

# Optimizing Artificial Neural Network Hyperparameters and Architecture

Ivar Thokle Hovden, ivarth@student.matnat.uio.no, University of Oslo

May 6, 2019

## 1 Introduction

The process of successfully creating and training an Artificial Neural Network (ANN) to solve a domain-specific task is highly dependent on the skill set of the researcher, which would typically be various experience gained from repetitive trying and failing of ANN modeling as well as domain knowledge from previous tasks. See Figure 1 to get an impression of the complexity in the field. In the deep learning community, architectures such as Convolutional Neural Networks (CNNs) and Autoencoders with many hidden layers have been proven to work well on large datasets [1], but limitations of architectures initially created with few layers, such as Recurrent Neural Networks (RNNs) have also inspired researchers to create completely new architectures such as highway networks [2]. Another example is Squeeze-and-Excitation networks, networks that better learn interdependencies between channels of convolutional features compared to traditional CNNs [3]. There exist great potential to learn from previous neural network research [4] to try to extend earlier ideas into Deep Neural Network (DNN) design, particularly those who are more biologically inspired than today's networks. One example is the Transformer network, which incorporates attention mechanisms [5]. Regardless of the number of layers, it can be hard to efficiently specify good non-learnable parameters of the ANN. In fact, how deep the network is should rather be regarded as a hyperparameter that is automatically tuned based on the problem at hand as well as available data. See Table 1 for an extensive list of ANN hyperparameters as well as various techniques to perform a successful ANN training. The goal of this essay is to show that the researcher should not only rely on previous experiences when defining hyperparameters, but also on ANN hyperparameter optimisation (HPO) and Neural Architecture Search (NAS) techniques such as Bayesian and Gradient-based Sequential

Optimization methods, Grid Search, Evolutionary Strategies (ES) and Reinforcement Learning (RL), for tuning into an optimal hyperparameter configuration and in this way commit to more reproducible research.

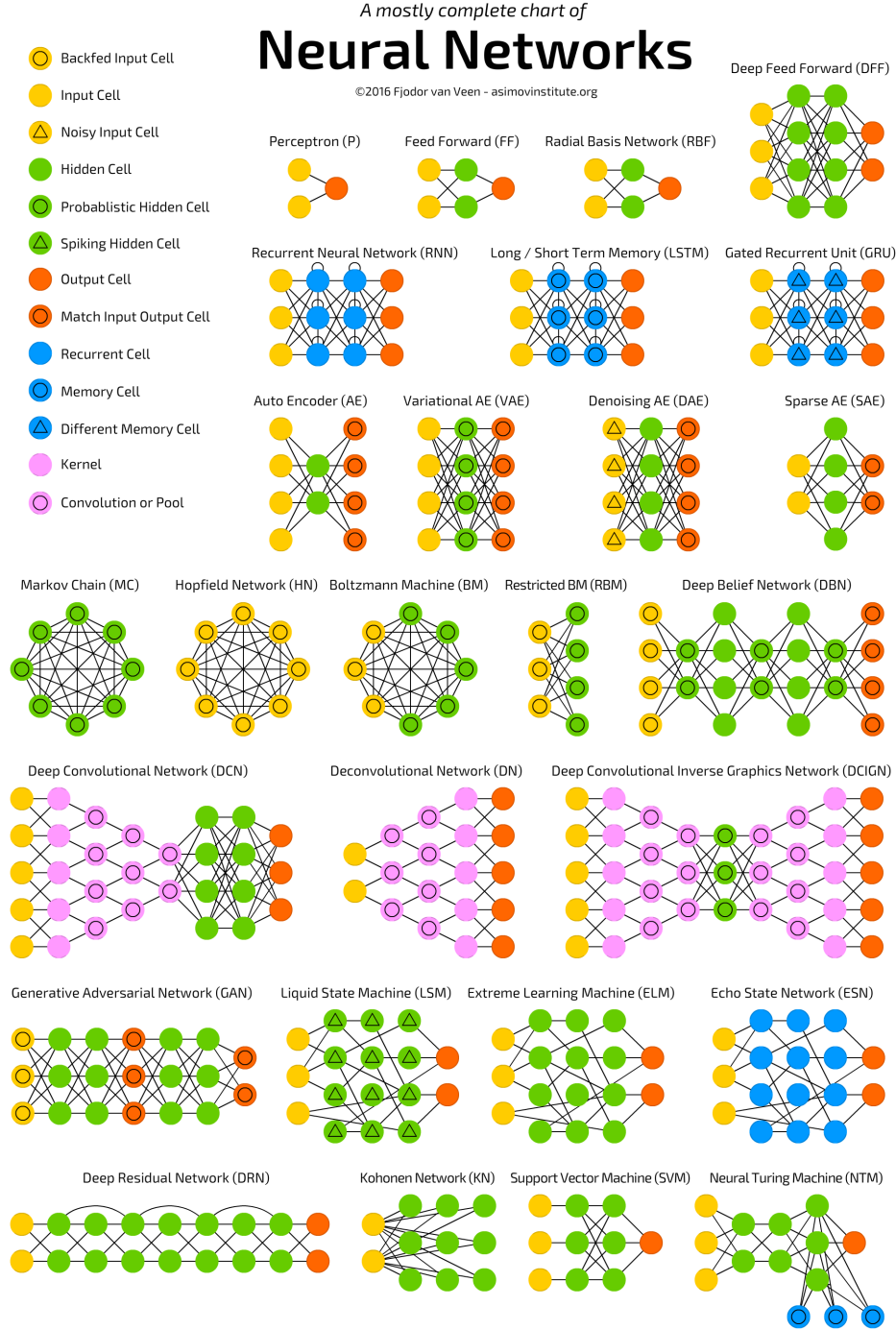


Figure 1: High-level visualization of common ANN architectures. In most of these networks weights and biases can be optimized through gradient based methods, such as backpropagation gradient descent (GD) [6]. Actual engineering of the network architecture is typically done manually, but the field of AutoML also considers optimizing the architecture [7]. A promising example is the evolutionary method CoDeepNEAT [8], which is suited for deep learning. Borrowed from [9].

Table 1: Some ANN hyperparameters and useful tricks for training ANNs. In this work, a hyperparameter is defined as any parameter in the ANN configuration that is not directly learnable by training through backpropagation and GD [6]. See [10] for useful tips on tuning the most common hyperparameters: learning rate, batch size, momentum and weight decay, as well as understanding the signs of overfitting and underfitting.

Parameter/method	Description	Example
Activation function	Defines how a neuron or group of neurons activate ("spiking") based on input connections and bias term(s).	Exponential Linear Unit (ELU) [11].
Number of neurons in a layer	-	1000
Number of layers	Typically layers between input and output layer, which are called hidden layers.	5
Number of epochs	The number of times all training examples have been passed through the network during training.	100
Mini-batch size	Number of training examples in each gradient descent (GD) update.	50
Number of filters in each layer	For Convolutional Neural Networks (CNNs). A filter is a collection of CNN layer weights that is convolved with the input of the layer.	10
Filter size	For CNNs. The spatial size of the filter.	5 x 5
Stride	For CNNs. The spatial step between successive convolutions with a filter. A stride larger than one will downsample the data in addition to convolving it with the filter.	2
Learning rate	Step length for GD update.	0.005
Learning rate decay	Incrementally decaying the learning rate parameter throughout the training to prevent overfitting. A regularization parameter.	-
Momentum	If using a momentum in the (GD) update rule, ex. Nesterov momentum. Acts as a smoothing of the trajectory of GD updates.	0.9

Dropout [12]	"Dropping out" some input connections to some neurons by a probability in order to make the network represent features more evenly throughout the network (some redundancy). ANN pruning. A type of regularization. Can be used for uncertainty quantification.	-
Early stopping	Stop the ANN training typically when training and validation error start to diverge from each other creating a "generalization gap". Prevents overfitting.	-
Loss function	Specifies how to calculate the error between prediction and label for a given training example. The error is backpropagated during training in order to update learnable parameters.	Multi-class cross-entropy.
Regularization on loss function	A (penalty) term added to the loss function or gradient update equation which can prevent overfitting and/or make the network more robust against noisy input data.	L2 or weight decay.
Initialization techniques of learnable parameters [13]	Learnable parameters such as weights and bias for a neuron need to have initial values before training starts. The initial values can affect the difficulty of the learning task by changing the probabilities of encountering local minima.	He et al. initialization of weights [14].
Batch normalization [15]	The distribution of each layer's inputs have a tendency to change during training leading to internal covariate shift. This can lead to saturating nonlinearities. Batch normalization attempts to prevent this by implementing a normalization layer after fully connected or convolutional layers that act on each mini-batch.	-

Transfer learning	Not always is it necessary to train the entire network. A pretrained network can be used. A selected number of layers of the pretrained network can be retrained on new or task-specific data for domain shift/adaptation where learnt features of the pretrained network are reused.	VGG16 [16] pretrained on the ImageNet [17] dataset.
Data augmentation	Create new data from existing data to create more training examples.	Rotating and flipping images.
Multitask learning	Train the same ANN on different datasets with different tasks.	Data/task 1: Object detection; data/task 2: re-coloring images.

Table 2: ANN parameters that are learnt through GD updates in backpropagation during training.

Parameter	Description
Weights	The amplification of input signals to a neuron.
Bias	An additive bias term to a neuron.

It must be noted that hyperparameter optimization (HPO) in the context of ANNs can be regarded as a subset of the broader task of specifying the best set of parameters of any algorithm that can be evaluated through a performance measure that is dependent on those parameters. A branch of optimization called algorithm configuration [18] is central in this regard, which deals with mathematically difficult search spaces consisting of both discrete and continuous variables as well as conditional variables that are only meaningful for some combination of other variables [19].

The fact that today’s supervised neural networks train on loss functions that are differ-

entiable is essentially what makes gradient based optimization through GD possible of the learnable parameters. However, it is reasonable that traditionally non-learnable parameters, namely hyperparameters, also play important roles in defining the overall learning capacity of the neural network. ANN hyperparameters typically share many of the properties with a variable in the algorithm configuration task (discrete or continuous, partially dependent on other variables, not differentiable w.r.t. the loss function / no gradient available, etc.). The ANN topology itself should also be regarded a hyperparameter.

Methods for finding an optimal architecture fall within the category of Neural Architecture Search (NAS). [20] divide the NAS process into three dimensions; search space, search strategy, and performance estimation strategy. One of the trickiest areas is the selection of performance estimation strategy, since evaluating the performance of an ANN architecture typically is computationally expensive. HPO including NAS can be regarded as a combined algorithm selection and hyperparameter optimization (CASH) problem, which aims to identify the combination of algorithm components with the best (cross-)validation performance [7].

