AutoML: Neural Architecture Search (NAS)

Part 2: One-shot Neural Architecture Search

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Outline

AutoML: Neural Architecture Search (NAS)

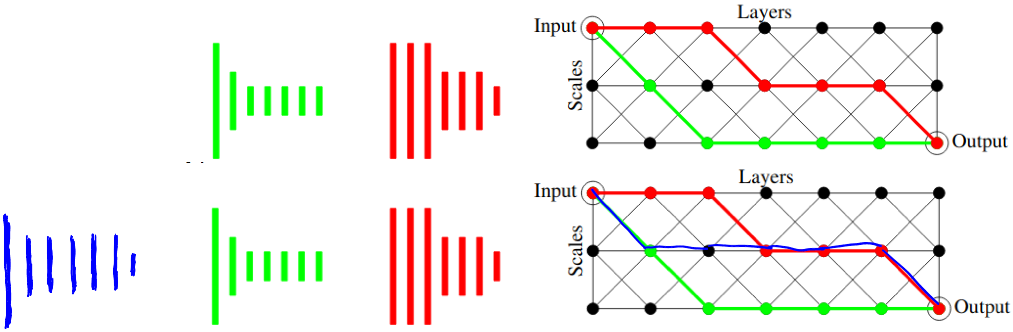
The One-Shot Model

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

One-shot models: convolutional neural fabrics [Saxena and Verbeek, 2017]

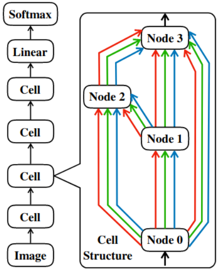
- A one-shot model is a big model that has all architectures in a search space as submodels
  - This allows weights sharing across architectures
  - One only needs to train the single one-shot model,
    and implicitly trains an exponential number of individual architectures

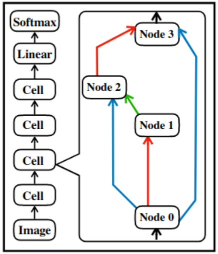
- The first type of one-shot models: convolutional neural fabrics

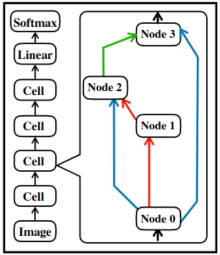


One-shot models for cell search spaces

- Directed acyclic multigraph to capture all (exponentially many) cell architectures
  - The nodes represent tensors
  - The edges represent computations (e.g., 3x3 conv, 5x5 conv, max pool, ...)
  - The results of operations on multiple edges between two nodes are combined (addition/concatenation)
- Individual architectures are subgraphs of this multigraph
  - Weights for the operation on an edge are shared
    across all (exponentially many) architectures that have that edge

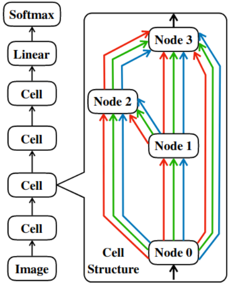






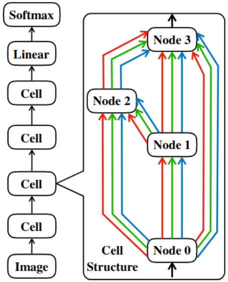
Training the one-shot model – standard SGD [Saxena and Verbeek, 2017]

- One-shot model is an acyclic graph; thus, backpropagation applies
  - Simplest method: standard training with SGD
  - This implicitly trains an exponential number of architectures
- Potential issue: co-adaptation of weights
  - Weights are implicitly optimized to work well on average across all architectures
  - They are not optimized specifically for the top-performing architecture

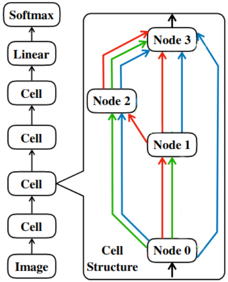


Training the one-shot model – DropPath [Bender et al., 2018]

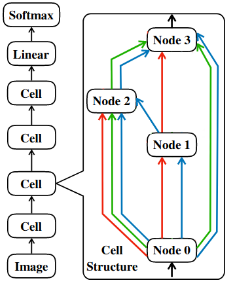
- To avoid coadaptation of weights, we can use DropPath, a technique analogous to Dropout [Srivastava et al., 2014]:
  - At each mini-batch iteration:
    for each operation connecting 2 nodes, zero it out with probability p
  - ScheduledDropPath: starts with $p = 0$ and increases p linearly to p_{max} at the end of training



