“Efficient Multi-Objective Neural Architecture Search with Evolutionary Algorithms”

Thomas Elsken - Bosch Center for Artificial Intelligence

Germany Area Group - March 10, 2021
tinyML Talks Sponsors

arm

tinyML Strategic Partner

EDGE IMPULSE

maxim integrated™

Qeexo

Reality AI

SynSense

Additional Sponsorships available – contact Olga@tinyML.org for info
Arm: The Software and Hardware Foundation for tinyML

1. Connect to high-level frameworks
   - Profiling and debugging tooling such as Arm Keil MDK

2. Supported by end-to-end tooling
   - Optimized models for embedded

3. Connect to Runtime
   - Runtime (e.g. TensorFlow Lite Micro)

- Optimized low-level NN libraries (i.e. CMSIS-NN)
- RTOS such as Mbed OS
- Arm Cortex-M CPUs and microNPUs

Stay Connected
- @ArmSoftwareDevelopers
- @ArmSoftwareDev

Resources: developer.arm.com/solutions/machine-learning-on-arm
WE USE AI TO MAKE OTHER AI FASTER, SMALLER AND MORE POWER EFFICIENT

Automatically compress SOTA models like MobileNet to <200KB with little to no drop in accuracy for inference on resource-limited MCUs

Reduce model optimization trial & error from weeks to days using Deeplite's design space exploration

Deploy more models to your device without sacrificing performance or battery life with our easy-to-use software

BECOME BETA USER bit.ly/testdeeplite
TinyML for all developers

Test

Edge Device
Real sensors in real time
Open source SDK

Acquire valuable training data securely

Dataset

Enrich data and train ML algorithms

Impulse

Test impulse with real-time device data flows

Embedded and edge compute deployment options

Get your free account at http://edgeimpulse.com
Health sensors measure PPG and ECG signals critical to understanding vital signs. Signal chain products enable measuring even the most sensitive signals.

The biggest (3MB flash and 1MB SRAM) and the smallest (256KB flash and 96KB SRAM) Cortex M4 microcontrollers enable algorithms and neural networks to run at wearable power levels.

The new MAX78000 implements AI inferences at over 100x lower energy than other embedded options. Now the edge can see and hear like never before.
Qeexo AutoML

Automated Machine Learning Platform that builds tinyML solutions for the Edge using sensor data

**Key Features**

- Supports 17 ML methods:
  - Multi-class algorithms: GBM, XGBoost, Random Forest, Logistic Regression, Gaussian Naive Bayes, Decision Tree, Polynomial SVM, RBF SVM, SVM, CNN, RNN, CRNN, ANN
  - Single-class algorithms: Local Outlier Factor, One Class SVM, One Class Random Forest, Isolation Forest
- Labels, records, validates, and visualizes time-series sensor data
- On-device inference optimized for low latency, low power consumption, and small memory footprint applications
- Supports Arm® Cortex™- M0 to M4 class MCUs

**End-to-End Machine Learning Platform**

For more information, visit: www.qeexo.com

**Target Markets/Applications**

- Industrial Predictive Maintenance
- Smart Home
- Wearables
- Automotive
- Mobile
- IoT
is for building products

https://reality.ai  info@reality.ai  @SensorAI  Reality AI

Reality AI Tools® software

- Automated Feature Exploration and Model Generation
- Bill-of-Materials Optimization
- Automated Data Assessment
- Edge AI / TinyML code for the smallest MCUs

Reality AI solutions

- Automotive sound recognition & localization
- Indoor/outdoor sound event recognition
- RealityCheck™ voice anti-spoofing

info@reality.ai  @SensorAI  Reality AI
SynSense builds **ultra-low-power** (sub-mW) **sensing and inference** hardware for **embedded, mobile and edge devices**. We design systems for **real-time always-on smart sensing**, for audio, vision, IMUs, bio-signals and more.