## 2 Definition of HPO for ANN

If we regard gradient based optimization of the  $N$  learnable parameters  $\theta$  in a configuration space  $\Theta = \Theta_1 \times \Theta_2 \times \dots \times \Theta_N$  as an inner optimization problem solved by GD, we can analogously define hyperparameter optimization (HPO) for ANN as an outer optimization problem of the  $M$  hyperparameters  $\lambda$  in a configuration space  $\Lambda = \Lambda_1 \times \Lambda_2 \times \dots \times \Lambda_M$  according to [21–23]

$$\lambda^* = \underset{\lambda \in \Lambda}{\operatorname{argmin}} \mathbb{E}_{(D_{train}, D_{val}) \sim \mathcal{D}} \mathbf{V}(\mathcal{L}, \mathcal{A}_\lambda, D_{train}, D_{val}) = \underset{\lambda \in \lambda^{(1)} \dots \lambda^{(S)}}{\operatorname{argmin}} \Psi(\lambda) \quad (1)$$

where  $\mathbf{V}(\mathcal{L}, \mathcal{A}_\lambda, D_{train}, D_{val})$  is a loss measure that we want to minimize, associated with a loss function  $\mathcal{L}$  and generated ANN model  $\mathcal{A}$  with hyperparameters  $\lambda$  trained on training data  $D_{train}$  and evaluated on validation data  $D_{val}$ . In practice,  $\mathbf{V}$  can be computed using k-fold Cross-Validation (CV). Since we don't have access to infinite data and computational resources, the optimal hyperparameters  $\lambda^*$  are also approximations of the true optimal hyperparameters on data  $\mathcal{D}$ . Nevertheless, we want to find an optimal ANN architecture  $\mathcal{A}$  with a hyperparameter configuration  $\lambda^*$  that minimizes a given loss measure  $\mathcal{L}$ . Moreover, the training  $D_{train}$  and validation  $D_{val}$  data need to represent sufficient distributions of relevant features needed to solve the task and thus minimize the loss. The HPO problem amounts to finding the best available hyperparameter configuration, which is minimizing the response surface function  $\Psi(\lambda)$

subject to all available hyperparameter configurations  $S$ . Equation 1 shows that not only are hyperparameters important in HPO, but also the selection of data and loss function.

### 3 Methods

#### 3.1 Grid search: Complete, Manual and Random

Perhaps the most basic form of HPO for ANNs is to perform a loss evaluation for each possible configuration of the  $M$  hyperparameters,  $S = \prod_{m=1}^M |\Lambda_m|$ , to find the configuration with the minimum loss. Here,  $S$  grows exponentially with the number of hyperparameters, thus a complete grid search would suffer from a common problem in machine learning of having to deal with a large amount of parameter states, often denoted the curse of dimensionality. A complete GD training would be needed on each loss evaluation by for instance regarding the final mini-batch cost function evaluation (after some number of epochs of training) as the loss evaluation in the HPO.

Having to evaluate every possible configuration makes complete grid search a slow process. For this reason, a subset of configurations can be selected manually by intuition before or during the grid search process. However, this introduces difficulties of reproducibility and also forces the HPO to be performed sequentially.

The most popular form of grid search today is random grid search where parameter configurations are instead selected randomly. [21] shows both theoretically and experimentally that if sampling hyperparameter configurations by modeling all hyperparameters as independent and identically uniformly distributed variables, random grid search will perform better in high-dimensional spaces than complete grid search. This is related to the fact that the loss function in hyperparameter space often has low effective dimensionality, and random grid search can cope better with low effective dimensionality. A method for speeding up random search as an infinite-armed bandit problem, HYPERBAND [24], also makes random search competitive to Bayesian optimization for HPO, which will be discussed next.



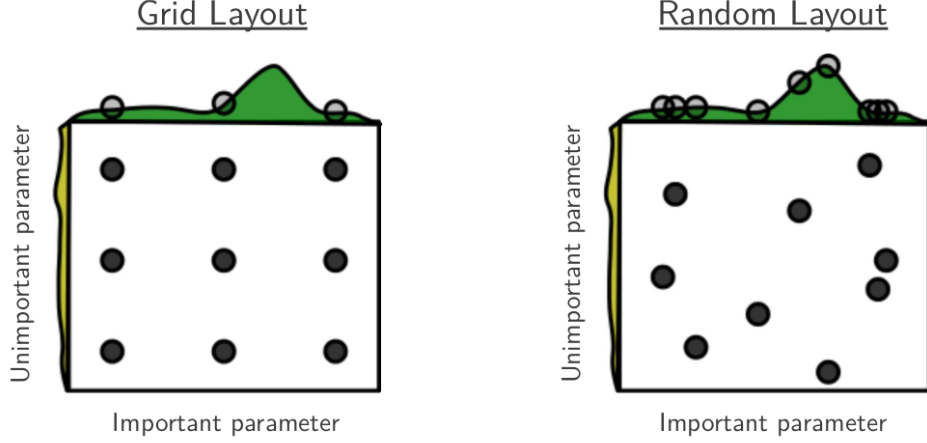


Figure 2: Complete and random grid search on two-dimensional hyperparameter space with lower effective dimensionality ( $D=1$ ). Random grid search is more likely to explore distinct values in lower dimensional space. This becomes more evident in high-dimensional spaces. Borrowed from [21].

### 3.2 Bayesian Optimization

Although having been around for a while [25], methods based on Bayesian optimization have become increasingly popular for ANN HPO during the last years [26–29]. A central reason for introducing Bayesian ANN HPO is that the loss function in Equation 1 is very expensive to evaluate since it needs a complete retraining of the ANN for each hyperparameter configuration.

The most popular Bayesian HPO for ANNs models the hyperparameter search process using a Gaussian Process (GP) surrogate model (Equation 2) together with an acquisition function (Equation 3) in order to infer a decision on which hyperparameter configuration to evaluate next in each iteration [22, 30–32]. In most simple form it is an adaptive Sequential Model-based Optimization (SMBO) method [33].

$$\Psi(\lambda) \sim \mathcal{GP}(m(\lambda), k(\lambda, \lambda')) \quad (2)$$

$$\alpha_{EI}(\lambda_{n+1}) = \mathbb{E}[\mathbb{I}(\lambda_{n+1})] = (\mathbf{y}_{min} - \mu_n(\lambda_{n+1}))\Phi(Z) + \sigma_n(\lambda_{n+1})\phi(Z) \quad (3)$$

where

$$Z = \frac{\mathbf{y}_{min} - \mu_n(\lambda_{n+1})}{\sigma_n(\lambda_{n+1})} \quad (4)$$

The GP in Equation 2 is a probabilistic model of  $\Psi$  in Equation 1 and is fully specified by a mean  $m(\lambda)$  and covariance  $k(\lambda, \lambda')$  function. The value of  $\Psi(\lambda)$  for each non-evaluated (unknown) hyperparameter configuration  $\lambda$  is modelled according to a mean and confidence interval prior to evaluation. Many different acquisition functions can be used with the GP model to predict new promising hyperparameter configurations to evaluate. The acquisition Equation 3 is called Expected Improvement [22, 30]. In each iteration,  $\mathbf{y}_{min}$  is the best observed value so far, while  $\phi(\cdot)$  and  $\Phi(\cdot)$  are the Probability Density Function (PDF) and Cumulative Distribution Function (CDF) of standard normal distribution, respectively.

The acquisition function 3 determines in each iteration  $n$  which parameter configuration  $\lambda$  to evaluate based on a trade-off between reasonably high prediction uncertainty, which is essentially measuring the confidence interval (exploration), and good objective function value which corresponds to the mean (exploitation) of the objective function approximating  $\Psi$ . In the noise-free case, this involves for each iteration  $n$  computing the predictive distribution of the GP for each possible hyperparameter configuration  $\lambda_{n+1}$  according to Equations 5, 6 [30, 31]

$$\mu_n(\lambda_{n+1}) = \mathbf{k}^T \mathbf{K}^{-1} \mathbf{y}_{1:n} \quad (5)$$

$$\sigma_n^2(\lambda_{n+1}) = k(\lambda_{n+1}, \lambda_{n+1}) - \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k} \quad (6)$$

where  $\mathbf{K}$  is the covariance matrix of all previously evaluated observations  $\mathbf{y}_{1:n}$  and  $\mathbf{k}$  a vector of covariances between  $\lambda_{n+1}$  and all previous configurations  $\lambda_{1:n}$ .

The hyperparameter configuration  $\lambda_{t+1}$  giving 5, 6 that lead to the most expected improvement when inserted into Equation 3 is then evaluated using  $\Psi(\lambda_{t+1})$  giving the new observation  $\mathbf{y}_{n+1}$ , which will be used in the next iteration [22], [34], Ch. 15.2.1. See Figure 3.

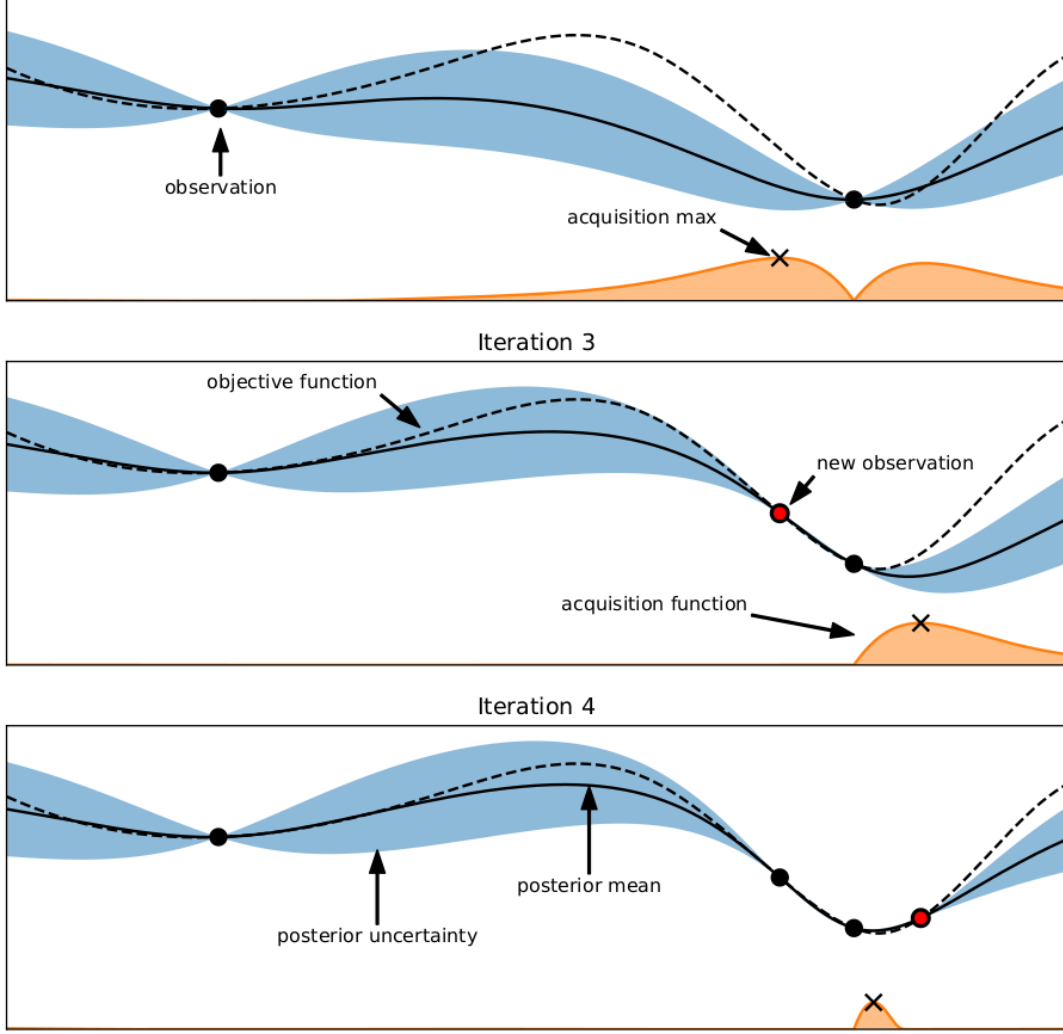


Figure 3: Illustration of objective function (blue) as well as acquisition function (orange) for one-dimensional Bayesian hyperparameter optimization over three iterations. In each iteration, the acquisition function determines the next point to evaluate based on a trade-off between large acquisition function value (exploitation) and high objective function uncertainty (exploration) (and thus minimized objective function value). The trade-off is achieved by finding the hyperparameter configuration giving a predictive mean and standard deviation of the GP model which maximizes the acquisition function. A new observation is then performed with this hyperparameter configuration. Lastly, the observation is used to update the posterior uncertainty and mean of the GP model. Borrowed from [22].

