Extending the Search from Architecture to Hyperparameter, Hardware, and System

Xuanyi Dong
http://xuanyidong.com
Auto-Architecture vs. Manual Architecture

 Automatically Designed

 Manually Designed
Auto-Architecture vs. Manual Architecture

<table>
<thead>
<tr>
<th>Automatically Designed</th>
<th>Manually Designed</th>
</tr>
</thead>
<tbody>
<tr>
<td>EffNetV2-XL(21k)</td>
<td>NFNet-F4</td>
</tr>
<tr>
<td>L(21k)</td>
<td>ViT-L/16(21k)</td>
</tr>
<tr>
<td>M(21k)</td>
<td>EffNet-B7(repro)</td>
</tr>
<tr>
<td>EffNetV2-L</td>
<td>lambdanet</td>
</tr>
<tr>
<td>F3</td>
<td>botnet</td>
</tr>
<tr>
<td>F0</td>
<td>ResNet-RS</td>
</tr>
<tr>
<td>F1</td>
<td>EffNet-B7</td>
</tr>
<tr>
<td>F2</td>
<td>B6</td>
</tr>
<tr>
<td>M</td>
<td>B5</td>
</tr>
<tr>
<td>B4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>Training time (TPU days)</td>
</tr>
</tbody>
</table>
Neural Architecture Search Grows Fast

The number of NAS papers rapidly increases.
What is Neural Architecture Search (NAS)?

- Search Space
- Child Program
- Search Algorithm

Train a ResNet on ImageNet

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020
What is Neural Architecture Search (NAS)?

Conv in ResNet can be selected from \{3x3, 5x5, sep..\}

**Search Space**

**Child Program**

**Search Algorithm**

Train a ResNet on ImageNet

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020
What is Neural Architecture Search (NAS)?

Conv in ResNet can be selected from \{3x3, 5x5, sep.\}

Search Space

Child Program

Search Algorithm

Train a ResNet on ImageNet

PPO, HyperBand, BOHB

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020
What is Neural Architecture Search (NAS)?

Conv in ResNet can be selected from \{3x3, 5x5, sep..\}

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020
What is important to NAS?

Search Space
- NASNet Search Space
- MBConv Search Space

Search Algorithm
- BayesOpt
- PPO, REINFORCE
- Evolution
- Differentiable Search
NAS’s performance is saturated - Search Space

