Neural Architecture Search of SPD Manifold Networks

Rhea Sanjay Sukthanker[1], **Zhiwu Huang**[1], **Suryansh Kumar**[1],
Erik Goron Endsjo[1], **Yan Wu**[1], **Luc Van Gool**[1,2]
[1]Computer Vision Lab, ETH Zürich, Switzerland
[2]PSI, KU Leuven, Belgium
rhea.sukthanker@inf.ethz.ch, {eendsjo, wuyan}@student.ethz.ch,
{zhiwu.huang, sukumar, vangool}@vision.ee.ethz.ch

Abstract

In this paper, we propose a new neural architecture search (NAS) problem of Symmetric Positive Definite (SPD) manifold networks, aiming to automate the design of SPD neural architectures. To address this problem, we first introduce a geometrically rich and diverse SPD neural architecture search space for an efficient SPD cell design. Further, we model our new NAS problem with a one-shot training process of a single supernet. Based on the supernet modeling, we exploit a differentiable NAS algorithm on our relaxed continuous search space for SPD neural architecture search. Statistical evaluation of our method on drone, action, and emotion recognition tasks mostly provides better results than the state-of-the-art SPD networks and traditional NAS algorithms. Empirical results show that our algorithm excels in discovering better performing SPD network design and provides models that are more than three times lighter than searched by the state-of-the-art NAS algorithms.

1 Introduction

Designing an efficient neural network architecture for a given application generally requires a significant amount of time, effort, and domain expertise. To mitigate this issue, there have been emerging a number of neural architecture search (NAS) algorithms to automate the design process of neural architectures [Zoph and Le, 2016; Liu *et al.*, 2017; Liu *et al.*, 2018a; Liu *et al.*, 2018b; Real *et al.*, 2019]. However, researchers have barely proposed NAS algorithms to optimize those neural network architecture designs that deal with non-Euclidean data representations and the corresponding set of operations —to the best of our knowledge.

It is well-known that Symmetric Positive Definite (SPD) manifold-valued data representation has shown overwhelming accomplishments in many real-world applications such as magnetic resonance imaging analysis [Pennec *et al.*, 2006], pedestrian detection [Tuzel *et al.*, 2008], human action recognition [Huang and Van Gool, 2017], hand gesture recognition [Nguyen *et al.*, 2019], etc. Also, in applications like diffusion tensor imaging of the brain, drone imaging, samples are collected directly as SPD's. As a result, neural network usage based on Euclidean data representation becomes inefficient for those applications. Consequently, this has led to the development of SPD neural networks (SPDNets) [Huang and Van Gool, 2017] to improve these research areas further. However, the SPDNets are handcrafted, and therefore, the operations or the parameters defined for these networks generally change as per the application. This motivates us to propose a new NAS problem of SPD manifold networks. A solution to this problem can reduce unwanted efforts in SPDNet architecture design. Compared to the traditional NAS problem, our NAS problem requires searching for a new basic neural computation cell modeled by a specific directed acyclic graph (DAG), where each node indicates a latent SPD representation, and each edge corresponds to a SPD candidate operation.

To solve the suggested NAS problem, we exploit a new supernet search strategy that models the architecture search problem as a one-shot training process of a supernet comprised of a mixture of SPD neural architectures. The supernet modeling enables a differential architecture search on a continuous relaxation of SPD neural architecture search space. We evaluate the proposed NAS method on three benchmark datasets, showing the automatically searched SPD neural architectures perform better than the state-of-the-art handcrafted SPDNets for radar, facial emotion, and skeletal action recognition. In summary, our work makes the following contributions:

- We introduce a brand-new NAS problem of SPD manifold networks that opens up a novel research problem at the intersection of automated machine learning and SPD manifold learning. For this problem, we exploit a new search space and a new supernet modeling, both of which respect the particular Riemannian geometry of SPD manifold networks.
- We propose a sparsemax-based NAS technique to optimize sparsely mixed architecture candidates during the search stage. This reduces the discrepancy between the search on mixed candidates and the training on one single candidate. To optimize the new NAS objective, we exploit a new bi-level algorithm with manifold- and convex-based updates.
- Evaluation on three benchmark datasets shows that our searched SPD neural architectures can outperform handcrafted SPDNets [Huang and Van Gool, 2017; Brooks *et al.*, 2019; Chakraborty *et al.*, 2020] and the state-of-the-art NAS methods [Liu *et al.*, 2018b; Chu *et al.*, 2020]. Our searched architecture is more than three times lighter than those searched using traditional NAS algorithms.