An important aspect of Bayesian GP HPO is the selection of an appropriate covariance function  $k(\lambda, \lambda')$ , which is the kernel of the GP. The covariance function dictates the structure of the response function that the GP can fit, and can be stationary, periodic or nonstationary

[31]. A common kernel is the Matern 5/2 kernel, which is a stationary kernel.

The works in [26] used a negative exponentiated distance as the kernel, which uses a (pseudo-)distance metric for ANN architectures that is efficiently computed via an Optimal Transport Program [35]: Optimal Transport Metrics for Architectures of Neural Networks (OTMANN). OTMANN introduces layer masses and path lengths into the ANN in order to efficiently estimate the amount of computation of the layers (as a performance measure) between two ANN architectures. They then used an Evolutionary Algorithm (EA) to maximize the Expected Improvement acquisition function 3 over a pool of ANNs. The resulting Bayesian Optimization (BO) framework was named Neural Architecture Search with Bayesian Optimisation and Optimal Transport (NASBOT), and it provided quite efficient (and generic) NAS compared to previous state-of-the-art evolutionary and reinforcement learning based NAS in terms of computation time [36, 37]. See Figure 4 for a comparison of CV results between random search, evolutionary algorithm, NASBOT and another BO framework, as well as Figure 11a for one of the resulting CNNs from NAS on the CIFAR-10 dataset [38]. For another example of Bayesian GP ANN HPO using CNNs and Transfer Learning, see the Master’s thesis at [39].

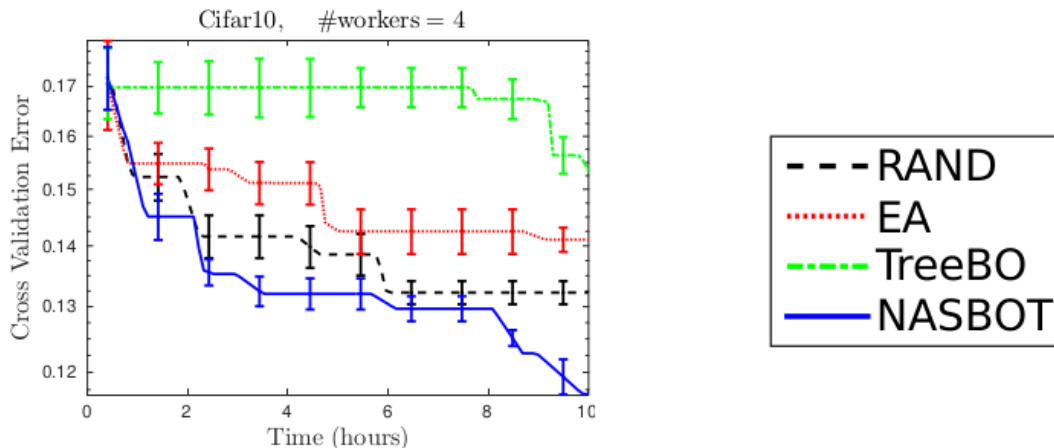


Figure 4: A comparison of CV error results from NAS on the CIFAR-10 dataset from [26] using four different NAS methods; random search, evolutionary algorithm, and the Bayesian methods TreeBO and NASBOT. Parallelized NAS and training was performed on four Nvidia Tesla K80 (12GB) GPUs with a budget of 10 hours. For NASBOT, the computation time (search cost) is several orders of magnitudes faster (less) than for instance Progressive NAS (PNAS) [36] (225 GPU days [40]).

Most kernels introduce their own hyperparameters, which of course is a downside of Bayesian

GP HPO for ANN. An attempt at handling GP kernel hyperparameters is to integrate them out [32, 41].

Another downside of Bayesian GP HPO for ANN is that GPs scale cubically with the number of observations. They also don't scale well to high dimensions [22]. [42] discussed using ANNs instead of GPs to overcome those limitations. [43] did similar work introducing a framework that they call Bayesian Optimization with Hamiltonian Monte Carlo Artificial Neural Networks (BOHAMIANN) reaching state-of-the art performance for a wide range of optimization tasks.

Some promising directions of Bayesian HPO are incorporating ideas from meta-learning [44], for instance in [45] to initialize Bayesian HPO, and in [46] where to introduce ranking. A comparable alternative to Bayesian optimization (suited for evaluating expensive functions) is perhaps derivative-free HPO methods such as in [47] based on Radial Basis Functions.

### 3.3 Population-based methods

It is natural to think that a search heuristic maintaining a population of architectural candidates can be beneficial when the goal is to find a good architecture and hyperparameter configuration based on available data. For example, the optimal configuration could be regarded as a result of having architectural candidates compete for best fitness in an environment with limited resources and constraints (the data). This imposes novelty, which motivates creativity and thus good configurations [48]. However, less competitive population-based methods have also been investigated for DNN HPO, such as Particle Swarm Optimization (PSO) [49, 50].

A group of algorithms building upon the idea of Evolutionary Strategies (ES) are Genetic Algorithms (GA), which use random mutations and replication to evolve the candidates in a population. Using ES to evolve ANNs is called Neuroevolution [48, 51–53].

With ES, the outer optimization problem in Equation 1 is relaxed by letting possible configurations evolve into new and unexplored configurations before they are eventually evaluated and the most fit reproduced. At the end, the most fit configuration or group of configurations is selected.

Evolutionary methods for training ANNs have existed for a long time [54] under the name Evolutionary ANNs (EANNs) [55, 56], and their fit with ANNs has been well known [53]. More recent research methods have also showed that ES can contribute to NAS, for instance the pioneering work with NeuroEvolution of Augmenting Topologies (NEAT) [57].

A link between learning and evolution has been discussed in the context of ANNs as well, for instance using GAs to initialize weights (evolution) before performing backpropagation gradient-

based training (learning), thereby learning on top of genetic search [58, 59]. This is motivated by the fact that in nature genetic encodings can compactly capture regularities such as symmetries in structure [48]. Moreover, the structures generated by genetic encodings are often more advanced than the genetic encodings, for example is our capacity of 30000 genes in our DNA-based genetic code [60] able to indirectly encode our biological neural network brain of about 100 trillion connections and 100 billion neurons [61].

Successful GAs try to mimic this indirect genotype-phenotype mapping through generations with survival of the fittest. One such example used for evolving ANNs is evolving Compositional Pattern Producing Networks (CPPN) [62] with NEAT [57] and then use CPPNs to generate patterns of weights for ANNs; HyperNEAT [63]. The main finding is that using CPPNs to determine the weight patterns for as ANN as an indirect encoding makes the ANN able to scale to new numbers of inputs and outputs without further evolution. An other example of providing an indirect encoding is Differentiable PPN (DPPN) [64] which was used to compress the weights of a denoising autoencoder. Perhaps the latest innovation of NEAT extending to DNNs is CoDeepNEAT [8]. In this work, NEAT is extended to coevolutionary optimization of components, topologies, and hyperparameters of DNNs.

CoDeepNEAT starts with a population of Directed Asyclic Graphs (DAGs) which they call chromosomes. Each node in a chromosome represents a DNN layer. Following NEAT, chromosomes evolve starting from minimal graphs into more complex graphs through mutations and crossover over generations. Genes from two chromosomes are lined up during crossover based on historical markings. Based on a similarity metric, the population of chromomes become divided into species. Then each species grows proportionally trough fitness and evolution occurs separately in each species. Unlike NEAT, in CoDeepNEAT, GD is used to determine the fitness of an evolved DNN. Each node in each chromosome (DAG) contains a table of real and binary valued hyperparameters that are mutated through uniform Gaussian distribution and random bit-flipping, respectively. These hyperparameters determine the type of layer (such as convolutional, fully connected, or recurrent) and the properties of that layer (such as number of neurons, kernel size, and activation function). In order to do a fitness evaluation a chromosome needs to be converted to a DNN. This is done by traversing the chromosome directed graph, replacing each node with the corresponding layer. See Figure 5 for an illustration of the DNN synthesis as well as Figure 11b for an evolved CNN architecture on the CIFAR-10 dataset.

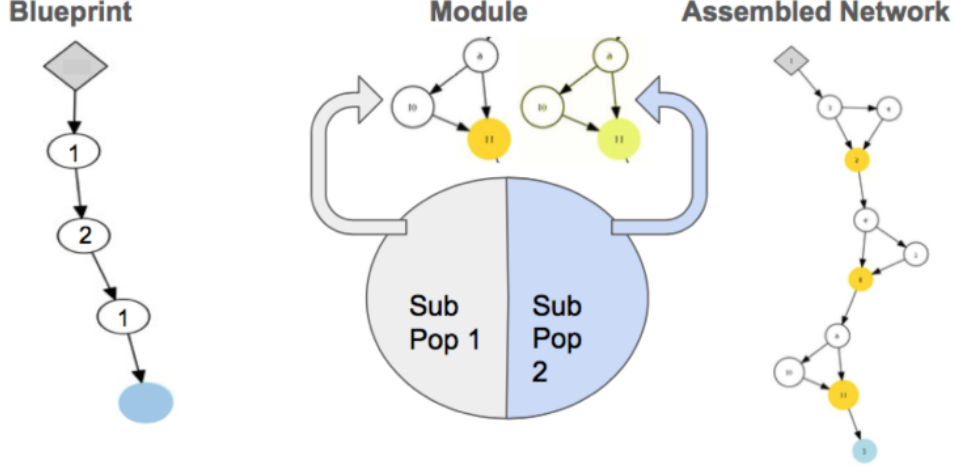


Figure 5: Synthesis of a DNN from a chromosome (Blueprint) in CoDeepNEAT by replacing the nodes of the chromosome with modules (such as convolutional layer) into an assembled DNN. Borrowed from [8].

The main downside of CoDeepNEAT is its extremely demanding computational cost. Although not specified exactly in the work, they report that a single DNN training takes a couple of days on a state-of-the-art GPU. The evolutionary strategy requires thousands of DNNs to be trained through the course of evolution, thus the computational cost makes CoDeepNEAT an infeasible solution for most people.

