EasyChair Preprint

№ 14726

Efficient Self-Supervised Neural Architecture Search

Sri Aditya Deevi, Asish Kumar Mishra, Deepak Mishra,
L Ravi Kumar, G V P Bharat Kumar and
G Murali Krishna Bhagavan

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

September 6, 2024

Efficient Self-Supervised Neural Architecture Search

Sri Aditya Deevi*
Mission Simulation Group
U.R. Rao Satellite Centre, ISRO
Bengaluru, India
saditya@ursc.gov.in

Asish Kumar Mishra*
Mission Simulation Group
U.R. Rao Satellite Centre, ISRO
Bengaluru, India
asishkm@ursc.gov.in

Deepak Mishra
Department of Avionics
Indian Institute of Space Science and Technology
Thiruvanathapuram, India
deepak.mishra@iist.ac.in

Ravi Kumar L
Mission Simulation Group
U.R. Rao Satellite Centre, ISRO
Bengaluru, India
rkkumarl@ursc.gov.in

Bharat Kumar G V P
Mission Simulation Group
U.R. Rao Satellite Centre, ISRO
Bengaluru, India
bharat@ursc.gov.in

Murali Krishna Bhagavan G
Mission Simulation Group
U.R. Rao Satellite Centre, ISRO
Bengaluru, India
bhagavan@ursc.gov.in

***Abstract*—Deep Neural Networks (DNNs) have successfully demonstrated superior performance on many tasks across multiple domains. Their success is made possible by expert practitioners' careful design of neural architectures. This manual handcrafted design requires a colossal number of computational resources, time, and memory to arrive at an optimal architecture. Automated Neural Architecture Search (NAS) is a promising area to explore to overcome these issues. However, optimizing a network for a job is a tedious task that requires lengthy search time, high processor needs, and a thorough examination of enormous possibilities. The need of the hour is to develop a strategy that saves time while maintaining an excellent level of accuracy. In this paper, we design, explore, and experiment with various differentiable NAS methods which are memory, time, and compute efficient. We also explore the role and efficacy of self-supervision to guide the search for optimal architectures. Self-Supervision offers numerous advantages such as facilitating the use of unlabelled data and making the "learning" non-task specific, thereby improving transfer to other tasks. To study the inclusion of self-supervision into the search process, we propose a simple loss function consisting of a convex combination of supervised cross-entropy loss and self-supervision loss. In addition, we carried out various analyses to characterize the performance of different approaches considered in this paper. The inspection of results obtained from various experiments on CIFAR-10 reveals that the proposed methodology balances time and accuracy while staying as near as possible to the state-of-the-art results.**

***Index Terms*—Neural Architecture Search, Self-Supervised Learning, Deep Learning, Efficiency**

I. Introduction

Human beings have had the zeal to automate various tasks in their daily lives and build remarkable things since time immemorial. Machine Learning approaches involve domain intensive handcrafting of features from data. Deep Learning automates feature extraction but requires handcrafting of architectures which require expert practical knowledge and massive compute resources. Neural Architecture Search takes one more step forward in this trend of automation by trying to find an optimal architecture for effective performance.

The idea of automating the discovery of architectures that achieve state-of-the-art performance has been further encouraged by the highly competitive performance of automatically searched architectures [1][2][3][4] on popular tasks such as Image Classification, Object Detection etc.

The different types of methods for Neural Architecture Search can be broadly classified into the following types: reinforcement learning (RL) based, genetic evolutionary algorithm based and gradient-based (differentiable search space). Most of the RL based and genetic algorithm-based methods are highly time-consuming and resource-intensive, requiring several GPU days (or weeks!) [5][2][1][6], making them infeasible even for moderately sized datasets.

In order to make the NAS techniques more efficient, [7] introduced a bilevel optimization-based differentiable framework (DARTS) that brought in a tremendous reduction in search times. Since then, several improvements [8][9][10] have been proposed for improving the performance and efficiency of DARTS.

The other vital part of our work, i.e. Self-Supervision, is necessary if we wish to train the networks with data abundantly. A lot of the data in the outside world is unlabelled and fully supervised algorithms cannot penetrate this space. Self-supervised learning has shown its importance in the field of Deep Learning in many of the recent papers [11] [12] [13]. Searching the network with self-supervision may aid in better transfer training and testing with different datasets.

In this work, we explore and experiment with methods to search for the optimal architectures in a memory, compute and time-efficient manner. We also illustrate the efficacy of self-supervision for guiding effective architecture exploration and operation selection. In addition, we try to provide an analytical understanding of the performance of different approaches from various vantage points. This paper is organized into the following sections : Section II is a basic overview of relevant theory. Various details related to the block level model design, experimental setup and other implementation details are provided as a part of Section III. Section IV provides

*These authors contributed equally to this work.

details about various design considerations.The qualitative and quantitative results obtained from various experiments are summarized in Section V. In Section VI, we provide analysis from various perspectives about the performance of different approaches that we experimented on. Section VII concludes the paper by reviewing some avenues of further research.