https://SynSense.ai
## Next tinyML Talks

<table>
<thead>
<tr>
<th>Date</th>
<th>Presenter</th>
<th>Topic / Title</th>
</tr>
</thead>
</table>
| Tuesday, March 16     | **Vijay Janapa Reddi**  
Associate Professor, Harvard University  | tinyMLPerf: Deep Learning Benchmarks for Embedded Devices          |

Webcast start time is 8 am Pacific time

Please contact [talks@tinyml.org](mailto:talks@tinyml.org) if you are interested in presenting
Local Committee in Germany

Alexis Veynachter,
Master Degree in Control Engineering, Senior Field Application Engineer
Infineon 32bits MCUs for Sensors, Fusion & Control

Carlos Hernandez-Vaquero
Software Project Manager, IoT devices
Robert Bosch

Prof. Dr. Daniel Mueller-Gritschneder
Interim Head - Chair of Real-time Computer Systems
Group Leader ESL - Chair of Electronic Design Automation
Technical University of Munich
Announcement

https://www.tinyml.org/event/summit-2021/

Highlights:
- Keywords: Premier Quality, Interactive, LIVE ... and FREE
- 5 days, 50+ presentations
- 4 Tutorials
- 2 Panel discussions: (i) VC and (ii) tinyML toolchains
- tinyML Research Symposium
- Late Breaking News
- 3 Best tinyML Awards (Paper, Product, Innovation)
- 10+ Breakout sessions on various topics
- tinyML Partner sessions
- tinyAI for (Good) Life
- LIVE coverage, starting at 8am Pacific time

What should I do about it:
- Check out the program – you will be impressed
- Register on-line (takes 5 min)
- If interested: Submit nominations for Best Awards and/or Late News – February 28 deadline
- Block out your calendar: March 22-26
- Become a sponsor (sponsorships@tinyML.org)
- Actively participate at the Summit
- Provide your feedback – we listen!
- Don’t worry about missing some talks – all videos will be posted on YouTube.com/tinyML
tinyML is growing fast

<table>
<thead>
<tr>
<th></th>
<th>2019 Summit (March 2019)</th>
<th>2020 Summit (Feb 2020)</th>
<th>2021 Summit (March 2021), expected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attendees</td>
<td>160</td>
<td>400+</td>
<td>3000+</td>
</tr>
<tr>
<td>Companies</td>
<td>90</td>
<td>172</td>
<td>300+ (?)</td>
</tr>
<tr>
<td>Linkedin members</td>
<td>0</td>
<td>798</td>
<td>~ 2000</td>
</tr>
<tr>
<td>Meetups members</td>
<td>0</td>
<td>1140</td>
<td>~ 5000</td>
</tr>
<tr>
<td>YouTube subscribers</td>
<td>0</td>
<td>0</td>
<td>~ 3000</td>
</tr>
</tbody>
</table>

also started in Asia: tinyML WeChat and BiliBili
Summit Sponsors
(as of Feb 15, 2021)

Contact: sponsorships@tinyML.org

multiple levels and benefits available
(also check www.tinyML.org)
Reminders

Slides & Videos will be posted tomorrow

tinyml.org/forums  youtube.com/tinyml

Please use the Q&A window for your questions
Thomas Elsken

Thomas Elsken is a Research Engineer at the Bosch Center for Artificial Intelligence. Thomas received his Master’s degree in Mathematics at the University of Muenster and did his PhD on neural architecture search at the University of Freiburg and the Bosch Center for Artificial Intelligence under the supervision of Prof. Frank Hutter. Thomas’ research interests lie in automated machine learning. His work focuses on automatically designing neural network architectures, also known as neural architecture search. More recently, Thomas gained interest in meta learning and how meta learning algorithms can be combined with neural architecture search methods in order to make them more efficient and practical.
Efficient Multi-Objective Neural Architecture Search with Evolutionary Algorithms

Thomas Elsken\textsuperscript{1,2},
joint work with Jan Hendrik Metzen\textsuperscript{1} and Frank Hutter\textsuperscript{2,1}

\textsuperscript{1}Bosch Center for AI, \textsuperscript{2}University of Freiburg
Motivation
Successes of Deep Learning

Reasoning in games

Computer vision in self-driving cars

Figure from Cityscapes data set

Speech recognition
Motivation

Bigger, more complex architectures…

[Canziani et al., preprint 2017]

Slide courtesy Nikhil Naik
Motivation

Bigger, more complex architectures...