A lot of recent research has been done on more efficient ES for DNN HPO and NAS [40, 65–74]. In order to simplify the NAS and HPO problem, many of these methods define the search space according to the NASNet search space [75] originally motivated by RL-based NAS [37]. This search space is inspired by the fact that successful deep learning architectures such as Inception and ResNet models [16, 76–79] have repetitively,  $N$  stacked convolutional cells with different configurations, such as number of filters  $F$  and pooling type and size. The NASNet search space tries to generalize this by introducing the overall ANN macro architecture that is known to perform well on two famous datasets: CIFAR-10 and ImageNet. A Cell in the NASNet search space is some combination of interconnected pooling, convolution, summation and concatenation operations, which can be of two types; Normal Cell which returns a feature map of the same dimension, and Reduction Cell which returns a feature map where the feature map height and width is reduced by a factor of two (stride of two) [75]. The idea is that when introduced to a new dataset, for instance the ImageNet dataset, an architecture found from NAS in the CIFAR-10 NASNet search space would be easier to transfer to the new dataset, by

altering a few free parameters such as the number of repeated layers  $N$  and number of filters  $F$  of the Normal Cell. Hence  $N$  and  $F$  are not optimized in the NAS but left to the user as easy design parameters for the model transfer. See Figure 6.

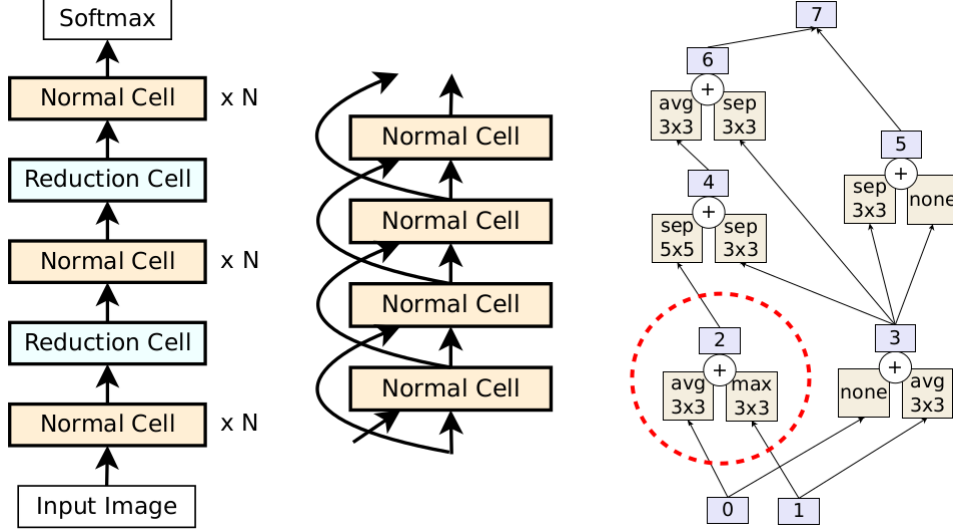


Figure 6: Left: The CIFAR-10 NASNet search space consisting of multiple  $N$  stacked Normal Cells as well as multiple Reduction Cells. Middle: Normal Cells stacked. Skip connections shall also be learned (depending on NAS search strategy). Right: Example of a Cell. Within the red circle is a summation of an average and max pooling operation. Opposed to NASBOT and CoDeepNEAT which search for entire networks, NAS using this kind of search space is considered micro search since it only searches for the best type of (convolutional) Cell. Borrowed from [40].

The creators behind the NASNet search space used RL to search for an optimal convolutional Cell, which resulted in NASNet-A [75]. They used a Proximal Policy Optimization (PPO) method involving a controller (a RNN) that repeatedly predicted the two convolutional Cells that were then synthesized into a candidate network and evaluated through gradient descent training. Others started quickly to adopt the NASNet search space into their own NAS research, for instance [80] (ENAS) using a similar RL approach but with additional parameter sharing between candidate networks speeding up the search by more than 1000x in terms of GPU hours compared to the original RL-based NAS work [37] (those who suggested the NASNet search space, but the original work was conducted before this suggestion and with slightly different search space), [36] (PNAS) which combined RL with an evolutionary algorithm to make a SMBO strategy resulting in 8 times more computational efficiency compared to NASNet-A



[75], and [40] (AmoebaNet-A) which used a tournament selection evolutionary algorithm with an added aging property to get comparable results to PNAS but with even better computational efficiency. Following the convention of NASNet, ENAS and PNAS, AmoebaNet-A was a CNN architecture optimized using NAS on the CIFAR-10 dataset, then augmented (transferred) into a model working on the ImageNet dataset (through retraining). Like initially proposed in NASNet-A [75], the augmentation involved adding a convolution with stride at the start of the architecture. See Figure 12b for a depiction of the evolved AmoebaNet-A CNN. The Cells of RL-based NASNet-A are shown in Figure 12a. Figure 7a shows testing accuracies of the EA (AmoebaNet-A, [40]), RL (PNAS, [36]) and random search ([21]) NAS methods on CIFAR-10 (before model augmentation), while Figure 7b shows final testing accuracies vs. model cost for the final augmented (into ImageNet) models.

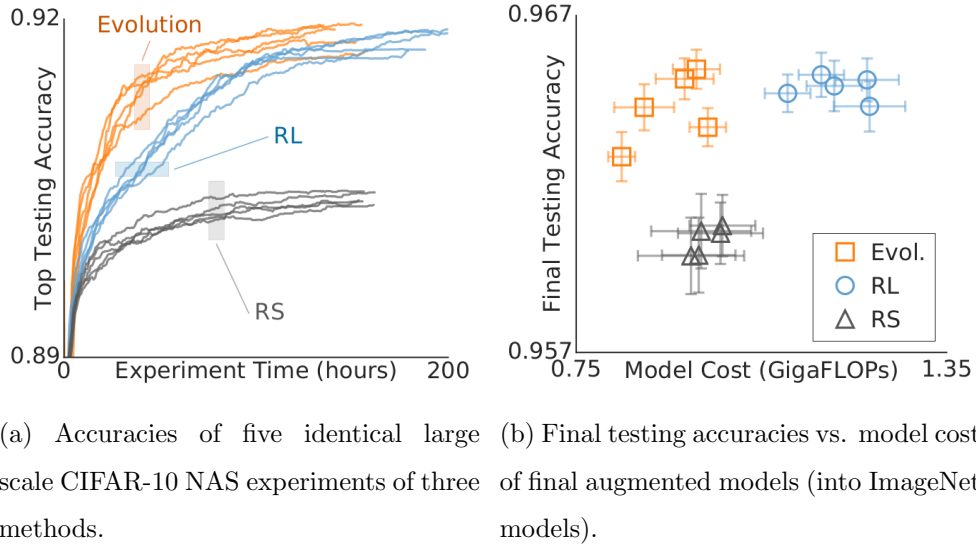


Figure 7: Evolution/Evol.: AmoebaNet-A, [40]. RL: PNAS [36]. RS: A model from random search NAS [21]. Note that RS did not use the NASNet search space, unlike the other two methods. Each large scale experiment ran on 450 Nvidia Tesla K40 GPUs for approx. 7 days to evaluate 20000 models. Borrowed from [40].

As discussed with the case of using ES for NAS, RL has also been investigated for NAS in more general ANN search spaces, like using Q-learning [81] and combining RL with Multi-Objective Optimization [82, 83]. RL has also been used for more narrowed, specific tasks, like searching for activation functions [84], searching for optimal update rules of optimization [85] and learning how to do optimization [86]. Ideas such as re-using weights in NAS using RL [87] and weight sharing in ENAS [80] correspond well to the idea of evolving genotypes for

genotype-phenotype mapping originating from traditional population-based methods discussed earlier [8, 57, 62–64]. Thus, many RL NAS methods incorporate ideas from ES. For this reason, the author decided to discuss RL for NAS as population-based methods.

The well suitedness of ideas from evolution for tuning ANN HPO and NAS has been discussed previously. Moreover, not only traditional population-based methods such as GAs benefit from those ideas in the context of NAS. One example was RL NAS. The fact that raw computational power has been a limiting factor for the success of NAS in early years [88] is exiting when taking into account how much more powerful computers we have today. The increase in available computation power is likely one of the main reasons why Deep Learning became so popular in the recent years. Nevertheless, lack of data and computation power in the early years also motivated creative ideas such as prediction risk for cheap model performance evaluation [88] and modifications of the NAS search space (to have less computationally demanding NAS), such as (neuron) sensitivity-based input pruning (SBP) and optimal brain damage (OBD) weight pruning [88].

In a recent Neuroevolution review article [48], the authors discussed in a future outlook that evolution in general is an open playground for testing and proving ideas, as it imposes few limitations on imaginable models. One such limitation that Neuroevolution methods such as GAs, do not impose, is that a model needs to be differentiable. They continue claiming that ultimately, if it is possible to translate such ideas (evolution) into gradient descent, then the effort may result in greater scalability and efficiency.

### 3.4 Gradient-based methods

Gradient-based NAS is perhaps the most interesting direction for ANN HPO and NAS. For example, [89] introduced HyperNetworks, networks that are used to generate the weights of another network. This is analogous to what was done in HyperNEAT [63] discussed previously, but unlike HyperNEAT, everything is now trained end-to-end with backpropagation gradient descent. The HyperNetwork thus incorporates the evolutionary idea of genotype(HyperNetwork)-phenotype mapping into a gradient-based method. The phenotype would be analogous to traditional gradient descent training of the neural network. [90] did similar work but with additional Multi-Objective Optimization, while [91] investigated using deconvolutional neural networks to design CNNs.

Some RL NAS methods use gradients, for instance PPO RL in NASNet-A [75] and MnasNet [83], and Policy Gradient RL in MONAS [82]. Moreover, an ES that use gradients [92], called

Natural ES (NES) [93] has been discussed as a scalable alternative to RL.

Some gradient-based methods that have entered HPO and NAS are a method of reversing stochastic gradient descent to get gradients of hyperparameters [94], NAS by network morphisms [95, 96] as well as an early conceptualization of gradient-based HPO [97].

More recently, [98] formulated the NAS problem as a continuous optimization problem giving it the name Neural Architecture Optimization (NAO), using a neural network based encoder, decoder and regression model (predictor) to optimize for best architecture. Differentiable Architecture Search (DARTS) [99] favours a simpler method of alternating gradient descent optimization of hyperparameter and traditional parameters to simultaneously optimize hyperparameters and (traditionally) learnable parameters (training). Like previously discussed RL-based (f. ex. NASNet [75]) and EA-based (f. ex. AmoebaNet-A [40]) methods, both DARTS [99] and NAO [98] perform NAS in the NASNet search space [75] presented earlier. The author favours DARTS for its simplicity and speed. Unlike earlier, gradient-based NAS enables the use of commodity hardware with few GPUs. See Figure 8 for a comparison of state-of-the-art CIFAR-10 image classifiers, mostly searched using NAS in the NASNet search space.