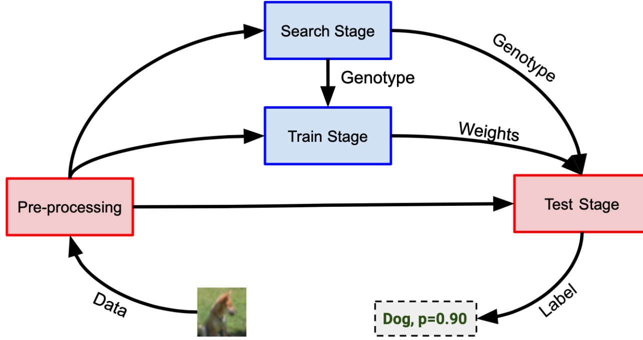

FIG. 1: High Level Block Diagram. In the test stage, the model utilizes genotype, which describes the optimized architecture from search and the weights, which are optimized during train stage.

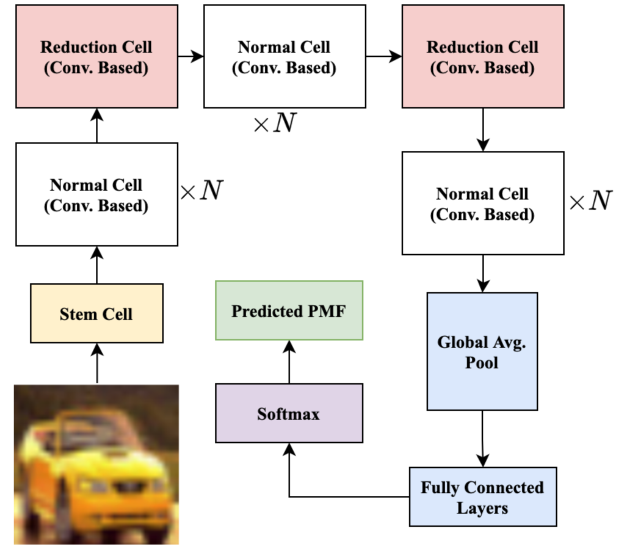

FIG. 2: Structure of the final architecture. Normal cells preserve the input spatial dimensions where each Reduction cell (present at roughly (1/3) and (2/3) of the total network depth) halve feature map's spatial dimensions. Note that, N can be different during search and train stages.

II. Theory and Methodology

The different stages of our approach are illustrated in FIG. 1. The pre-processed images taken from a given dataset (can be proxy or final) are utilized for performing *search* of an optimal architecture denoted by a genotype. The discovered architecture is then trained from scratch to optimize the parameters during the *train* stage. The genotype and optimized parameters can be utilized to make class predictions in the *test* stage.

A. Differentiable Neural Architecture Search (DARTS)

In order to make the search process efficient, it is formulated as a bi-level optimization [7] over architectural weights (α) and network weights (w) as follows :

$$\begin{array}{ll} \min_{\alpha} & \mathcal{L}_{\text{val}}\left(w^{*}(\alpha), \alpha\right) \\ \text{s.t.} & w^{*}(\alpha)=\operatorname{argmin}_{w} \mathcal{L}_{\text{train}}(w, \alpha) \end{array}$$

where $\mathcal{L}_{\text{train}}$ and $\mathcal{L}_{\text{val}}$ are the training and validation losses respectively. The efficiency is ensured as the search space is made continuous rather than expensive exploration from a set of discrete architectures.

The search can be termed as "micro-search" for finding effective computational cells that can be stacked to form the full architecture (See FIG. 2). A cell can be seen as a directed acyclic graph consisting of an ordered sequence of 'P' nodes. Nodes are a set of latent feature maps. Each intermediate node x_j is represented as :

$$x_{j}=\sum_{i<j} f_{i, j}\left(x_{i}\right)$$

At each node, we compute a weighted summation of different operations $o(\cdot) \in \mathcal{O}$ (such as `conv 3×3, maxpool` etc) :

$$f_{i, j}\left(x_{i}\right)=\sum_{o \in \mathcal{O}_{i, j}} \frac{\exp \left(\alpha_{o}^{(i, j)}\right)}{\sum_{o^{\prime} \in \mathcal{O}} \exp \left(\alpha_{o^{\prime}}^{(i, j)}\right)} o\left(x_{i}\right)$$

The intention is to choose the best operation for connecting any given pair of nodes after a search. Each cell has two input nodes `c_{k-1}` and `c_{k-2}` (feature maps from previous cells) and the output of a cell (`c_{k}`) is given by concatenating the outputs of non-input cell nodes.