AlexNet

[Krizhevsky et al., NeurIPS 2012]

Inception-V4

[Szegedy et al., AAAI 2017]
Motivation

Can we automatically learn neural network architectures?

‘There is a repeating theme in […] AI research.
When we want to create an intelligent system, we […] first try to hand-design it […].
Once we realize that it’s too hard, we then try to hand-code some components of
the system and have machine learning figure out how to best use those hand-
designed components to solve a task.
Ultimately, we realize that with sufficient data and computation, we can learn the
entire system.’

Jeff Clune, *AI-generating algorithms, an alternate paradigm for producing general artificial intelligence*, 2019
Agenda

Neural Architecture Search (NAS)

1. Motivation

2. Brief Introduction to NAS
   1. Search Space Design
   2. NAS via Evolution
   3. NAS via Reinforcement Learning

3. Efficient NAS with Network Morphisms
   1. NASH: Single-Objective Neural Architecture Search
   2. LEMONADE: Multi-Objective Neural Architecture Search

4. Conclusion
Agenda

Neural Architecture Search (NAS)

1. Motivation

2. Brief Introduction to NAS
   1. Search Space Design
   2. NAS via Evolution
   3. NAS via Reinforcement Learning

3. Efficient NAS with Network Morphisms
   1. NASH: Single-Objective Neural Architecture Search
   2. LEMONADE: Multi-Objective Neural Architecture Search

4. Conclusion
Brief Introduction to NAS

Search Space Design

Chain-structured space
(different colours: different layer types)

Categorical & conditional search space!

E.g., ['maxpool', 3] vs. ['conv', 5, 16]

Layer type
Kernel size

# filters (only for layer type ‘conv’)

More complex space
with multiple branches and skip connections

Figures from [Elsken et al., JMLR 2019]
Brief Introduction to NAS

Search Space Design – Cell Search Spaces

Introduced by Zoph et al. [CVPR 2018]

Two possible cells

normal cell: preserves spatial resolution

reduction cell: reduce spatial resolution

Architecture composed of stacking together individual cells

Figures from [Elsken et al., JMLR 2019]
Brief Introduction to NAS
Neural Architecture Search via Evolution [Real et al., ICML 2017]

- Evolve population of neural network architectures over time
- Generate child networks by applying mutations to parent networks

Mutations:
- add layer
- remove layer
- change # filters in conv layer,
- add skip connection
- …
Brief Introduction to NAS
Neural Architecture Search via RL [Zoph & Le, ICLR 2017]

Sample architecture A with probability p

The controller (RNN)

Trains a child network with architecture A to get accuracy R

Compute gradient of p and scale it by R to update the controller

Softmax

Hidden State

Embedding

Layer N-1

Layer N

Layer N+1
Brief Introduction to NAS

Computational costs of black box methods

- Large computational costs: 800 GPUs for 3-4 weeks, 12,800 architectures trained

Analysis of the compute used in selected deep learning papers

Figure from https://openai.com/blog/ai-and-compute

Neural Architecture Search with RL

[Zoph & Le, ICLR 2017]

*A petaflop/s-day (pfs-day) consists of performing $10^{15}$ neural network operations per second for one day, or a total of about $10^{20}$ operations.
## Brief Introduction to NAS

