DHA: End-to-End Joint Optimization of Data Augmentation Policy, Hyper-parameter and Architecture

Kaichen Zhou[1] **Lanqing Hong**[2] **Shoukang Hu**[3] **Fengwei Zhou**[2] **Binxin Ru**[1] **Jiashi Feng**[4] **Zhenguo Li**[2]

[1]University of Oxford
[2]Huawei Noah's Ark Lab
[3]The Chinese University of Hong Kong
[4]National University of Singapore

Abstract Automated machine learning (AutoML) usually involves several crucial components, such as Data Augmentation (DA) policy, Hyper-Parameter Optimization (HPO), and Neural Architecture Search (NAS). However joint optimization of these components remains challenging due to the largely increased search dimension and the variant input types of each component. In parallel to this, the common practice of *searching* for the optimal architecture first and then *retraining* it before deployment in NAS often suffers from low performance correlation between the search and retraining stages. An end-to-end solution that integrates the AutoML components and returns a ready-to-use model at the end of the search is desirable. In view of these, we propose **DHA**, which achieves joint optimization of **D**ata augmentation policy, **H**yper-parameter and **A**rchitecture. Specifically, end-to-end NAS is achieved in a differentiable manner by optimizing a compressed lower-dimensional feature space, while DA policy and HPO are updated dynamically at the same time.

1 Introduction

While deep learning has achieved remarkable progress in various tasks such as computer vision and natural language processing, the design and training of a well-performing deep neural architecture for a specific task usually requires tremendous human involvement He et al. (2016); Sandler et al. (2018). To alleviate such burden on human users, AutoML algorithms have been proposed in recent years to automate the pipeline of designing and training a model, such as automated Data Augmentation (DA), Hyper-Parameter Optimization (HPO), and Neural Architecture Search (NAS) Cubuk et al. (2018); Mittal et al. (2020); Chen et al. (2019).) All of these AutoML components are normally processed independently and the naive solution of applying them sequentially in separate stages, not only suffers from low efficiency but also leads to sub-optimal results Dai et al. (2020); Dong et al. (2020). Indeed, how to achieve full-pipeline "from data to model" automation efficiently and effectively is still a challenging and open problem.

One of the main difficulties lies in understanding how to automatically combine the different AutoML components (e.g., NAS and HPO) appropriately without human expertise. Another main challenge of achieving the automated pipeline "from data to model" is understanding how to perform end-to-end searching and training of models without the need of parameter retraining. Current approaches, even those considering only one AutoML component such as NAS algorithms, usually require two stages, one for searching and one for retraining Liu et al. (2019); Xie et al. (2019).

Targeting the challenging task-specific end-to-end AutoML, we propose DHA, a differentiable joint optimization solution for efficient end-to-end AutoML components, including the DA, HPO and NAS. In DHA, the optimization strategy **weight-sharing** Xie et al. (2020) is delicately adopted in DA, HPO and NAS by respectively introducing the probability matrix, the continuous hyper-parameter setting and the super-network. Specifically, the DA and HPO are regarded as dynamic schedulers,

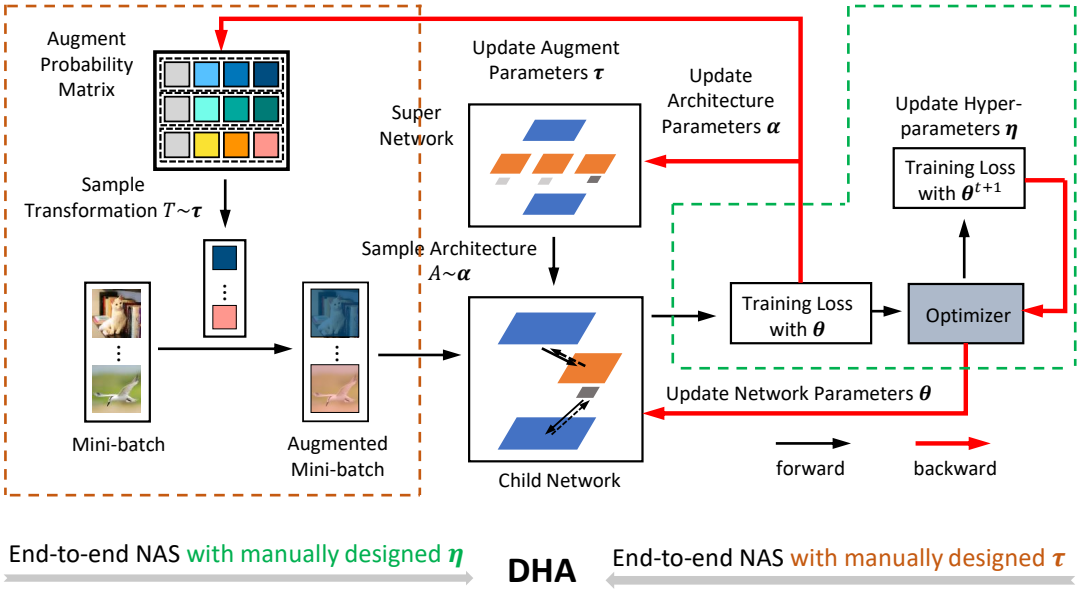

Figure 1: An overview of DHA. We first sample the DA operations for each sample based on the data transformation parameters τ. Then, a child network is sampled based on the architecture parameters α, which will be used to process the transformed mini-batch. Training loss is calculated to update τ, α, and θ. Finaly, the training loss based on updated networks' weights θ^{t+1} is used to update hyper-parameters η.

which adapt themselves to the update of network parameters and network architecture. At the same time, the end-to-end NAS optimization is realized in a differentiable manner with the help of sparse coding method. Instead of performing our search in a high-dimensional network architecture space, we optimize a compressed lower-dimensional feature space. With this differentiable manner, DHA can effectively deal with the huge search space and the high optimization complexity caused by the joint optimization problem.

2 Methodology

Consider a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, where N is the size of this dataset, and y_i is the label of the input sample x_i. We aim to train a neural network $f(\cdot)$, which can achieve the best accuracy on the test dataset $\mathcal{D}^{test}$. Multiple AutoML components are considered, including DA, HPO, and NAS. Let τ, η, α, and θ represent the data augmentation parameters, the hyper-parameters, the architecture parameters, and the objective neural network parameters, respectively. This problem can be formulated as

$$\begin{aligned} &argmin_{\tau,\eta,\alpha,\theta}\mathcal{L}(\tau,\eta,\alpha,\theta;\mathcal{D}) \\ s.t. \quad &c_i(\alpha) \leq C_i, i = 1, ..., \gamma, \end{aligned} \tag{1}$$

where $\mathcal{L}(\cdot)$ represents the loss function, $\mathcal{D}$ denotes the input data, $c_i(\cdot)$ refers to the resource cost (e.g., storage or computation cost) of the current architecture α, which is restricted by the i-th resource constraints C_i, and γ denotes the total number of resource constraints. Considering the huge search space, it is challenging to achieve the joint optimization of τ, η, α, and θ within one-stage without parameter retraining. In this work, we propose to use the differentiable method to provide a computationally efficient solution. See Fig. 1 for an illustration.

2.1 Data augmentation parameters

For every mini-batch of training data $\mathcal{B}^{tr} = \{(x_k, y_k)\}_{k=1}^{n^{tr}}$, we conduct data augmentation to increase the diversity of the training data. For t-th iteration, we sample n^{tr} transformations

according to data augmentation parameter $\boldsymbol{\tau}^t$ to generate the corresponding augmented samples in the batch. Given a sampled architecture, the loss function for each augmented sample is denoted by $\mathcal{L}^{tr}(f(\boldsymbol{\alpha}^t, \boldsymbol{\theta}^t; \mathcal{T}_k(x_k)))$, where $\mathcal{T}_k$ represents the selected transformation. In order to relax $\boldsymbol{\tau}$ to be differentiable, we regard $p_k(\boldsymbol{\tau}^t)$, the probability as an importance weight for the loss function. The objective of data augmentation is to minimize the following loss function:

$$\mathcal{L}^{DA}(\boldsymbol{\tau}^t) = -\sum_{k=1}^{n^{tr}} p_k(\boldsymbol{\tau}^t)\mathcal{L}^{tr}(f(\boldsymbol{\alpha}^t, \boldsymbol{\theta}^t; \mathcal{T}_k(x_k))). \tag{2}$$

In this way, DHA aims to increase the probability of those transformations with high **training loss**.

2.2 Hyper-parameters

At the training stage, we alternatively update $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$. In t-th iteration, we can update $\boldsymbol{\theta}^t$ based on the gradient of the unweighted training loss $\mathcal{L}^{tr}(f(\boldsymbol{\alpha}^t, \boldsymbol{\theta}^t; \mathcal{B}^{tr})) = \frac{1}{n^{tr}}\sum_{k=1}^{n^{tr}} \mathcal{L}^{tr}(f(\boldsymbol{\alpha}^t, \boldsymbol{\theta}^t; \mathcal{T}_k(x_k)))$, which can be written as $\boldsymbol{\theta}^{t+1} = OP(\boldsymbol{\theta}^t, \boldsymbol{\eta}^t, \nabla_{\boldsymbol{\theta}}\mathcal{L}^{tr}(f(\boldsymbol{\alpha}^t, \boldsymbol{\theta}^t; \mathcal{B}^{tr})))$, where $OP(\cdot)$ is the optimizer. To update the hyper-parameters $\boldsymbol{\eta}$, we regard $\boldsymbol{\theta}^{t+1}$ as a function of $\boldsymbol{\eta}$ and compute the **training loss** $\mathcal{L}^{tr}(f(\boldsymbol{\alpha}^t, \boldsymbol{\theta}^{t+1}(\boldsymbol{\eta}^t); \mathcal{B}^{tr}))$ with network parameters $\boldsymbol{\theta}^{t+1}(\boldsymbol{\eta}^t)$ on a mini-batch of training data $\mathcal{B}^{tr}$. Then, $\boldsymbol{\eta}^t$ is updated with $\nabla_{\boldsymbol{\eta}}\mathcal{L}^{tr}(f(\boldsymbol{\alpha}^t, \boldsymbol{\theta}^{t+1}(\boldsymbol{\eta}^t); \mathcal{B}^{tr}))$ by:

$$\boldsymbol{\eta}^{t+1} = \boldsymbol{\eta}^t - \beta\nabla_{\boldsymbol{\eta}}\mathcal{L}^{tr}(f(\boldsymbol{\alpha}^t, \boldsymbol{\theta}^{t+1}(\boldsymbol{\eta}^t); \mathcal{B}^{tr})), \tag{3}$$

where β is a learning rate. Even $\boldsymbol{\theta}^t$ can also be deployed to $\boldsymbol{\theta}^{t-1}$ whose calculation also involves $\boldsymbol{\eta}^{t-1}$, we take an approximation method and regard $\boldsymbol{\theta}^t$ here as a variable independent of $\boldsymbol{\eta}^{t-1}$.

2.3 Architecture parameters

Following Liu et al. (2019), we denote the each space as a single directed acyclic graph (DAG), where the probability matrix $\boldsymbol{\alpha}$ consists of vector $\boldsymbol{\alpha}_{i,j}^T = [\alpha_{i,j}^1, ..., \alpha_{i,j}^r, ..., \alpha_{i,j}^k]$ and $\alpha_{i,j}^r$ represents the probability of choosing r^{th} operation associated with the edge (i, j). Instead of directly optimizing $\boldsymbol{\alpha} \in \mathbb{R}^n$, we adopt ISTA-NAS to optimize its compressed representation $b \in \mathbb{R}^m$ where $m << n$, which can be written as $b = A\boldsymbol{\alpha} + \epsilon$, where $\epsilon \in \mathbb{R}^m$ represents the noise and $A \in \mathbb{R}^{m\times n}$ is the measurement matrix which is randomly initialized. $\boldsymbol{\alpha}$ is optimized by using iterative shrinkage thresholding algorithm Daubechies et al. (2004), which can be written as:

$$\boldsymbol{\alpha}^{t+1} = \eta_{\lambda/L}(\boldsymbol{\alpha}^t - \frac{1}{L}A^T(A\boldsymbol{\alpha} - b)), t = 0, 1, ..., \tag{4}$$

where L represents the LASSO formulation which can be written as $\min_{\boldsymbol{\alpha}} \frac{1}{2}||A\boldsymbol{\alpha} - b||_2^2 + \lambda||\boldsymbol{\alpha}||_1$; the λ represents the regularization parameters and the $\eta_{\lambda/L}$ is the shrinkage operator as defined in Beck and Teboulle (2009). Thus we have $\boldsymbol{\alpha}_j^T\boldsymbol{o}_j = (b_j^T A_j - [\boldsymbol{\alpha}_j(b_j)]^T E_j)o_j$, where o_j refer to all possible operations connected to note j and $E_j = A_j^T A_j - I$. With this relaxation, b can be optimized through calculating the gradient concerning **training loss**.