One-shot model



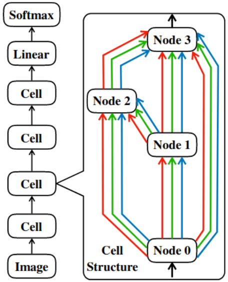
Architecture for batch 1



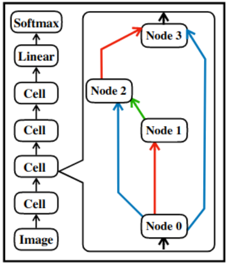
Architecture for batch 2

Training the one-shot model – Sampling

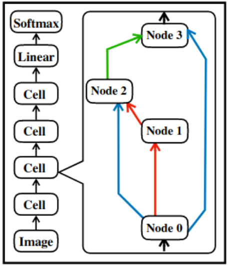
- At each mini-batch iteration during the training of the one-shot model sample a single architecture from the search space
  - **Random Search with Weight Sharing** [Li and Talwalkar, 2020] $\longrightarrow$ sample from uniform distribution
  - **ENAS** [Pham et al., 2018] $\longrightarrow$ sample from the learned policy of a RNN controller
- Update the parameters of the one-shot model corresponding to only that architecture



One-shot model



Architecture for batch 1



Architecture for batch 2

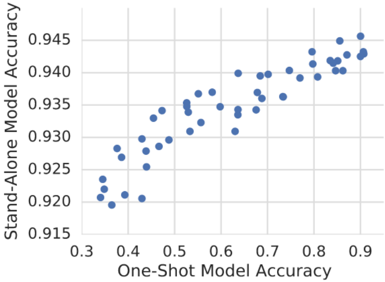
How to utilize the trained one-shot model?

- After training the one-shot model we have to select the best individual architecture from it
- There are multiple ways one can approach this. Some of these are:
  1. Sample uniformly at random M architectures and rank them based on their validation error using the one-shot model parameters

  $1b$. (Optional) Select top K $(K < M)$ and retrain them from scratch for a couple of epochs

  2. Return the top performing architecture to retrain from scratch for longer

- **Pitfall:** the correlation between architectures evaluated with the one-shot weights and retrained from scratch (stand-alone models) should be high
- If not, **selecting the best architecture based on the one-shot weights** is sub-optimal.



From [Bender et al., 2018]

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  How are the weights shared in the one-shot model?
- Repetition:
  What is the difference between Random Search with Weight Sharing and ENAS?
- Discussion:
  What migth be some downsides of using the one-shot model for NAS?

Outline

AutoML: Neural Architecture Search (NAS)

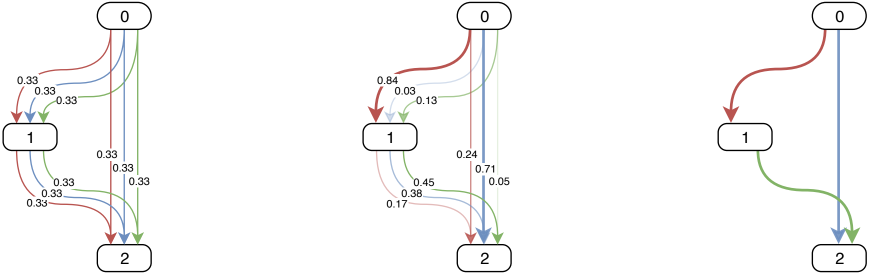
DARTS: Differentiable Architecture Search

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

DARTS: Differentiable Architecture Search [Liu et al, 2018]

- Use one-shot model with continuous architecture weight α for each operator

$$x^{(j)} = \sum_{i<j} \tilde{o}^{(i,j)}(x^{(i)}) = \sum_{i<j} \sum_{o \in \mathcal{O}} \frac{exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} exp(\alpha_{o'}^{(i,j)})} o(x^{(i)})$$



(a) Initialization (b) Search end (c) Final cell

- By optimizing the architecture weights α, DARTS assigns importance to each operation
  - Since the α are continuous, we can optimize them with gradient descent
- In the end, DARTS discretizes to obtain a single architecture (c)