**Computational costs of black box methods**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Error (%)</th>
<th>Params (Millions)</th>
<th>GPU Days</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoph and Le (2017)</td>
<td>3.65</td>
<td>37.4</td>
<td>22,400</td>
</tr>
<tr>
<td>Zoph et al. (2018)</td>
<td>3.41</td>
<td>3.3</td>
<td>2,000</td>
</tr>
<tr>
<td>Real et al. (2017)</td>
<td>5.40</td>
<td>5.4</td>
<td>2,600</td>
</tr>
<tr>
<td>Real et al. (2019)</td>
<td>3.34</td>
<td>3.2</td>
<td>3,150</td>
</tr>
</tbody>
</table>

### It’s expensive!

Comparison of different NAS methods (Cifar-10)

[Wistuba et al., preprint 2019]
Agenda

Neural Architecture Search (NAS)

1. Motivation
2. Brief Introduction to NAS
   1. Search Space Design
   2. NAS via Evolution
   3. NAS via Reinforcement Learning
3. Efficient NAS with Network Morphisms
   1. NASH: Single-Objective Neural Architecture Search
   2. LEMONADE: Multi-Objective Neural Architecture Search
4. Conclusion
Efficient NAS with Network Morphisms

Network morphisms [Chen et al., ICLR 2016; Wei et al., ICML 2016]

- Change the network structure, but not the modelled function (i.e., for every input the network yields the same output as before applying the network morphism)

- Can use this in NAS algorithms as operations to generate new networks
- Avoids costly training from scratch
Efficient NAS with Network Morphisms

Network morphisms – an example

► We have trained a network

\[ N_1(x) = \text{Softmax}_{w_{1,1}} \circ \text{ReLU} \circ \text{Conv}_{w_{1,2}}(x) \]

► Want to add another Relu-Conv block

\[ N_2(x) = \text{Softmax}_{w_{2,1}} \circ \text{ReLU} \circ \text{Conv}_{w_{2,2}} \circ \text{ReLU} \circ \text{Conv}_{w_{2,3}}(x) \]

► Our goal: by initialization, \( N_2(x) = N_1(x) \) for every \( x \)

► Step 1: copy weights of unchanged parts of network:

\[ w_{2,1} = w_{1,1}, \quad w_{2,3} = w_{1,2} \]

► Step 2: initialize new conv layer to be identity mapping:

\[ \text{Conv}_{w_{2,2}}(x) = x \]

► Then:
Efficient NAS with Network Morphisms
Neural Architecture Search by Hill Climbing (NASH)

[Elksen et al., NeurIPS MetaLearn 2017]
# Efficient NAS with Network Morphisms

**NASH – Results on Cifar-10**  
(from end of 2017, not state of the art anymore)

<table>
<thead>
<tr>
<th>Model</th>
<th>GPU Days</th>
<th>Params (M)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RESNet (He et al., 2016)</strong></td>
<td>10.2</td>
<td>4.6</td>
<td></td>
</tr>
<tr>
<td><strong>DENSENET (Huang et al., 2017)</strong></td>
<td>7.0</td>
<td>4.10</td>
<td></td>
</tr>
<tr>
<td><strong>DENSENET (Huang et al., 2017)</strong></td>
<td>27.2</td>
<td>3.74</td>
<td></td>
</tr>
<tr>
<td><strong>SHAKE-SHAKE (Gastaldi, 2017)</strong></td>
<td>26.3</td>
<td>2.9</td>
<td></td>
</tr>
<tr>
<td><strong>Zoph and Le (2017)</strong></td>
<td>22,400</td>
<td>37.4</td>
<td>3.6</td>
</tr>
<tr>
<td><strong>Zoph et al. (2017)</strong></td>
<td>2,000</td>
<td>3.3</td>
<td>3.41</td>
</tr>
<tr>
<td><strong>Zoph et al. (2017)</strong></td>
<td>2,000</td>
<td>27.6</td>
<td>3.0</td>
</tr>
<tr>
<td><strong>Cai et al. (2018)</strong></td>
<td>15</td>
<td>19.7</td>
<td>5.7</td>
</tr>
<tr>
<td><strong>Real et al. (2017)</strong></td>
<td>2,600</td>
<td>5.4</td>
<td>5.4</td>
</tr>
<tr>
<td><strong>Real et al. (2018)</strong></td>
<td>3,150</td>
<td>3.2</td>
<td>3.3</td>
</tr>
<tr>
<td><strong>NASH (ours)</strong></td>
<td>0.5</td>
<td>4.9</td>
<td>4.3</td>
</tr>
<tr>
<td><strong>NASH (ours)</strong></td>
<td>1</td>
<td>7.9</td>
<td>4.0</td>
</tr>
<tr>
<td><strong>NASH (ours)</strong></td>
<td>3</td>
<td>21.2</td>
<td>3.6</td>
</tr>
</tbody>
</table>