<table>
<thead>
<tr>
<th>Search Space</th>
<th>Methods</th>
<th>CIFAR-10</th>
<th></th>
<th>CIFAR-100</th>
<th></th>
<th>ImageNet-16-120</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type</td>
<td>validation</td>
<td>test</td>
<td>validation</td>
<td>test</td>
<td>validation</td>
<td>test</td>
</tr>
<tr>
<td>Multi-trial</td>
<td>REA</td>
<td>91.25±0.31</td>
<td>94.02±0.31</td>
<td>72.28±0.95</td>
<td>72.23±0.84</td>
<td>45.71±0.77</td>
<td>45.77±0.80</td>
</tr>
<tr>
<td></td>
<td>REINFORCE</td>
<td>91.12±0.25</td>
<td>93.90±0.26</td>
<td>71.80±0.94</td>
<td>71.86±0.89</td>
<td>45.37±0.74</td>
<td>45.64±0.78</td>
</tr>
<tr>
<td></td>
<td>RANDOM</td>
<td>91.07±0.26</td>
<td>93.86±0.23</td>
<td>71.46±0.97</td>
<td>71.55±0.97</td>
<td>45.03±0.91</td>
<td>45.28±0.97</td>
</tr>
<tr>
<td></td>
<td>BOHB</td>
<td>91.17±0.27</td>
<td>93.94±0.28</td>
<td>72.04±0.93</td>
<td>72.00±0.86</td>
<td>45.55±0.79</td>
<td>45.70±0.86</td>
</tr>
<tr>
<td>Topology Search Space ( \mathcal{S}_t )</td>
<td>RSPS</td>
<td>87.60±0.61</td>
<td>91.05±0.66</td>
<td>68.27±0.72</td>
<td>68.26±0.96</td>
<td>39.73±0.34</td>
<td>40.69±0.36</td>
</tr>
<tr>
<td></td>
<td>DARTS (1st)</td>
<td>49.27±13.44</td>
<td>59.84±7.84</td>
<td>61.08±4.37</td>
<td>61.26±4.43</td>
<td>38.07±2.90</td>
<td>37.88±2.91</td>
</tr>
<tr>
<td></td>
<td>DARTS (2nd)</td>
<td>58.78±13.44</td>
<td>65.38±7.84</td>
<td>59.48±5.13</td>
<td>60.49±4.95</td>
<td>37.56±7.10</td>
<td>36.79±7.59</td>
</tr>
<tr>
<td></td>
<td>GDAS</td>
<td>89.68±0.72</td>
<td>93.23±0.58</td>
<td>68.35±2.71</td>
<td>68.17±2.50</td>
<td>39.55±0.00</td>
<td>39.40±0.00</td>
</tr>
<tr>
<td></td>
<td>SETN</td>
<td>90.00±0.97</td>
<td>92.72±0.73</td>
<td>69.19±1.42</td>
<td>69.36±1.72</td>
<td>39.77±0.33</td>
<td>39.51±0.33</td>
</tr>
<tr>
<td></td>
<td>ENAS</td>
<td>90.20±0.00</td>
<td>93.76±0.00</td>
<td>70.21±0.71</td>
<td>70.67±0.62</td>
<td>40.78±0.00</td>
<td>41.44±0.00</td>
</tr>
<tr>
<td></td>
<td>ResNet Optimal</td>
<td>90.86</td>
<td>93.91</td>
<td>70.50</td>
<td>70.89</td>
<td>44.10</td>
<td>44.23</td>
</tr>
<tr>
<td></td>
<td>Optimal</td>
<td>91.61</td>
<td>94.37 (94.37)</td>
<td>73.49</td>
<td>73.51 (73.51)</td>
<td>46.73</td>
<td>46.20 (47.31)</td>
</tr>
<tr>
<td>Size Search Space ( \mathcal{S}_s )</td>
<td>REA</td>
<td>90.37±0.20</td>
<td>93.22±0.16</td>
<td>70.23±0.50</td>
<td>70.11±0.61</td>
<td>45.30±0.69</td>
<td>45.94±0.92</td>
</tr>
<tr>
<td></td>
<td>REINFORCE</td>
<td>90.25±0.23</td>
<td>93.16±0.21</td>
<td>69.84±0.59</td>
<td>69.96±0.57</td>
<td>45.06±0.77</td>
<td>45.71±0.93</td>
</tr>
<tr>
<td></td>
<td>RANDOM</td>
<td>90.10±0.26</td>
<td>93.03±0.25</td>
<td>69.57±0.57</td>
<td>69.72±0.61</td>
<td>45.01±0.74</td>
<td>45.42±0.86</td>
</tr>
<tr>
<td></td>
<td>BOHB</td>
<td>90.07±0.28</td>
<td>93.01±0.24</td>
<td>69.75±0.60</td>
<td>69.90±0.65</td>
<td>45.11±0.69</td>
<td>45.56±0.81</td>
</tr>
<tr>
<td>Weight Sharing</td>
<td>channel-wise interpolation</td>
<td>90.71±0.00</td>
<td>93.40±0.00</td>
<td>70.30±0.00</td>
<td>70.72±0.00</td>
<td>44.73±0.00</td>
<td>47.17±0.00</td>
</tr>
<tr>
<td></td>
<td>masking + Gumbel-Softmax</td>
<td>90.41±0.10</td>
<td>93.14±0.13</td>
<td>70.30±0.00</td>
<td>70.72±0.00</td>
<td>45.71±0.39</td>
<td>46.38±0.27</td>
</tr>
<tr>
<td></td>
<td>masking + sampling</td>
<td>89.73±0.37</td>
<td>92.78±0.30</td>
<td>69.67±0.22</td>
<td>70.11±0.33</td>
<td>44.70±0.60</td>
<td>45.11±0.76</td>
</tr>
<tr>
<td>Largest Candidate</td>
<td>Optimal</td>
<td>90.71</td>
<td>93.40 (93.65)</td>
<td>70.92</td>
<td>70.12 (71.34)</td>
<td>46.73</td>
<td>45.10 (47.40)</td>
</tr>
</tbody>
</table>

NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size, IEEE TPAMI 2021
NAS’s performance is saturated - Search Algorithm

Multi-trial Search

Weight-sharing Search

NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size, IEEE TPAMI 2021
NAS is sub-optimal

L(21k) = NAS + Manually Designed Training Strategies
NAS is sub-optimal

\[ \text{EffNetV2-XL}(21k) \]

\[ \text{L}(21k) = \text{NAS} + \text{Manually Designed Training Strategies} \]
NAS is sub-optimal

<table>
<thead>
<tr>
<th>HP-1</th>
<th>Rank</th>
<th>HP-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(LR=5.5, L2=1.5e-4)</td>
<td>56.9%</td>
<td>&gt;</td>
</tr>
<tr>
<td>(LR=1.1, L2=8.4e-4)</td>
<td>54.7%</td>
<td>&lt;</td>
</tr>
</tbody>
</table>
## AutoHAS: Efficient and Joint Search

<table>
<thead>
<tr>
<th></th>
<th>learning rate</th>
<th>weight decay</th>
<th>augmentation</th>
<th>dropout</th>
<th>architecture</th>
<th>efficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bayesian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>RL or Evolution</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>PBT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Gradient Descent on LR</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Hypergradient</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NAS (Weight Sharing)</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>AutoHAS</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
AutoHAS: Efficient and Joint Search

Integrate REINFORCE into AutoHAS
AutoHAS: Efficient and Joint Search

**Integrate Differentiable Search into AutoHAS**

**AutoHAS Controller**

- Sample
- Candidate HP (RMSProp, LR=0.1)

**Candidate Architecture**

- $W_\alpha$

**Training Loss**

**Validation Loss**

**Update the AutoHAS controller**

**Update $W_\alpha$ using $W_\alpha$ and HP**

**Compute $W^*_\alpha$ using $W_\alpha$ and HP**

**Update $W$ using the sampled HP**

**AutoHAS: Efficient and Joint Search**
AutoHAS: Weight Sharing

AutoHAS Controller

Sample

Candidate HP (RMSProp, LR=0.1)

Candidate Architecture

$W_\alpha$

Sample

Layer-0

Layer-1

Layer-2

Super Model

Layer-0

Layer-1

Layer-2

Candidate Architecture

AutoHAS: Efficient Hyperparameter and Architecture Search, NAS@ICLR 2021
AutoHAS: Efficient Hyperparameter and Architecture Search, NAS@ICLR 2021

AutoHAS: Differentiable vs. REINFORCE

<table>
<thead>
<tr>
<th></th>
<th>#Params (MB)</th>
<th>#FLOPs (M)</th>
<th>Accuracy (%)</th>
<th>Search Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline model</td>
<td>1.5</td>
<td>35.9</td>
<td>50.96</td>
<td>1.0</td>
</tr>
<tr>
<td>AutoHAS (Differentiable)</td>
<td>1.5</td>
<td>36.1</td>
<td>52.17</td>
<td>6.1</td>
</tr>
<tr>
<td>AutoHAS (REINFORCE)</td>
<td>1.5</td>
<td>36.3</td>
<td>53.01</td>
<td>1.8</td>
</tr>
</tbody>
</table>

Search for architecture, learning rate, weight decay
# AutoHAS improves SoTA models