2 Background

As our work is directed towards solving a new NAS problem, we confine our discussion to the work that has greatly influenced our method *i.e.*, one-shot NAS methods and SPD networks. There are mainly two types of one-shot NAS methods based on the architecture modeling [Elsken *et al.*, 2018] *(i) parameterized architecture* [Liu *et al.*, 2018b; Zheng *et al.*, 2019; Wu *et al.*, 2019; Chu *et al.*, 2020], and *(ii) sampled architecture* [Deb *et al.*, 2002; Chu *et al.*, 2019]. In this paper, we adhere to the parametric modeling due to its promising results on conventional neural architectures. A majority of the previous work on NAS with continuous search space fine-tunes the explicit feature of specific architectures [Saxena and Verbeek, 2016; Veniat and Denoyer, 2018; Ahmed and Torresani, 2017; Shin *et al.*, 2018]. On the contrary, [Liu *et al.*, 2018b; Liang *et al.*, 2019; Zhou *et al.*, 2019; Zhang *et al.*, 2020; Wu *et al.*, 2021; Chu *et al.*, 2020] provides architectural diversity for NAS with highly competitive performances. The other part of our work focuses on SPD network architectures. There exist algorithms to develop handcrafted SPDNets like [Huang and Van Gool, 2017; Brooks *et al.*, 2019; Chakraborty *et al.*, 2020]. To automate the process of SPD network design, in this work, we choose the most promising approaches from the field of NAS [Liu *et al.*, 2018b] and SPD networks [Huang and Van Gool, 2017]).

Next, we summarize some of the essential notions of Riemannian geometry of SPD manifolds, followed by introducing some basic SPDNet operations and layers. As some of the introduced operations and layers have been well-studied by the existing literature, we apply them directly to define our SPD neural architectures' search space.

Representation and Operation. We denote $n \times n$ real SPD as $\boldsymbol{X} \in \mathcal{S}_{++}^n$. A real SPD matrix $\boldsymbol{X} \in \mathcal{S}_{++}^n$ satisfies the property that for any non-zero $z \in \mathbb{R}^n$, $z^T \boldsymbol{X} z > 0$. We denote $\mathcal{T}_{\boldsymbol{X}}\mathcal{M}$ as the tangent space of the manifold $\mathcal{M}$ at $\boldsymbol{X} \in \mathcal{S}_{++}^n$ and log corresponds to matrix logarithm.

Let $\boldsymbol{X}_1, \boldsymbol{X}_2$ be any two points on the SPD manifold then the distance between them is given by $\delta_{\mathcal{M}}(\boldsymbol{X}_1, \boldsymbol{X}_2) = 0.5\|\log(\boldsymbol{X}_1^{-\frac{1}{2}}\boldsymbol{X}_2\boldsymbol{X}_1^{-\frac{1}{2}})\|_F$. There are other efficient methods to compute distance between two points on the SPD manifold, however, their discussion is beyond the scope of our work [Gao *et al.*, 2019; Dong *et al.*, 2017]. Other property of the Riemannian manifold of our interest is local diffeomorphism of geodesics which is a one-to-one mapping from the point on the tangent space of the manifold to the manifold [Pennec, 2020; Lackenby, 2020]. To define such notions, let $\boldsymbol{X} \in \mathcal{S}_{++}^n$ be the base point and, $\boldsymbol{Y} \in \mathcal{T}_{\boldsymbol{X}}\mathcal{S}_{++}^n$, then $\boldsymbol{Y} \in \mathcal{T}_{\boldsymbol{X}}\mathcal{S}_{++}^n$ is associated to a point on the SPD manifold [Pennec, 2020] by the map