Model	B	N	F	#op	Error(%)	#params	M	GPU Days
DenseNet-BC [19]		100	40	/	3.46	25.6M	/	/
ResNeXt-29 [41]				/	3.58	68.1M	/	/
NASNet-A [45]	5	6	32	13	3.41	3.3M	20000	2000
NASNet-B [45]	5	4	N/A	13	3.73	2.6M	20000	2000
NASNet-C [45]	5	4	N/A	13	3.59	3.1M	20000	2000
Hier-EA [27]	5	2	64	6	3.75	15.7M	7000	300
AmoebaNet-A [35]	5	6	36	10	3.34	3.2M	20000	3150
AmoebaNet-B [35]	5	6	36	19	3.37	2.8M	27000	3150
AmoebaNet-B [35]	5	6	80	19	3.04	13.7M	27000	3150
AmoebaNet-B [35]	5	6	128	19	2.98	34.9M	27000	3150
AmoebaNet-B + Cutout [35]	5	6	128	19	2.13	34.9M	27000	3150
PNAS [26]	5	3	48	8	3.41	3.2M	1280	225
ENAS [34]	5	5	36	5	3.54	4.6M	/	0.45
Random-WS	5	5	36	5	3.92	3.9M	/	0.25
DARTS + Cutout [28]	5	6	36	7	2.83	4.6M	/	4
NAONet	5	6	36	11	3.18	10.6M	1000	200
NAONet	5	6	64	11	2.98	28.6M	1000	200
NAONet + Cutout	5	6	128	11	2.11	128M	1000	200
NAONet-WS	5	5	36	5	3.53	2.5M	/	0.3

Figure 8: A comparison of state-of-the-art image classifiers on CIFAR-10 conducted in [98]. The most interesting columns are Error, which is the top-1 validation error on CIFAR-10, as well as GPU Days. All architectures except the two on the top were optimized using varying NAS methods in the NASNet search space. Please consult [98] for cited works. The Bayesian NAS discussed previously, NASBOT [26], has a computational cost that is comparable with DARTS [99]. However, NASBOT does not provide comparable accuracy, see Figure 4 and 9. The following figure description was taken from [98] and slightly modified: 'N' is the number of times the discovered Normal Cell is unrolled to form the final CNN architecture. 'F' represents the filter size. 'op' is the number of different operation for one branch in the Cell, which is an indicator of the scale of architecture space for automatic architecture design algorithm. 'M' is the total number of network architectures that are trained to obtain the claimed performance. '/' denotes that the criteria is meaningless for a particular algorithm. 'NAONet-WS' represents the architecture discovered by NAO ([98]) and the weight sharing method as described in the ENAS work [80] (See NAO article [98], section 3.4). 'Random-WS' represents the random search baseline, conducted in the weight sharing setting of ENAS [80].

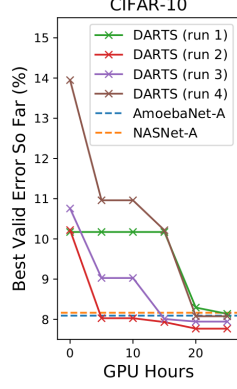


Figure 9: Validation errors during four runs of the DARTS NAS [99] with different random seeds, as well as competing NAS methods, on CIFAR-10. Each run is on a single NVidia GTX 1080Ti. As can be seen, a NAS run on CIFAR-10 takes approximately one day. In Figure 8, results from DARTS is shown only for the 2. order version. There the search cost is four GPU days.

The main idea behind DARTS is a continuous relaxation of the (categorical) candidate operations (for instance convolution and max pooling) within the Cell by a softmax over all possible operations (see [99] Equation 2). A weight for each candidate operation is then associated with all possible edges between each possible combination of nodes within a Cell, together to form the architecture configuration  $\alpha$ . When the NAS is finished, final single edge operations between nodes are determined by taking operations with largest weight between nodes in  $\alpha$ . With this formulation of possible operations, the NAS reduces to finding the optimal architecture  $\alpha^*$  with the optimal weights  $w^*$  according to the bilevel optimization problem (with  $\alpha$  as upper-level variable and  $w$  as the lower-level variable)

$$\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha) \quad (7)$$

$$s.t. \quad w^*(\alpha) = \operatorname{argmin}_w \mathcal{L}_{train}(w, \alpha) \quad (8)$$

where

$$w^* = \operatorname{argmin}_w \mathcal{L}_{train}(w, \alpha^*) \quad (9)$$

The bilevel optimization problem defined in Equations 7, 8 and 9 is then solved by an approximate iterative optimization procedure where  $w$  and  $\alpha$  are optimized by alternating

between gradient descent steps in the weight and architecture spaces respectively. See Figure 10 for an overall illustration of the NAS, as well as Figure 12c and Figure 12d for optimized architecture in the NASNet search space based on CIFAR-10 data using this method. Relating to the HPO for ANN definition in Equation 1,  $\alpha$  corresponds to  $\lambda$ , while  $w$  corresponds to  $\theta$ .

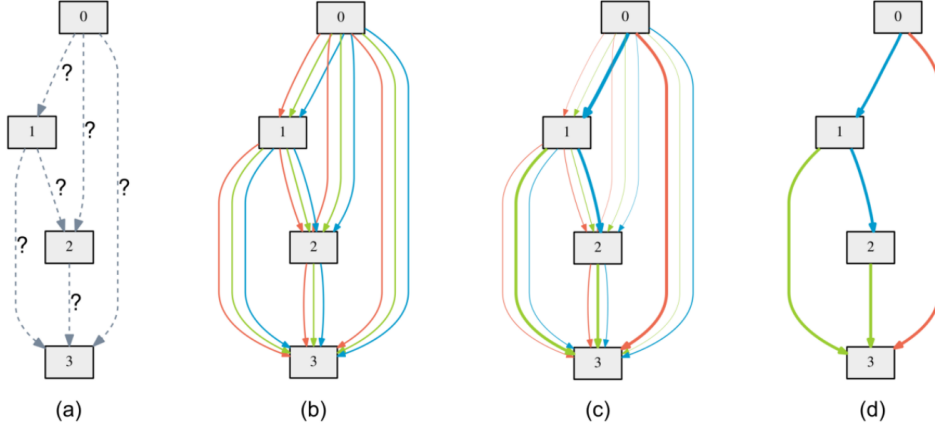
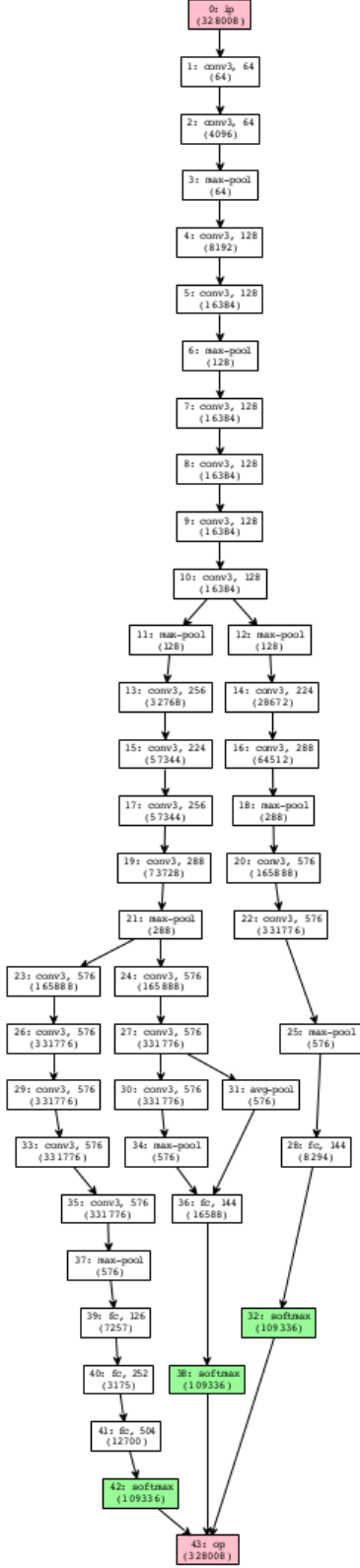
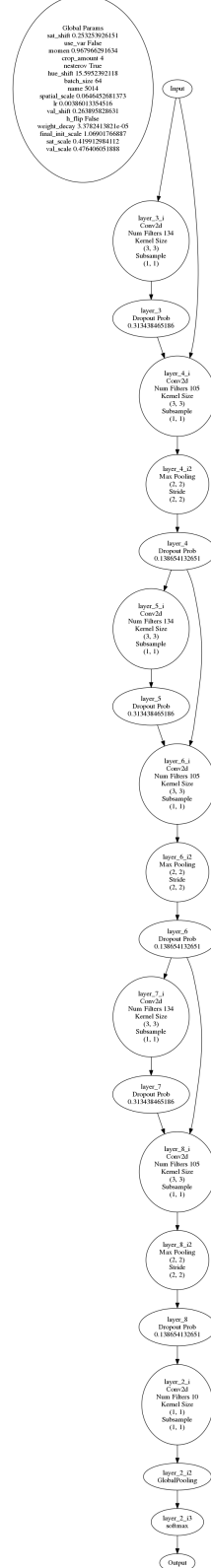


Figure 10: An overview of the DARTS method [99]. Normal and Reduction Cell according to [75] is optimized using gradient descent. (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Inducing the final architecture from the learned mixing probabilities. Figure and description borrowed from [99].

DARTS has already been improved by performing some adjustments and additions creating sharpDARTS [100], for instance a new type of Normal and Reduction Cell called SharpSetConv, Cosine Power Annealing learning rate schedule, as well as a method for handling hyperparameters of a Cell (hyperhyperparameters?). SharpDARTS also makes use of an automatic data augmentation method, AutoAugment [101], which improves the validation results. Nevertheless, at the time of this writing, gradient-based HPO and NAS such as DARTS is an exiting topic of NAS that provides state-of-the-art automatically optimized neural network architectures suited for Deep Learning on relatively cheap hardware. This ultimately points in the direction of removing the sub-optimal human dependence in neural network engineering and thus more interpretable and reproducible Deep Learning.