<table>
<thead>
<tr>
<th>Model</th>
<th>Method</th>
<th>#Params (M)</th>
<th>#FLOPs (M)</th>
<th>Top-1 Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50</td>
<td>Human</td>
<td>25.6</td>
<td>4110</td>
<td>77.20</td>
</tr>
<tr>
<td></td>
<td>AutoHAS</td>
<td>25.6</td>
<td>4110</td>
<td>77.83 (+0.63)</td>
</tr>
<tr>
<td>EfficientNet-B0</td>
<td>NAS</td>
<td>5.3</td>
<td>398</td>
<td>77.15</td>
</tr>
<tr>
<td></td>
<td>AutoHAS</td>
<td>5.2</td>
<td>418</td>
<td>77.92 (+0.77)</td>
</tr>
</tbody>
</table>
AutoHAS-discovered Hyperparameters

Search for drop-path ratio in EfficientNet
AutoHAS-discovered Hyperparameters

Search for learning rate for MobileNet and ResNet

AutoHAS: Efficient Hyperparameter and Architecture Search, NAS@ICLR 2021
AutoHAS-discovered Hyperparameters

Search for learning rate and weight decay for BERT
AutoHAS works on different datasets

<table>
<thead>
<tr>
<th>Method</th>
<th>Search Space</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
<th>Stanford Cars</th>
<th>Oxford Flower</th>
<th>SUN-397</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNetV2 (baseline)</td>
<td></td>
<td>94.1</td>
<td>76.3</td>
<td>83.8</td>
<td>74.0</td>
<td>46.3</td>
</tr>
<tr>
<td>AutoHAS</td>
<td>Weight Decay</td>
<td>95.0</td>
<td>77.8</td>
<td>89.0</td>
<td>84.4</td>
<td>49.1</td>
</tr>
<tr>
<td>AutoHAS</td>
<td>MixUp</td>
<td>94.1</td>
<td>77.0</td>
<td>85.2</td>
<td>79.6</td>
<td>47.4</td>
</tr>
<tr>
<td>AutoHAS</td>
<td>Arch</td>
<td>94.5</td>
<td>76.8</td>
<td>84.1</td>
<td>76.4</td>
<td>46.3</td>
</tr>
<tr>
<td>AutoHAS</td>
<td>MixUp + Arch</td>
<td>94.4</td>
<td>77.4</td>
<td>84.8</td>
<td>78.2</td>
<td>47.3</td>
</tr>
<tr>
<td>AutoHAS</td>
<td>Weight Decay + MixUp</td>
<td><strong>95.0 (+0.9)</strong></td>
<td><strong>78.4 (+2.1)</strong></td>
<td><strong>89.9 (+6.1)</strong></td>
<td><strong>84.4 (+10.4)</strong></td>
<td><strong>50.5 (+4.2)</strong></td>
</tr>
</tbody>
</table>
AutoHAS works on different datasets