B. Partial Channel Connections

The idea of partial channel connections [8] is to utilize randomly sampled (1/K) of the total features channels for mixed operation computation in a node (See FIG. 3) :

$$f_{i, j}^{\mathrm{PC}}\left(\mathbf{x}_{i} ; \mathbf{S}_{i, j}\right)=\sum_{o \in \mathcal{O}} \frac{\exp \left\{\alpha_{i, i}^{o}\right\}}{\sum_{o^{\prime} \in \mathcal{O}} \exp \left\{\alpha_{i, j}^{o}\right\}} \cdot o\left(\mathbf{S}_{i, j} * \mathbf{x}_{i}\right) + \left(1-\mathbf{S}_{i, j}\right) * \mathbf{x}_{i}$$

where $\mathbf{S}_{i,j}$ is the sampling mask.

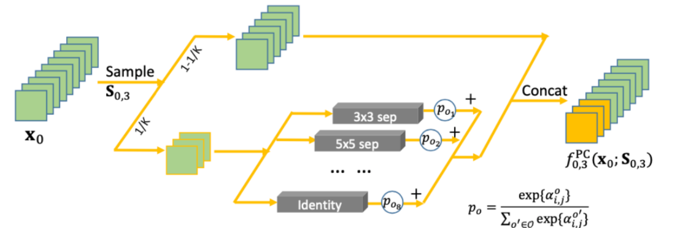

FIG. 3: Illustration of Partial Channel Connections [8]. We can see that the (1-1/K) of the total channels are bypassed without participating in the mixed sum.

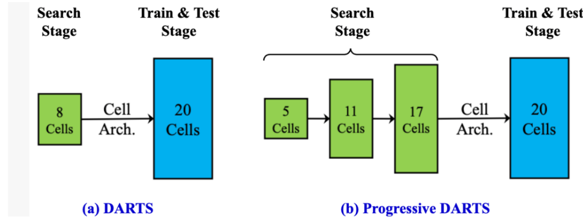

FIG. 4: Illustration of Progressive Search. Note that the difference in the number of cells used during the search stage is much more severe in DARTS as compared to the Progressive case.

Using a small fraction of channels reduces the memory requirement, by enabling usage of a larger batch size. However, the use of partial connections can cause the connectivity between nodes to fluctuate due to random channel sampling, so in order to avoid that edge normalization is performed. The incoming edges into any intermediate node are weighted by the softmax version of the shared edge parameters β :

$$\mathbf{x}_j^{\mathrm{PC}} = \sum_{i<j} \frac{\exp\left\{\beta_{i,j}\right\}}{\sum_{i'<j} \exp\left\{\beta_{i',j}\right\}} \cdot f_{i,j}\left(\mathbf{x}_i\right)$$

Note that $beta$'s can be optimized during the search stage along with the $alpha$'s. It is also observed [8] that the use of partial channel connections provides an indirect regularization for the algorithm to avoid choosing many parameter-free operations.

C. Progressive Search

It is observed that there exists a certain "depth gap" in the performance of the model during search and evaluation [9]. This might be because of the reason that certain operations are preferred more in deeper networks as compared to shallow networks. Empirical results suggest that shallow search (search using less number of cells) leads to shallow connections in the cells, leading to degraded transfer performance of discovered architectures on difficult datasets (e.g., ImageNet) compared to a proxy.

So, the idea is to search in a progressive manner by dividing it into different phases, increasing the depth in each phase gradually as illustrated in FIG. 4.

However, increasing the depth during search increases the computational overhead, so *search space approximation* is done, where as the different phases of the search proceed the cardinality of the operation set $\mathcal{O}$ is reduced. This is done by dropping out the operations with a lower architectural weight (α) in the previous phase.

It should be noted that at the start of each search, the weights are trained from scratch as the deeper networks may have altered preferences. Also, it is observed that a certain kind of over-fitting can happen during the search stage, where more skip-connections are preferred as initially, they propagate more consistent information and lead to rapid gradient descent. So to mitigate this, operation level dropout is employed for `skip_connects`, which acts as a *search space regularizer*. The dropout rate is initially higher to partially block the preference of skip connections, which is decayed with iterations to remove any bias against them.

