COCO

Architecture Decisions for Detection Architecture Search

Learn the connections between blocks

Architect

Search

State-of-the-art accuracy


Table 1: State-of-the-art action classification performances on Charades [19].

<table>
<thead>
<tr>
<th>Method</th>
<th>modality</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-Strm. [20] (from [18])</td>
<td>RGB+Flow</td>
<td>18.6</td>
</tr>
<tr>
<td>Asyn-TF [18]</td>
<td>RGB+Flow</td>
<td>22.4</td>
</tr>
<tr>
<td>CoViAR [28]</td>
<td>Compressed</td>
<td>21.9</td>
</tr>
<tr>
<td>MultiScale TRN [33]</td>
<td>RGB</td>
<td>25.2</td>
</tr>
<tr>
<td>I3D [3]</td>
<td>RGB</td>
<td>32.9</td>
</tr>
<tr>
<td>I3D [3] (from [25])</td>
<td>RGB</td>
<td>35.5</td>
</tr>
<tr>
<td>I3D-NL [25]</td>
<td>RGB</td>
<td>37.5</td>
</tr>
<tr>
<td>STRG [26]</td>
<td>RGB</td>
<td>39.7</td>
</tr>
<tr>
<td>LFB [27]</td>
<td>RGB</td>
<td>42.5</td>
</tr>
<tr>
<td>SlowFast [6]</td>
<td>RGB+RGB</td>
<td>45.2</td>
</tr>
<tr>
<td>Two-stream (2+1)D ResNet</td>
<td>RGB+Flow</td>
<td>46.5</td>
</tr>
<tr>
<td>AssembleNet</td>
<td>RGB+Flow</td>
<td>51.6</td>
</tr>
</tbody>
</table>
Translation: WMT

256 input words + 256 output words
So, et al. The Evolved Transformer, 2019,
arxiv.org/abs/1901.11117
Using more convolutions in earlier layers
Platform-aware search

Tan et al., MnasNet: Platform-Aware Neural Architecture Search for Mobile. CVPR, 2019
arxiv.org/abs/1807.11626
Collaboration between Waymo and Google Brain:

- 20–30% lower latency / same quality.
- 8–10% lower error rate / same latency.

‘Interesting’ architectures:

https://medium.com/waymo/automl-automating-the-design-of-machine-learning-models-for-autonomous-driving-141a5583ec2a
Tabular Data

trees, neural nets, 
#layers, activation 
functions, connectivity

Automated Feature Engineering
Automated Architecture Search
Automated Hyper-parameter Tuning
Automated Model Selection
Automated Model Ensembling
Automated Model Distillation and Export for Serving

Normalization, Transformation (log, cosine)

Can distill to decision trees for interpretability

https://ai.googleblog.com/2019/05/an-end-to-end-automl-solution-for.html
Internal Benchmark on Kaggle Competitions

AutoML placed 2nd in a live one-day competition against 76 teams
Problems of NAS

● Enormous compute consumption
  ○ Requires ~10k training trials to coverage on a carefully designed search space
  ○ Not applicable if single trial’s computation is heavy

● Works inefficiently on arbitrary and giant search space
  ○ Feature selection (search space $2^{100}$ if there are 100 features)
  ○ Per feature transform (search space $c^{100}$ if there are 100 features and each has $c$ types of transform)
  ○ Embedding and hidden layer size
Efficient NAS: Addressing the efficiency

Key idea:
1. One path inside a big model is a child model
2. **Controller selects a path** inside a big model and train for a few steps
3. Controller selects another path inside a big model and train for a few steps, **reusing the weights** produced by the previous step
4. Etc.

Results: Can **save 100->1000x compute**

Related works: DARTS, SMASH, One-shot architecture search,

Learning Data Augmentation Procedures

Data → Data Processing → Machine Learning Model

- Very important but manually tuned
- Focus of machine learning research
Data Augmentation

Enlarge your Dataset
AutoAugment Search Algorithm

**Controller**: proposes augmentation policy

**Train & evaluate models with the augmentation policy**

Iterate to find the most accurate policy

20K times

AutoAugment: Example Learned Policy

AutoAugment Learns: \((\text{Operation, Probability, Magnitude})\)

