Deep Neural Network Weight Initialization from Hyperparameter Tuning Trials

Sina Sheikholeslami[1][0000−0001−7236−4637], Tianze Wang[1][0000−0003−0422−6560], Amir H. Payberah[1][0000−0002−2748−8929], Jim Dowling[2][0000−0002−9484−6714], and Vladimir Vlassov[1][0000−0002−6779−7435]

[1] KTH Royal Institute of Technology, Stockholm, Sweden
{sinash,tianzew,payberah,vladv}@kth.se
[2] Hopsworks AB, Stockholm, Sweden
jim@hopsworks.ai

Abstract. Training of deep neural networks from scratch requires initialization of the neural network weights as a first step. Over the years, many policies and techniques for weight initialization have been proposed and widely used, including Kaiming initialization and different variants of random initialization. On the other hand, another requirement for starting the training stage is to choose and set suitable hyperparameter values, which are usually obtained by performing several hyperparameter tuning trials. In this paper, we study the suitability of weight initialization using weights obtained from different epochs of hyperparameter tuning trials and compare it to Kaiming uniform (random) weight initialization for image classification tasks. Based on an experimental evaluation using ResNet-18, ResNet-152, and InceptionV3 models, and CIFAR-10, CIFAR-100, Tiny ImageNet, and Food-101 datasets, we show that weight initialization from hyperparameter tuning trials can speed up the training of deep neural networks by up to 2x while maintaining or improving the best test accuracy of the trained models, when compared to random initialization.

Keywords: weight initialization · deep neural network training · hyperparameter tuning · model training

1 Introduction

Training deep neural networks (DNNs) requires setting values for some parameters of the training process, e.g., learning rate, dropout rate, number of hidden layers, the amount of weight decay, and various settings of the model optimization algorithm, before starting the training stage. These parameters are referred to as hyperparameters (HPs), or meta-parameters for DNN model training, and their values are typically found during a stage called hyperparameter tuning (or hyperparameter optimization), where our goal is to find the right combination for the value of different hyperparameters that maximize the prediction accuracy of the model.

The HP tuning stage is usually started by specifying a search space of hyperparameters, which includes the list of hyperparameters and the possible or allowed values each can take. Then, an HP tuning algorithm (or approach) is selected based on the downstream task and the computational and time constraints. The search process involves multiple trials, each training a model with a combination of hyperparameter values. Over the years, many hyperparameter tuning algorithms have been proposed [7], with grid search, random search [3], Bayesian optimization [26], and Asynchronous Successive Halving Algorithm (ASHA) [14] being the most popular. In the HP tuning stage, the training data is usually divided into (smaller) training and validation sets. Then, the model will be trained for a reduced number of epochs (usually a fraction of the number of epochs used for full model training) on the (smaller) training set using different combinations of hyperparameter values. The resulting performance from various combinations is then evaluated against the validation sets. At the end of this stage, the hyperparameters required for model training are chosen, and we proceed to the training stage.

Prior to the full model training stage, however, we need to perform one additional step, i.e., weight initialization [18], after deciding the values of hyperparameters. This involves setting appropriate initial values for the DNN model for effective training, e.g., avoiding issues like exploding and vanishing gradients [1]. Over the years, researchers and practitioners have proposed numerous approaches for weight initialization [18], including random initialization, Xavier (Glorot) initialization [8], and Kaiming (He) initialization [10]. Different initialization approaches use different heuristics and techniques to provide a better starting point for the model training stage, based on, e.g., information about the model, priors on the distribution of the dataset, etc. Nevertheless, to the best of our knowledge, there is a lack of weight initialization schemes that would use computations or results from the hyperparameter tuning stage, where the model from the best-performing trial has already "learned" some useful information from the dataset used for hyperparameter tuning, and can be potentially reused to provide a more appropriate starting point for the full model training.

Inspired by the idea of reusing computations or results of different stages of creating a deep learning (DL) system [15], in this paper we try to understand how weight initialization using the results of the hyperparameter tuning stage compares to the current best practices for weight initialization, in particular, the default weight initialization in PyTorch which is a variant of Kaiming uniform initialization for convolutional layers[3]. Our driving motivation was to see if we could somehow "reuse" some of the computations of the hyperparameter tuning stage in the training stage.

Contributions. In this paper, we:

[3] The list of available initializations in PyTorch: https://pytorch.org/docs/stable/nn.init.html

- propose a novel weight initialization approach that uses computation results (i.e., model weights) of the hyperparameter tuning stage to speed up and enhance the model training stage, and
- through an experimental evaluation consisting of 232 training runs, we show that for some combinations of models and datasets, weight initialization from hyperparameter tuning trials can outperform random initialization in terms of time-to-target-accuracy (which translates into speedup of the training stage by up to 2x), while achieving similar or even better best test accuracy.

2 Background

Weight Initialization. Neural network weight initialization is a very crucial and well-studied subject in the machine learning and deep learning literature [23, 1, 25, 18]. Weights are parameters that the neural network learns during the training process. Each weight represents the strength of connections between neurons, and the weights in a neural network determine how it will react in the output given a certain input. The primary goal of the training stage is to learn the appropriate values of all the weights from the data so that the neural network would perform well on the training data and generalize well on unseen test data.

Weight initialization is a crucial step in the training process of neural networks. The training process of neural networks is iterative by nature and is dominated by methods of stochastic gradient descent and its variants, and most neural networks are strongly affected by the choice of initialization [9]; hence, we need a suitable initial point for the training process where weights are initialized. The choice of the initial point can affect the speed of convergence, determine whether the neural network converges to a point with low and high cost, and sometimes, whether it converges at all. Common weight initialization techniques implemented and used in different deep learning software frameworks include variants of Xavier (Glorot) and Kaiming (He) initialization. In this work, we propose a new method that uses the best weight point found during hyperparameter tuning trials as the initial point for the model training stage.

Throughout the years, several weight initialization approaches based on "pre-training" have been proposed [13, 2, 12, 24, 21]. Our approach differs from this body of work, particularly in that they all include various computations and calculations (e.g., using Autoencoders) on the pre-trained neural network, whereas we directly use the exact weights from the winning hyperparameter tuning trial. We refer readers to [18] for a recent review of weight initialization strategies and approaches.

Hyperparameter Tuning and Optimization. Apart from model weights (parameters) that determine the model itself, most machine learning models have settings that we usually refer to as hyperparameters, e.g., the number of hidden layers in the model, choice of the optimizer, learning rate, etc. Hyperparameters specify the details of the learning process but are not part of the result of training

the model. Some hyperparameters, e.g., batch size and number of hidden layers, affect the time and memory cost of the training process. Other hyperparameters, e.g., choice of optimizer and floating point precision, affect the quality of the learned model after the training process.

One can choose between two basic approaches for selecting hyperparameters, i.e., manually or automatically. Choosing a hyperparameter manually requires domain knowledge and a deep understanding of the model, the downstream task, and the training process itself. Apart from that, choosing manually would often involve a tedious process of trial and error by trying out different potential values. Automated selection of hyperparameters, often referred to as hyperparameter tuning (or optimization), lifts the requirement of domain knowledge and requires less manual effort.

In a typical setting of HP tuning, we need to define a search space, a search strategy, and a computation budget. The search space specifies the ranges of values of the hyperparameters that we are interested in optimizing. A search strategy defines how we navigate through the search space to find the optimal set of values for the given hyperparameters. Finally, the computation budget limits the amount of time and computation used for the search. Common search strategies include grid search, random search, Bayesian optimization, Asynchronous Successive Halving (ASHA), etc. The computation budget can be specified in wall-clock time, number of epochs, number of evaluations, etc. We refer readers to [7, 27, 4] for a detailed review of hyperparameter tuning and optimization methods.

Meta-learning and Weight Initialization. A number of approaches have been proposed to learn "initializers" or "policies" that can suggest suitable initial parameters. Among these proposals, [6] introduce MetaInit, an algorithm that learns to suitably initialize the parameters of a given neural network for a given task. Our work is different from this line of research in that we do not attempt to "learn" any set of weights for the purpose of initialization; instead, we propose to "reuse" the model weights that are already learned during hyperparameter tuning trials as initial weights for the model training stage.

3 Methodology

Based on the idea of reusing results and computations of one stage of DL systems in another, we propose a novel weight initialization approach that uses weights from the top-performing or "winning" hyperparameter tuning trials to initialize the model weights for the model training stage. Generally, the performance of the HP tuning trial is measured using the same performance metric as the training task, e.g., validation (test) accuracy for classification tasks. The assumption is that for one trial in the hyperparameter tuning stage to be the winner, the weights in the model of that trial have already accomplished learning from the data to some degree compared to random initialization. Furthermore, it might

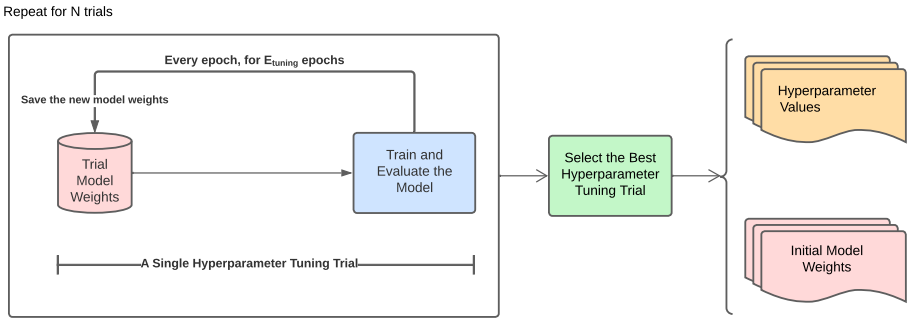

Fig. 1. The modified hyperparameter tuning experiment, where in addition to suitable hyperparameter values, we also use the weights from the winning hyperparameter tuning trial to initialize the model for the model training stage.

even be potentially a decent set of weights as it is the winner among all the other trials allowed in the hyperparameter tuning budget.

Reusing the weights of the best hyperparameter tuning trial requires a minor modification to the typical hyperparameter tuning loop to save the weights of the model at certain epochs during the trials. Figure 1 and Algorithm 1 show the needed modification. This essentially means that in addition to suitable hyperparameter values, we also save (and later reuse) the model weights from the best hyperparameter tuning trial.