<table>
<thead>
<tr>
<th>Method</th>
<th>Search Space</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
<th>Stanford Cars</th>
<th>Oxford Flower</th>
<th>SUN-397</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNetV2 (baseline)</td>
<td></td>
<td>94.1</td>
<td>76.3</td>
<td>83.8</td>
<td>74.0</td>
<td>46.3</td>
</tr>
<tr>
<td>AutoHAS</td>
<td>Weight Decay</td>
<td>95.0</td>
<td>77.8</td>
<td>89.0</td>
<td>84.4</td>
<td>49.1</td>
</tr>
<tr>
<td>AutoHAS</td>
<td>MixUp</td>
<td>94.1</td>
<td>77.0</td>
<td>85.2</td>
<td>79.6</td>
<td>47.4</td>
</tr>
<tr>
<td>AutoHAS</td>
<td>Arch</td>
<td>94.5</td>
<td>76.8</td>
<td>84.1</td>
<td>76.4</td>
<td>46.3</td>
</tr>
<tr>
<td>AutoHAS (Joint)</td>
<td>MixUp + Arch</td>
<td>94.4</td>
<td>77.4</td>
<td>84.8</td>
<td>78.2</td>
<td>47.3</td>
</tr>
<tr>
<td>AutoHAS (Sequential)</td>
<td>MixUp + Arch</td>
<td>94.4</td>
<td>77.6</td>
<td>85.5</td>
<td>79.6</td>
<td>48.3</td>
</tr>
<tr>
<td>AutoHAS (Joint)</td>
<td>Weight Decay + MixUp</td>
<td><strong>95.0 (+0.9)</strong></td>
<td><strong>78.4 (+2.1)</strong></td>
<td>89.9</td>
<td>84.4</td>
<td>50.5</td>
</tr>
<tr>
<td>AutoHAS (Sequential)</td>
<td>Weight Decay + MixUp</td>
<td>94.9</td>
<td>78.2</td>
<td><strong>90.5 (+6.8)</strong></td>
<td><strong>85.4 (+11.4)</strong></td>
<td><strong>50.8 (+4.5)</strong></td>
</tr>
</tbody>
</table>
AutoHAS vs. other HPO methods

ImageNet Accuracy (%)

Search Time Cost (TPU Hours)

AutoHAS
IFT
Bayesian Optimization
Random Search
HGD
What’s else?
What’s else?
NAHAS: Better Pareto Frontier
Joint Architecture and Accelerator Search

AutoHAS cannot handle hardware design

Rethinking Co-design of Neural Architectures and Hardware Accelerators, arXiv 2021.02
Joint Architecture and Accelerator Search

Controller

Accelerator Configuration

HAS Configuration

PEs_in_x_dimension
PEs_in_y_dimension
SIMD_units
register_file_KB
local_memory_MB
compute_lanes
io_bandwidth_gbps

Trainer

Performance Estimator

Accuracy

Reward Engine

Reward

Latency

Rethinking Co-design of Neural Architectures and Hardware Accelerators, arXiv 2021.02
Joint Search vs. Phase Search

Rethinking Co-design of Neural Architectures and Hardware Accelerators, arXiv 2021.02
## Multi-trial vs. One-shot

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1 Acc.</th>
<th>Latency in ms (Ratio-to-best)</th>
<th>Energy in mJ (Ratio-to-best)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EfficientNet-B0 \cite{Tan2019}, MobileNetV2 \cite{Sandler2018}, MnasNet-B1 \cite{Tan2019}, ProxylessNAS \cite{Cai2019}, Manual-EdgeTPU-small, IBN-only fixed accelerator</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBN-only NAHAS multi-trial</td>
<td>74.9%</td>
<td>0.30 (1.17x)</td>
<td>0.75 (1.23x)</td>
</tr>
<tr>
<td>IBN-only NAHAS oneshot</td>
<td><strong>76.5%</strong></td>
<td>0.35 (1.17x)</td>
<td><strong>0.61</strong></td>
</tr>
<tr>
<td>EfficientNet-B1 \cite{Tan2019}, MnasNet-D1 \cite{Tan2019}, Fixed accelerator multi-trial w fused-IBN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBN-only NAHAS multi-trial</td>
<td>77.4%</td>
<td>0.52 (1.06x)</td>
<td>1.50 (1.53x)</td>
</tr>
<tr>
<td>IBN-only NANAS oneshot</td>
<td><strong>76.8%</strong></td>
<td><strong>0.49</strong></td>
<td><strong>0.98</strong></td>
</tr>
<tr>
<td>EfficientNet-B3 \cite{Tan2019}, Manual-EdgeTPU-medium, MobilenetV3 w SE, Fixed accelerator multi-trial w fused-IBN, NAHAS multi-trial w fused-IBN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAHAS multi-trial w fused-IBN</td>
<td><strong>79.5%</strong></td>
<td>0.72 (1.12x)</td>
<td><strong>1.46</strong></td>
</tr>
</tbody>
</table>