D. Barlow Twins

The Barlow Twins [14] method was used for implementing self-supervision in terms of the architectural search. In this methodology, we try to learn the embedding or latent vectors invariant to distortion in the images. We do this by feeding two identical networks with two different distortions of an input image. We then compute the cross-correlation between the two outputs of the network as a mode of comparison. The elements of the cross-correlation matrix $\mathcal{C}$ is shown in equation below.

$$\mathcal{C}_{ij} \triangleq \frac{\sum_b z_{b,i}^A z_{b,j}^B}{\sqrt{\sum_b \left(z_{b,i}^A\right)^2} \sqrt{\left(\sum_b z_{b,j}^B\right)^2}}$$

where b is the index for batch samples, i, j are the index of the vector dimension of the networks' outputs and $z_{b,i}^A, z_{b,j}^B$ superscripts denotes the two different outputs (latent vectors) of the network. If the dimension of each output (latent vector) is n, $\mathcal{C}$ is a square matrix of size $n \times n$, and with values comprised between -1 (perfect anti-correlation) and 1 (perfect correlation)

We now make the network learn that both the images are similar by computing loss through the difference between the cross-correlation matrix and an identity matrix of size $n \times n$. The Barlow Twins loss function $\mathcal{L}_{\mathcal{BT}}$ is shown in equation below.

$$\mathcal{L}_{\mathcal{BT}} \triangleq \underbrace{\sum_i \left(1-\mathcal{C}_{ii}\right)^2}_{\text{Invariance term}} + \lambda \underbrace{\sum_i \sum_{j \neq i} \mathcal{C}_{ij}^2}_{\text{Redundancy Reduction term}}$$

where $\mathcal{C}$ is the cross-correlation matrix, i, j represent the index of the elements in the cross correlation matrix and λ is a positive constant trading off the importance of the two different terms of the loss. As shown in equation above, the first term represents the loss due to variance of the outputs and the second term represents the loss due to redundancy between the latent vectors.

By taking $\mathcal{L}_{\mathcal{BT}}$ as loss function the network learns to extract better latent feature vectors, ignore the distortions in the images and reduce the redundancy terms between the latent vectors (or embeddings). FIG. 5 shows the complete mechanism of the operation of Barlow Twins. This type of mechanism and loss function is implemented in Self-Supervised networks as explained in Section IV.

III. Experimental Setup

A. Datasets

We have used CIFAR-10 [15] and CIFAR-100 [15] datasets for architectural search and training purposes and another popular dataset such as Fashion MNIST [16] for transfer training purposes (architecture search was conducted on CIFAR-10,

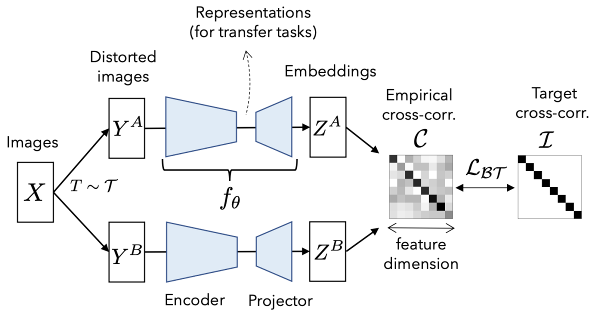

FIG. 5: The Barlow Twins Self-Supervision Mechanism[14]. It improves the quality of features extracted by reducing redundancy while being task-independent.

but training and testing were done with Fashion MNIST) in some stages.

B. Implementation details

Pytorch framework was used for architectural search, training, and testing purpose. We used Tesla V100 PCl-E 32GB GPU for Search, Train and Test Stages, NVIDIA RTX GeForce 2060 GPU, and Google Colab's Tesla GPU for analysis.

PyTorch's Loader and Data Transformer was used to pre-process images such as Random Cropping, resizing, converting to tensors and normalizing for search and train stages in the model. An additional Random Horizontal Flipping was used exclusively for search and train stages. Cutout Regularization in random image regions was also used during training stages for better performance. For the Barlow Twins case, random transformations like Random Crop, Random Horizontal Flip, Random Color jitter, Random Grayscale, Random distortion, and Random Erasing were applied with different probabilities for generating the two distorted images that should be fed into the two identical networks.

The Train-Validation Split considered 50K and 10K respectively for the searching as well as training purpose. For the purpose of searching, we use the Cross-Entropy Loss or a convex combination of Barlow Twins Loss and Cross-entropy loss based on the experiment's stage(See Section IV) and *SGD* optimizer (l.r.=0.025) with *weight decay* and *Cosine LR Scheduler* is considered as Optimizer. For search stages, we have considered a batch size of 256 images. For training stages, we have used Cross-entropy loss and *Adam* Optimizer (l.r.=6e-4, β_1=0.5, β_2=0.999) with weight decay as Optimizer. Note that check-pointing of models is done whenever there is a decrease in the validation loss after every epoch. Auxiliary Loss Towers are also considered in the training stages for boosting gradient flow during backpropagation.

IV. Approach Design

A. Design Considerations for Architecture Search

In this section, we can look at some of the aspects associated with the design of the search stage.

1) Operation Space:

We have taken a list of operations for the Neural Architecture Search. All the stages implemented in this work use this operation space to search the architecture. The operation space was kept small and simple for easy and faster searching of architecture, also keeping accuracy in view. The various operations used are `max_pool_3×3`, `avg_pool_3×3`, `skip_connect`, `sep_conv_3×3`, `sep_conv_5×5`, `dil_conv_3×3`, and `dil_conv_5×5`. The `sep_conv` here represents a separable convolution operation whereas `dil_conv` represents a diluted convolution operation. For reduction cells, the operations are strided, since we want to reduce the spatial dimensions.

2) Learning Algorithm:

We update the parameters by utilizing the *First Order Approximation* as mentioned in [7]. For the progressive case, the network parameters are optimized for all epochs in the search stage, whereas the architecture parameters are frozen for the first 15 epochs. Also, in this case, as shown in FIG. 4, we consider eight cells during the search and 20 cells during the evaluation stage.

For the non-progressive cases, we consider the following setup as given in [9] :

Phases	# Normal Cells	# Operations
I	5	8
II	11	5
III	17	3

TABLE I: Features of different search phases. The number of cells increase (due to Progressive nature) and number of operations decrease (due to search space approx.) as the phase number increases.

The network parameters are optimized for all epochs in this search phase, whereas the architecture parameters are frozen for the first ten epochs.

3) End of Search:

For choosing operations, on each edge (i, j), operation with the largest $\alpha^o_{i,j}$ value is preserved. For choosing the inter-node connections, we consider :

- For non-partial connection case, node x_j needs to be pick two links (highest weighted) from $\{\max_o \alpha^o_{0,j}, \ldots, \max_o \alpha^o_{j-1,j}\}$.
- For partial connection case, node x_j needs to be pick two links (highest weighted) from normalised values of the set $\{(\max_o \alpha^o_{0,j})\, \beta_{0,j}, \ldots, (\max_o \alpha^o_{j-1,j})\, \beta_{j-1,j}\}$.

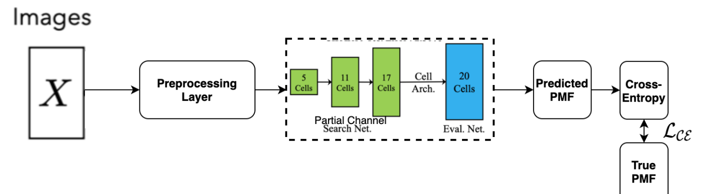

FIG. 6: Illustrative Overview of Vanilla PCP-DARTS. It combines the ideas of progressive search and partial channel connections.

B. Loss Function

1) Architecture Weights: We consider both self-supervised and fully supervised stages in our work (See Section II), for optimizing the architectural weights. For fully supervised stages, we consider the cross-entropy loss over the different classes of interest. For self-supervised stages, we consider the following mixture loss function :

$$\mathcal{L} = \text{ss_factor} \times \mathcal{L}_{\mathcal{BT}} + (1 - \text{ss_factor}) \times \mathcal{L}_{\mathcal{CE}}$$

where ss_factor decides the weight of self-supervision (ss_factor=1 implies fully self-supervised search).

2) Network Weights: We consider a supervised setup with cross-entropy loss to optimize the network weights for all the stages.

C. Design of Stages

*1) **Vanilla PCP-DARTS**:* The features of this stage are that it is a progressive, partial channel connection based search, and supervised loss is considered for both architecture search and network weight optimization.

*2) **SS PCP-DARTS**:* The features of this stage are that it is a progressive, partial channel connection based search. Supervised loss is considered for network weight optimization, and mixture loss, discussed in Section. IV.B, is used.

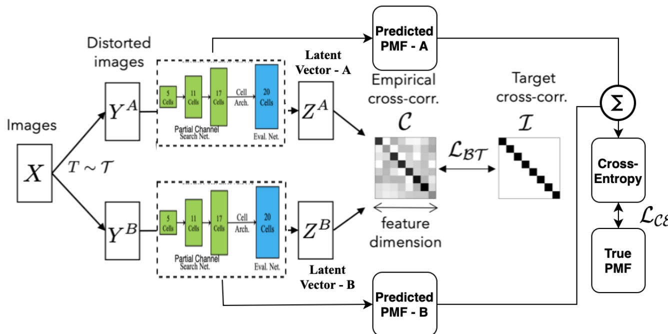

FIG. 7: Illustrative Overview of SS PCP-DARTS. It combines the idea of progressive search and partial channel connections in a self-supervised setting.

V. Results

In this section, we present the results of our approach.

A. Architectural Search Results

The search for architecture was done in a fully-supervised and self-supervised fashion as covered in Section IV and Section II in detail. Each stage has two types of cells - Normal and Reduction Cells. Both in normal as well as in reduction cells, we have two inputs, c_{k-2} and c_{k-1} and an output c_{k} and four nodes. These nodes are connected by edge functions, which are taken from the Neural Architectural Search Operation Space Section. IV. FIG. 8 and FIG. 9 shows the architecture of the final cells of some of the stages achieved from the architectural search. The architecture of all the Normal and Reduction Cells are covered in Supplementary Material.

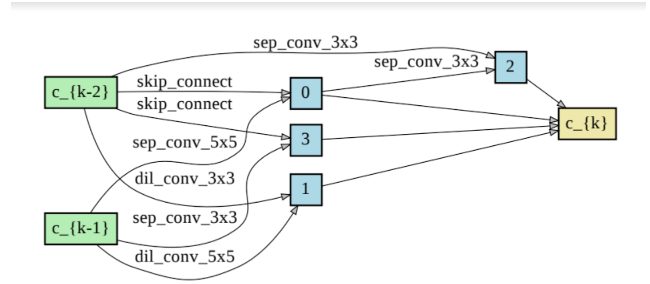

FIG. 8: Normal Cell Architecture for PC-DARTS. It is less deeper (3-layers) as compared to other algorithms. The unlabelled arrows denote an concatenation operation carried out at the c_{k} node.

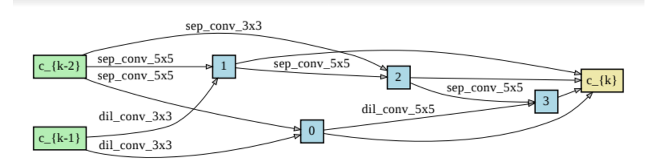

FIG. 9: Normal Cell Architecture for Vanilla PCP-DARTS. It is 4-layers deep, which is comparatively more than PC-DARTS.

B. Train Results

After optimizing the network architecture as part of Neural Architecture Search, we trained and tested the network using the CIFAR-10 test dataset. TABLE II shows a comparison of accuracy, search time and the number of parameters between various stages. Note that the network search time is over different GPUs with different batch sizes.

We also conducted some of the experiments using Vanilla PCP-DARTS, P-DARTS and SS PCP-DARTS with a ss_factor of 1 on the CIFAR-100 Dataset. TABLE III shows a comparison of accuracy and search time between these three models.

To show the usefulness of self-supervision, we also did a transfer train on the Fashion MNIST dataset. The architecture was searched using the CIFAR-10 dataset, but the training and testing were done on Fashion MNIST. The Vanilla PCP-DARTS gave a test accuracy of 95.97%, whereas the SS PCP-DARTS with ss_factor=1 gave a slightly higher test accuracy of 96.06%. In comparison to these, the PC-DARTS gave a lower test accuracy of 95.65%.

VI. Analysis

In this section, we have analyzed our results from various angles and explained the reason for some of the occurrences. A particular Adversarial Attack Analysis was carried out to understand Self-Supervision better, as presented in this section.

A. General Analysis

Based on the results presented in Section V, we have analyzed the performance and general trends followed by the network during the search, train and test stages. Some of the common observations and their possible reason is listed below

- PCP-DARTS is comparable to PC-DARTS and P-DARTS when we take an overall performance like search time

Various Architectures	Test Err. (in %)	Params (in $\mathbf{M}$)	Search Cost (in $\mathbf{GPU}$-days)	Search Method
DenseNet-BC [17]	3.46	25.6	–	Manual
NASNet-A [18]	2.65	3.3	1800	Reinforcement Learning
AmoebaNet-B [19]	2.55	2.8	3150	Evolutionary
Hireachical Evolution [20]	3.75	15.7	300	Evolutionary
NAONet-WS [21]	3.53	3.1	0.4	Neural Arch. Optimization
DARTS [7]	3.00	3.3	0.4	Gradient-based
SNAS [22]	2.85	2.8	1.5	Gradient-based
P-DARTS† [9]	2.71	3.4	0.2391	Gradient-based
PC-DARTS† [8]	2.57	3.5	0.1624	Gradient-based
Vanilla PCP-DARTS	3.12	4.5	0.1631	Gradient-based
SS PCP-DARTS (ssf=0.25)	3.91	3.9	0.2201	Gradient-based
SS PCP-DARTS (ssf=0.5)	3.57	3.7	0.1535	Gradient-based
SS PCP-DARTS (ssf=0.75)	2.99	3.7	0.1989	Gradient-based
SS PCP-DARTS (ssf=1)	3.28	4.2	0.1948	Gradient-based

TABLE II: Comparison of different architectures on CIFAR-10. † - We ran the code released by authors for fair comparison

	PC-DARTS	**Vanilla PCP-DARTS**	**P-DARTS**	**SS PCP-DARTS**
ss_factor	-	-	-	1
Test Accuracy		81.92	82.48	74.65
Network Search Days	0.1	0.2647	0.3	0.2464

TABLE III: Comparison of various algorithms on CIFAR-100 Dataset

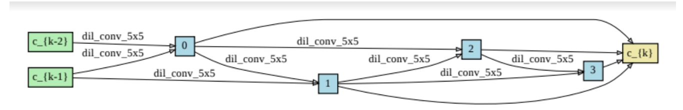

FIG. 11: Reduction Cell Architecture for SS PCP-DARTS with ss_factor=0.75. This is the deepest cell network that can be formed using 4-nodes and Self-Supervision triggers it.

and accuracy into account which can be observed from TABLE II and TABLE III. PCP-DARTS takes almost comparable time to PC-DARTS but gives an accuracy similar to nearby P-DARTS. This performance combines both the concepts of the partial channel and progressive search in the network.

- PC-DARTS prefers parameter-less operations. It is a observation that parameter-less operations like `max_pool`, `avg_pool` and `skip_connect` are more in final architectures of PC-DARTS than in PCP-DARTS. This is evident from FIG. 10 and FIG. 11. We suspect that this occurrence maybe because of the fundamental property of PC-DARTS to exploit inter-node interactions.

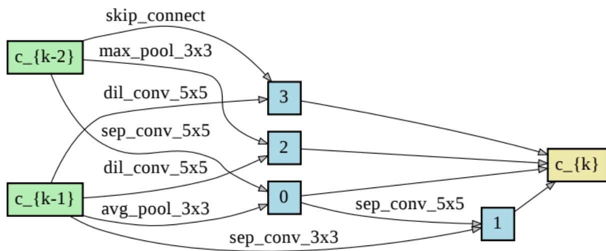

FIG. 10: Reduction Cell Architecture for PC-DARTS. It is 3-layers deep cell containing more `pool`, `skip_connect` and `sep_conv` operations.

- PCP-DARTS has more representational power than PC-DARTS. It is an observation that PCP-DARTS contains more number of `sep_conv` operations whereas PC-DARTS contains more number of `dil_conv` with pooling layers succeeding those often. This is also evident from FIG. 10 and FIG. 11.This may be considered an attempt by both the networks to extend their receptive fields except for the difference that PC-DARTS uses two operations whereas PCP-DARTS only uses one approximately equivalent operation. This reduction in operation gives the PCP-DARTS more representation power. Such behaviour may be attributed to the inherent nature of PCP-DARTS to increase the depth and reduce the number of operations as the network search progresses.
- Vanilla PCP-DARTS has better depth and lesser number of skip-connections than PC-DARTS as can be observed from FIG. 8 and FIG. 9. This kind of affiliation may be attributed to the nature of the network architecture. While PC-DARTS was designed to prefer more skip-connections to capture inter-node interactions, the PCP-DARTS was designed to better the depth of the networks as the architecture search progresses.
- Self-Supervised Learning may help in increasing the depth of networks. PCP-DARTS show that whenever we use Self-Supervision for architectural search, the architecture shows more depth. We can observe this from FIG. 9 and FIG. 11. This observation may be possible because of the improvement of diversity, richness and quality of features with depth, which Barlow Twins Loss ensures.
- Based on the results shown for CIFAR-10 in Section V, we can observe that using both supervised and self-supervised learning for architecture search gives lesser accuracy than using only one of them. We suspect that the reason for this may be because of the possible upward shift in minima as a result of the convex combination of two-loss functions that have different minima locations

in the network search space.

FIG. 12: Illustration of FGSM method for efficiently generating adversarial examples. Note that, we are essentially try to perturb the image along the direction of most likely misclassification with a magnitude ϵ.

B. Adversarial Attack Analysis

We also perform adversarial attack analysis on our approaches to understand the robustness of the different stages. We generate adversarial images X_{adv}, which are maliciously designed to be perceptually indistinguishable from original input X but is misclassified by the model. To perform this, we use the Fast Gradient Method (FGSM) as described in [23] (See FIG. 12).

$$X^{adv} = X + \epsilon \operatorname{sign}\left(\nabla_X J\left(X, Y_{\text{true}}\right)\right)$$

where ϵ is the magnitude of adversarial perturbation and $\nabla_X J\left(X, Y_{\text{true}}\right)$ is the gradient of loss function w.r.t to clean input X.