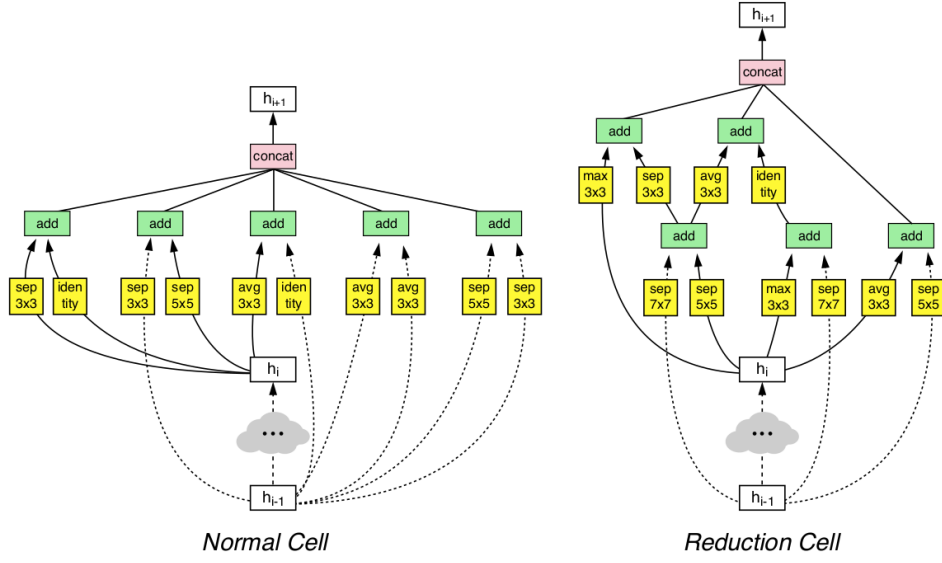


(a) NASBOT (Bayesian Optimization) [26]

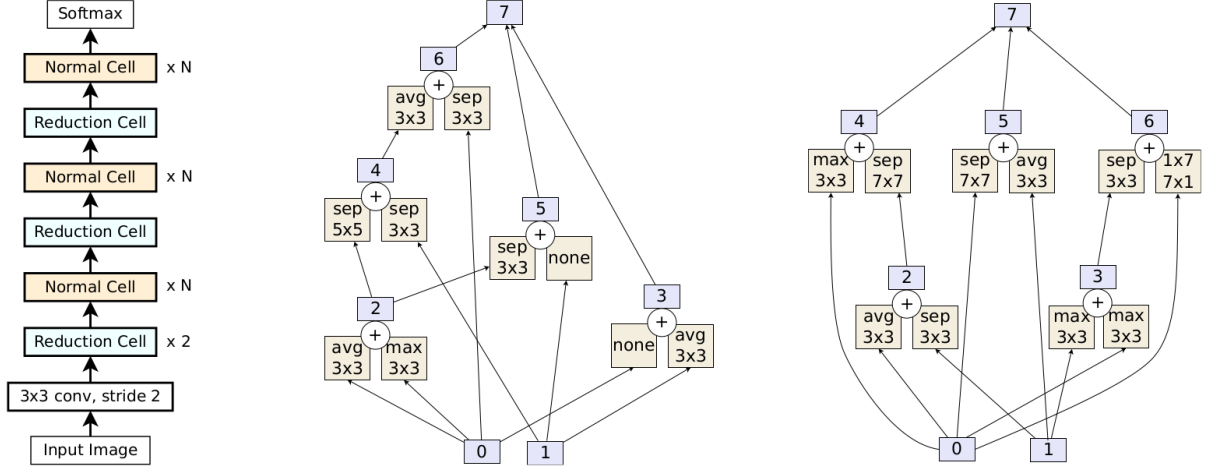


(b) CoDeepNEAT (EA) [8]

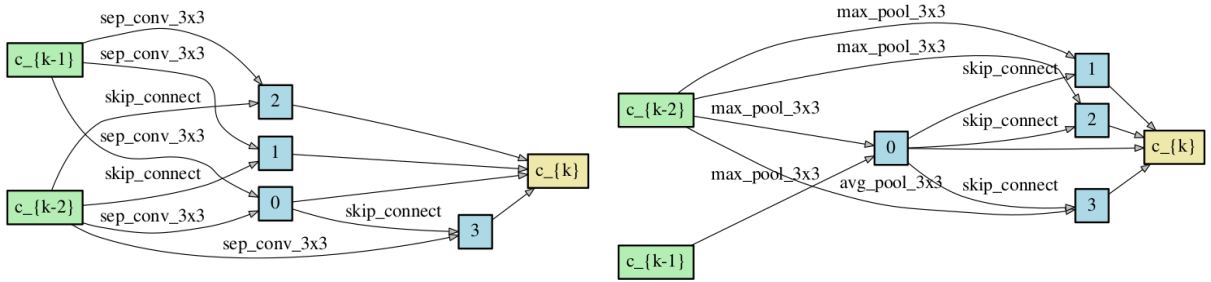
Figure 11: CNNs from (entire network, macro) NAS on CIFAR-10.



(a) NASNet-A Cells (RL) [75]. The Cells are used in the same augmented network as in 12b Left.



(b) AmoebaNet-A (EA) [40]. Left: Augmented network for ImageNet. Middle: Normal Cell. Right: Reduction Cell.



(c) DARTS Normal Cell (gradient-based) [99]

(d) DARTS Reduction Cell (gradient-based) [99]

Figure 12: CNNs from (convolutional Cell search, micro) NAS on CIFAR-10 and transferred into an ImageNet model by macro-model augmentation and gradient descent re-training.



## References

- [1] Liu, W. et al. “A survey of deep neural network architectures and their applications”. In: *Neurocomputing* 234 (Apr. 19, 2017), pp. 11–26. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2016.12.038. URL: <http://www.sciencedirect.com/science/article/pii/S0925231216315533> (visited on 04/12/2019).
- [2] Srivastava, R. K., Greff, K., and Schmidhuber, J. “Training Very Deep Networks”. In: *Advances in Neural Information Processing Systems 28*. Ed. by Cortes, C. et al. Curran Associates, Inc., 2015, pp. 2377–2385. URL: <http://papers.nips.cc/paper/5850-training-very-deep-networks.pdf> (visited on 04/09/2019).
- [3] Hu, J., Shen, L., and Sun, G. “Squeeze-and-Excitation Networks”. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018, pp. 7132–7141. URL: [http://openaccess.thecvf.com/content\\_cvpr\\_2018/html/Hu\\_Squeeze-and-Excitation\\_Networks\\_CVPR\\_2018\\_paper.html](http://openaccess.thecvf.com/content_cvpr_2018/html/Hu_Squeeze-and-Excitation_Networks_CVPR_2018_paper.html) (visited on 04/09/2019).
- [4] Schmidhuber, J. “Deep Learning in Neural Networks: An Overview”. In: *Neural Networks* 61 (Jan. 2015), pp. 85–117. ISSN: 08936080. DOI: 10.1016/j.neunet.2014.09.003. URL: <http://arxiv.org/abs/1404.7828> (visited on 04/12/2019).
- [5] Vaswani, A. et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30*. Ed. by Guyon, I. et al. Curran Associates, Inc., 2017, pp. 5998–6008. URL: <http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf> (visited on 05/02/2019).
- [6] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. “Learning representations by back-propagating errors”. In: *Cognitive modeling* 5.3 (1988), p. 1.
- [7] Mendoza, H. et al. “Towards Automatically-Tuned Neural Networks”. In: (2016), p. 8.
- [8] Miikkulainen, R. et al. “Chapter 15 - Evolving Deep Neural Networks”. In: *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Ed. by Kozma, R. et al. Academic Press, Jan. 1, 2019, pp. 293–312. ISBN: 978-0-12-815480-9. URL: <http://www.sciencedirect.com/science/article/pii/B9780128154809000153> (visited on 04/09/2019).
- [9] Veen, F. v. *The Neural Network Zoo*. The Asimov Institute. Sept. 14, 2016. URL: <http://www.asimovinstitute.org/neural-network-zoo/> (visited on 04/22/2019).

- [10] Smith, L. N. “A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay”. In: *arXiv:1803.09820 [cs, stat]* (Mar. 26, 2018). URL: <http://arxiv.org/abs/1803.09820> (visited on 04/24/2019).
- [11] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)”. In: *arXiv:1511.07289 [cs]* (Nov. 23, 2015). URL: <http://arxiv.org/abs/1511.07289> (visited on 05/02/2019).
- [12] Srivastava, N. et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html> (visited on 04/09/2019).
- [13] Salimans, T. and Kingma, D. P. “Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks”. In: *arXiv:1602.07868 [cs]* (Feb. 25, 2016). URL: <http://arxiv.org/abs/1602.07868> (visited on 04/09/2019).
- [14] He, K. et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1026–1034. URL: [https://www.cv-foundation.org/openaccess/content\\_iccv\\_2015/html/He\\_Delving\\_Deep\\_into\\_ICCV\\_2015\\_paper.html](https://www.cv-foundation.org/openaccess/content_iccv_2015/html/He_Delving_Deep_into_ICCV_2015_paper.html) (visited on 04/30/2019).
- [15] Ioffe, S. and Szegedy, C. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *arXiv:1502.03167 [cs]* (Feb. 10, 2015). URL: <http://arxiv.org/abs/1502.03167> (visited on 04/30/2019).
- [16] Simonyan, K. and Zisserman, A. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *arXiv:1409.1556 [cs]* (Sept. 4, 2014). URL: <http://arxiv.org/abs/1409.1556> (visited on 05/04/2019).
- [17] Deng, J. et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops). Miami, FL: IEEE, June 2009, pp. 248–255. ISBN: 978-1-4244-3992-8. DOI: 10.1109/CVPR.2009.5206848. URL: <https://ieeexplore.ieee.org/document/5206848/> (visited on 05/04/2019).
- [18] Eggenberger, K., Lindauer, M., and Hutter, F. “Pitfalls and Best Practices in Algorithm Configuration”. In: *Journal of Artificial Intelligence Research* 64 (Apr. 16, 2019),

- pp. 861–893. ISSN: 1076-9757. DOI: 10.1613/jair.1.11420. URL: <https://www.jair.org/index.php/jair/article/view/11420> (visited on 05/02/2019).
- [19] Bergstra, J., Yamins, D., and Cox, D. D. “Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures”. In: (2013), p. 9.
  - [20] Elsken, T., Metzen, J. H., and Hutter, F. “Neural Architecture Search: A Survey”. In: *arXiv:1808.05377 [cs, stat]* (Aug. 16, 2018). URL: <http://arxiv.org/abs/1808.05377> (visited on 04/12/2019).
  - [21] Bergstra, J. and Bengio, Y. “Random Search for Hyper-Parameter Optimization”. In: *Journal of Machine Learning Research* 13 (Feb 2012), pp. 281–305. ISSN: ISSN 1533-7928. URL: <http://www.jmlr.org/papers/v13/bergstra12a.html> (visited on 04/09/2019).
  - [22] Hutter, F., Kotthoff, L., and Vanschoren, J. *Automatic machine learning: methods, systems, challenges*. Springer, 2019. ISBN: 978-3-030-05317-8 978-3-030-05318-5. DOI: 10.1007/978-3-030-05318-5. URL: <https://research.tue.nl/en/publications/automatic-machine-learning-methods-systems-challenges> (visited on 04/09/2019).
  - [23] Claesen, M. and De Moor, B. “Hyperparameter Search in Machine Learning”. In: *arXiv:1502.02127 [cs, stat]* (Feb. 7, 2015). URL: <http://arxiv.org/abs/1502.02127> (visited on 04/09/2019).
  - [24] Li, L. et al. “Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization”. In: (2016), p. 52.
  - [25] Mackay, D. J. C. “Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks”. In: *Network: Computation in Neural Systems* 6.3 (Jan. 1, 1995), pp. 469–505. ISSN: 0954-898X. DOI: 10.1088/0954-898X\_6\_3\_011. URL: [https://doi.org/10.1088/0954-898X\\_6\\_3\\_011](https://doi.org/10.1088/0954-898X_6_3_011) (visited on 04/09/2019).
  - [26] Kandasamy, K. et al. “Neural Architecture Search with Bayesian Optimisation and Optimal Transport”. In: *Advances in Neural Information Processing Systems 31*. Ed. by Bengio, S. et al. Curran Associates, Inc., 2018, pp. 2016–2025. URL: <http://papers.nips.cc/paper/7472-neural-architecture-search-with-bayesian-optimisation-and-optimal-transport.pdf> (visited on 04/12/2019).
  - [27] Kotthoff, L. et al. “Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA”. In: (2017), p. 5.