Algorithm 1 Hyperparameter Tuning with Weight Saving

Require: Total number of hyperparameter tuning trials N, number of epochs per trial E

1: **for** $i = 1$ to N **do**
2: Choose a set of hyperparameter values for the trial
3: **for** $e = 1$ to E **do**
4: Perform a forward and backward pass on the model
5: Update the model weights
6: Save the model weights to storage as `weights_i_e`
7: **end for**
8: **end for**

We can see in Algorithm 1 that our only modification in a typical hyperparameter tuning trial corresponds to line 6, in which we save the model weights to storage after each epoch. Based on our specific initialization policy, we do this to choose a set of initial weights from any given epoch for the training stage. As an example, in Figure 2, we have shown the best test accuracy of models initialized with different configurations. The numbers on the X axis, i.e., 2, 5, 10, 15, 20, and 25, correspond to different values for `e` when initializing the model with `weights_i_e` before starting the training stage, where the value of `i`

corresponds to the index of the "winning" hyperparameter tuning trial (i.e., the trial with the best final validation accuracy). We should mention that for very large models, saving all the weights of different model variants might require considerable storage space and time to write to external storage. However, one can mitigate this by keeping track of the performance metric of top-T models (with T being a predefined constant number and only triggering the model save function when better models, e.g., with regards to test accuracy, are found).

4 Experimental Evaluation

To evaluate our weight initialization approach, we perform a number of experiments on different models and datasets and compare weight initialization from hyperparameter tuning to PyTorch's default initialization scheme, which uses a combination of various techniques, including Kaiming uniform [10] for convolutional and linear layers. We repeat the experiments several times as a way to control for randomness.

Our two main evaluation metrics are (i) best test accuracy, which is the maximum (top-1) accuracy of the model on the test set during training, and (ii) time-to-target-accuracy (TTA), which is measured as the number of epochs it takes for the model to surpass a relatively high test accuracy as a milestone. The former metric indicates the model performance for a configuration, while the latter suggests the effect of a configuration on training time. To summarize, we want to know if our initialization approach can result in better models (higher test accuracy) while speeding up training (lower TTA). We choose the target accuracies based on our observations from the training curves of each model/dataset pair.

4.1 Experiment Setup

Hyperparameter Search Space. For the ResNet models, we use SGD with momentum (0.9) and follow a search space inspired by Zhang et al. [28] and common practice: a set of possible learning rate values of {0.01, 0.03, 0.05, 0.1, 0.2, 0.3} and a set of possible weight decay values of {0.0003, 0.001, 0.003}. Details about the hyperparameter tuning for the InceptionV3 model can be found in 4.6. We try a subset of the different combinations using random search. We use PyTorch [19] on Ray [16, 17] for hyperparameter tuning and parallel execution of trials while assigning a different random seed to each trial. For hyperparameter tuning of experiments that use the CIFAR-10, CIFAR-100, and Food-101 datasets, the training datasets provided by PyTorch were randomly partitioned into 80/20 train/validation splits. The corresponding subsections below explain more details on hyperparameter tuning for each task.

Weight Initialization Experiments. The goal of this set of experiments is to investigate if we can use the model weights obtained during the hyperparameter tuning to initialize the model before starting the training round. We select a

number of common model/dataset combinations (e.g., ResNet-18 on CIFAR-10). For each combination, we tune a number of model-independent hyperparameters (i.e., learning rate and weight decay) for a number of trials using random search, with no early stopping. Within each hyperparameter tuning trial, we save the model weights after every epochf. We then rank the trials in descending order of final validation accuracy and specify the winning hyperparameter tuning trial. Moving on to the model training stage, for weight initialization we use weights from several epochs of the winning trial. We then train the model for a number of epochs, and report the best test accuracy of each training trial. As the baseline for comparison, we use random initialization as implemented in PyTorch, which in particular uses Kaiming uniform initialization for convolutional layers.

Randomness Control and Reproducibility. When training deep neural networks, we should deal with many sources of randomness, including non-determinisms in hardware, software frameworks, and optimization algorithms and computations [29, 20]. This stochasticity can drastically influence the performance of models and make it hard for researchers to draw strong conclusions from experimental evaluations. To alleviate this and allow for reasonable reproducibility of our results, We use a predefined set of global random seeds, repeat each set of experiments several times, and report the averages and standard deviations for each set of results[4].

4.2 EXP1: ResNet-18 on CIFAR-10

Our first set of experiments deals with tuning and training ResNet-18 on the CIFAR-10 dataset[5]. The CIFAR-10 dataset contains 60000 32×32 color images in 10 classes (6000 images per class). There are 50000 training images and 10000 test images.

Hyperparameter Tuning. For hyperparameter tuning, we used the same search space described in 4.1. We performed 10 tuning trials, with each trial consisting of 30 epochs of training with a batch size of 256. The winning trial had achieved a final validation accuracy of 88.48% with a learning rate of 0.01 and weight decay rate of 0.0003.

Training Configurations. Using the hyperparameters and weights from the winning hyperparameter tuning trial, we try 7 different weight initializations and compare them to a baseline in which we initialize the model weights using PyTorch's default scheme. These 7 different sets of weights are taken from epochs #2, #5, #10, #15, #20, #25, and the final epoch (HP Final, #30). Each configuration is trained for 200 epochs, and we repeat the training 8 times

[4] Code and raw results from our experiments can be found in https://github.com/ssheikholeslami/dnn-weight-initialization-from-hp-tuning.

[5] CIFAR-10 and CIFAR-100 datasets: https://www.cs.toronto.edu/ kriz/cifar.html

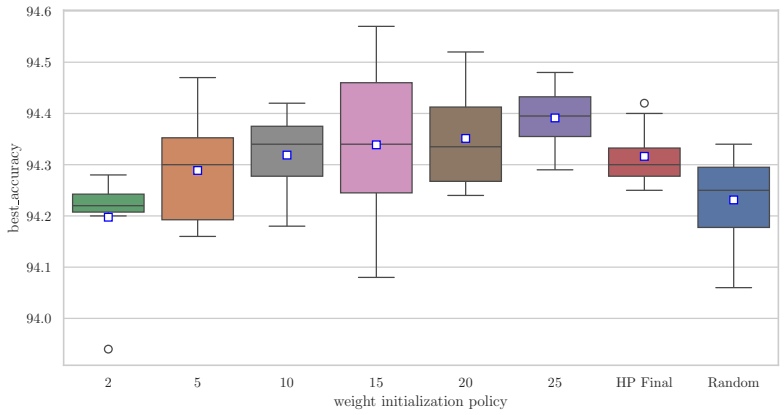

Fig. 2. Best test accuracy after 200 epochs of training ResNet-18 on CIFAR-10, using different weight initialization configurations. Numerical values are reported in Table 1.

(with 8 different global seeds). Based on the training curves of the models we set the target to 90.00%. The results of this experiment are also presented in Table 1.

We can see that all the models that use weight initialization from the hyperparameter tuning stage achieve this milestone significantly faster than the baseline approach (random initialization), and most of them achieve a higher best test accuracy on average compared to the baseline. To verify the significance of the results, we performed a paired t-test and Mann–Whitney U-test on results from "Epoch 25" and "Random" configurations, and the results were: t-statistic=3.521, p-value=0.00970; U1=3.0, p-value=0.00108, both indicating a significant difference in terms of best test accuracy. This specific configuration also shows an speedup of 2.027x in terms of TTA compared to random initialization.

4.3 EXP2: ResNet-18 on CIFAR-100

In this experiment, we tune and train ResNet-18 this time on the CIFAR-100 dataset. CIFAR-100 is similar to CIFAR-10 in terms of dimensions and total number of examples, but it has 100 classes containing 600 images each.

Hyperparameter Tuning. For hyperparameter tuning, we used the same search space described in 4.1. We performed 10 tuning trials, with each trial consisting of 40 epochs of training with a batch size of 256. The winning trial had achieved a final validation accuracy of 57.74% with a learning rate of 0.1 and weight decay rate of 0.0003.

Training Configurations. Using the hyperparameters and weights from the winning hyperparameter tuning trial, we try 6 different weight initializations

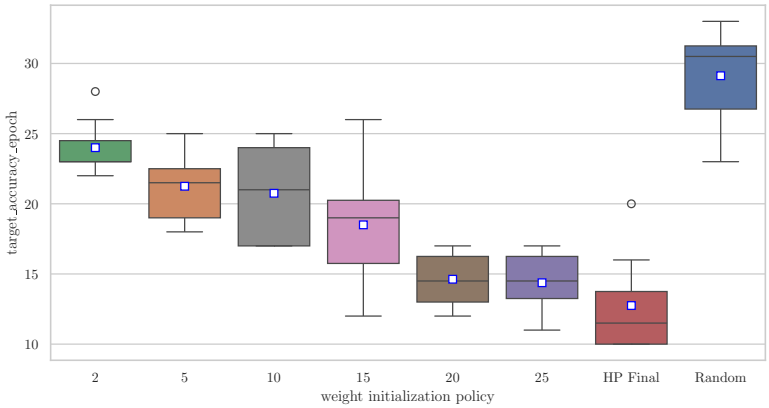

Fig. 3. First epoch to reach the target (90%) test accuracy when training ResNet-18 on CIFAR-10, using different weight initialization configurations.

and compare them to a baseline in which we initialize the model weights using PyTorch's default scheme. These 6 different sets of weights are taken from epochs #5, #10, #15, #30, #35, and the final epoch (HP Final, #40). Each configuration is trained for 200 epochs, and we repeat the training 8 times (with 8 different global seeds). Based on the training curves of the models we set the target to 75.00%. The results of this experiment are presented in Figure 4 and Table 2.

4.4 EXP3: ResNet-18 on Tiny ImageNet

In this experiment, we tune and train ResNet-18 on the Tiny ImageNet[6] dataset. Tiny ImageNet is a small-scale version of the larger ImageNet dataset, and contains 100000 downsized 64×64 color images in 200 classes as the training set, as well as 50 images for validation and 50 images for test in each class.

Hyperparameter Tuning. For hyperparameter tuning, we used the same search space described in 4.1. We performed 12 tuning trials, with each trial consisting of 40 epochs of training with a batch size of 256. The winning trial had achieved a final validation accuracy of 35.94% with a learning rate of 0.3 and a weight decay rate of 0.0003.

Training Configurations. Using the hyperparameters and weights from the winning hyperparameter tuning trial, we try 6 different weight initializations and compare them to a baseline in which we initialize the model weights using PyTorch's default scheme. These 6 different sets of weights are taken from

[6] Accessible from https://image-net.org/download-images.php

Table 1. Results from the experiments on ResNet-18 and CIFAR-10. Each trial consists of training the model for 200 epochs. The experiments for each configuration (row) are repeated with 8 different random seeds, and the average values are reported. The TTA in this table indicates the first epoch in which the model achieves at least a 90.00% test accuracy.

Weight Initialization	Best Test Accuracy	TTA (#Epoch)
Epoch 2	94.198±0.108	24.0
Epoch 5	94.289±0.107	21.25
Epoch 10	94.319±0.085	20.75
Epoch 15	94.339±0.166	18.5
Epoch 20	94.351±0.103	14.625
Epoch 25	**94.391±0.061**	14.375
HP Final Epoch	94.316±0.061	**12.75**
Random (Kaiming + Uniform)	94.231±0.098	29.125

Table 2. Results from the experiments on ResNet-18 and CIFAR-100. Each trial consists of training the model for 200 epochs. The experiments for each configuration (row) have been repeated with 8 different random seeds and the average values are reported. The TTA in this table indicates the first epoch in which the model achieves at least a 75.00% test accuracy.

Weight Initialization	Best Test Accuracy	TTA (#Epoch)
Epoch 5	77.044±0.191	122.75
Epoch 10	77.152±0.219	123.25
Epoch 15	77.112±0.138	123.125
Epoch 30	**77.292±0.181**	124.5
Epoch 35	77.124±0.287	124.25
HP Final Epoch	76.947±0.193	124.25
Random (Kaiming + Uniform)	77.080±0.299	**122.375**

epochs #5, #10, #15, #30, #35, and the final epoch (HP Final, #40). Each configuration is trained for 200 epochs, and we repeat the training 8 times (with 8 different global seeds). Based on the final accuracies and training curves, we set the target to 40.00%. The results of this experiment are presented in Figure 5 and Table 3.

4.5 EXP4: ResNet-152 on CIFAR-100

For this experiment, we change the model from ResNet-18 to ResNet-152, which is the larger variant of the ResNet family of models [11], and train it on the CIFAR-100 dataset. Tiny ImageNet is a small-scale version of the larger ImageNet dataset, and contains 100000 downsized 64×64 color images in 200 classes as the training set, as well as 50 images for validation and 50 images for test in each class.

Hyperparameter Tuning. For hyperparameter tuning, we used the same search space described in 4.1. We performed 10 tuning trials, with each trial

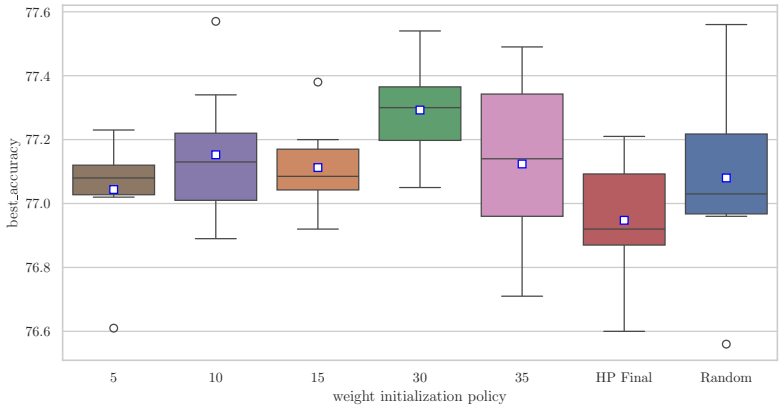

Fig. 4. Best test accuracy after 200 epochs of training ResNet-18 on CIFAR-100, using different weight initialization configurations. Numerical values are reported in Table 2.

Table 3. Results from the experiments on ResNet-18 and Tiny ImageNet. Each trial consists of training the model for 200 epochs. The experiments for each configuration (row) are repeated with 8 different random seeds and the average values are reported. The TTA in this table indicates the first epoch in which the model achieves at least a 40.00% test accuracy.

Weight Initialization	Best Test Accuracy	TTA (#Epoch)
Epoch 5	44.062±0.536	153.75
Epoch 10	44.085±0.403	**153.25**
Epoch 15	43.885±0.245	153.5
Epoch 30	44.328±0.479	153.875
Epoch 35	44.362±0.415	154.125
HP Final Epoch	44.223±0.275	154.25
Random (Kaiming + Uniform)	**44.428±0.485**	154.5

consisting of 80 epochs of training with a batch size of 128. The winning trial had achieved a final validation accuracy of 51.13% with a learning rate of 0.01 and a weight decay rate of 0.003.

Training Configurations. Using the hyperparameters and weights from the winning hyperparameter tuning trial, we try three different weight initializations and compare them to a baseline in which we initialize the model weights using PyTorch's default scheme. These three sets of weights are taken from epochs #65, #75 and the final epoch (HP Final, #80). Each configuration is trained for 200 epochs, and we repeat the training four times (with four different global seeds). Based on the final accuracies and training curves, we set the target to 75.00%. The results of this experiment are presented in Figure 6 and Table 4.

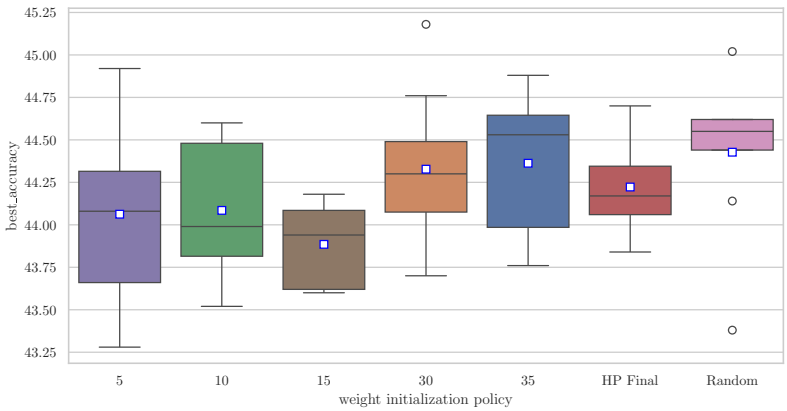

Fig. 5. Best test accuracy after 200 epochs of training ResNet-18 on Tiny ImageNet, using different weight initialization configurations. Numerical values are reported in Table 3.

Table 4. Results from the experiments on ResNet-152 and CIFAR-100. Each trial consists of training the model for 200 epochs. The experiments for each configuration (row) are repeated with 4 different random seeds and the average values are reported. The TTA in this table indicates the first epoch in which the model achieves at least a 75.00% test accuracy.

Weight Initialization	Best Test Accuracy	TTA (#Epoch)
Epoch 65	80.23±0.189	144.0
Epoch 75	80.168±0.107	143.25
HP Final Epoch	80.072±0.285	**142.5**
Random (Kaiming + Uniform)	**80.372±0.169**	143.25

4.6 EXP5: InceptionV3 on Food-101

For this experiment, we use the InceptionV3 network [22], and train it on the Food-101 dataset [5]. The Food-101 dataset contains 101000 color images in 101 classes and 750 training and 250 test images per each class.

Hyperparameter Tuning. For hyperparameter tuning, we used the following search space: learning rate values sampled from a loguniform distribution between 0.0001 and 0.1, momentum from a uniform distribution between 0.8 and 0.99, weight decay from a loguniform distribution between 0.00001 and 0.001, and the maximum number of iterations (T_max) for the Cosine Annealing learning rate scheduler from the set of possible values of 50, 100, 200. We performed 12 tuning trials, with each trial consisting of 20 epochs of training with a batch size of 64. The winning trial had achieved a final validation accuracy of 40.50%

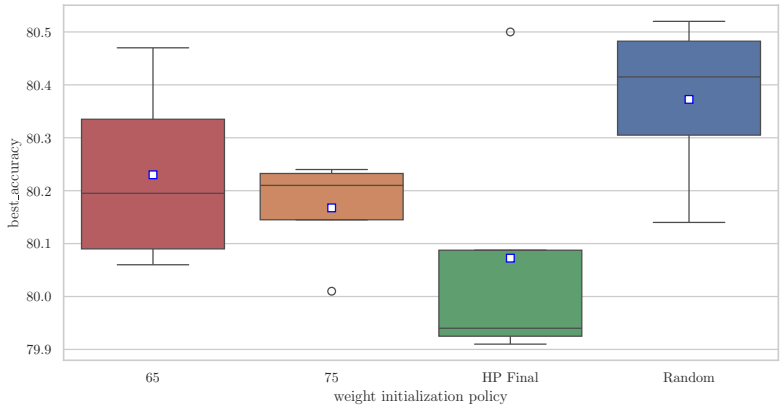

Fig. 6. Best test accuracy after 200 epochs of training ResNet-152 on CIFAR-100, using different weight initialization configurations. Numerical values are reported in Table 4.

with a learning rate of 0.021.

Training Configurations. Using the hyperparameters and weights from the winning hyperparameter tuning trial, we try four different weight initializations and compare them to a baseline in which we initialize the model weights using PyTorch's default scheme. These four sets of weights are taken from epochs #5, #10, #15, and the final epoch (HP Final, #20). Each configuration is trained for 50 epochs, and we repeat the training for each configuration eight times (with eight different global seeds). Based on the final accuracies and training curves, we set the target to 70.00%. The results of this experiment are presented in Figure 7, Figure 8, and Table 5. Based on the results, we can see that for this combination of model and dataset, weight initialization from hyperparameter tuning trials clearly outperforms random initialization in terms of both TTA and best test accuracy.

4.7 Further Discussion

Summary of Findings. Overall, our experimental evaluation shows that for some models and datasets, e.g., ResNet-18 on CIFAR-10 or CIFAR-100, and InceptionV3 on Food-101, weight initialization using hyperparameter tuning trials can outperform random initialization in terms of time-to-target-accuracy, as well as best test accuracy. It is also interesting to note that using weights from a later epoch of the hyperparameter tuning trial does not necessarily result in a better test accuracy; e.g., we can see in Figure 2 that initializing the model with weights from epoch #25 result in the best performance, and better than HP Final.

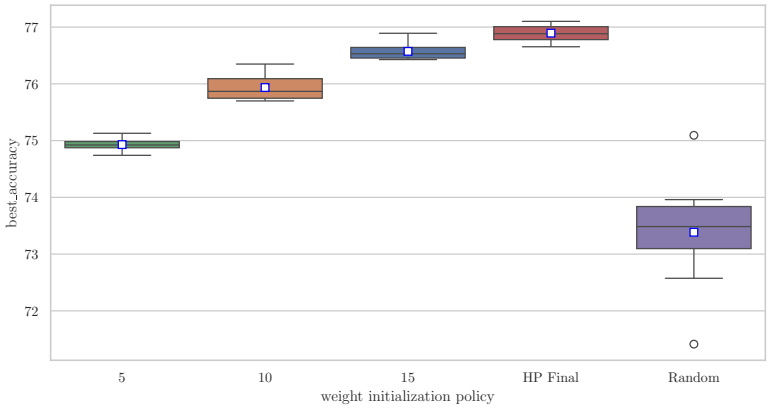

Fig. 7. Best test accuracy after 50 epochs of training InceptionV3 on Food-101, using different weight initialization configurations. Numerical values are reported in Table 5. Training with each configuration was repeated 8 times.

Table 5. Results from the experiments on InceptionV3 and Food-101. Each trial consists of training the model for 50 epochs. The experiments for each configuration (row) are repeated with 8 different random seeds and the average values are reported. The TTA in this table indicates the first epoch in which the model achieves at least a 70.00% test accuracy.

Weight Initialization	Best Test Accuracy	TTA (#Epoch)
Epoch 5	74.93±0.115	28.75
Epoch 10	75.935±0.23	23.25
Epoch 15	76.572±0.155	20.5
HP Final Epoch	**76.894±0.158**	**18.375**
Random (Kaiming + Uniform)	73.385±1.073	33.875

For ResNet-18 on Tiny ImageNet, and ResNet-152 on CIFAR-100, random initialization achieves a higher best test accuracy, but other initializations can achieve better TTAs. One possible reason for this difference in performance is that for ResNet-18 on Tiny ImageNet, and ResNet-152 on CIFAR-100, our underlying training regimes (irrespective of weight initialization policy) do not result in a test accuracy in the so-called state-of-the-art region, which is not the case for ResNet-18 on CIFAR-10 and CIFAR-100. This can be investigated using further, more thorough experiments; however, we believe our findings from these experiments are interesting enough to motivate further research on this topic.

Storage Requirements. Our modified hyperparameter tuning algorithm, in its general form, as specified in Algorithm 1, requires that we save the model weights to storage after each epoch, hence using storage space. When tuning

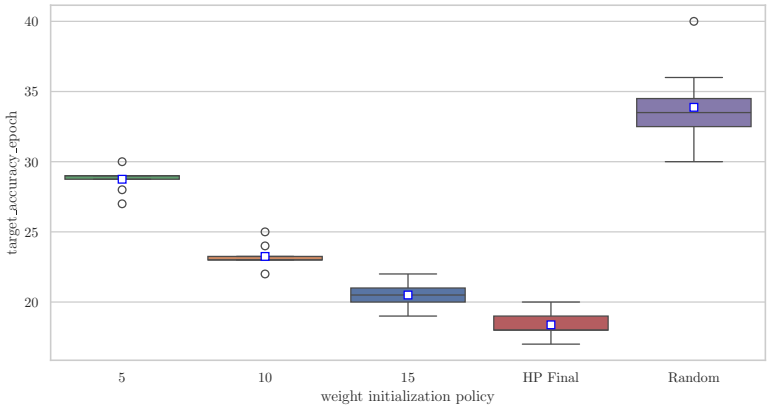

Fig. 8. First epoch to reach the target (70%) test accuracy when training InceptionV3 on Food-101, using different weight initialization configurations.

ResNet-152 on CIFAR-100, the size of each set of weights is 224 Megabytes (MB), so a complete hyperparameter tuning experiment in which we run 10 trials with 80 epochs each would require 179.2 Gigabytes (GB) of storage to save all the weights. Saving the weights after each epoch can become impractical or lead to a bottleneck when tuning considerably larger models; however, one can always modify the algorithm so that only weights from specific epochs are saved, e.g., the weights from the final epoch.

5 Conclusion

In this paper, we proposed a novel weight initialization approach that uses computation results (i.e., model weights) of the hyperparameter tuning stage to speed up and enhance the model training stage and evaluated its performance through a number of experiments on common models and datasets in the image classification domain. Our main research question was to understand how to effectively perform weight initialization using the weights from the hyperparameter tuning stage and how it compares to the current best practices for weight initialization, in particular, the default weight initialization in PyTorch. The results of our experiments using ResNet-18 and ResNet-152 models and CIFAR-10, CIFAR-100, and Tiny ImageNet datasets show that for some combinations of models and datasets, weight initialization from hyperparameter tuning trials can outperform random initialization in terms of time-to-target-accuracy, while maintaining or improving the best test accuracy of the learned model.

Our work serves as a starting point in the research direction of reusing computation results in hyperparameter tuning of DNN models to speed up the training process after hyperparameter tuning. Based on our empirical study, we envision

that further investigation of our approach can result in weight initialization approaches that, in turn, lead to possibly faster and more efficient training pipelines that also train better models. Future work includes expanding our studies in both theoretical and empirical dimensions to further understand how to effectively reuse computation results from hyperparameter tuning. To this end, performing more experiments using more types of DNN models, datasets, and downstream tasks in different domains can be an interesting starting point.

References

1. Arpit, D., Campos, V., Bengio, Y.: How to initialize your network? robust initialization for weightnorm & resnets. Advances in Neural Information Processing Systems **32** (2019)
2. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. Advances in neural information processing systems **19** (2006)
3. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of machine learning research **13**(2) (2012)
4. Bischl, B., Binder, M., Lang, M., Pielok, T., Richter, J., Coors, S., Thomas, J., Ullmann, T., Becker, M., Boulesteix, A.L., et al.: Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **13**(2), e1484 (2023)
5. Bossard, L., Guillaumin, M., Van Gool, L.: Food-101 – mining discriminative components with random forests. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) Computer Vision – ECCV 2014. pp. 446–461. Springer International Publishing, Cham (2014)
6. Dauphin, Y.N., Schoenholz, S.: Metainit: Initializing learning by learning to initialize. Advances in Neural Information Processing Systems **32** (2019)
7. Feurer, M., Hutter, F.: Hyperparameter optimization. Automated machine learning: Methods, systems, challenges pp. 3–33 (2019)
8. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. pp. 249–256. JMLR Workshop and Conference Proceedings (2010)
9. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)
10. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015)
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
12. Le Paine, T., Khorrami, P., Han, W., Huang, T.S.: An analysis of unsupervised pre-training in light of recent advances. In: 3rd International Conference on Learning Representations, ICLR 2015 (2015)
13. Li, G., Alnuweiri, H., Wu, Y., Li, H.: Acceleration of back propagation through initial weight pre-training with delta rule. In: IEEE International Conference on neural networks. pp. 580–585. IEEE (1993)
14. Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-Tzur, J., Hardt, M., Recht, B., Talwalkar, A.: A system for massively parallel hyperparameter tuning. Proceedings of Machine Learning and Systems **2**, 230–246 (2020)

15. Li, L., Sparks, E., Jamieson, K., Talwalkar, A.: Exploiting reuse in pipeline-aware hyperparameter tuning. arXiv preprint arXiv:1903.05176 (2019)
16. Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J.E., Stoica, I.: Tune: A research platform for distributed model selection and training. arXiv preprint arXiv:1807.05118 (2018)
17. Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M.I., et al.: Ray: A distributed framework for emerging {AI} applications. In: 13th USENIX symposium on operating systems design and implementation (OSDI 18). pp. 561–577 (2018)
18. Narkhede, M.V., Bartakke, P.P., Sutaone, M.S.: A review on weight initialization strategies for neural networks. Artificial intelligence review **55**(1), 291–322 (2022)
19. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32** (2019)
20. Picard, D.: Torch. manual_seed (3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision. arXiv preprint arXiv:2109.08203 (2021)
21. Ruiz-Garcia, A., Elshaw, M., Altahhan, A., Palade, V.: Stacked deep convolutional auto-encoders for emotion recognition from facial expressions. In: 2017 International Joint Conference on Neural Networks (IJCNN). pp. 1586–1593. IEEE (2017)
22. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2818–2826 (2016)
23. Thimm, G., Fiesler, E.: Neural network initialization. In: From Natural to Artificial Neural Computation: International Workshop on Artificial Neural Networks Malaga-Torremolinos, Spain, June 7–9, 1995 Proceedings 3. pp. 535–542. Springer (1995)
24. Trinh, T.H., Luong, M.T., Le, Q.V.: Selfie: Self-supervised pretraining for image embedding. arXiv preprint arXiv:1906.02940 (2019)
25. Weymaere, N., Martens, J.P.: On the initialization and optimization of multilayer perceptrons. IEEE Transactions on Neural Networks **5**(5), 738–751 (1994)
26. Wu, J., Chen, X.Y., Zhang, H., Xiong, L.D., Lei, H., Deng, S.H.: Hyperparameter optimization for machine learning models based on bayesian optimization. Journal of Electronic Science and Technology **17**(1), 26–40 (2019)
27. Yu, T., Zhu, H.: Hyper-parameter optimization: A review of algorithms and applications. arXiv preprint arXiv:2003.05689 (2020)
28. Zhang, M., Lucas, J., Ba, J., Hinton, G.E.: Lookahead optimizer: k steps forward, 1 step back. Advances in neural information processing systems **32** (2019)
29. Zhuang, D., Zhang, X., Song, S., Hooker, S.: Randomness in neural network training: Characterizing the impact of tooling. Proceedings of Machine Learning and Systems **4**, 316–336 (2022)