Rethinking Co-design of Neural Architectures and Hardware Accelerators, arXiv 2021.02
Co-design Improves > 1% ImageNet Accuracy

Rethinking Co-design of Neural Architectures and Hardware Accelerators, arXiv 2021.02
AutoML: System Design

Target: Scale AutoML horizontally and vertically

Design more algorithms

Improve Programming Interface of AutoML

Apply to more applications
Example 0: Triple-level Search

- Operation Search Space
  - Architecture Search Space
    - Architecture
      - + - x / sqrt ...
      - 3x3-conv, depthwise-conv, dilated-conv, pool, att., ...

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020
Example 1: Coupling between CP and SS

class ResidualBlock:
    def call(self, inputs):
        op = Conv(...)
        return Add(
            inputs, op(inputs))
Example 1: Coupling between CP and SS

class ResidualBlock:
    def call(self, inputs):
        op = Conv(...)
        return Add(inputs, op(inputs))

class SearchableResidualBlock:
    def call(self, inputs, hps):
        if hps.op_type == 'conv':
            op = Conv(...)
        elif hps.op_type == 'dense':
            op = Dense(...)
        elif ...
        return Add(inputs, op(inputs))
Example 2: Coupling in Efficient NAS

```python
class MobileModel(object):
    def call(self, inputs):
        ...
        layer = MBConv()
        ...
        return Sequential(
            [layer, ...])

class EnasMobileModel(object):
    def call(self, inputs, hps):
        ...
        layer = Switch([MBConv((3, 3), 3), MBConv((5, 5), 3), MBConv((7, 7), 3), ...], selected=hps.op_choice0)
        ...
        return Sequential([layer, ...])
```

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020
Example 2: Coupling in Efficient NAS

```python
class MobileModel(object):
    def call(self, inputs):
        ...  
        layer = MBConv()
        ...  
        return Sequential([layer, ...])

class EnasMobileModel(object):
    def call(self, inputs, hps):
        ...
        layer = WeightedSum([MBConv((3, 3), 3), MBConv((5, 5), 3), MBConv((7, 7), 3), ...], weights=hps.op_weights0)
        ...
        return Sequential([layer, ...])

class DartsMobileModel(object):
    def call(self, inputs, hps):
        ...  
        layer = MBConv()
        ...  
        return Sequential([layer, ...])
```

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020
Example 2: Coupling in Efficient NAS

```python
class MobileModel(object):
    def call(self, inputs):
        ...
        layer = MBConv()
        ...
        return Sequential([layer, ...])
```

```python
class EnasMobileModel(object):
    def call(self, inputs, hps):
        ...
        layer = WeightedSum([MBConv((3, 3), 3), MBConv((5, 5), 3), MBConv((7, 7), 3), ...], weights=hps.op_weights0)
        ...
        return Sequential([layer, ...])
```

```python
class DartsMobileModel(object):
    def call(self, inputs, hps):
        ...
        return Sequential([layer, ...])
```
The Fluidity of Couplings

Change Search Space

Change Search Algorithm
What if?

Fixed Coupling

Dynamic Coupling

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020
Symbolic Programming for AutoML

Simple and unified interfaces

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020
Symbolize: Make regular program symbolically programmable

Conv = \texttt{symbolize}(tf.keras.layers.Conv2D)

@\texttt{symbolize}
class \texttt{Trainer}(object):
    def \_\_init\_\_(self, model, optimizer):
        ...
    def train(self):
        return \texttt{trainer_impl}(Self.optimizer,
                                    self.model)
Symbolic objects are mutable

Program parts are not only compositional, but also can be modified programmatically.

```python
Trainer(  
    model=Stacked(  
        op=Conv(8, (3, 3)),  
        repeats=2),  
    optimizer=Adam(2e-2)  
)    
Trainer(  
    model=Stacked(  
        op=MaxPool((3, 3)),  
        repeats=2),  
    optimizer=Adam(2e-2)  
)
```
Programming interfaces are provided for symbolic manipulation

```python
def swap(k, v, parent):
    if isinstance(v, Conv):
        return MaxPool(v.kernel)

trainer.clone().rebind(swap)
```