DARTS: Architecture Optimization

- The optimization problem (a $\rightarrow$ b) is a bi-level optimization problem:

$$\begin{aligned} &\text{min}_{\alpha}\mathcal{L}_{\text{val}}(w^*(\alpha), \alpha) \\ s.t.\ w^*(\alpha) \ &\in \ \text{argmin}_{w}\mathcal{L}_{\text{train}}(w, \alpha) \end{aligned}$$

- This is solved using alternating SGD steps on architectural parameters α and weights w

---

Algorithm: DARTS 1st order

---

while *not converged* **do**
- Update one-shot weights $\mathbf{w}$ by $\nabla_{\mathbf{w}}\mathcal{L}_{train}(\mathbf{w}, \alpha)$
- Update architectural parameters α by $\nabla_{\alpha}\mathcal{L}_{valid}(\mathbf{w}, \alpha)$

return $\arg\max_{o \in \mathcal{O}} \alpha_o^{(i,j)}$ for each edge (i, j)

- Note: there is no theory showing that this process converges

Strong performance on some benchmarks

- E.g., original CNN search space
  - 8 operations on each MixedOp
  - 28 MixedOps in total
  - $> 10^{23}$ possible architectures
- Performance
  - $< 3\%$ error on CIFAR-10 in less than 1 GPU day of search

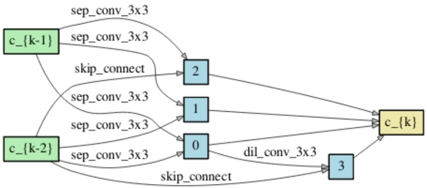


Figure 4: Normal cell learned on CIFAR-10.

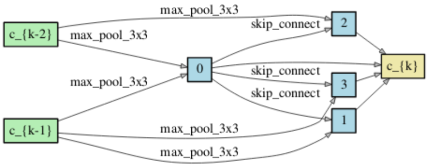
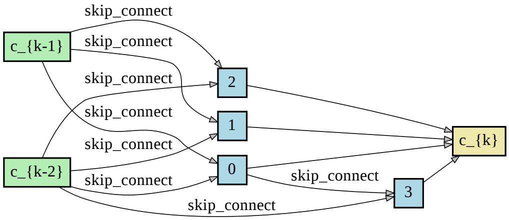


Figure 5: Reduction cell learned on CIFAR-10.

Issues – Non-robust behaviour

- DARTS is very sensitive w.r.t. its own hyperparameters (e.g. one-shot learning rate, L_2 regularization, etc.)
  - Tuning these hyperparameters for every new task/search space is computationally expensive
  - DARTS may return degenerate architectures, e.g., cells composed with only skip connections



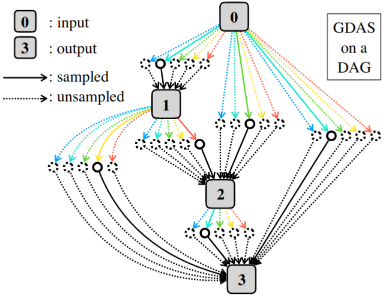
- RobustDARTS [Zela et al, 2020] – tracks the curvature of the validation loss and stops the search early based on that
- SmoothDARTS [Chen and Hsieh, 2020] – applies random perturbation and adversarial training to avoid bad regions

Issues – Memory constraints

- DARTS keeps the entire one-shot model in memory, together with its computed tensors
  - This constrains the search space size and the fidelity used to train the one-shot model
  - Impossible to run on large datasets as ImageNet

A lot of research aims to fix this issue:

- GDAS [Dong et al, 2019] – samples from a Gumbel Softmax distribution to keep only a single architecture in memory
- ProxylessNAS [Cai et al, 2019] – computes approximate gradients on α keeping only 2 edges between two nodes in memory at a time
- PC-DARTS [Xu et al, 2020] – performs the search on a subset of the channels in the one-shot model



Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  What is the main difference between DARTS and the other one-shot NAS methods we saw before?

- Repetition:
  How does DARTS optimize the architectural weights and one-shot weights?

- Repetition:
  What are DARTS' main issues and how can they be fixed?

