Autotuned Classification Based on Knowledge Transferred from Self-supervised Models

Jaydeep Kishore
Shiv Nadar University, Delhi NCR
Snehasis Mukherjee (snehasis.mukherjee@snu.edu.in)
Shiv Nadar University, Delhi NCR

Research Article

Keywords: Neural Architecture Search, Autotuning, Self supervised Learning, Hyperparameter optimization, TPE

Posted Date: April 27th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1600269/v1

Autotuned Classification Based on Knowledge Transferred from Self-supervised Models

Jaydeep Kishore
Department of Computer Science
Shiv Nadar University, Delhi NCR, India
jk452@snu.edu.in

Snehasis Mukherjee
Department of Computer Science
Shiv Nadar University, Delhi NCR, India
snehasis.mukherjee@snu.edu.in

***Abstract*—With the recent advancements of deep learning-based methods in image classification, the requirement of a huge amount of training data is inevitable to avoid overfitting problems. Moreover, supervised deep learning models require labeled datasets. Preparing such a huge amount of labeled data takes a lot of human effort and time. In this scenario, self-supervised models are becoming popular because of their ability to learn even from unlabelled datasets. This paper presents a method to automatically tune the hyperparameters of a self-supervised model for image classification. The learned features are transferred from a self-supervised model to a set of Fully Connected (FC) layer, where the hyperparameters related to the architecture are tuned. A softmax layer is used following the FC layers for classification. The hyperparameters such as the number of layers, number of units in each layer, learning rate, dropout, and optimizer are automatically tuned in these FC layers, using a Bayesian optimization technique called the Tree-structured Parzen Estimator Approach (TPE) algorithm. To evaluate the performance of the proposed method, state-of-the-art self-supervised models such as Simclr and SWAV are used to extract the learned features. Experiments are carried out on CIFAR-10, CIFAR-100, and Tiny Imagenet datasets. The proposed method outperforms the state-of-the-art.**

***Index Terms*—Neural Architecture Search, Autotuning, Self supervised Learning, Hyperparameter optimization, TPE**

I. Introduction

During the last few years, deep learning-based methods have been gaining popularity among researchers, especially for solving supervised classification problems in the image processing and computer vision domains. However, supervised deep learning models need a huge amount of annotated data for training. The annotation of data takes a lot of human effort and time. Self-supervised models are becoming popular in such circumstances where providing sufficient manual measures in the annotation process is challenging.

Self-supervised learning models have reduced human effort and time significantly through their ability to learn from unlabeled data for classification. The self-supervised models extract the learned features from a CNN and use the features as pretext tasks. Finally, the learned features from the self-supervised models can be further employed for classification, object detection, and segmentation tasks. A few areas where self-supervised models were successfully applied include image colourization [1], predicting patch position [2], jigsaw puzzle [3], image inpainting [4], predicting image rotation [5], placing frames in right order [6], and contrastive self-supervised learning [7]–[10]. Contrastive self-supervised learning models have shown state-of-the-art performance. Some of the self-supervised models based on contrastive learning include SimClr [7], MOCO [8], [9], and SWAV [10]. Fig. 1 presents the simple idea of a contrastive self-supervised model where unlabelled images are augmented with different transformations. Two transformations are chosen randomly to be fed parallelly into the model. The learned features from the two differently augmented images are compared and dragged closer to each other based on the loss function. In this process the model is trained to learn similar features of similar objects, even without the use of annotation.

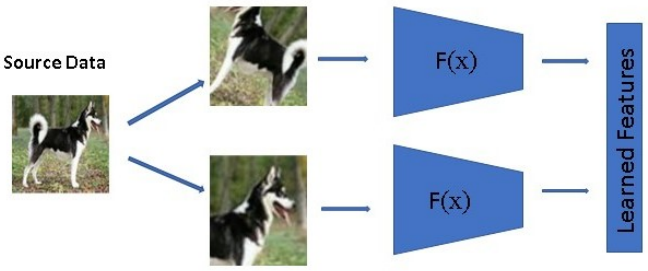

Fig. 1. A typical Contrastive self-supervised model.

Learned features from self-supervised models can be transferred to perform downstream tasks like classification, object detection, and segmentation. We design the self-supervised models for target datasets, by fine-tuning some layers (i.e., tuning the hyperparameters related to the model) for getting better results. Fine-tuning can be done in two ways: manually or automatically. Manual hyperparameter tuning is based upon heuristics and takes a lot of computational time and human effort. Whereas the methods to tune hyperparameters automatically (we call "autotuning"), require less human effort and provide better results than manual tuning. Examples of hyperparameters that can be autotuned include the number of layers, category of optimizers, number of channels, number of units, dropout rate, learning rate, etc.

Despite several efforts made for autotuning the hyperparameters related to CNNs, no attempts are made for hyperparameter tuning using a self-supervised model. We attempt to adjust some essential hyperparameters for the downstream task

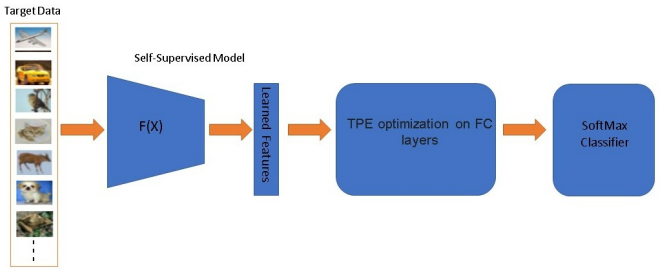

Fig. 2. The overall idea of the proposed algorithm.

(image classification) by autotuning the fully connected (FC) layers followed by a self-supervised model. Fig. 2 presents The overall idea of the work. The learned features from the self-supervised model are taken as input for the FC layers, followed by a softmax classifier. The hyperparameters are optimized using the Tree-structured Parzen Estimator (TPE) [11].

Neural Architecture Search (NAS) algorithms [12]–[14] are used to tune the hyperparameters automatically. NAS methods find the structure of the model from scratch and hence require a lot of GPU hours. Basha et al. [15] and [16] proposed a method to autotune CNN models for improved transfer learning using a Bayesian optimization technique. Motivated by the success of [15] for tuning FC layers following a CNN, in terms of accuracy and GPU hours, we apply the TPE algorithm, which is a variant of Bayesian optimization, to fine-tune FC layers following a self-supervised model. Next, we make a detailed survey of literature in the related areas.

II. Related Work

Tuning hyperparameters related to machine learning models, has been an active area of research during the last few years [12], [17]. Deep learning architectures, due to the huge number of hyperparameters, require an efficient procedure for tuning before applying to any specific task. Deep learning architectures can be tuned either manually or automatically. The manual tuning of architectures require a lot of human effort and time. Hence, researchers are recently focusing on automatically searching for hyperparameter values for making better deep learning models specific to a given problem or dataset. NAS [12], [13] is the area of research to search the hyperparameters related to an architecture automatically.

Recent NAS procedures can be categorized broadly into four classes: Heuristics based [18], [19], Reinforcement learning-based [20], evolutionary algorithm based [21], and gradient-based [22] approaches. Heuristics based approaches are experimental approaches, and hence, require a large computation time [19]. In [19], Polap et al. experimented with several heuristic algorithm to tune the hyperparameters for a specific task. Subramanian et al. applied heuristics to find out suitable hyperparameter values related to a specific problem of identification of diseases in leaves [18].

The NAS algorithms generally require a lot of GPU and computational overhead. Reinforcement learning-based algorithms are computationally most expensive among all the three categories of approaches for NAS. For instance, Zoph et al. [23] took 12,800 proxy CNN models, 128 days and 800 GPUs. In [21], Zoph et al. took 20000 samples of CNN models and used 500 GPUs for four days. Gong et al. [24] proposed AutoGAN, where they introduced the NAS method to GAN architecture. They used an RNN controller to guide the search of hyperparameter values. Their model achieved better performance compared to the handcrafted (intuition based) GAN models.

Our goal in this study is to apply NAS to fine-tune a pre-trained self-supervised architecture while transferring the learned features from a source task to a target task. Transfer learning facilitates to reuse of the intellectual knowledge from the source task and produces satisfactory results after fine-tuning these models for the target task. The pre-trained model designed for the source task may not be suitable for the target task, and hence an efficient fine-tuning approach is necessary to make it ideal for the target task, especially when training sample size in the target data is less, leading to overfitting. In [25], Han et al. combined transfer learning and web data augmentation for reducing the overfitting. Further, [25] tuned the hyperparameters such as type of optimizer, learning rate, drop-out rate, etc. using Bayesian optimization. This paper employed Bayesian optimization but tuned only a few hyperparameters, leaving many other hyperparameters such as number of units in a layer, number of layers, etc., untuned, which affect the classification performance. Mendoza et al. [26] proposed Auto-net to provide an automatically tuned feed-forward neural network, and shown superior performance on some datasets. They automatically tuned several hyperparameters such as number of FC layers, number of neurons in each FC layer, batch size, learning rate, and many more.

Recently Bayesian optimization (BO) based approaches [27] are becoming popular because of their less computational overhead compared to the other approaches to learn architectures automatically. Cho et al. [28] explained early termination, diversification, cost function transformation, and parallelization, which are four basic strategies of BO. They examined the robustness of BO for tuning problems for various deep neural networks by applying the basic strategies on benchmark datasets with pre-evaluated performance. However, [28] could not choose the hyperparameter values, range of hyperparameters and were unable to set up the maximum number of epochs. Victoria et al. [29] used the BO algorithm to optimize hyperparameters to enhance the performance of a CNN model. In [15] and [16], Basha et al. presented a method for autotuning hyperparameters using Bayesian optimization. The scope of [15] is limited to autotune the FC layers of CNN models while [16] automatically tuned some CNN layers as well. They optimized the hyperparameters such as filter size in CNN layers, number of filters, filter size in max pooling, Number of FC layers and neurons, and many more, to obtain significantly enhanced performances of CNNs. However, no such efforts on self-supervised architectures, are found in the literature.

Nguyen et al. [30] tuned the parameters in LSTM network

using the TPE algorithm. They proposed the LSTM prediction model and applied it with the long-term dependencies in time-series data. In this model, The hyperparameters were autotuned using a variant of Bayesian optimization called the TPE algorithm. The TPE-based model showed promising performances when applied to the LSTM and RNN models. The success of the TPE algorithm motivates us to use it on self-supervised models.

All the approaches for autotuning the hyperparameters of deep learning models discussed above, are applied on CNNs and RNNs. However, fine-tuning a self-supervised model is relatively a less explored area of study. Kaplan et al. [14] proposed a self-supervised neural architecture search (SSNAS) algorithm for finding network models with unlabelled data. This approach is carried out using a single GPU. This paper used the self-supervised approach of SimClr model [7], and the same setup as in DARTS [22]. The model was trained for a few epochs (100) and picked the best model based on validation accuracy. However, [14] was limited to a smaller model. Liang et al. [31] proposed self-supervised vertical federated neural architecture search (SS-VFNAS), to apply the self-supervised neural architecture search technique in VFNAS to improve the vertical federated learning (VFL) privacy.

We observe that only a limited number of efforts are made for autotuning on self-supervised models. Although two recent approaches are found on fine-tuning self-supervised models, they are not very rigorous about the search space of the hyperparameters and lack experimentation on various self-supervised models. This study is an effort to fine-tune self-supervised models using the TPE algorithm, with a wide spectrum of search spaces for the hyperparameters. Our research contributions are as follows:

- This is the first attempt to tune the learned features obtained from the self supervised models.
- We propose a method for fine-tuning a set of FC layers followed by a softmax classification layer, to efficiently transfer the knowledge learned by a self-supervised model, to perform image classification.
- We effectively apply TPE algorithm for fine-tuning self-supervised classification models based on CNNs.
- Recent self-supervised models such as SimClr and SWAV are used for experimenting with the proposed approach.
- The experiments are carried out with benchmark datasets.

Next we illustrate the proposed approach.

III. Proposed Method

The objective of the proposed method is to fine-tune the hyperparameters of the Image classifier using the pretrained self-supervised learning model with transfer learning. Figure 3 shows a simple framework of the proposed approach where the self-supervised model is used as the backbone, followed by the FC layers and a softmax layer. The proposed technique automatically tunes hyperparameters such as the number of layers, number of channels in each layer, dropout rate, learning rate, etc., of the FC layers. To find the optimized hyperparameters, We employ the Tree-structured parzen estimator Approach (TPE) [11], [30].

A. TPE algorithm

In the proposed Bayesian Optimization based TPE procedure, we can not directly access the objective function F for obtaining the optimized hyperparameter values. We take validation accuracy as the objective function F. Let us define F as $F : \mathbb{R}^d \rightarrow \mathbb{R}$. The Mathematical definition for optimizing the hyperparameters may be as follows,

$$x_* = \arg\max_{x \in H} F(x). \tag{1}$$

Here $x \in \mathbb{R}^d$ is the input vector consisting of the initial hyperparameter values, H indicates the set of the hyperparameter search space, and x_* is the optimal configuration of hyperparameters learned by the Bayesian optimization method. Then, x_* is responsible for obtaining the best performance on validation data with less overfit.

Hyperparameter optimization is the problem of optimizing a loss function over a graph-structured configuration space. In this work we restrict ourselves to tree-structured configuration spaces. Configuration spaces are tree-structured in the sense that some leaf variables (e.g., the number of hidden units in the 2nd layer of a network) are only well-defined when node variables (e.g., a discrete choice of how many layers to use) take a particular value. A hyperparameter optimization algorithm not only optimizes over variables which are discrete, ordinal, and continuous, but also it simultaneously choose which variables to optimize.

Conventional Bayesian optimization has the limitation to choose the hyperparameter values based on the probability score for the value, for a given fitness function. This probability score provide less tolerance of error in the training data. Instead, TPE algorithm chooses the hyperparameter value [30] based on the ratio between the probability of the value is optimal and the suboptimal probability values, providing more tolerance for error in the training data. This tolerance to the errors provide more reliable performance in self-supervised models. With Bayesian reasoning and updating scheme of the surrogate model, TPE tries to get better hyperparameter values in each trial. The empirical data is assumed to follow a Gaussian distribution. We choose the initial samples from the response surface randomly. The response surface is given as $\{h_i, f_i \; where \; i = 1, ..., m_0\}$, where h is the hyperparameter values set up, f is the value of response surface called fitness function, assumed to follow the Gaussian distribution, and m_i is the ith iteration. Then, based on the predefined threshold value f^* on f, the hyperparameter space is divided into two groups. The first group contains the search space with the best score (giving the optimal value of f), and the second group contains other possible (non-optimal) values from the search space. We choose f^* to be some quantile γ of the experimental f values in the way that $p(f < f^*) = \gamma$. Surrogate function in TPE algorithm directly follows the Bayes rule hence approximated by $P(h/f)$ (probability of

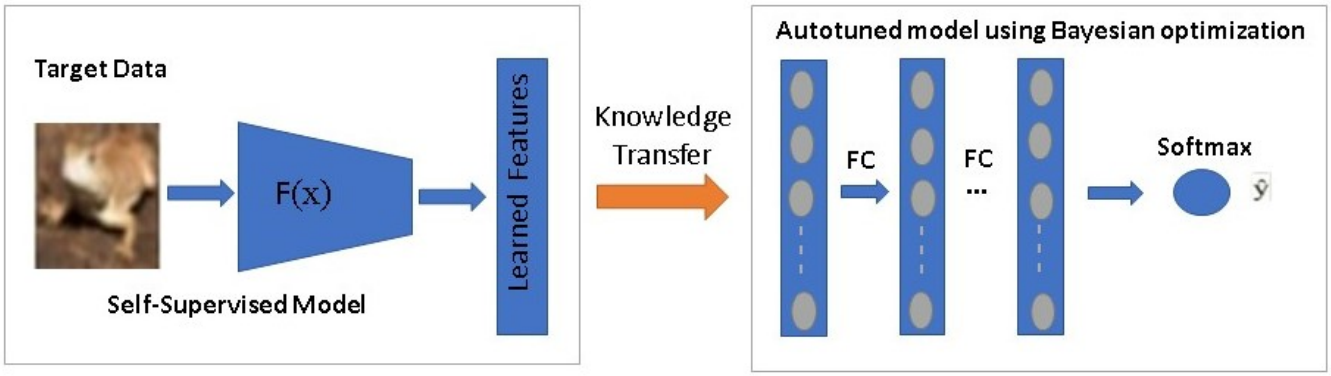

Fig. 3. A diagram of the proposed method for classification tasks by automatically tuning the hyperparameters such as number of layers, number of units in each layer, dropout rate, and learning rate after transferring the knowledge from a self-supervised model over target datasets.

hyperparameter set for the given fitness function f) and $P(f)$ (probability of fitness function). TPE defines the conditional probability score $P(h/f)$ using two densities distributions of h and f as follows:

$$P(h/f) = \begin{cases} p_{optimal}(h) & iff < f^* \\ p_{other}(h) & iff \geq f^*. \end{cases} \tag{2}$$

Here $p_{optimal}(h)$ is the density formed by the observation that for $\{h^i\}$, the loss for the f is less than f^*. Subsequently, $p_{other}(h)$ is the density with the remaining observations in the search space. The parameter selection is based on the best observations and their distributions. Hence, we get better distribution at the starting if the initial iteration is more. We get the next hyperparameter set by maximizing Expected Improvement (EI), an acquisition function. Finally, we define EI to maximize by the following expression,

$$EI = \frac{p_{optimal}(h)}{p_{other}(h)} \tag{3}$$

In TPE algorithm, EI is taken as the score of the hyperparameter values unlike the convensional BO based methods which rely only on the probability of the optimal score. In each iteration, the proposed algorithm returns h^* with the greatest EI score. Next we discuss about the search space for the hyperparameter values.

B. Hyperparameter Search Space

Obtaining a suitable search space for the hyperparameters is crucial. The number of FC layers, number of units in each FC layer, Dropout, and learning rate, are the hyperparameters that are autotuned by the proposed method. Table 1 shows the hyperparameter search spaces corresponding to each hyperparameter. By observing the popular CNN-based architectures, we can conclude that number of FC layers can vary between 1 and 3 to design a classification model with better accuracy [32], [33]. Moreover, the study made in [16] suggests that the maximum required number of FC layers followed by a deep architecture is 3. Hence in this study, we take 1,2,3 as the search space for the number of FC layers. For the number of units in each layer, we choose the search space as {128 to 1024, with step 64}. Dropout rates are chosen from the range .3 to .5, with step .1, and the learning rate is chosen from 1e-5 to 1e-1. Most hyperparameter search space choices are inspired by the papers [15] and [16]. Next we discuss the experimental setup for evaluating the proposed model.

TABLE I
HYPERPARAMETER SEARCH SPACE USED IN OUR EXPERIMENTS.

S.No.	HP_Name	HP_Range	HP_Type
1	Number of FC layers	$\{1, 2, 3\}$	int
2	no. of neurons	[128,1024,with step 64]	int
3	learning rate	[1e-4,1e-1]	loguniform
4	dropout rate	[0.3,0.5,with step .1]	float

IV. Experimental Setting

We start with the details about the training procedures for experimenting on the proposed approach, followed by a brief discussion on the self-supervised models used in our experiments. Finally we provide a brief description on the datasets used in this study for performing the experiments.

A. Training Details

We perform all the experiments using the software framework PyTorch lightening [34]. We use the pre-trained Self-supervised models from Pytorch lightning bolts [35] . We implement the TPE algorithm using optuna library [36].

We consider the recent self-supervised models such as Simclr and SWAV models for experimentation. All these models are trained over imagenet dataset [37]. These models are autotuned over the target datasets such as CIFAR-10 [38], CIFAR-100 [38], and Tiny ImageNet [39] for image classification with transfer learning.

Data preprocessing plays a vital role in reducing overfitting problems. As the self-supervised models are trained with

TABLE II

PERFORMANCE OF THE PROPOSED METHOD FOR AUTOTUNING THE HYPERPARAMETERS WHILE TRANSFERRING THE LEARNED FEATURES OF POPULAR SELF SUPERVISED MODELS. THE PROPOSED AUTUNED CLASSIFIER IS COMPARED WITH LINEAR CLASSIFIER, WHEN APPLIED ON THE BENCHMARK DATASETS, IN TERMS OF PERCENTAGE OF ACCURACY.

S.No	Model	Dataset	Classifier	#FC Layers	#units	Learning_Rate	Dropout	Validation _Accuracy
1	SimClr	CIFAR-10	Autotuned	2	[832,768]	0.00594	0.3	**89.14**
2			Linear	NA	NA	0.00594	NA	88.24
3		CIFAR-100	Autotuned	2	[1024,832]	0.00687	0.5	**73.48**
4			Linear	NA	NA	0.00687	NA	71.64
5		Tiny ImageNet	Autotuned	1	[768]	0.00489	0.5	**77.03**
6			Linear	NA	NA	0.00489	NA	76.12
7	SWAV	CIFAR-10	Autotuned	2	[1024,256]	0.006	0.4	**91.2**
8			Linear	NA	NA	0.006	NA	88.56
9		CIFAR-100	Autotuned	1	[1024]	0.01242	0.4	**75.7**
10			Linear	NA	NA	0.01242	NA	74.74
11		Tiny ImageNet	Autotuned	1	[512]	0.003789	0.5	**80.8**
12			Linear	NA	NA	0.003789	NA	78.35

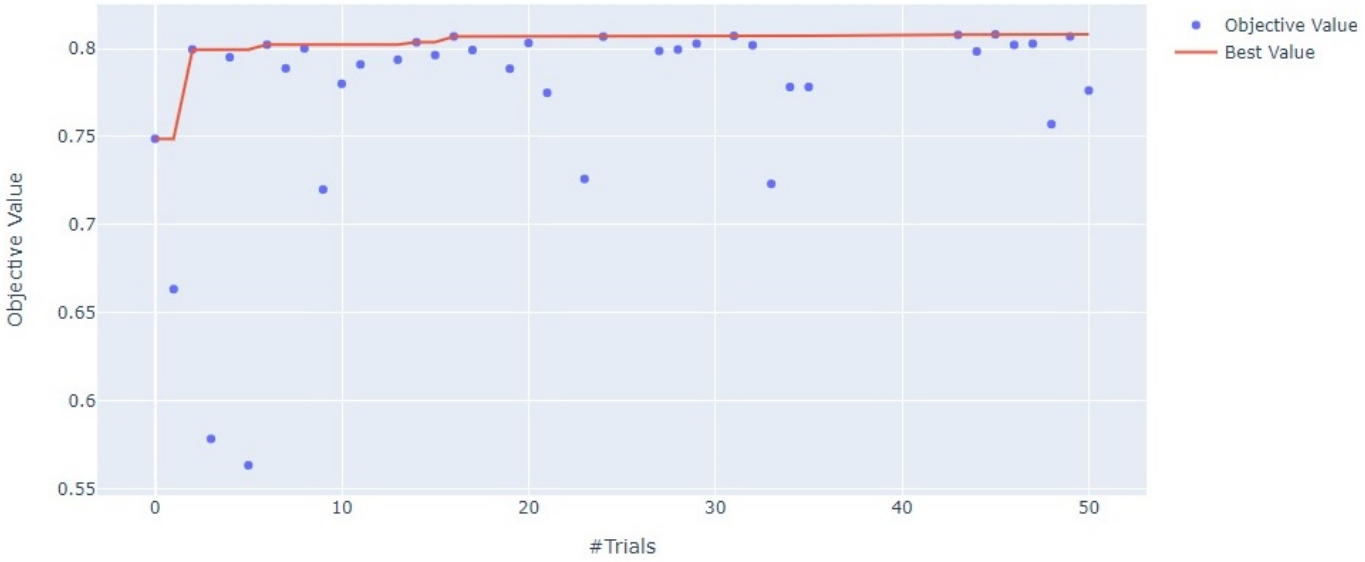

Fig. 4. Optimization History Plot when the SWAV model [10] is autotuned with the proposed method and tested on the Tiny ImageNet dataset [39]. Here the X-axis denotes the number of trials, and Y-axis refers to the validation accuracy. The blue points denote the validation accuracy with respect to the number of trials. The blue points which fall on the red line indicate the best possible value (i.e., accuracy) after completing the particular trials.

image size 224×224, we up-sample the images of any dataset to 160×160. Experimentally we observe that up-sampling more than size 160×160 does not influence the performance of our model. Whereas downsampling below the size of 160×160 causes overfitting. The kind of augmentations on the images we applied for training the self-supervised models include horizontal flip with probability .5 and data normalization.

We trained our model for 30 epochs with batch size 64 using Nesterov momentum SGD with a momentum parameter of .9, inspired from [7]. We select Cosine Annealing learning rate schedular with weight decay (L2 penalty multiplier) of 1e-4.While training our model, we have observed that after nine epochs, the accuracy was saturating, so we have modified the weights of the entire model at epoch ten only.Each Experiment is carried out for 50 trials, and the best set of hyperparameters is chosen from these trials.

B. Self-supervised models used as backbone

Several self-supervised learning (SSL) models can be found in the literature solving different image related problems. Among them, contrastive learning-based models are the most popular SSL models because of their performances. In this study we experiment with two contrastive SSL models: Simclr [7] and SWAV [10].

1) SimClr [7]: The main ConvNet backbone in Simclr is ResNet-50 [40]. The augmented image of shape (224,224,3) is the input for ResNet-50, and the network provides the output of 2048 dimensional embedding vector h. Then a projection head $g(.)$ is applied to the embedding vector h, which produces a final representation $z = g(h)$. The projection head $g(.)$ is a Multilayer Perceptron (MLP) consisting of two dense layers. Both layers have 2048 units, and the hidden layer has a non-linear (ReLU) activation function. The number of outputs in the last hidden layer of the projection head is 128. For training,

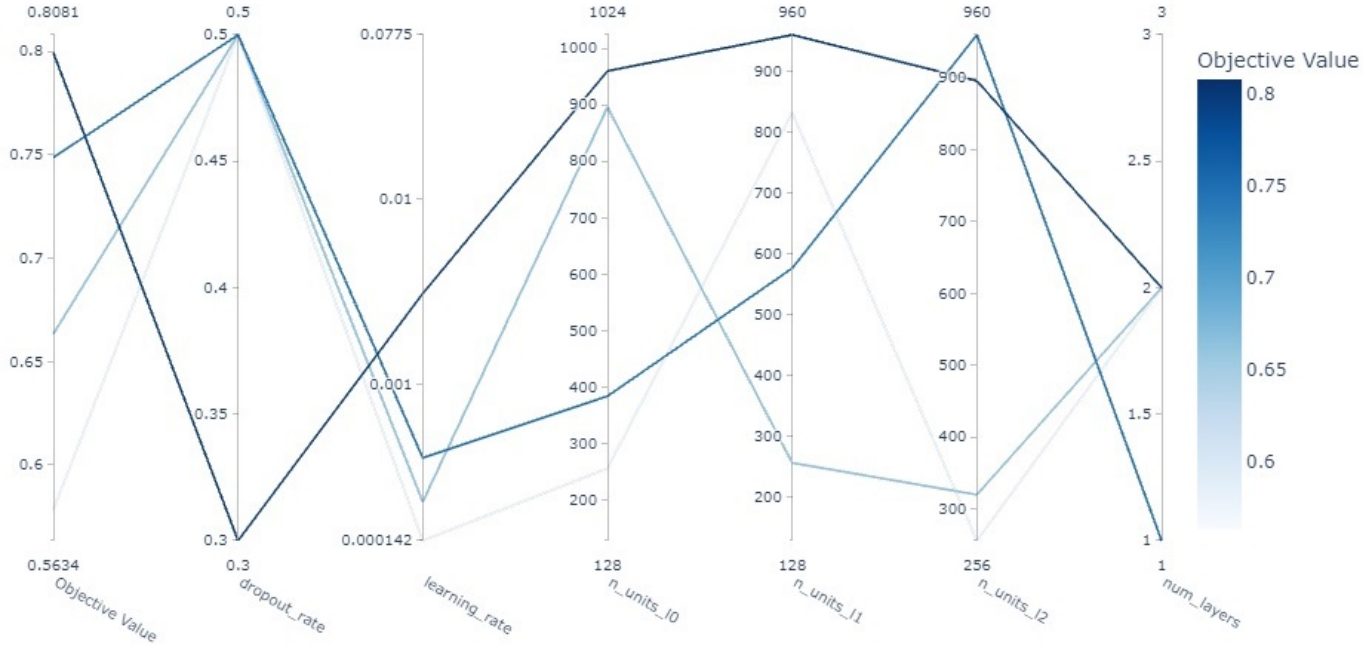

Fig. 5. Parallel Coordinate Plot for the proposed autotuning method when applied on the SWAV model [10], and experimented with the Tiny ImageNet dataset [39]. Vertical line refers to the objective values (validation accuracy) and a set of hyperparameters. Numbers indicated along the vertical lines represent the hyperparameter search space. We can visualize a few examples of the mapping of different hyperparameter set up, and their influence on the objective values.

They used normalized temperature-scaled cross-entropy loss (NT-Xent).

2) SWAV [10]: SwAV proposed a multi-crop augmentation policy where the same image is randomly cropped to get a pair of high-resolution images (224x224) and trimmed to get additional views of low-resolution images (96x96). The augmented images are passed through the ResNet50 backbone to get the embedding vector. This embedding vector is then passed to an external non-linear network. The output of this network is the projection vector denoted by Z. The projection vector is fed to a single linear layer. This linear layer is the prototype layer and is characterized by C. This layer maps Z to K trainable prototype vectors. The layer's output is the dot product between Z and the prototypes. This layer's associated weight matrix can be considered as a learnable prototype bank. Finally, a part of the output of this linear layer is used for cluster assignment using the Sinkhorn Knopp algorithm [41], and a swapped prediction problem is set up.

C. Datasets

We used three publicly available benchmarked datasets, which include CIFAR-10, CIFAR-100, and Tiny ImageNet.

1) CIFAR-10: CIFAR-10 dataset was proposed by Krizhevsky et al. [38]. The dataset contains images from 10 classes. CIFAR-10 dataset consists of 60000 32x32 color images, with 6000 images per class. This dataset contains 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each having 10000 images. The test batch contains 1000 randomly selected images from each class. The training batches contain the remaining images in a random order, but some training batches may contain more images from one category than another. Between them, the training batches contain exactly 5000 images from each class.

2) CIFAR-100: CIFAR-100 dataset was also proposed in [38]. The dataset contains images from 100 classes. It has 600 images in each category. This dataset includes 500 training images and 100 testing images per category. The 100 types in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a good label and a coarse label.

3) Tiny ImageNet [39]: This is a subset of the ImageNet dataset. It contains images from 200 categories with 64x64 image sizes. It includes 500 training images, 50 validation images, and 50 test images for each class.

V. Results and Analysis

We perform several experiments to validate the efficacy of the proposed autotuning method. The results of the experiments are compared using the same setup offered by the [7] for transfer learning. Table II shows that the performance of our auto-tuned classifier is better than the linear classifier on all the three datasets, when applied with both the simCLR and SWAV models. Further, Table II shows the optimized hyperparameter values obtained from the proposed method for each dataset. As linear classifier, we use the same learning rate as in the corresponding autotuned classifier. Next we discuss the performance of the proposed method compared to the state-of-the-art, on three benchmark datasets.

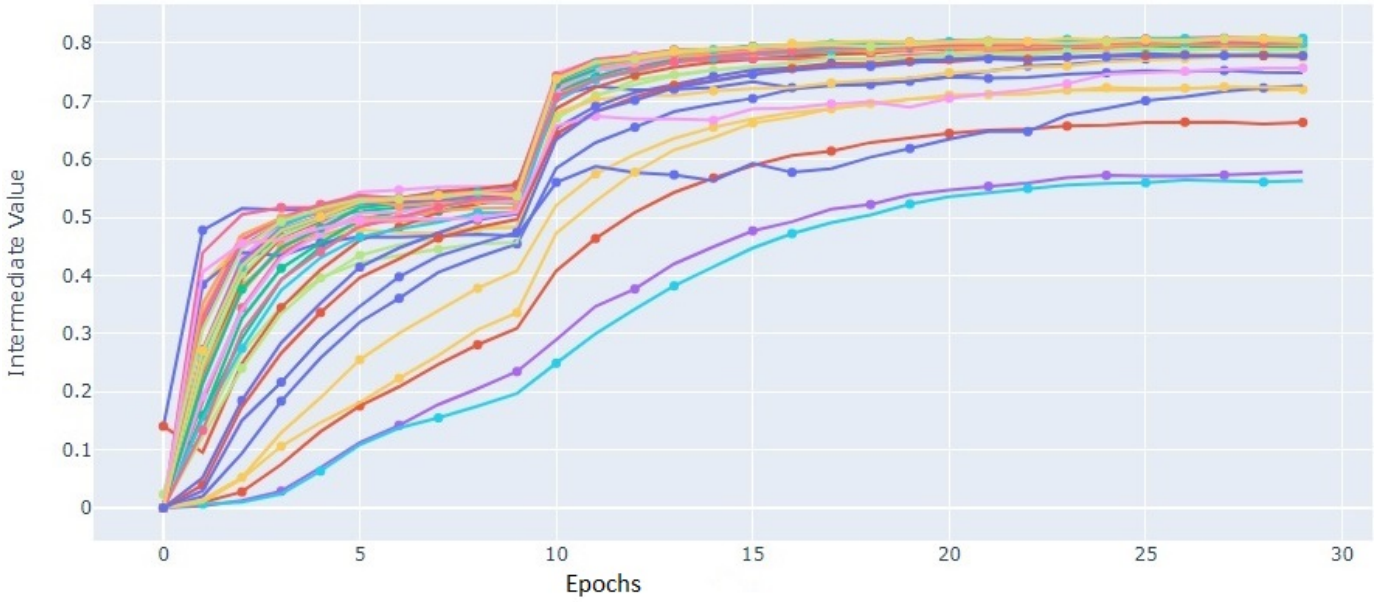

Fig. 6. Intermediate Value Plot for the proposed autotuning model when applied on SWAV model [10] and experimented with Tiny ImageNet dataset [39]. The X-axis refers to the number of epochs in each trial, and Y-axis refers to Validation accuracy. The various lines in this figure show the validation accuracy obtained by different trials with the hyperparameter set up, with respect to the number of epochs.

TABLE III
PERFORMANCE OF THE PROPOSED AUTOTUNING METHOD ON THE CIFAR-10 DATASET, COMPARED TO THE STATE-OF-THE-ART.

Dataset	Method	Accuracy
CIFAR-10	SimClr [35]	88.5
	CPC(V2) [35]	84.5
	Ours(SimClr + Autotuned Classifier)	**89.14**
	Ours(SWAV + Autotuned Classifier)	**91.2**

TABLE IV
PERFORMANCE OF THE PROPOSED AUTOTUNING METHOD ON THE CIFAR-100 DATASET, COMPARED TO THE STATE-OF-THE-ART.

Dataset	Method	Accuracy
CIFAR-100	SimClr [7]	71.6
	AMDIM-small [42]	71.5
	AMDIM-large [42]	72.8
	Ours(SimClr + Autotuned Classifier)	**73.48**
	Ours(SWAV + Autotuned Classifier)	**75.7**

A. CIFAR-10 Image Classification Results

We compare the performance of the proposed method with the state-of-the-art self-supervised methods, as shown in Table III. The comparison clearly shows that our method outperforms the popular self supervised models. Further, the proposed method could significantly improve the performance of the self supervised models as depicted from Table III.

B. CIFAR-100 Image Classification Results

CIFAR-100 dataset contains the same images contained in the CIFAR-10 dataset, whereas the number of classes are increased to 100 instead of 10, requiring more fine classification of features of images. The comparison of the proposed method with the state-of-the-art self-supervised methods, as shown in Table IV, clearly shows that our method outperforms the popular self supervised models. Further, the proposed method could significantly improve the performance of the self supervised models.

C. Tiny ImageNet Image Classification Results

Table V shows the comparative result of our autotuned classifier with the state-of-the-art methods. Further, the proposed method could significantly improve the performance of the self supervised models. In case of a huge dataset like ImageNet, we can further observe that, with the optimized hyperparameter values provided by the proposed autotuning model, the popular self supervised model can even outperform the state-of-the-art supervised model such as UPANets [43], as shown in Table V. We can further observe that, both the self supervised models have outperformed the UPANet model with significant margin.

Figures 4, 5, and 6 visualizes the process of getting optimized hyperparameters using the proposed autotuning method with SWAV model when applied on the Tiny ImageNet dataset. While autotuning the hyperparameters, we get objective value (validation accuracy) with respect to each trial (hyperparameter set up). Figure 4 refers to the optimization history plot where the points that appear in the path of the red line indicate the best objective values, which are decided by Maximizing the Expected Improvement (EI) using the Equation (3). Our target is to maximize the objective value using the TPE algorithm, and the trial for which we get maximum validation accuracy is considered to be the best. The blue points that lie on the red line indicate the best possible objective value obtained with the particular trial. After completing the specific trial, the objective value of that trial will be compared with the objective

TABLE V
PERFORMANCE OF THE PROPOSED AUTOTUNING METHOD ON THE TINY IMAGENET DATASET, COMPARED TO THE STATE-OF-THE-ART.

Dataset	Method	Accuracy
Tiny ImageNet	UPANets [43]	67.67
	PreActResNet18 [44]	63.48
	Ours(SimClr + Autotuned Classifier)	**77.03**
	Ours(SWAV + Autotuned Classifier)	**80.8**

value of previously completed trials. If that value is maximum among other values, the red line will pass through that point. Otherwise the objective value obtained for the particular trial lie below the red line. Figure 4 shows that, for most of the trials, the blue dots either lie on the red line, or are just below the red line. This means, the proposed method can efficiently search for potential optimized hyperparameter values.

Figure 5 few examples of the objective values obtained from specific hyperparameter values. Here we can visualize the mapping of different hyperparameter set with the objective values (validation accuracy).

We execute our model for 30 epochs for each trial in our experiments. Figure 6 indicates the accuracy of different hyperparameter settings (with respect to the number of epochs) suggested by the proposed method, with SWAV model applied on Tiny ImageNet dataset. Figure 6 shows that, the proposed method can achieve best possible accuracy in only around 20 epochs, thus saving the computational overhead.

VI. CONCLUSIONS

We propose a method to automatically tune the hyperparameters of a self supervised model. We apply TPE algorithm for tuning the self supervised model. We experiment with two state-of-the-art self supervised models and observe that, the proposed method can improve the performances of both the self supervised models in terms of accuracy in image classification. We experiment on three benchmark datasets to establish the efficacy of the proposed method. Further we observe that, with the hyperparameters set up by the proposed model, the self-supervised model could produce even better results that the state-of-the-art supervised models on the Tiny ImageNet dataset. In future the proposed autotuning method can be applied on self supervised transformer networks.

REFERENCES

[1] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *ECCV*, 2016, pp. 649–666.

[2] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *ICCV*, 2015, pp. 1422–1430.

[3] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *ECCV*, 2016, pp. 69–84.

[4] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "context encoders: feature learning by inpainting," in *CVPR*, 2016, pp. 2536–2544.

[5] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," in *ICLR*, 2018.

[6] I. Misra, C. L. Zitnick, and M. Hebert, "Shuffle and learn: unsupervised learning using temporal order verification," in *ECCV*, 2016, pp. 527–544.

[7] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *ICML*, 2020, pp. 1597–1607.

[8] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *CVPR*, 2020, pp. 9729–9738.

[9] X. Chen, H. Fan, R. Girshick, and K. He, "Improved baselines with momentum contrastive learning," *arXiv preprint arXiv:2003.04297*, 2020.

[10] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised learning of visual features by contrasting cluster assignments," in *NIPS*, 2020.

[11] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *NIPS*, vol. 24, 2011.

[12] T. Elsken, J. H. Metzen, F. Hutter *et al.*, "neural architecture search: a survey." *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.

[13] M. Wistuba, A. Rawat, and T. Pedapati, "A survey on neural architecture search," *arXiv preprint arXiv:1905.01392*, 2019.

[14] S. Kaplan and R. Giryes, "Self-supervised neural architecture search," *arXiv preprint arXiv:2007.01500*, 2020.

[15] S. H. Basha, S. K. Vinakota, S. R. Dubey, V. Pulabaigari, and S. Mukherjee, "autofcl: automatically tuning fully connected layers for handling small dataset," *Neural Computing and Applications*, 2021.

[16] S. H. Shabbeer Basha, S. K. Vinakota, V. Pulabaigari, S. Mukherjee, and S. R. Dubey, "autotune: automatically tuning convolutional neural networks for improved transfer learning," *Neural Networks*, vol. 133, pp. 112–122, 2021.

[17] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[18] M. Subramanian, K. Shanmugavadivel, and P. Nandhini, "On fine-tuning deep learning models using transfer learning and hyper-parameters optimization for disease identification in maize leaves," *Neural Computing and Applications*, 2022.

[19] D. Polap, M. Wozniak, W. Hołubowski, and R. Damasevicius, "A heuristic approach to the hyperparameters in training spiking neural networks using spike-timing-dependent plasticity," *Neural Computing and Applications*, 2021.

[20] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *ECCV*, 2018, pp. 19–34.

[21] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *CVPR*, 2018, pp. 8697–8710.

[22] H. Liu, K. Simonyan, and Y. Yang, "darts: differentiable architecture search," in *ICLR*, 2019.

[23] b. zoph and q. v. le, "Neural architecture search with reinforcement learning," in *ICLR*, 2017.

[24] X. Gong, S. Chang, Y. Jiang, and Z. Wang, "autogan: neural architecture search for generative adversarial networks," in *ICCV*, 2019, pp. 3224–3234.

[25] D. Han, Q. Liu, and W. Fan, "a new image classification method using cnn transfer learning and web data augmentation," *Expert Systems with Applications*, vol. 95, pp. 43–56, 2018.

[26] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter, "Towards automatically-tuned neural networks," in *Workshop on Automatic Machine Learning*, 2016, pp. 58–65.

[27] p. i. frazier, "a tutorial on bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[28] H. Cho, Y. Kim, E. Lee, D. Choi, Y. Lee, and W. Rhee, "Basic enhancement strategies when using bayesian optimization for hyperparameter tuning of deep neural networks," *IEEE Access*, vol. 8, pp. 52 588–52 608, 2020.

[29] A. H. Victoria and G. Maragatham, "automatic tuning of hyperparameters using bayesian optimization," *Evolving Systems*, vol. 12, pp. 217–223, 2021.

[30] H.-P. Nguyen, J. Liu, and E. Zio, "a long-term prediction approach based on long short-term memory neural networks with automatic parameter optimization by tree-structured parzen estimator and applied to time-series data of npp steam generators," *Applied Soft Computing*, vol. 89, p. 106116, 2020.

[31] X. Liang, Y. Liu, J. Luo, Y. He, T. Chen, and Q. Yang, "Self-supervised cross-silo federated neural architecture search," *arXiv preprint arXiv:2101.11896*, 2021.

[32] a. krizhevsky, i. sutskever, and g. e. hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.
[33] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[34] e. a. Falcon, WA, "pytorch lightning," *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, vol. 3, 2019.
[35] W. Falcon and K. Cho, "a framework for contrastive self-supervised learning and designing a new approach," *arXiv preprint arXiv:2009.00104*, 2020.
[36] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "optuna: a next-generation hyperparameter optimization framework," in *International Conference on Knowledge Discovery and Data Mining*, 2019.
[37] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.
[38] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.
[39] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," *CS 231N*, vol. 7, no. 7, p. 3, 2015.
[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.
[41] p. a. knight, "the sinkhorn–knopp algorithm: convergence and applications," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 1, pp. 261–275, 2008.
[42] P. Bachman, R. D. Hjelm, and W. Buchwalter, "Learning representations by maximizing mutual information across views," in *NIPS*, 2019, pp. 15 535–15 545.
[43] C.-H. Tseng, S.-J. Lee, J.-N. Feng, S. Mao, Y.-P. Wu, J.-Y. Shang, M.-C. Tseng, and X.-J. Zeng, "Upanets: Learning from the universal pixel attention networks," *arXiv preprint arXiv:2103.08640*, 2021.
[44] J.-H. Kim, W. Choo, and H. O. Song, "Puzzle mix: Exploiting saliency and local statistics for optimal mixup," in *ICML*, 2020, pp. 5275–5285.