Original

Equalize, 0.4, 4  
Rotate, 0.8, 8

Solarize, 0.6, 3  
Equalize, 0.6, 7

Posterize, 0.8, 5  
Equalize, 1.0, 2

Rotate, 0.2, 3  
Solarize, 0.6, 8

Posterize, 0.4, 6

Probability of applying

Magnitude
AutoAugment: Example Learned Policy

For each Sub-Policy (5 Sub-Policies = Policy): AutoAugment Learns: (Operation, Probability, Magnitude)
## AutoAugment CIFAR Results

### Full CIFAR-10

<table>
<thead>
<tr>
<th>Model</th>
<th>No data aug</th>
<th>Standard data-aug</th>
<th>AutoAugment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wide-ResNet-28-10</td>
<td>3.87</td>
<td>3.08</td>
<td>2.68</td>
</tr>
<tr>
<td>Shake-Shake (26 2x32d)</td>
<td>3.55</td>
<td>3.02</td>
<td>2.47</td>
</tr>
<tr>
<td>Shake-Shake (26 2x96d)</td>
<td>2.86</td>
<td>2.56</td>
<td>1.99</td>
</tr>
<tr>
<td>Shake-Shake (26 2x112d)</td>
<td>2.82</td>
<td>2.57</td>
<td>1.89</td>
</tr>
<tr>
<td>AmoebaNet-B (6,128)</td>
<td>2.98</td>
<td>2.13</td>
<td>1.75</td>
</tr>
<tr>
<td>PyramidNet+ShakeDrop</td>
<td>2.67</td>
<td>2.31</td>
<td><strong>1.48</strong></td>
</tr>
</tbody>
</table>

### CIFAR-100

<table>
<thead>
<tr>
<th>Model</th>
<th>No data aug</th>
<th>Standard data-aug</th>
<th>AutoAugment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wide-ResNet-28-10</td>
<td>18.80</td>
<td>18.41</td>
<td>17.09</td>
</tr>
<tr>
<td>Shake-Shake (26 2x96d)</td>
<td>17.05</td>
<td>16.00</td>
<td>14.28</td>
</tr>
<tr>
<td>PyramidNet+ShakeDrop</td>
<td>13.99</td>
<td>12.19</td>
<td><strong>10.67</strong></td>
</tr>
</tbody>
</table>

*State-of-the-art accuracy*
## AutoAugment ImageNet Results (Top5 error rate)

<table>
<thead>
<tr>
<th>Model</th>
<th>No data augmentation</th>
<th>Standard data augmentation</th>
<th>AutoAugment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50</td>
<td>7.80</td>
<td>6.92</td>
<td>6.18</td>
</tr>
<tr>
<td>ResNet-200</td>
<td>5.85</td>
<td>4.99</td>
<td></td>
</tr>
<tr>
<td>AmoebaNet-B</td>
<td>3.97</td>
<td>3.78</td>
<td></td>
</tr>
<tr>
<td>AmoebaNet-C</td>
<td>3.90</td>
<td></td>
<td><strong>3.52</strong></td>
</tr>
</tbody>
</table>

**Code is opensourced:**
[https://github.com/tensorflow/models/tree/master/research/autoaugment](https://github.com/tensorflow/models/tree/master/research/autoaugment)
Expanded AutoAugment for Object Detection

Zoph et al. 2019, Learning Data Augmentation Strategies for Object Detection, 

Google
Learn Augmentation on COCO Results

<table>
<thead>
<tr>
<th>Method</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>36.7</td>
</tr>
<tr>
<td>baseline + DropBlock [13]</td>
<td>38.4</td>
</tr>
<tr>
<td>Augmentation policy with color operations</td>
<td>37.5</td>
</tr>
<tr>
<td>+ geometric operations</td>
<td>38.6</td>
</tr>
<tr>
<td>+ bbox-only operations</td>
<td><strong>39.0</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Baseline</th>
<th>Our result</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50</td>
<td>36.7</td>
<td>39.0</td>
<td>+2.3</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>38.8</td>
<td>40.4</td>
<td>+1.6</td>
</tr>
<tr>
<td>ResNet-200</td>
<td>39.9</td>
<td>42.1</td>
<td>+2.2</td>
</tr>
</tbody>
</table>
Learn Augmentation on COCO Results