- Manually designed architectures
- Blackbox NAS with RL
- RL with network morphisms
- Blackbox NAS with Evolution
- Evolution with network morphisms
Efficient NAS with Network Morphisms

Deep learning in practice

- Deploying deep neural networks on, e.g.,
  mobile devices,
  autonomous vehicles,
  robots, …
  usually comes with hardware constraints & limitations

- Hardware from deployment scenario is often very different from high end GPUs

- Previous work only optimizes for a single objective: accuracy

- Can we incorporate these (hardware) objectives during architecture search?
Efficient NAS with Network Morphisms

Designing Efficient Architectures: LEMONADE [Elsken et al., ICLR 2019]
(Lamarckian Evolutionary algorithm for Multi-Objective Neural Architecture DEsign)

- LEMONADE extends NASH by optimizing multiple objectives concurrently, e.g. accuracy and number of parameters (as a proxy for efficiency)

Population in every generation = Pareto front w.r.t. multiple objectives

Check out our blog post at automl.org/automl-blog/
Efficient NAS with Network Morphisms

LEMONADE in a nutshell

In every iteration of LEMONADE:

- Sample networks from current population
- Generate children by mutating parents
- Evaluate these children with respect to the multiple objectives.
- Update the population: new population = Pareto front w.r.t. multiple objectives

Check out our blog post at automl.org/automl-blog/

[Elskens et al., ICLR 2019]
Efficient NAS with Network Morphisms

LEMONADE in a nutshell

- LEMONADE extends NASH by optimizing multiple objectives (4-5) concurrently, e.g., accuracy and number of parameters (as a proxy for efficiency)

- Observation: some objectives expensive to evaluate (e.g., accuracy – requires training), while others are very cheap (e.g., number of parameters)
  - Exploit observation by sampling promising children based on cheap objectives

- Also uses network morphisms (NM)

- Additionally, use approximate network morphisms (ANM) to also allow compressing networks

[Check out our blog post at automl.org/automl-blog/]

[Elssen et al., ICLR 2019]
Efficient NAS with Network Morphisms

LEMONADE details

Sample networks from current population to generate children

Check out our blog post at automl.org/automl-blog/

[Elsken et al., ICLR 2019]
Efficient NAS with Network Morphisms

LEMONADE details

Children Generation

- Pareto-front
- Population
- Children

Sampling probability = inversely proportional to population density
Efficient NAS with Network Morphisms

LEMONADE details

A. Children Generation

- Pareto-front
- Population
- Children

B. Training and Evaluation

- New Pareto-front
- Rejected Children
- Accepted Children
- Trained Children
- New Population

accept / reject children, fine tune accepted children

Check out our blog post at automl.org/automl-blog/

[Elsken et al., ICLR 2019]
Efficient NAS with Network Morphisms

LEMONADE details

Efficient NAS with Network Morphisms

LEMONADE details

Check out our blog post at automl.org/automl-blog/

[Elksen et al., ICLR 2019]

acceptance probability = inversely proportional to density
Efficient NAS with Network Morphisms