- [28] Golovin, D. et al. “Google Vizier: A Service for Black-Box Optimization”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’17. New York, NY, USA: ACM, 2017, pp. 1487–1495. ISBN: 978-1-4503-4887-4. DOI: 10.1145/3097983.3098043. URL: <http://doi.acm.org/10.1145/3097983.3098043> (visited on 04/09/2019).
- [29] Klein, A. et al. “Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets”. In: *arXiv:1605.07079 [cs, stat]* (May 23, 2016). arXiv: 1605.07079. URL: <http://arxiv.org/abs/1605.07079> (visited on 04/09/2019).
- [30] Brochu, E., Cora, V. M., and Freitas, N. de. “A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning”. In: *arXiv:1012.2599 [cs]* (Dec. 12, 2010). URL: <http://arxiv.org/abs/1012.2599> (visited on 04/29/2019).
- [31] Shahriari, B. et al. “Taking the Human Out of the Loop: A Review of Bayesian Optimization”. In: *Proceedings of the IEEE* 104.1 (Jan. 2016), pp. 148–175. ISSN: 0018-9219. DOI: 10.1109/JPROC.2015.2494218.
- [32] Snoek, J., Larochelle, H., and Adams, R. P. “Practical Bayesian Optimization of Machine Learning Algorithms”. In: *Advances in Neural Information Processing Systems 25*. Ed. by Pereira, F. et al. Curran Associates, Inc., 2012, pp. 2951–2959. URL: <http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf> (visited on 04/09/2019).
- [33] Bergstra, J. S. et al. “Algorithms for Hyper-Parameter Optimization”. In: *Advances in Neural Information Processing Systems 24*. Ed. by Shawe-Taylor, J. et al. Curran Associates, Inc., 2011, pp. 2546–2554. URL: <http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf> (visited on 04/09/2019).
- [34] Murphy, K. P. *Machine learning: a probabilistic perspective*. MIT press, 2012. ISBN: 0-262-30432-5.
- [35] Villani, C. *Optimal transport: old and new*. Vol. 338. Springer Science & Business Media, 2008. ISBN: 3-540-71050-7.
- [36] Liu, C. et al. “Progressive Neural Architecture Search”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 19–34. URL: <http://openaccess.>

- thecvf.com/content\_ECCV\_2018/html/Chenxi\_Liu\_Progressive\_Neural\_Architecture\_ECCV\_2018\_paper.html (visited on 04/12/2019).
- [37] Zoph, B. and Le, Q. V. “Neural Architecture Search with Reinforcement Learning”. In: *arXiv:1611.01578 [cs]* (Nov. 4, 2016). arXiv: 1611.01578. URL: <http://arxiv.org/abs/1611.01578> (visited on 04/09/2019).
  - [38] Krizhevsky, A. *Learning multiple layers of features from tiny images*. 2009.
  - [39] Borgli, R. J. “Hyperparameter optimization using Bayesian optimization on transfer learning for medical image classification”. In: (2018). URL: <https://www.duo.uio.no/handle/10852/64146> (visited on 12/12/2018).
  - [40] Real, E. et al. “Regularized Evolution for Image Classifier Architecture Search”. In: *arXiv:1802.01548 [cs]* (Feb. 5, 2018). URL: <http://arxiv.org/abs/1802.01548> (visited on 05/03/2019).
  - [41] MacKay, D. J. C. “Hyperparameters: Optimize, or Integrate Out?” In: *Maximum Entropy and Bayesian Methods*. Ed. by Heidbreder, G. R. 141. Dordrecht: Springer Netherlands, 1996, pp. 43–59. ISBN: 978-90-481-4407-5 978-94-015-8729-7. URL: [http://link.springer.com/10.1007/978-94-015-8729-7\\_2](http://link.springer.com/10.1007/978-94-015-8729-7_2) (visited on 04/09/2019).
  - [42] Snoek, J. et al. “Scalable Bayesian Optimization Using Deep Neural Networks”. In: *arXiv:1502.05700 [stat]* (Feb. 19, 2015). URL: <http://arxiv.org/abs/1502.05700> (visited on 04/09/2019).
  - [43] Springenberg, J. T. et al. “Bayesian Optimization with Robust Bayesian Neural Networks”. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS’16. USA: Curran Associates Inc., 2016, pp. 4141–4149. ISBN: 978-1-5108-3881-9. URL: <http://dl.acm.org/citation.cfm?id=3157382.3157560> (visited on 04/09/2019).
  - [44] Brazdil, P. et al. *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008. ISBN: 3-540-73262-4.
  - [45] Feurer, M., Springenberg, J. T., and Hutter, F. “Initializing Bayesian Hyperparameter Optimization via Meta-Learning”. In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. Twenty-Ninth AAAI Conference on Artificial Intelligence. Feb. 16, 2015. URL: <https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/10029> (visited on 04/09/2019).

- [46] Bardenet, R. et al. “Collaborative hyperparameter tuning”. In: (2013), p. 9.
- [47] Diaz, G. I. et al. “An effective algorithm for hyperparameter optimization of neural networks”. In: *IBM Journal of Research and Development* 61.4 (July 2017), 9:1–9:11. ISSN: 0018-8646. DOI: 10.1147/JRD.2017.2709578.
- [48] Stanley, K. O. et al. “Designing neural networks through neuroevolution”. In: *Nature Machine Intelligence* 1.1 (Jan. 2019), p. 24. ISSN: 2522-5839. DOI: 10.1038/s42256-018-0006-z. URL: <https://www.nature.com/articles/s42256-018-0006-z> (visited on 04/12/2019).
- [49] Lorenzo, P. R. et al. “Particle Swarm Optimization for Hyper-parameter Selection in Deep Neural Networks”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO ’17. New York, NY, USA: ACM, 2017, pp. 481–488. ISBN: 978-1-4503-4920-8. DOI: 10.1145/3071178.3071208. URL: <http://doi.acm.org/10.1145/3071178.3071208> (visited on 04/09/2019).
- [50] Lorenzo, P. R. et al. “Hyper-parameter Selection in Deep Neural Networks Using Parallel Particle Swarm Optimization”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO ’17. New York, NY, USA: ACM, 2017, pp. 1864–1871. ISBN: 978-1-4503-4939-0. DOI: 10.1145/3067695.3084211. URL: <http://doi.acm.org/10.1145/3067695.3084211> (visited on 05/03/2019).
- [51] Hoekstra, V. “An overview of neuroevolution techniques”. In: (2011), p. 36.
- [52] Floreano, D., Dürr, P., and Mattiussi, C. “Neuroevolution: from architectures to learning”. In: *Evolutionary Intelligence* 1.1 (Mar. 1, 2008), pp. 47–62. ISSN: 1864-5917. DOI: 10.1007/s12065-007-0002-4. URL: <https://doi.org/10.1007/s12065-007-0002-4> (visited on 04/12/2019).
- [53] Yao, X. “Evolving artificial neural networks”. In: *Proceedings of the IEEE* 87.9 (Sept. 1999), pp. 1423–1447. ISSN: 0018-9219. DOI: 10.1109/5.784219.
- [54] Miller, G. F., Todd, P. M., and Hegde, S. U. “Designing Neural Networks Using Genetic Algorithms”. In: *Proceedings of the 3rd International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 379–384. ISBN: 978-1-55860-066-9. URL: <http://dl.acm.org/citation.cfm?id=645512.657097> (visited on 04/12/2019).

- [55] Yao, X. and Liu, Y. “Towards designing artificial neural networks by evolution”. In: *Applied Mathematics and Computation* 91.1 (Apr. 1, 1998), pp. 83–90. ISSN: 0096-3003. DOI: 10.1016/S0096-3003(97)10005-4. URL: <http://www.sciencedirect.com/science/article/pii/S0096300397100054> (visited on 04/12/2019).
- [56] Carvalho, A. R., Ramos, F. M., and Chaves, A. A. “Metaheuristics for the feedforward artificial neural network (ANN) architecture optimization problem”. In: *Neural Computing and Applications* 20.8 (Nov. 1, 2011), pp. 1273–1284. ISSN: 1433-3058. DOI: 10.1007/s00521-010-0504-3. URL: <https://doi.org/10.1007/s00521-010-0504-3> (visited on 04/12/2019).
- [57] Stanley, K. O. and Miikkulainen, R. “Evolving Neural Networks through Augmenting Topologies”. In: *Evolutionary Computation* 10.2 (June 1, 2002), pp. 99–127. ISSN: 1063-6560. DOI: 10.1162/106365602320169811. URL: <https://doi.org/10.1162/106365602320169811> (visited on 04/12/2019).
- [58] Belew, R. K., Mcinerney, J., and Schraudolph, N. N. “Evolving Networks: Using the Genetic Algorithm with Connectionist Learning”. In: *In*. Addison-Wesley, 1990, pp. 511–547.
- [59] Whitley, D. “Genetic Algorithms and Neural Networks”. In: *Genetic Algorithms in Engineering and Computer Science*. 195. John Wiley, pp. 191–201.
- [60] Venter, J. C. et al. “The Sequence of the Human Genome”. In: *Science* 291.5507 (Feb. 16, 2001), pp. 1304–1351. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1058040. URL: <https://science.sciencemag.org/content/291/5507/1304> (visited on 05/04/2019).
- [61] Herculano-Houzel, S. “The human brain in numbers: a linearly scaled-up primate brain”. In: *Frontiers in Human Neuroscience* 3 (2009). ISSN: 1662-5161. DOI: 10.3389/neuro.09.031.2009. URL: <https://www.frontiersin.org/articles/10.3389/neuro.09.031.2009/full> (visited on 05/04/2019).
- [62] Stanley, K. O. “Compositional pattern producing networks: A novel abstraction of development”. In: *Genetic Programming and Evolvable Machines* 8.2 (June 1, 2007), pp. 131–162. ISSN: 1573-7632. DOI: 10.1007/s10710-007-9028-8. URL: <https://doi.org/10.1007/s10710-007-9028-8> (visited on 04/12/2019).
- [63] Stanley, K. O., D’Ambrosio, D. B., and Gauci, J. “A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks”. In: *Artificial Life* 15.2 (Jan. 14, 2009), pp. 185–