- Discussion:
  RobustDARTS stops the architecture optimization early, before the curvature of the validation loss becomes high. Why do you think this works?
  [Hint: think about the discretization step after the DARTS search.]

Outline

AutoML: Neural Architecture Search (NAS)

NASLib: A Modular and Extensible NAS Library

Bernd Bischl   Frank Hutter   Lars Kotthoff
Marius Lindauer   Joaquin Vanschoren

Motivation for NASLib [Zela et al., 2020]

NASLib is a framework for easily implementing different NAS methods, aiming to:

- Allow fair comparisons without confounding factors, which could be due to
  - Different codebases
  - Different search and evaluation pipelines
  - Different hyperparameter settings
  - Other confounding factors, e.g., library versions, GPU types, etc.
- Modularize different components of NAS optimizers to allow combining them
- Offer researchers a convenient way of prototyping new NAS methods
- Offer users reliable implementations of NAS methods
  - Facilitate the use of NAS for new search spaces
  - Develop a robust true AutoML framework

NASLib building blocks: Search Spaces, Optimizers, Evaluators

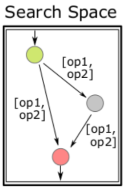
- NASLib implements a broad range of NAS optimizers
  - Blackbox NAS methods, e.g., Regularized Evolution
  - One-shot NAS methods, e.g., DARTS
- The optimizers are modularized
  - This allows to switch from, e.g., DARTS to GDAS or PC-DARTS by just one method call
- The evaluators are agnostic to the origin of an architecture
  - The final architecture is run using exactly the same object to evaluate on the test set
- NASLib's main building block is the graph object represented as a NetworkX [1] graph
  - Easily manipulate the graph by adding/removing nodes/edges
  - Hide complexity of dealing with the PyTorch computational graph
  - Easy high-level way of creating complex structures, e.g., hierarchical search spaces

[1] https://networkx.github.io/

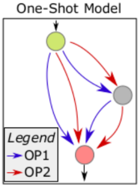
NASLib building blocks: Search Spaces, Optimizers, Evaluators

- The optimizers are agnostic to the search space they are running on
  - This facilitates their use for new types of search spaces

- An optimizer takes the search space as a NetworkX object and builds the PyTorch computational graph

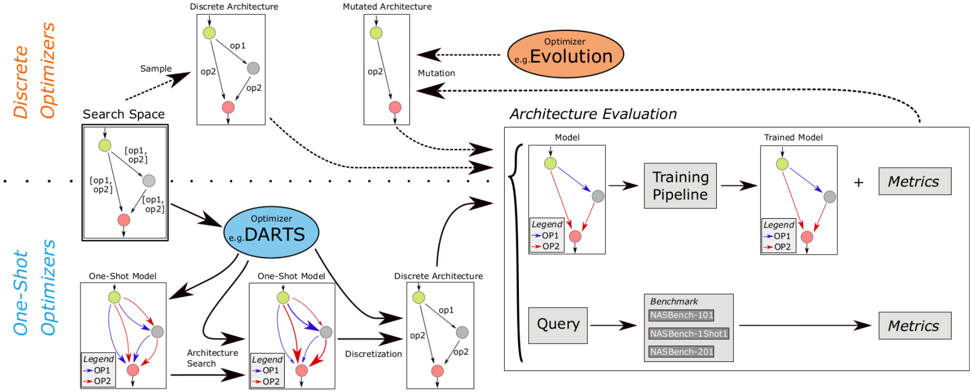


$\Rightarrow$



- Depending on the optimizer, each operation choice in the NetworkX object becomes:
  - a `MixedOp` – for one-shot NAS optimizers, e.g. DARTS
  - a `CategoricalOp` – for black-box optimizers, e.g. Regularized Evolution

NASLib: Overview



Tabular benchmarks for one-shot NAS

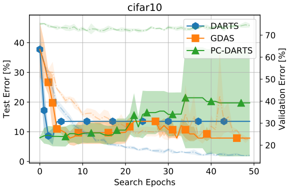
- NAS-Bench-101 [Ying et al. 2019] is not directly compatible with one-shot NAS methods
  - Mainly due to the constraint of at most 9 edges in the cell