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Change</th>
<th># Scales</th>
<th>mAP</th>
<th>mAP&lt;sub&gt;S&lt;/sub&gt;</th>
<th>mAP&lt;sub&gt;M&lt;/sub&gt;</th>
<th>mAP&lt;sub&gt;L&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>MegDet [32]</td>
<td>multiple</td>
<td></td>
<td>50.5</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>AmoebaNet + NAS-FPN</td>
<td>baseline [14]</td>
<td>1</td>
<td>47.0</td>
<td>30.6</td>
<td>50.9</td>
<td>61.3</td>
</tr>
<tr>
<td></td>
<td>+ learned augmentation</td>
<td>1</td>
<td>48.6</td>
<td>32.0</td>
<td>53.4</td>
<td>62.7</td>
</tr>
<tr>
<td></td>
<td>+ ↑ anchors, ↑ image size</td>
<td>1</td>
<td>50.7</td>
<td>34.2</td>
<td>55.5</td>
<td>64.5</td>
</tr>
</tbody>
</table>

State-of-the-art accuracy at the time for a single model

Code is opensourced:  
https://github.com/tensorflow/tpu/tree/master/models/official/detection
RandAugment: Practical data augmentation with no separate search

Faster AutoAugment w/ vastly reduced search space!

Only two tunable parameters now: Magnitude and Policy Length

RandAugment: Practical data augmentation with no separate search

Match or surpass AA with significantly less cost!

<table>
<thead>
<tr>
<th>Dataset</th>
<th>baseline</th>
<th>PBA</th>
<th>Fast AA</th>
<th>AA</th>
<th>RA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wide-ResNet-28-2</td>
<td>94.9</td>
<td>-</td>
<td>-</td>
<td><strong>95.9</strong></td>
<td>95.8</td>
</tr>
<tr>
<td>Wide-ResNet-28-10</td>
<td>96.1</td>
<td><strong>97.4</strong></td>
<td>97.3</td>
<td><strong>97.4</strong></td>
<td>97.3</td>
</tr>
<tr>
<td>Shake-Shake</td>
<td>97.1</td>
<td><strong>98.0</strong></td>
<td>98.0</td>
<td><strong>98.0</strong></td>
<td>98.0</td>
</tr>
<tr>
<td>PyramidNet</td>
<td>97.3</td>
<td><strong>98.5</strong></td>
<td>98.3</td>
<td><strong>98.5</strong></td>
<td>98.5</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wide-ResNet-28-2</td>
<td>75.4</td>
<td>-</td>
<td>-</td>
<td><strong>78.5</strong></td>
<td>78.3</td>
</tr>
<tr>
<td>Wide-ResNet-28-10</td>
<td>81.2</td>
<td><strong>83.3</strong></td>
<td>82.7</td>
<td>82.9</td>
<td><strong>83.3</strong></td>
</tr>
<tr>
<td>SVHN (core set)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wide-ResNet-28-2</td>
<td>96.7</td>
<td>-</td>
<td>-</td>
<td><strong>98.0</strong></td>
<td>98.3</td>
</tr>
<tr>
<td>Wide-ResNet-28-10</td>
<td>96.9</td>
<td>-</td>
<td>-</td>
<td><strong>98.1</strong></td>
<td>98.3</td>
</tr>
<tr>
<td>SVHN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wide-ResNet-28-2</td>
<td>98.2</td>
<td>-</td>
<td>-</td>
<td><strong>98.7</strong></td>
<td>98.7</td>
</tr>
<tr>
<td>Wide-ResNet-28-10</td>
<td>98.5</td>
<td>98.9</td>
<td>98.8</td>
<td>98.9</td>
<td><strong>99.0</strong></td>
</tr>
</tbody>
</table>
RandAugment: Practical data augmentation with no separate search

<table>
<thead>
<tr>
<th></th>
<th>baseline</th>
<th>Fast AA</th>
<th>AA</th>
<th>RA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50</td>
<td>76.3 / 93.1</td>
<td>77.6 / 93.7</td>
<td>77.6 / 93.8</td>
<td>77.6 / 93.8</td>
</tr>
<tr>
<td>EfficientNet-B5</td>
<td>83.2 / 96.7</td>
<td>-</td>
<td>83.3 / 96.7</td>
<td>83.9 / 96.8</td>
</tr>
<tr>
<td>EfficientNet-B7</td>
<td>84.0 / 96.9</td>
<td>-</td>
<td>84.4 / 97.1</td>
<td>85.0 / 97.2</td>
</tr>
</tbody>
</table>

Can easily scale regularization strength when model size changes!

State-of-the-art accuracy

Code and Models Opensourced:
https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet