DEHB-WS: Joint Architecture and Hyperparameter Search with Weight Sharing

Zahra Padar[1] **Sharat Patil**[1] **Sai Prasanna**[1]

[1]Albert Ludwigs University of Freiburg

Abstract Joint Neural architecture search (NAS) and hyperparameter optimization (HPO) is shown to be essential for obtaining strong performance in deep neural networks [2]. Additionally, using a shared supernetwork (supernet) while doing NAS can speed up the search process [5]. In this work, we introduce a new approach to joint optimization: Differential Evolution Hyperband (DEHB) with Weight sharing (WS) or DEHB-WS. We conduct multi-fidelity search over both architecture and hyperparameter space using the DEHB algorithm while sampling architectural blocks from a shared supernet. Using a supernet we can find configurations that could be trained from scratch faster than vanilla DEHB with similar performance. Furthermore, fine-tuning the incumbent architecture with the weights inherited from the supernet with the incumbent hyperparameters leads to better performance within fewer epochs compared to training the final configuration from scratch. Our code and results are available here .

1 Introduction

Hyperparameter optimization (HPO) has proven to be successful in obtaining top performance in neural networks. Current state-of-the-art hyperparameter search approaches such as SMAC [11], BOHB[8], and DEHB [1] do multi-fidelity optimization. They use the performance of models trained in lower fidelities, such as epochs, as a proxy for the performance of a hyperparameter configuration at a higher fidelity. Since performance at lower and higher fidelities are generally correlated, early stopping poor configurations at lower fidelity allows room to search for more configurations at a given computational budget.

Neural architecture search (NAS) [7] comes with the promise of automating the choice of neural network architecture. It has been demonstrated that tuning hyperparameters as a post-processing step after architecture search is sub-optimal because of the interaction between hyperparameters and the architectural choice [13]. This motivates recent works [9], [4], [5], [14] on jointly optimizing hyperparameter and architecture search.

Joint search becomes expensive as the search space grows. To mitigate this problem, the AutoHAS paper [5] samples architectures from a single supernet allowing them to avoid training new configurations from scratch. They use a reinforcement-learning-based controller to sample architecture and hyperparameters. Our approach follows AutoHAS in using a single shared supernet during our joint search. We use vanilla Differential Evolution Hyperband (DEHB) [1] algorithm for the joint search. This evolution-based algorithm has obtained state-of-the-art results for architecture search [1]. Our approach combines the advantages of multi-fidelity optimization, weight sharing, and evolution for effective joint search.

We evaluate our approach by training a randomly initialized final incumbant architecture with the incumbant hyperparameter configuration found by our method. And we evaluate the model obtained by fine-tuning the final configuration with weights inherited from the final supernet.

2 Our Approach: DEHB-WS

2.1 DEHB

DEHB is a multi-fidelity hyperparameter optimization algorithm that uses differential evolution (DE) in conjunction with Hyperband (HB) [10]. DEHB runs multiple iterations of HB. Each iteration consists of multiple Successive Halving (SH) brackets. Each SH bracket has rungs corresponding to its budget levels. The budget describes the fidelity of the training run.

DEHB takes one vanilla HB iteration to initialize the population required to perform DE. In the first iteration, the good performing configurations are simply promoted to the higher budgets, i.e, the next rung. In subsequent HB iterations, evolution is used to get configurations for the next rung in a given bracket.

2.2 Supernet

Our supernet is based on the Once For All architecture [3]. Using the supernet allows us to sample a subnet for any configuration within its search space. Our Convolutional Neural Network (CNN) supernet weights are shared based on the kernel size, the number of filters, and the depth of the model.

2.3 DEHB with Weight Sharing (DEHB-WS)

Due to the use of a supernet the effective training that a configuration undergoes is more than the budget for which it is evaluated. This makes it difficult to use BOHB or SMAC4MF without modification, as they model the performance of training a configuration from scratch for a given budget. However, DEHB does not explicitly model the performance at a particular budget. It instead uses a parent pool which acts as an indicator of a good search space to explore. This motivates us to use it for the joint search with weight sharing.

We pre-train the supernet for ten epochs. This warmstarts the initial population in the first bracket, as all the configurations are sampled from the supernet.