212. ISSN: 1064-5462. DOI: 10.1162/artl.2009.15.2.15202. URL: <https://doi.org/10.1162/artl.2009.15.2.15202> (visited on 04/12/2019).
- [64] Fernando, C. et al. “Convolution by Evolution: Differentiable Pattern Producing Networks”. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. GECCO ’16. New York, NY, USA: ACM, 2016, pp. 109–116. ISBN: 978-1-4503-4206-3. DOI: 10.1145/2908812.2908890. URL: <http://doi.acm.org/10.1145/2908812.2908890> (visited on 04/12/2019).
- [65] Real, E. et al. “Large-scale Evolution of Image Classifiers”. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML’17. JMLR.org, 2017, pp. 2902–2911. URL: <http://dl.acm.org/citation.cfm?id=3305890.3305981> (visited on 04/12/2019).
- [66] Liu, H. et al. “Hierarchical Representations for Efficient Architecture Search”. In: *arXiv:1711.00436 [cs, stat]* (Nov. 1, 2017). URL: <http://arxiv.org/abs/1711.00436> (visited on 05/03/2019).
- [67] Jaderberg, M. et al. “Population Based Training of Neural Networks”. In: *arXiv:1711.09846 [cs]* (Nov. 27, 2017). URL: <http://arxiv.org/abs/1711.09846> (visited on 04/09/2019).
- [68] Desell, T. “Large Scale Evolution of Convolutional Neural Networks Using Volunteer Computing”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO ’17. New York, NY, USA: ACM, 2017, pp. 127–128. ISBN: 978-1-4503-4939-0. DOI: 10.1145/3067695.3076002. URL: <http://doi.acm.org/10.1145/3067695.3076002> (visited on 04/12/2019).
- [69] Xie, L. and Yuille, A. “Genetic CNN”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1379–1388. URL: [http://openaccess.thecvf.com/content\\_iccv\\_2017/html/Xie\\_Genetic\\_CNN\\_ICCV\\_2017\\_paper.html](http://openaccess.thecvf.com/content_iccv_2017/html/Xie_Genetic_CNN_ICCV_2017_paper.html) (visited on 04/12/2019).
- [70] Suganuma, M., Shirakawa, S., and Nagao, T. “A genetic programming approach to designing convolutional neural network architectures”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, July 1, 2017, pp. 497–504. ISBN: 978-1-4503-4920-8. DOI: 10.1145/3071178.3071229. URL: <http://dl.acm.org/citation.cfm?id=3071178.3071229> (visited on 04/09/2019).



- [71] Ghazi-Zahedi, K. “NMODE — Neuro-MODule Evolution”. In: *arXiv:1701.05121 [cs]* (Jan. 18, 2017). URL: <http://arxiv.org/abs/1701.05121> (visited on 04/12/2019).
- [72] Loshchilov, I. and Hutter, F. “CMA-ES for Hyperparameter Optimization of Deep Neural Networks”. In: *arXiv:1604.07269 [cs]* (Apr. 25, 2016). URL: <http://arxiv.org/abs/1604.07269> (visited on 04/09/2019).
- [73] Assunção, F. et al. “DENSER: deep evolutionary network structured representation”. In: *Genetic Programming and Evolvable Machines* 20.1 (Mar. 1, 2019), pp. 5–35. ISSN: 1573-7632. DOI: 10.1007/s10710-018-9339-y. URL: <https://doi.org/10.1007/s10710-018-9339-y> (visited on 04/12/2019).
- [74] Young, S. R. et al. “Optimizing Deep Learning Hyper-parameters Through an Evolutionary Algorithm”. In: *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*. MLHPC ’15. New York, NY, USA: ACM, 2015, 4:1–4:5. ISBN: 978-1-4503-4006-9. DOI: 10.1145/2834892.2834896. URL: <http://doi.acm.org/10.1145/2834892.2834896> (visited on 04/09/2019).
- [75] Zoph, B. et al. “Learning Transferable Architectures for Scalable Image Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8697–8710. URL: [http://openaccess.thecvf.com/content\\_cvpr\\_2018/html/Zoph\\_Learning\\_Transferable\\_Architectures\\_CVPR\\_2018\\_paper.html](http://openaccess.thecvf.com/content_cvpr_2018/html/Zoph_Learning_Transferable_Architectures_CVPR_2018_paper.html) (visited on 04/12/2019).
- [76] Szegedy, C. et al. “Going Deeper With Convolutions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9. URL: [https://www.cv-foundation.org/openaccess/content\\_cvpr\\_2015/html/Szegedy\\_Going\\_Deeper\\_With\\_2015\\_CVPR\\_paper.html](https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html) (visited on 05/04/2019).
- [77] He, K. et al. “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778. URL: [http://openaccess.thecvf.com/content\\_cvpr\\_2016/html/He\\_Deep\\_Residual\\_Learning\\_CVPR\\_2016\\_paper.html](http://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html) (visited on 05/04/2019).
- [78] Szegedy, C. et al. “Rethinking the Inception Architecture for Computer Vision”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3673. 2016, pp. 2818–2826. URL: [https://www.cv-foundation.org/openaccess/content\\_cvpr\\_2016/html/Szegedy\\_Rethinking\\_the\\_Inception\\_CVPR\\_2016\\_paper.html](https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Szegedy_Rethinking_the_Inception_CVPR_2016_paper.html) (visited on 05/04/2019).

- [79] Szegedy, C. et al. “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”. In: *Thirty-First AAAI Conference on Artificial Intelligence*. Thirty-First AAAI Conference on Artificial Intelligence. Feb. 12, 2017. URL: <https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14806> (visited on 05/04/2019).
- [80] Pham, H. et al. “Efficient Neural Architecture Search via Parameter Sharing”. In: *arXiv:1802.03268 [cs, stat]* (Feb. 9, 2018). URL: <http://arxiv.org/abs/1802.03268> (visited on 04/12/2019).
- [81] Baker, B. et al. “Designing Neural Network Architectures using Reinforcement Learning”. In: *arXiv:1611.02167 [cs]* (Nov. 7, 2016). URL: <http://arxiv.org/abs/1611.02167> (visited on 04/09/2019).
- [82] Hsu, C.-H. et al. “MONAS: Multi-Objective Neural Architecture Search using Reinforcement Learning”. In: *arXiv:1806.10332 [cs, stat]* (June 27, 2018). URL: <http://arxiv.org/abs/1806.10332> (visited on 04/12/2019).
- [83] Tan, M. et al. “MnasNet: Platform-Aware Neural Architecture Search for Mobile”. In: *arXiv:1807.11626 [cs]* (July 30, 2018). URL: <http://arxiv.org/abs/1807.11626> (visited on 04/12/2019).
- [84] Ramachandran, P., Zoph, B., and Le, Q. V. “Searching for Activation Functions”. In: *arXiv:1710.05941 [cs]* (Oct. 16, 2017). URL: <http://arxiv.org/abs/1710.05941> (visited on 04/12/2019).
- [85] Bello, I. et al. “Neural Optimizer Search with Reinforcement Learning”. In: *arXiv:1709.07417 [cs, stat]* (Sept. 21, 2017). URL: <http://arxiv.org/abs/1709.07417> (visited on 04/12/2019).
- [86] Li, K. and Malik, J. “Learning to Optimize”. In: *arXiv:1606.01885 [cs, math, stat]* (June 6, 2016). URL: <http://arxiv.org/abs/1606.01885> (visited on 04/09/2019).
- [87] Cai, H. et al. “Efficient Architecture Search by Network Transformation”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. Thirty-Second AAAI Conference on Artificial Intelligence. Apr. 29, 2018. URL: <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16755> (visited on 04/12/2019).
- [88] Moody, J. “Prediction risk and architecture selection for neural networks”. In: Springer-Verlag, 1994, pp. 147–165.

- [89] Ha, D., Dai, A., and Le, Q. V. “HyperNetworks”. In: *arXiv:1609.09106 [cs]* (Sept. 27, 2016). URL: <http://arxiv.org/abs/1609.09106> (visited on 04/12/2019).
- [90] Smithson, S. C. et al. “Neural networks designing neural networks: multi-objective hyperparameter optimization”. In: Proceedings of the 35th International Conference on Computer-Aided Design. ACM, Nov. 7, 2016, p. 104. ISBN: 978-1-4503-4466-1. DOI: 10.1145/2966986.2967058. URL: <http://dl.acm.org/citation.cfm?id=2966986.2967058> (visited on 04/09/2019).
- [91] Albelwi, S. and Mahmood, A. “A Framework for Designing the Architectures of Deep Convolutional Neural Networks”. In: *Entropy* 19.6 (June 2017), p. 242. DOI: 10.3390/e19060242. URL: <https://www.mdpi.com/1099-4300/19/6/242> (visited on 04/09/2019).
- [92] Salimans, T. et al. “Evolution Strategies as a Scalable Alternative to Reinforcement Learning”. In: *arXiv:1703.03864 [cs, stat]* (Mar. 10, 2017). URL: <http://arxiv.org/abs/1703.03864> (visited on 04/12/2019).
- [93] Wierstra, D. et al. “Natural Evolution Strategies”. In: *Journal of Machine Learning Research* 15 (2014), pp. 949–980. URL: <http://jmlr.org/papers/v15/wierstra14a.html> (visited on 05/05/2019).
- [94] Maclaurin, D., Duvenaud, D., and Adams, R. P. “Gradient-based Hyperparameter Optimization through Reversible Learning”. In: *arXiv:1502.03492 [cs, stat]* (Feb. 11, 2015). arXiv: 1502.03492. URL: <http://arxiv.org/abs/1502.03492> (visited on 04/09/2019).
- [95] Gordon, A. et al. “MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks”. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018, pp. 1586–1595. URL: [http://openaccess.thecvf.com/content\\_cvpr\\_2018/html/Gordon\\_MorphNet\\_Fast\\_\\_CVPR\\_2018\\_paper.html](http://openaccess.thecvf.com/content_cvpr_2018/html/Gordon_MorphNet_Fast__CVPR_2018_paper.html) (visited on 05/03/2019).
- [96] Elsken, T., Metzen, J.-H., and Hutter, F. “Simple And Efficient Architecture Search for Convolutional Neural Networks”. In: *arXiv:1711.04528 [cs, stat]* (Nov. 13, 2017). URL: <http://arxiv.org/abs/1711.04528> (visited on 04/09/2019).
- [97] Bengio, Y. “Gradient-Based Optimization of Hyperparameters”. In: *Neural Computation* 12.8 (Aug. 2000), pp. 1889–1900. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/089976600300015187. URL: <http://www.mitpressjournals.org/doi/10.1162/089976600300015187> (visited on 04/09/2019).

- [98] Luo, R. et al. “Neural Architecture Optimization”. In: *Advances in Neural Information Processing Systems 31*. Ed. by Bengio, S. et al. Curran Associates, Inc., 2018, pp. 7816–7827. URL: <http://papers.nips.cc/paper/8007-neural-architecture-optimization.pdf> (visited on 04/12/2019).
- [99] Liu, H., Simonyan, K., and Yang, Y. “DARTS: Differentiable Architecture Search”. In: *arXiv:1806.09055 [cs, stat]* (June 23, 2018). URL: <http://arxiv.org/abs/1806.09055> (visited on 04/12/2019).
- [100] Hundt, A., Jain, V., and Hager, G. D. “sharpDARTS: Faster and More Accurate Differentiable Architecture Search”. In: *arXiv:1903.09900 [cs]* (Mar. 23, 2019). URL: <http://arxiv.org/abs/1903.09900> (visited on 05/05/2019).
- [101] Cubuk, E. D. et al. “AutoAugment: Learning Augmentation Policies from Data”. In: *arXiv:1805.09501 [cs, stat]* (May 24, 2018). URL: <http://arxiv.org/abs/1805.09501> (visited on 05/05/2019).