Clone trainer and replace all the Conv layers into MaxPools
From Static Program to Search Space

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020
From Static Program to Search Space

```
trainer = Trainer(
    model=Stacked(
        op=Conv(8, (3, 3)),
        repeats=3),
    optimizer=Adam(2e-2)
)

hyper_trainer = Trainer(
    model=Stacked(
        op=Oneof([Identity(), MaxPool((3, 3)), Conv(Oneof([[4, 8]], (3, 3))), repeats=3]),
    optimizer=Oneof([Adam(2e-2), RMSProp(Scalar(1e-6, 1e-3))])
)
```

Static Child Program  Search Space
Search Expressed as a For-loop

```
for trainer, feedback in sample(
    search_space=hyper_trainer,
    algorithm=PPO()):
    reward = trainer.train()
    feedback(reward)
```

Search as a feedback loop with sampled child programs
How Sample Works?

```
Trainer(
    model=Stacked(op=oneof([Identity(),
                              MaxPool((3, 3)),
                              Conv(oneof([4, 8]), (3, 3)),
                              repeats=3)),
    optimizer=oneof([Adam(2e-2),
                     RMSProp(floatv(1e-6, 1e-3))]))
```

Search Space

Inputs

Outputs

Child Program

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020
### AutoML: System Design

#### 3 Search Spaces:
- **$S_1$**: Search the kernel size & expansion factor of the inverted bottleneck units in MobileNetV2
- **$S_2$**: Search the output filters of the inverted bottleneck units in MobileNetV2
- **$S_3$**: $S_1 + S_2$

#### 3 Search Algorithms:
- **RS**: Random Search
- **Bayesian**: Bayesian Optimisation
- **TuNAS**: Efficient Search Algorithm

<table>
<thead>
<tr>
<th>#</th>
<th>Search space</th>
<th>Search algorithm</th>
<th>Lines of codes</th>
<th>Search cost</th>
<th>Train cost</th>
<th>Test accuracy</th>
<th># of MAdds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(static)</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>1</td>
<td>73.1</td>
<td>300M</td>
</tr>
<tr>
<td>2</td>
<td>(static) $\rightarrow S_1$</td>
<td>RS</td>
<td>+23</td>
<td>25</td>
<td>1</td>
<td>73.7 (↑ 0.6)</td>
<td>299M</td>
</tr>
<tr>
<td>3</td>
<td>$S_1$</td>
<td>RS $\rightarrow$ Bayesian</td>
<td>+1</td>
<td>25</td>
<td>1</td>
<td>73.9 (↑ 0.8)</td>
<td>305M</td>
</tr>
<tr>
<td>4</td>
<td>$S_1$</td>
<td>Bayesian $\rightarrow$ TuNAS</td>
<td>+1</td>
<td>1</td>
<td>1</td>
<td>74.2 (↑ 1.1)</td>
<td>301M</td>
</tr>
<tr>
<td>5</td>
<td>(static) $\rightarrow S_2$</td>
<td>TuNAS</td>
<td>+10</td>
<td>1</td>
<td>1</td>
<td>73.3 (↑ 0.2)</td>
<td>307M</td>
</tr>
<tr>
<td>6</td>
<td>$S_1, S_2 \rightarrow S_3$</td>
<td>TuNAS</td>
<td>+1</td>
<td>2</td>
<td>1</td>
<td>73.8 (↑ 0.7)</td>
<td>303M</td>
</tr>
</tbody>
</table>

PyGlove lets you use ~10 LoCs to switch between different spaces and algorithms.
Take Away

The Future of AutoDL:
– Symbolic Programming Based Infra
– Architecture -> Hyperparameter
– Architecture -> Hardware