2.4 Joint-optimization

Based on the above analysis of each AutoML module, DHA realizes end-to-end joint optimization of automated data augmentation parameters $\boldsymbol{\tau}$, hyper-parameters $\boldsymbol{\eta}$, and architecture parameters $\boldsymbol{\alpha}$. The main reason that DHA could optimize large-scale search space in an effective manner, is that DHA delicately adopt **weight-sharing** in the joint-optimization for different parameters. Instead of only optimizing a sub-network with a DA strategy and a hyper-parameter setting to check the performance of certain setting, we realize the joint-optimization with the help of a super-net network, a DA probability matrix and continuous hyper-parameter setting. In that way, DHA can make use of previous trained parameter weights to check the performance setting, which largely decreases the computational request.

3 Experiments

3.1 Experiment setting

Datasets. Following Ru et al. (2020), we conducted experiments on various datasets, including CIFAR10, CIFAR100, SPORT8, MIT67 and IMAGNET for the object classification task Krizhevsky et al. (2009); Li and Fei-Fei (2007); Quattoni and Torralba (2009); Nilsback and Zisserman (2008).
Search space. **(1) Automated DA**. Following Ho et al. (2019), we consider 14 different operations for data augmentation, such as AutoContrast and Equalize. The magnitude of each operation is randomly sampled from the uniform distribution. **(2) NAS**. Following Liu et al. (2019) and Cai et al. (2019), we consider both the cell-based and the MobileNet search space, which regards the whole architecture as a stack similar cells. **(3) HPO**. We consider both the L2 regularization (i.e., weight decay) and the learning rate in the experiments.
Baselines. We compare DHA with various AutoML algorithms (see Table 1). To further demonstrate the benefits of joint optimization of multiple AutoML components, we also include a baseline, **Sequential DHA**, which resembles the common practice by human to optimize different components in sequence. Specifically, Sequential DHA consists of two stages. During the first stage, Sequential DHA performs NAS to find the optimal architecture under certain hyper-parameter settings. In the next stage, Sequential DHA performs the online DA and HPO strategy proposed in our paper and trains the architecture derived from the first stage from scratch.

Table 1: Top-1 accuracy (%) and computational time (GPU hour) of different AutoML algorithms on CIFAR10, CIFAR100, SPORT8, MIT67 and IMAGNET with Cell-Based Search Space.

Model	CIFAR10 Acc	CIFAR100 Acc	SPORT8 Acc	MIT67 Acc	IMAGNET Acc
ENAS Pham et al. (2018)	95.85±0.17	78.02±0.55	94.54±0.35	71.05±0.29	-
NSGA-NET Lu et al. (2019)	96.18±0.37	77.31±0.14	92.53±0.34	70.20±0.41	-
DARTS Liu et al. (2019)	97.24±0.10	82.37±0.34	93.87±0.35	70.73±0.24	73.30
P-DARTS Chen et al. (2019)	97.13±0.07	82.46±0.37	92.45±0.66	70.70±0.29	75.30
MANAS Carlucci et al. (2019)	97.18±0.07	82.07±0.14	94.46±0.22	71.36±0.19	73.85
One-Stage ISTA Yang et al. (2020)	97.64±0.20	83.10±0.11	94.33±0.12	72.12±0.03	76.00
Sequential DHA	97.77±0.14	83.51±0.12	94.55±0.09	72.34±0.14	76.70
DHA	**98.11**±0.26	**83.93**±0.23	**95.06**±0.13	**73.35**±0.19	**77.40**

3.2 Results

As shown in Table 1, methods optimizing all of DA, HPO and NAS automatically (i.e, Sequential DHA and DHA) consistently outperform those NAS algorithms with manual designed DA and HPO. Specifically, DHA achieves SOTA results on all datasets. This shows the clear performance gain of extending the search scope from architecture to including also data augmentation and hyper-parameters, justifying the need for multi-component optimization in AutoML. Moreover, despite optimising over a larger search space, DHA remains cost efficient. For example, on CIFAR100, DHA enjoys 1.56% higher test accuracy than DARTS but requires 42% less time. Besides, the comparison between DHA and Sequential DHA reveals the evident advantage of doing DA, HPO and NAS jointly over doing them separately in different stages.