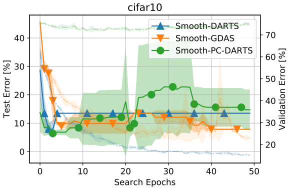
- NAS-Bench-1Shot1 [Zela et al. 2020]
  - 3 sub-spaces of NAS-Bench-101 that are compatible with one-shot methods
    - ⋆ 6 240, 29 160, and 363 648 architectures, respectively
  - Currently the largest one-shot NAS tabular benchmark

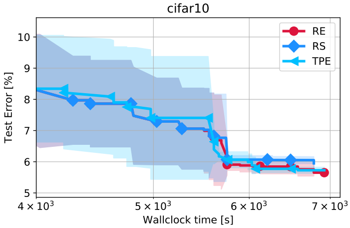
- NAS-Bench-201 [Dong and Yang. 2020]
  - Much smaller than NAS-Bench-101 and largest NAS-Bench-1Shot1 subspace
    - ⋆ 15 625 architectures
  - Every architecture in the search space evaluated on 3 image classification datasets

NASLib case study: Results on NAS-Bench-201

- NAS-Bench-201 is already integrated in NASLib and we can run any one-shot optimizer on it
- We can also combine random perturbations [Chen and Hsieh, 2020] with any one-shot optimizer
- We can also evaluate black-box optimizers cheaply with a tabular benchmark







Opportunities with NASLib

Room for many interesting projects and theses

- Applications of NAS to your problem of interest, with interesting search spaces
  - NASLib is the first library that separates the NAS method from the search spaces
    - Therefore, no changes are required in the NAS methods
    - This should make new applications much easier
- Studying hierarchical search spaces in detail
- Combining different components of existing NAS methods
  - So far, it has been very hard on a code level to mix and match components
  - It ought to be possible to design the world's best NAS method by combining the right components

Room for interesting Hiwi projects

- Not everything is perfect yet, we can use lots of support by great programmers

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  What would one have to do in order to apply the methods in NASLib to a new search space?

- Discussion:
  Is there a problem of your interest that you would you like to apply the methods in NASLib to?

- Discussion:
  Given that NASLib's modular design allows mixing and matching components of one-shot NAS methods, which of the methods we discussed might make sense to combine?

Outline

AutoML: Neural Architecture Search (NAS)

Practical Recommendations for NAS and HPO

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Maturity of the Fields of NAS and HPO

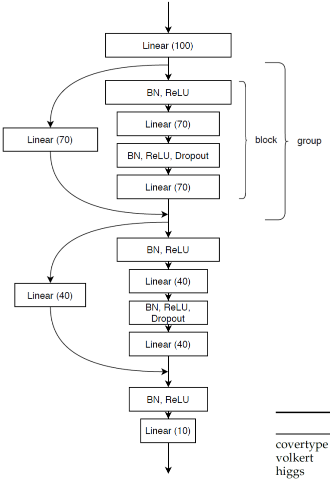
- Hyperparameter optimization is a mature field
  - Blackbox HPO has been researched for decades;
    there are many software packages
  - Multi-fidelity HPO has also become quite mature
- Neural architecture search is still a very young field
  - Blackbox is quite mature, but slow
  - Multi-fidelity NAS is also quite mature and faster
  - Meta-learning + multi-fidelity NAS is fast, but is still a very young field
  - Gradient-based NAS is fast, but can have failure modes with terrible performance
  - Gradient-based NAS hasn't reached the hands-off AutoML stage yet
- NAS is mainly used to create new architectures that many others can reuse
- Given a new dataset, HPO is crucial for good performance; NAS may not be necessary
  - The biggest gains typically come from tuning key hyperparameters (learning rate, etc)
  - Reusing a previous achitecture often yields competitive results

Practical Recommendations for NAS and HPO

- Recommendations for a new dataset
  - → Always run HPO
  - → Try NAS if you can
- How to combine NAS & HPO
  - If the compute budget suffices, optimize them jointly, e.g., using BOHB
    - + Auto-PyTorch Tabular [Zimmer, Lindauer & Hutter, 2020]
    - + Auto-RL [Runge et al, 2019]
  - Else
    - + If you have decent hyperparameters: run NAS, followed by HPO for fine-tuning [Saikat et al, 2019]
    - + If you don't have decent hyperparameters: first run HPO to get competitive

Case Study: NAS & HPO in Auto-PyTorch Tabular [Zimmer, Lindauer & Hutter, 2020]