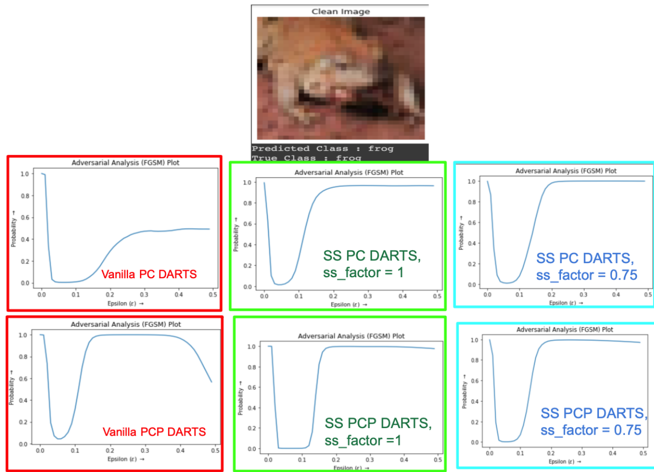

FIG. 13: Probability of True Class vs Magnitude of Adversarial perturbation (ϵ). Note that, all the architectures found in each stage predict the same class, but the variation of probability with ϵ is interestingly distinct.

We also plot the curves for probability against ϵ for all the stages on some input samples. One of the examples is shown in FIG. 13.

From these plots, we can try to justify our observations by making the following statements:

- For most of the epsilon values considered, we have a higher probability for the correct class when self-supervision is incorporated into the loss function of the architecture search. This may mean that some amount of Barlow Twin Self Supervision makes the found architectures slightly robust to adversarial attacks like FGSM.
- We can observe a dip in the plot, of prob vs epsilon, for a small range of epsilon values. For the case of self-supervision, we can consider that because of the non-task specific loss function, the mixture loss function may contain multiple minima for the same class compared to the vanilla (Fully-supervised) loss function.
- Higher probability predictions for the chosen predicted class as seen in FIG. 13 can be interpreted as self-supervision might be forcing the algorithm to choose operations that extract higher quality, differentiating features among classes. There might be some decorrelation of decision regions to some extent.

VII. Conclusion and Future Work

This work presents architecture search approaches utilizing a differentiable formulation and various concepts of self-supervision focused on improving efficiency while maintaining their efficacy. The results provided for various experiments and the analyses of these results reveal interesting characteristics about our approaches and open new avenues for future work. Exploring and analyzing the incorporation of self-supervision for classification, transfer performance of different tasks, and more complex datasets can produce exciting results and insights. Another promising direction for future work can be to make the capsule layers [24] [25] [6] less computational expensive so that they can be incorporated in the operation space.

Supplementary Information

Code and supplementary material is available upon request.

Bibliography

[1] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, “Regularized evolution for image classifier architecture search,” in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4780–4789.

[2] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” *arXiv preprint arXiv:1611.01578*, 2016.

[3] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, “Efficient neural architecture search via parameters sharing,” in *International Conference on Machine Learning*. PMLR, 2018, pp. 4095–4104.

[4] Y. Zhao, L. Wang, Y. Tian, R. Fonseca, and T. Guo, “Few-shot neural architecture search,” in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 707–12 718.

[5] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[6] A. Marchisio, A. Massa, V. Mrazek, B. Bussolino, M. Martina, and M. Shafique, “Nascaps: A framework for neural architecture search to optimize the accuracy and hardware efficiency of convolutional capsule networks,”

in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.
[7] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *International Conference on Learning Representations*, 2018.
[8] Y. Xu, L. Xie, W. Dai, X. Zhang, X. Chen, G.-J. Qi, H. Xiong, and Q. Tian, "Partially-connected neural architecture search for reduced computational redundancy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, pp. 2953–2970, 2021.
[9] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1294–1303.
[10] X. Chu, T. Zhou, B. Zhang, and J. Li, "Fair darts: Eliminating unfair advantages in differentiable architecture search," in *European conference on computer vision*. Springer, 2020, pp. 465–480.
[11] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.
[12] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar *et al.*, "Bootstrap your own latent-a new approach to self-supervised learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 271–21 284, 2020.
[13] I. Misra and L. v. d. Maaten, "Self-supervised learning of pretext-invariant representations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6707–6717.
[14] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow twins: Self-supervised learning via redundancy reduction," in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 310–12 320.
[15] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 and cifar-100 datasets," *URL: https://www. cs. toronto. edu/kriz/cifar. html*, vol. 6, no. 1, p. 1, 2009.
[16] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017.
[17] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.
[18] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.
[19] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4780–4789.
[20] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *International Conference on Learning Representations*, 2018.
[21] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," *Advances in neural information processing systems*, vol. 31, 2018.
[22] S. Xie, H. Zheng, C. Liu, and L. Lin, "Snas: stochastic neural architecture search," in *International Conference on Learning Representations*, 2018.
[23] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *stat*, vol. 1050, p. 20, 2015.
[24] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *Advances in neural information processing systems*, vol. 30, 2017.
[25] G. E. Hinton, S. Sabour, and N. Frosst, "Matrix capsules with em routing," in *International conference on learning representations*, 2018.