For each of the configurations to be evaluated, the weights are sampled from the supernet and trained independently, thereby allowing us to do a fair comparison in a rung. We update the supernet only at the end of each rung with the weights of the configurations that got selected into the parent pool. This prevents worse-performing configurations from affecting the supernet. The weights of the selected configurations are averaged and copied into the supernet.

The Hyperband (HB) component is modified to use cumulative training budgets, i.e, subtracting the previous rung's budget from the current budget. This can be implemented where there is a direct promotion by saving the model and continuing its training. However in DEHB, after the initial HB iteration, evolution takes place at each rung. Therefore, continued training is not possible after the first HB iteration.

In DEHB-WS we leverage the supernet to use cumulative training budgets even after the initialization HB iteration. A mutated configuration will be in a similar part of the configuration space as its parents. Since the parents contributed to the supernet's weights, we can train the mutated configuration using inherited weights for fewer epochs while still achieving good estimates of the score. Doing this reduces the number of epochs each evaluation takes without undermining performance.

3 Experiments

3.1 Search Space and Dataset

We use the joint search space (Table: 1) defined by the JAHS-Bench-201 [2] paper. The JAHS-Bench-201 is a collection of surrogate benchmarks for Joint Architecture and Hyperparameter Search (JAHS) over three datasets.

JAHS-Bench-201 follows NAS-bench-201 [6] in using a cell search space with each cell comprising six connections and each connection, in turn, having five choices for operations. The cell configuration is then used to form a network by repeatedly stacking cells and one residual network block (B.5).

Space	Property	Description
Architecture	Cell Space	Nas-Bench-201
Hyperparameter	Activation	ReLU/Hardswish/Mish
	Learning Rate	[10^-3,10^0]
	Weight Decay	[10^-5,10^-2]
	Trivial Augment	On/Off
Fidelity	Epochs	[0,200]

Table 1: Joint Search Space of JAHS-Bench-201

The JAHS-Bench-201 search space consists of four fidelity parameters: the depth and width of the neural network, the resolution of input images, and the number of epochs. Although our Supernet can support the entire search space we only use epochs as the fidelity for optimization, setting the rest to their maximum setting. This choice makes the search space tractable for our computational budget.

Dataset: We use an upscaled version of the Fashion-MNIST [12] dataset used by JAHS-Bench-201. The task is image classification with ten labels. The images are upscaled to 32x32 pixels.

3.2 Baselines and training pipeline

We train one baseline with fixed CNN architecture and hyperparameters (B.6). For the joint-search, we use DEHB[1] and SMAC4MF[11] as baselines. In the DEHB and the SMAC4MF baselines, we query the JAHS-bench-201 benchmark for obtaining validation performance instead of actual training. To enable a fair comparison of baselines, we replicate the exact training and evaluation pipeline of the benchmark comprising of learning rate scheduler, optimizer, and evaluation dataset split. To account for the variance in our results due to random factors such as random initialization, we run three seeds and report the mean and standard deviation for all the metrics.

We cannot fairly compare the wall-clock runtime of our approach and the baselines due to the difference in the hardware used by us and the JAHS benchmark. Instead, we use the total number of objective function evaluations. We limit this to sixty. For our DEHB-WS approach, this takes two hours and twenty minutes on average in our hardware (B.3).

4 Results

We find that the final incumbent of our DEHB-WS (Table 2) obtains the least validation regret $(1 - \text{accuracy}/100)$. It is also able to achieve a lower validation regret faster than DEHB (Figure 2).

For the final test accuracy (Table 3) the DEHB incumbent outperforms the DEHB-WS incumbent when we retrain them for 50 epochs from scratch. By inheriting weights from the supernet and training the incumbent configuration we can obtain better test accuracy (Table 3) with only fifteen fine-tuning epochs. The total epochs required for the overall search using DEHB-WS + fine-tuning is less than the epochs required to search using vanilla DEHB. Our approach results in a considerable speed-up compared to DEHB. Note that our retraining accuracies are queried from the benchmark while the fine-tuned accuracies were tested on real data.

Table 2: Final Incumbent Validation Regret

Approach	Validation Regret
DEHB	0.0762±0.0048
SMAC4MF	0.0746±0.01
DEHB-WS	**0.0675±.0086**

Table 3: Final Incumbent Test set accuracy

Approach	Test Accuracy
Fixed CNN Baseline	90.45 ± 0.47
DEHB (Retrain 50)	92.62 ± 0.80
SMAC4MF (Retrain 50)	91.01 ± 1.46
DEHB-WS (Retrain 50)	92.11 ± 0.73
DEHB-WS (Inherit+Finetune 10)	92.43 ± 1.52
DEHB-WS (Inherit+Finetune 15)	**93.37 ± 1.16**

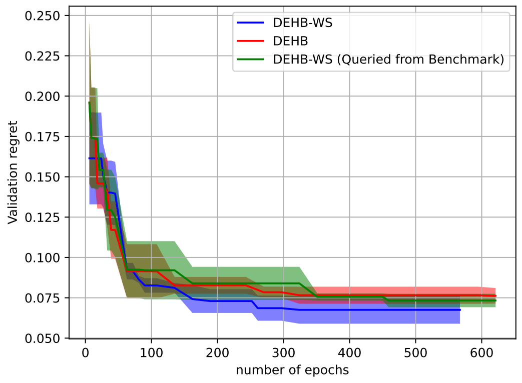

Figure 1: Validation regret of incumbent over cumulative epochs

In Figure 2, we plot the validation regret of incumbents over time for DEHB and DEHB-WS. We use the cumulative number of training epochs at a given function evaluation as a proxy for time. We report the validation regret of DEHB (red) and DEHB-WS with inheriting weights (blue). We also report the performance of DEHB-WS incumbent configurations trained from scratch by querying the benchmark (green). The query is done for the same number of training epochs as vanilla DEHB for that function evaluation. This allows us to check the performance of the incumbent configuration found by DEHB-WS without the effects of weight sharing. Around 200 epochs, we see that the DEHB-WS (inherited weights) incumbent reaches the least validation regret. In addition, after 567 epochs the final incumbent of the DEHB-WS outperforms DEHB in both cases of inheriting and training from scratch. We also see that the inherited weights of DEHB-WS incumbents are always better than the queried counterparts. This indicates that our method searches for joint configurations with inherited weights that reach good performance on finetuning.

5 Future Work

Due to the computational, and time constraints testing on more seeds, datasets, search spaces, and higher budgets would be necessary. The JAHS-bench-201 supernet does not have a lot of trainable parameters in the blocks being searched. Most of the weights are within the ResNet reduction blocks where only the activation functions are a search choice. We also see that inheriting weights without fine-tuning gives poor results. This might be due to the nature of our cell search space. Testing our approach on a search space that allows more weight sharing is therefore necessary. Since the supernet keeps getting trained in the latter iterations the budget levels are not informative. This means using the budget level to do comparisons of the configurations is not possible. Limiting the supernet to one SH bracket which might prevent this or take advantage of the supernet the latter HB iterations can be modified.

Acknowledgements. We thank the authors of the JAHS-Bench-201 for providing their datasets, and the authors of DEHB for providing additional information on the algorithm.

References

[1] Awad, N. H., Mallik, N., and Hutter, F. (2021). Dehb: Evolutionary hyberband for scalable, robust and efficient hyperparameter optimization. In *IJCAI*.

[2] Bansal, A., Stoll, D., Janowski, M., Zela, A., and Hutter, F. (2022). Jahs-bench-201: A foundation for research on joint architecture and hyperparameter search. *openreview.net*.

[3] Cai, H., Gan, C., and Han, S. (2020). Once for all: Train one network and specialize it for efficient deployment. *ArXiv*, abs/1908.09791.

[4] Dai, X., Wan, A., Zhang, P., Wu, B., He, Z., Wei, Z., Chen, K., Tian, Y., Yu, M., Vajda, P., and Gonzalez, J. E. (2021). Fbnetv3: Joint architecture-recipe search using predictor pretraining. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16271–16280.

[5] Dong, X., Tan, M., Yu, A. W., Peng, D., Gabrys, B., and Le, Q. V. (2020). Autohas: Differentiable hyper-parameter and architecture search. *CoRR*, abs/2006.03656.

[6] Dong, X. and Yang, Y. (2020). Nas-bench-201: Extending the scope of reproducible neural architecture search. *ArXiv*, abs/2001.00326.

[7] Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *ArXiv*, abs/1808.05377.

[8] Falkner, S., Klein, A., and Hutter, F. (2018). Bohb: Robust and efficient hyperparameter optimization at scale. In *ICML*.

[9] Guerrero-Viu, J., Hauns, S., Izquierdo, S., Miotto, G., Schrodi, S., Biedenkapp, A., Elsken, T., Deng, D., Lindauer, M. T., and Hutter, F. (2021). Bag of baselines for multi-objective joint neural architecture search and hyperparameter optimization. *ArXiv*, abs/2105.01015.

[10] Li, L., Jamieson, K. G., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. S. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18:185:1–185:52.

[11] Lindauer, M. T., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Sass, R., and Hutter, F. (2022). Smac3: A versatile bayesian optimization package for hyperparameter optimization. *J. Mach. Learn. Res.*, 23:54:1–54:9.

[12] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[13] Zela, A., Klein, A., Falkner, S., and Hutter, F. (2018). Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *ArXiv*, abs/1807.06906.

[14] Zimmer, L., Lindauer, M., and Hutter, F. (2020). Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl. *CoRR*, abs/2006.13799.

A Author Contributions

Zahra Padar.

- Literature survey.
- Replicated JAHS-bench-201 training and evaluation pipeline.
- Writing Experiments and Results section.
- Explored analysis with DeepCave.
- Collating results

Sharat Patil.

- Literature survey.
- Initial idea to combine DEHB and weight sharing.
- Implementing the supernet.
- Implementing DEHB-WS.
- Writing Our Approach and Future Work section.

Sai Prasanna.

- Literature survey.
- Setting up JAHS benchmark.
- Implementing baseline SMAC and DEHB with the JAHS-201 search space.
- Writing the Abstract, Introduction and Experiments section.
- Docstrings and code cleanup.

B Experimental Setup

B.1 Warmstarting

The Supernet was pretrained for 10 epochs before starting the search to allow warmstarting DEHB-WS.

B.2 Hyperband Configuration

These settings cause the each of the algorithms to run for three iterations.

Min Budget	3
Max Budget	27
eta	3
Max Function Evaluations	60

Table 4: Settings for hyperband in all three approaches

B.3 Hardware

The experiments were run in on a single NVIDIA 3090 RTX GPU.

B.4 Dataset splits

The train, validation, and test split size of Fashion-MNIST used by JAHS-Bench-201 and this work.

Split	Size
Train	44100
Validation	18900
Test	7000

B.5 Cell Search Space

The cell search space of NAS-Bench-201, JAHS-Bench-201, and DEHB-WS.

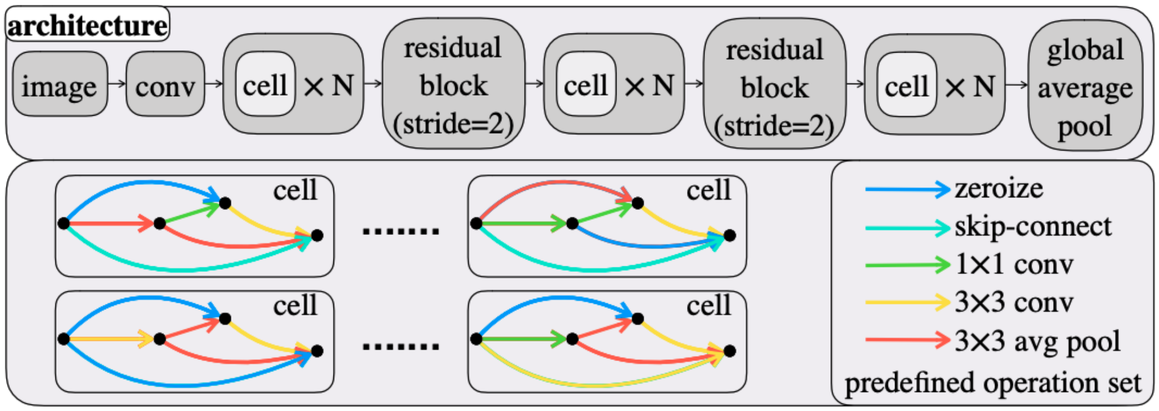

Figure 2: NAS-201 Cell Search Space

B.6 Baseline CNN Architecture

The architecture of the CNN used as baseline without NAS or HPO consists of three 2D convolutional layers with ReLU activation function, each followed by Batch Normalization and Max-pooling, as well as an Identity and a linear layer in the end.