Comparison of **LEMONADE**, **NASNet** (other NAS method, 50x more expensive) and **MobileNet V2** (manually designed architecture to be efficient)
Efficient NAS with Network Morphisms
Optimizing for robustness with respect to hardware faults

[Schorn et al., Springer Neural Computing and Applications, 2020]
Agenda

Neural Architecture Search (NAS)

1. Motivation

2. Brief Introduction to NAS
   1. Search Space Design
   2. NAS via Evolution
   3. NAS via Reinforcement Learning

3. Efficient NAS with Network Morphisms
   1. NASH: Single-Objective Neural Architecture Search
   2. LEMONADE: Multi-Objective Neural Architecture Search

4. Conclusion
Conclusion

- NAS has become very popular in the deep learning community in last 1-2 years

- NAS is not necessarily super expensive anymore

- Multi-objective NAS allows to search for architectures tailored towards specific deployment scenarios

Figure from [Lindauer et al., JMLR 2020]
THANK YOU!
Automated design of error-resilient and hardware-efficient deep neural networks

Christoph Schorn\textsuperscript{1,2}, Thomas Elsken\textsuperscript{1,3}, Sebastian Vogel\textsuperscript{1,2}, Armin Runge\textsuperscript{1}, Andre Guntoro\textsuperscript{1}, Gerd Ascheid\textsuperscript{2}

\textsuperscript{1}Bosch, \textsuperscript{2}RWTH Aachen, \textsuperscript{3}University of Freiburg
Automated design of error-resilient and hardware-efficient deep neural networks [Schorn et al., Springer Neural Computing and Applications, 2020]

- Based on
  Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators,
  Schorn et al., DATE 2018
  Efficient Multi-Objective Neural Architecture Search via Lamarckian Evolution,
  Elsken et al., ICLR 2019
  Self-supervised quantization of pre-trained neural networks for multiplierless acceleration,
  Vogel et al., DATE 2019

- Multi-objective optimization of
  - error rate
  - architecture sensitivity index (ASI)
  - operations per frame
  - data transfer
  - bandwidth requirement

+ a-posteriori quantization
Automated design of error-resilient and hardware-efficient deep neural networks [Schorn et al., Springer Neural Computing and Applications, 2020]

▶ Architecture Sensitify Index

Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators, Schorn et al., DATE 2018

\[ f_{ASI}(N) = \sum_{l \in L_N} \frac{\lambda(l) \zeta(l)}{n_{\text{outputs}}} \]

- pooling factor
- skip connection factor
- layers in network
- # neurons in layers
Automated design of error-resilient and hardware-efficient deep neural networks [Schorn et al., Springer Neural Computing and Applications, 2020]

CIFAR-10

GTSRB
Automated design of error-resilient and hardware-efficient deep neural networks [Schorn et al., Springer Neural Computing and Applications, 2020]
Automated design of error-resilient and hardware-efficient deep neural networks [Schorn et al., under review, 2019]
Automated design of error-resilient and hardware-efficient deep neural networks [Schorn et al., under review, 2019]
Impact of other hyperparameters & data augmentation

Performance of randomly sampled architectures with different training pipelines on Cifar-10.

Figure from

[Yang et al., NAS Evaluation is frustratingly hard, ICLR 2020]
Brief Introduction to NAS
Evolution vs. Reinforcement Learning vs. Random Search
during architecture search

[Real et al., AAAI 2019]
THANK YOU!
Copyright Notice

This presentation in this publication was presented as a tinyML® Talks webcast. The content reflects the opinion of the author(s) and their respective companies. The inclusion of presentations in this publication does not constitute an endorsement by tinyML Foundation or the sponsors.

There is no copyright protection claimed by this publication. However, each presentation is the work of the authors and their respective companies and may contain copyrighted material. As such, it is strongly encouraged that any use reflect proper acknowledgement to the appropriate source. Any questions regarding the use of any materials presented should be directed to the author(s) or their companies.

tinyML is a registered trademark of the tinyML Foundation.

www.tinyML.org