4 Conclusion

In this work, we present DHA, an end-to-end joint-optimization method for three important components of AutoML, including DA, HPO and NAS. This differentiable joint-optimization method can efficiently optimize larger search space than previous AutoML methods and achieve SOTA results on various datasets with a relatively low computation cost.

5 Limitations and Broader Impact Statement

Conventional neural architecture search methods perform a search over a fixed set of architecture candidates and then apply or search for a separate set of hyper-parameters when retraining the best architecture derived from the architecture search phase. Such search protocol may lead to sub-optimal results Zela et al. (2018); Dong et al. (2020) as it neglects the influence of training hyper-parameters on architecture performance and ignores superior architectures under alternative hyper-parameter values Dai et al. (2020). Given this, several works have been proposed to jointly optimize architecture structure and training hyper-parameters Dai et al. (2020); Wang et al. (2020); Dong et al. (2020).

However, previous AutoML algorithms usually focus on one or two components and ignores the coupling relationship among different components. DHA firstly presents an end-to-end joint-optimization method for three important components of AutoML including Data Augmentation, Neural Architecture Search, and Hyperparameter Optimization. This differentiable joint-optimization method can efficiently optimize larger search space with the help of the weight sharing strategy than previous joint-optimization AutoML methods and achieve a better result with a relatively lower computation cost. Moreover, experiments in this paper have also proved that the performance of simultaneous joint-optimization AutoML method outperforms pipeline-optimization AutoML method. Our method mainly concentrates on optimizing the differentiable hyper-parameter like weight decay and momentum. According to Li et al. (2020), learning rate and weight decay are the two most important hyper-parameters in model training. Experimental results also show the effectiveness of our search space.

6 Appendix

6.1 Experimental Settings

6.1.1 Search Space. **(1) Automated DA**. Following Ho et al. (2019), we consider 14 different operations for data augmentation, including: ● AutoContrast ● Equalize ● Rotate ● Posterize ● Solarize ● Color ● Contrast ● Brightness ● Sharpness ● Shear X ● Shear Y ● Translate X ● Translate Y ● Identity. The magnitude ranging from 0 to 10 of each operation is randomly sampled from a uniform distribution. At each time, two operations would be sampled according to τ and would be successively applied to each sample.
(2) NAS. Following Liu et al. (2019), we consider the cell-based search space, which regards the whole architecture as a stack of similar cells. Each cell consists of a fixed number of nodes and our model tries to find the best operation combination between different nodes. One difference worth mentioning is that in contrast to DARTS Liu et al. (2019) which has 8 different operations between two nodes, we adopt the setting in Yang et al. (2020) which only considers 7 different operation options between two nodes including: ● 3 × 3 Separable Convolutions ● 5 × 5 Separable Convolutions ● 3 × 3 Dilated Separable Convolutions ● 5 × 5 Dilated Separable Convolutions ● 3 × 3 Max Pooling ● 3 × 3 Average Pooling ● Identity. To check the generalization of DHA, we also have tested our model with MobilenetV2 search space Tan and Le (2019). MobilenetV2 search space consists of Mobilenet blocks with kernel size $\{3, 5, 7\}$, expansion ratio $\{3, 6\}$ and identity operation.
(3) HPO. In our model, we consider both the L2 regularization(i.e., weight decay) and the learning rate in the experiments involving the HPO.

6.1.2 Setting. Experiments are run on 8 NVIDIA V100s under PyTorch-1.3.0 and python3.6. We adopt the hyper-parameter setting in Liu et al. (2019). As to the two-stage NAS, for experiments with CIFAR10, CIFAR100, SPORT8, MIT67, and FLOWERS102 during the search phase, the initial channel number and the cell number of the architecture are respectively 16 and 8. During the retraining/tuning phase, the final channel number and the cell number of the architecture are respectively 36 and 20. For experiments with ImageNet, the initial channel number of the architecture is 48 and the cell number of the architecture is 4. During the final retraining/tuning phase, the final channel number and the cell number of the architecture are respectively 48 and 14. As to the one-stage NAS, for experiments with CIFAR10, CIFAR100, SPORT8, MIT67, and FLOWERS102, the channel number and the cell number of the architecture are respectively 36 and 20. For experiments with ImageNet, the channel number and the cell number of the architecture are respectively 48 and 14.