- Joint Architecture Search and Hyperparameter Optimization
  - Purely using HPO techniques: very similar methods as in Auto-sklearn 2.0
  - Multi-fidelity optimization with BOHB
  - Meta-learning with task-independent recommendations
  - Ensembling of neural nets and traditional ML

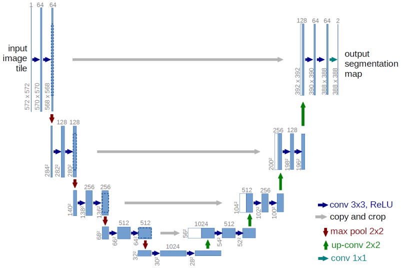


	Name	Range	log	type	cond.
Architecture	network type	[ResNet, MLPNet]	-	cat	-
	num layers (MLP)	[1, 6]	-	int	✓
	max units (MLP)	[64, 1024]	✓	int	✓
	max dropout (MLP)	[0, 1]	-	float	✓
	num groups (Res)	[1, 5]	-	int	✓
	blocks per group (Res)	[1, 3]	-	int	✓
	max units (Res)	[32, 512]	✓	int	✓
	use dropout (Res)	[F, T]	-	bool	✓
	use shake drop	[F, T]	-	bool	✓
	use shake shake	[F, T]	-	bool	✓
	max dropout (Res)	[0, 1]	-	float	✓
	max shake drop (Res)	[0, 1]	-	float	✓
Hyper-parameters	batch size	[16, 512]	✓	int	-
	optimizer	[SGD, Adam]	-	cat	-
	learning rate (SGD)	[1e-4, 1e-1]	✓	float	✓
	L2 reg. (SGD)	[1e-5, 1e-1]	-	float	✓
	momentum	[0.1, 0.999]	-	float	✓
	learning rate (Adam)	[1e-4, 1e-1]	✓	float	✓
	L2 reg. (Adam)	[1e-5, 1e-1]	-	float	✓
	training technique	[standard, mixup]	-	cat	-
	mixup alpha	[0, 1]	-	float	✓
	preprocessor	[none, trunc. SVD]	-	cat	-
	SVD target dim	[10, 256]	-	int	✓

	Auto-PyTorch	AutoGluon	AutoKeras	Auto-Sklearn	hyperopt-sklearn
covertype	**96.86 ± 0.41**	-	61.61 ± 3.52	-	-
volkert	**79.46 ± 0.43**	68.34 ± 0.10	44.25 ± 2.38	67.32 ± 0.46	-
higgs	**73.01 ± 0.09**	72.6 ± 0.00	71.25 ± 0.29	72.03 ± 0.33	-
car	**99.22 ± 0.02**	97.19 ± 0.35	93.39 ± 2.82	98.42 ± 0.62	98.95 ± 0.96
mfeat-factors	**99.10 ± 0.18**	98.03 ± 0.23	97.73 ± 0.23	98.64 ± 0.39	97.88 ± 38.48
apsfailure	99.32 ± 0.01	**99.5 ± 0.03**	-	99.43 ± 0.04	-
phoneme	**90.59 ± 0.13**	89.62 ± 0.06	86.76 ± 0.12	89.26 ± 0.14	89.79 ± 4.54
dibert	**99.04 ± 0.15**	98.17 ± 0.05	96.51 ± 0.62	98.14 ± 0.47	-

Case Study: NAS & HPO in Auto-DispNet

- Problem: disparity estimation
  - Estimate depth from stereo images
- Background: U-Net
  - Skip connections from similar spatial resolution to avoid loosing information
- Search space for DARTS
  - 3 cells: keeping spatial resolution, downsampling, and new upsampling cell that supports U-Net like skip connections
- Both NAS and HPO improved the state of the art [Saikat et al, 2019]:
  - End-point-error (EPE) on Sintel dataset: 2.36 → 2.14 (by DARTS)
  - Subsequent HPO: 2.14 → 1.94 (by BOHB)



Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  If you want to use both HPO and NAS for your problem, how could you proceed?
- Discussion:
  Think of a problem of your particular interest. For that problem, which approach would you use to combine HPO and NAS, and why?

Further Reading

Survey on NAS: [Elsken and Metzen and Hutter, 2019]