Moreover, the Adam optimizer is used in the DHA optimization process. Learning rate and weight decay are randomly initialized. The main reason for setting the lower bound and upper bound is to prevent negative values and large values. We have set the number of epochs as our stopping criterion. Variances shown in experiment results are mainly caused by setting different random seeds, which are used to show the stability of our model.

6.2 Details of Sequential DHA

Sequential DHA consists of two stages. During the first stage, Sequential DHA performs NAS to find the optimal architecture under certain hyper-parameter settings. In the next stage, Sequential DHA performs the online DA and HPO strategy proposed in our paper and trains the architecture derived from the first stage from scratch.

References

Beck, A. and Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202.

Cai, H., Zhu, L., and Han, S. (2019). Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*.

Carlucci, F. M., Esperança, P. M., Singh, M., Gabillon, V., Yang, A., Xu, H., Chen, Z., and Wang, J. (2019). Manas: Multi-agent neural architecture search. *arXiv preprint arXiv:1909.01051*.

Chen, X., Xie, L., Wu, J., and Tian, Q. (2019). Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *CVPR*.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. (2018). Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*.

Dai, X., Wan, A., Zhang, P., Wu, B., He, Z., Wei, Z., Chen, K., Tian, Y., Yu, M., Vajda, P., et al. (2020). FBNetV3: Joint architecture-recipe search using neural acquisition function. In *CVPR*.

Daubechies, I., Defrise, M., and De Mol, C. (2004). An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 57(11):1413–1457.

Dong, X., Tan, M., Yu, A. W., Peng, D., Gabrys, B., and Le, Q. V. (2020). AutoHAS: Differentiable hyper-parameter and architecture search. *arXiv preprint arXiv:2006.03656*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*.

Ho, D., Liang, E., Chen, X., Stoica, I., and Abbeel, P. (2019). Population based augmentation: Efficient learning of augmentation policy schedules. In *ICML*.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. *Tech Report*.

Li, H., Chaudhari, P., Yang, H., Lam, M., Ravichandran, A., Bhotika, R., and Soatto, S. (2020). Rethinking the hyperparameters for fine-tuning. *arXiv preprint arXiv:2002.11770*.

Li, L.-J. and Fei-Fei, L. (2007). What, where and who? classifying events by scene and object recognition. In *ICCV*.

Liu, H., Simonyan, K., and Yang, Y. (2019). DARTS: Differentiable architecture search. In *ICLR*.

Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., and Banzhaf, W. (2019). Nsga-net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 419–427.

Mittal, G., Liu, C., Karianakis, N., Fragoso, V., Chen, M., and Fu, Y. (2020). HyperSTAR: Task-aware hyperparameters for deep networks. In *CVPR*.

Nilsback, M.-E. and Zisserman, A. (2008). Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics & Image Processing*.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. (2018). Efficient neural architecture search via parameters sharing. In *ICML*.

Quattoni, A. and Torralba, A. (2009). Recognizing indoor scenes. In *CVPR*.

Ru, B., Esperanca, P., and Carlucci, F. (2020). Neural architecture generator optimization. In *NeurIPS*.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). MobilenetV2: Inverted residuals and linear bottlenecks. In *CVPR*.

Tan, M. and Le, Q. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. In *ICML*.

Wang, T., Wang, K., Cai, H., Lin, J., Liu, Z., Wang, H., Lin, Y., and Han, S. (2020). APQ: Joint search for network architecture, pruning and quantization policy. In *CVPR*.

Xie, L., Chen, X., Bi, K., Wei, L., Xu, Y., Chen, Z., Wang, L., Xiao, A., Chang, J., Zhang, X., and Tian, Q. (2020). Weight-sharing neural architecture search: A battle to shrink the optimization gap. *CoRR*, abs/2008.01475.

Xie, S., Zheng, H., Liu, C., and Lin, L. (2019). SNAS: Stochastic neural architecture search. In *ICLR*.

Yang, Y., Li, H., You, S., Wang, F., Qian, C., and Lin, Z. (2020). Ista-nas: Efficient and consistent neural architecture search by sparse coding. *arXiv preprint arXiv:2010.06176*.

Zela, A., Klein, A., Falkner, S., and Hutter, F. (2018). Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *arXiv preprint arXiv:1807.06906*.