$$\exp_{\boldsymbol{X}}(\boldsymbol{Y}) = \boldsymbol{X}^{\frac{1}{2}} \exp(\boldsymbol{X}^{-\frac{1}{2}}\boldsymbol{Y}\boldsymbol{X}^{-\frac{1}{2}})\boldsymbol{X}^{\frac{1}{2}} \in \mathcal{S}_{++}^n, \ \forall \boldsymbol{Y} \in \mathcal{T}_{\boldsymbol{X}}. \tag{1}$$

Likewise,

$$\log_{\boldsymbol{X}}(\boldsymbol{Z}) = \boldsymbol{X}^{\frac{1}{2}} \log(\boldsymbol{X}^{-\frac{1}{2}}\boldsymbol{Z}\boldsymbol{X}^{-\frac{1}{2}})\boldsymbol{X}^{\frac{1}{2}} \in \mathcal{T}_{\boldsymbol{X}}, \ \forall \boldsymbol{Z} \in \mathcal{S}_{++}^n \tag{2}$$

is defined as its inverse map.

1) Basic operations of SPD Network. Operations such as mean centralization, normalization, and adding bias to a batch of data are inherent performance booster for neural networks. Accordingly, works like [Brooks *et al.*, 2019; Chakraborty, 2020] use the notion of such operations for the manifold-valued data to define analogous operations on manifolds. Below we introduce them following [Brooks *et al.*, 2019] work.

- *Batch mean, centering and bias*: Given a batch of N SPD matrices $\{\boldsymbol{X}_i\}_{i=1}^N$, we can compute its Riemannian barycenter ($\mathscr{B}$) as:

$$\mathscr{B} = \underset{\boldsymbol{X}_\mu \in \mathcal{S}_{++}^n}{\operatorname{argmin}} \sum_{i=1}^{N} \delta_{\mathcal{M}}^2(\boldsymbol{X}_i, \boldsymbol{X}_\mu). \tag{3}$$

It is sometimes referred to as Fréchet mean [Moakher, 2005; Bhatia and Holbrook, 2006]. This definition can be extended to compute the weighted Riemannian Barycenter also known as weighted Fréchet Mean (wFM)[1]

$$\mathscr{B} = \underset{\boldsymbol{X}_\mu \in \mathcal{S}_{++}^n}{\operatorname{argmin}} \sum_{i=1}^{N} w_i \delta_{\mathcal{M}}^2(\boldsymbol{X}_i, \boldsymbol{X}_\mu); \ \text{ s.t. } w_i \geq 0, \ \sum_{i=1}^{N} w_i = 1. \tag{4}$$

Eq:(4) can be estimated using Karcher flow [Karcher, 1977; Bonnabel, 2013; Brooks *et al.*, 2019] or recursive geodesic mean [Cheng *et al.*, 2016; Chakraborty *et al.*, 2020].

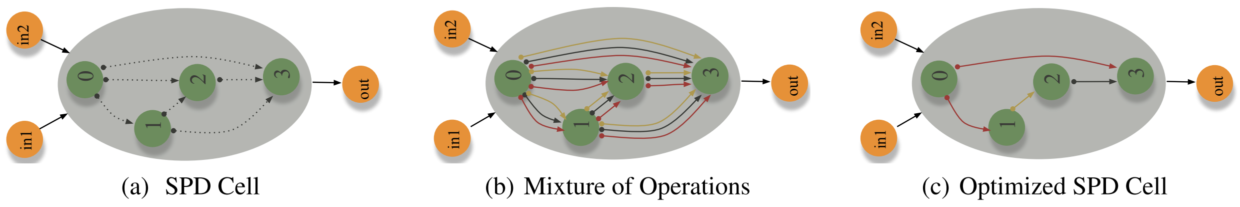
2) Basic layers of SPD Network. Analogous to standard convolutional networks (ConvNets), [Huang and Van Gool, 2017; Brooks *et al.*, 2019; Chakraborty *et al.*, 2020] designed SPD layers to perform operations that respect SPD manifold constraints. Assuming $\boldsymbol{X}_{k-1} \in \mathcal{S}_{++}^n$ be the input SPD matix to the k^{th} layer, the SPD layers are defined as follows:

- *BiMap layer*: This layer corresponds to a dense layer for SPD data. It reduces the dimension of a input SPD matrix via a transformation matrix $\boldsymbol{W}_k$ as $\boldsymbol{X}_k = \boldsymbol{W}_k\boldsymbol{X}_{k-1}\boldsymbol{W}_k^T$. To ensure $\boldsymbol{X}_k$ to be SPD, $\boldsymbol{W}_k$ is commonly required to be of full row-rank through an orthogonality constraint on it.
- *Batch normalization layer:* To perform batch normalization after each BiMap layer, we first compute the Riemannian barycenter of one batch of SPD samples followed by a running mean update step, which is Riemannian weighted average between the batch mean and the current running mean, with the weights $(1 - \theta)$ and (θ) respectively. With the mean, we centralize and add a bias to each SPD sample in the batch using Eq:(5), where $\mathscr{P}$ is the notation used for parallel transport and I is the identity matrix:

$$\begin{aligned} &\text{Centering the } \mathscr{B} : \boldsymbol{X}_i^c = \mathscr{P}_{\mathscr{B}\to I}(\boldsymbol{X}_i) = \mathscr{B}^{-\frac{1}{2}}\boldsymbol{X}_i\mathscr{B}^{-\frac{1}{2}}, \\ &\text{Bias towards } \boldsymbol{G} : \boldsymbol{X}_i^b = \mathscr{P}_{I\to \boldsymbol{G}}(\boldsymbol{X}_i^c) = \boldsymbol{G}^{\frac{1}{2}}\boldsymbol{X}_i^c\boldsymbol{G}^{\frac{1}{2}}. \end{aligned} \tag{5}$$

- *ReEig layer*: The ReEig layer is analogous to ReLU layers presented in the classical ConvNets. It aims to introduce non-linearity to SPD networks. The ReEig for the k^{th} layer is defined as: $\boldsymbol{X}_k = \boldsymbol{U}_{k-1}\max(\epsilon\boldsymbol{I}, \Sigma_{k-1})\boldsymbol{U}_{k-1}^T$ where, $\boldsymbol{X}_{k-1} = \boldsymbol{U}_{k-1}\Sigma_{k-1}\boldsymbol{U}_{k-1}^T$, $\boldsymbol{I}$ is the identity matrix, and $\epsilon > 0$ is a rectification threshold value. U_{k-1}, Σ_{k-1} are the orthonormal matrix and singular-value matrix respectively, and obtained via matrix factorization of $\boldsymbol{X}_{k-1}$.

[1]Following [Tuzel *et al.*, 2008; Brooks *et al.*, 2019], we focus on the estimate of wFM using Karcher flow.



(a) SPD Cell (b) Mixture of Operations (c) Optimized SPD Cell

Figure 1: (a) A SPD cell structure composed of 4 SPD nodes, 2 input node and 1 output node. Initially the edges are unknown (b) Mixture of candidate SPD operations between nodes (c) Optimal cell architecture obtained after solving the bi-level optimization formulation.

- *LogEig layer*: To map the manifold representation of SPDs to a flat space so that a Euclidean operation can be performed, LogEig layer is introduced. The LogEig layer is defined as: $\boldsymbol{X}_k = \boldsymbol{U}_{k-1}\log(\Sigma_{k-1})\boldsymbol{U}_{k-1}^T$ where, $\boldsymbol{X}_{k-1} = \boldsymbol{U}_{k-1}\Sigma_{k-1}\boldsymbol{U}_{k-1}^T$. The LogEig layer is used with fully connected layers to solve tasks with SPD representation.
- *ExpEig layer*: The ExpEig layer is designed to map the corresponding SPD representation from a flat space back to an SPD manifold space. It is defined as $\boldsymbol{X}_k = \boldsymbol{U}_{k-1}\exp(\Sigma_{k-1})\boldsymbol{U}_{k-1}^T$, where $\boldsymbol{X}_{k-1} = \boldsymbol{U}_{k-1}\Sigma_{k-1}\boldsymbol{U}_{k-1}^T$.
- *Weighted Riemannian pooling layer*: It uses wFM to perform the weighted pooling on SPD samples. To compute wFM, we follow [Brooks *et al.*, 2019] to use Karcher flow [Karcher, 1977] due to its simplicity and wide usage.

3 Proposed Method

Firstly, we introduce a new definition of the computation cell for our task. In contrast to traditional NAS computation cell design [Liu *et al.*, 2018b; Chu *et al.*, 2020], our computational cell (called SPD cell), incorporates the notion of SPD manifold geometry while performing any SPD operations. Similar to the basic NAS cell design, our SPD cell can either be a **normal cell** that returns SPD feature maps of the same width and height or, a **reduction cell** in which the SPD feature maps are reduced by a certain factor in width and height. Secondly, solving our new NAS problem will require an appropriate and diverse SPD *search space* that can help NAS method to optimize for an effective SPD cell, which can then be stacked and trained to build an efficient SPD neural network architecture.

Concretely, an SPD cell is modeled by a directed asyclic graph (DAG) which is composed of nodes and edges. In our DAG each *node* indicates an *latent representation* of the SPD manifold-valued data *i.e.*, an intermediate SPD feature map, and each *edge* corresponds to a *valid candidate operation* on the SPD manifold (see Fig.1(a)). Each edge of a SPD cell is associated with a set of candidate SPD manifold operations ($\mathscr{O}_{\mathcal{M}}$) that transforms the SPD-valued latent representation from the source node (say $\boldsymbol{X}_{\mathcal{M}}^{(i)}$) to the target node (say $\boldsymbol{X}_{\mathcal{M}}^{(j)}$). We define the intermediate transformation between the nodes in our SPD cell as:

$$\boldsymbol{X}_{\mathcal{M}}^{(j)} = \underset{\boldsymbol{X}_{\mathcal{M}}^{(j)}}{\operatorname{argmin}} \sum_{i<j} \delta_{\mathcal{M}}^2\left(\mathscr{O}_{\mathcal{M}}^{(i,j)}\left(\boldsymbol{X}_{\mathcal{M}}^{(i)}\right), \boldsymbol{X}_{\mathcal{M}}^{(j)}\right), \quad (6)$$

where $\delta_{\mathcal{M}}$ denotes the geodesic distance. This transformation result corresponds to the unweighted Fréchet mean of the operations based on the predecessors, such that the mixture of all operations still reside on SPD manifolds. Note that our definition of SPD cell ensures that each computational graph preserves the appropriate geometric structure of the SPD manifold. Equipped with the notion of SPD cell and its intermediate transformation, we are ready to propose our search space (§3.1) followed by the solution to our new NAS problem (§3.2) and its results (§4).

3.1 Search Space

Our search space consists of a set of valid SPD network operations. It includes some existing SPD operations, *e.g.*, BiMap, Batch Normalization, ReEig, LogEig, ExpEig, and weighted Riemannian pooling layers, all of which are introduced in Sec.2. Though existing works have well explored those individual operations (e.g., BiMap, LogEig, ExpEig), their different combinations are still understudied and are essential to enrich our search space. To enhance the search space, following traditional NAS methods [Liu *et al.*, 2018b; Gong *et al.*, 2019], we apply the SPD batch normalization to every SPD convolution operation (*i.e.*, BiMap), and design three variants of convolution blocks including the one without activation (*i.e.*, ReEig), the one using post-activation and the one using pre-activation (see Table 1). In addition, we introduce **five new operations** analogous to [Liu *et al.*, 2018b] to enrich the search space in the context of SPD networks. These are, skip normal, none normal, average pooling, max pooling, and skip reduced. The effect of such diverse operation choices has not been fully explored for SPD networks. All the candidate operations are illustrated in Table (1), and their definitions are as follows:

(a) Skip normal: It preserves the input representation and is similar to regular skip connections. **(b) None normal**: It corresponds to the operation that returns identity as the output i.e, the notion of zero in the SPD space. **(c) Max pooling**: Given a set of SPD matrices, the max pooling operation first projects these samples to a flat space via a LogEig operation, where a standard max pooling operation is performed. Finally, an ExpEig operation is used to map the samples back to the SPD manifold. **(d) Average pooling**: Similar to Max pooling, the average pooling operation first projects the samples to the flat space using a LogEig operation, where a standard average pooling is employed. To map the sample back to the SPD manifold, an ExpEig operation is used. **(e) Skip reduced**: It is similar to 'skip_normal' but in contrast, it decomposes the input into small matrices to reduces the inter-dependency between channels. Our definition of the reduce operation is in line with the work of [Liu *et al.*, 2018b].

Operation	Definition	Operation	Definition
BiMap_0	{BiMap, Batch Normalization}	WeightedReimannPooling_normal	{wFM on SPD multiple times}
BiMap_1	{BiMap,Batch Normalization, ReEig}	AveragePooling_reduced	{LogEig, AveragePooling, ExpEig}
BiMap_2	{ReEig, BiMap, Batch Normalization}	MaxPooling_reduced	{LogEig, MaxPooling, ExpEig}
Skip_normal	{Output same as input}	Skip_reduced = $\{\mathrm{C_{in}} = \mathrm{BiMap}(\mathrm{X_{in}}), [\mathrm{U_{in}}, \mathrm{D_{in}}, \sim] = \mathrm{svd}(\mathrm{C_{in}}); \mathrm{in} = 1, 2\}$,	
None_normal	{Return identity matrix}	$\mathrm{C_{out}} = \mathrm{U_b D_b U_b^T}$, where, $\mathrm{U_b} = \mathrm{diag}(\mathrm{U_1}, \mathrm{U_2})$ and $\mathrm{D_b} = \mathrm{diag}(\mathrm{D_1}, \mathrm{D_2})$	

Table 1: Search space for the proposed SPD architecture search method.

The newly introduced operations allow us to generate a more diverse discrete search space. As presented in Table (2), the randomly selected architecture (generally consisting of the newly introduced SPD operations) shows some improvement over the handcrafted SPDNets, which only contain conventional SPD operations. This establishes the effectiveness of the introduced rich search space. For more details about the proposed SPD operations, please refer to our Appendix[2].

3.2 Supernet Search

To solve the new NAS problem, one of the most promising methodologies is supernet modeling. In general, the supernet method models the architecture search problem as a one-shot training process of a single supernet that consists of all candidate architectures. Based on the supernet modeling, we search for the optimal SPD neural architecture by parameterizing the design of the supernet architectures, which is based on the continuous relaxation of the SPD neural architecture representation. Such an approach allows for an efficient search of architecture using the gradient descent approach. Next, we introduce our supernet search method, followed by a solution to our proposed bi-level optimization problem. Fig.1(b) and Fig.1(c) illustrate an overview of our proposed method.

To search for an optimal SPD architecture parameterized by α, we optimize the over parameterized supernet. In essence, it stacks the basic computation cells with the parameterized candidate operations from our search space in a one-shot search manner. The contribution of specific subnets to the supernet helps in deriving the optimal architecture from the supernet. Since the proposed operation search space is discrete in nature, we relax the explicit choice of an operation to make the search space continuous. To do so, we use wFM over all possible candidate operations. Mathematically,

$$\bar{\mathscr{O}}_{\mathcal{M}}(\boldsymbol{X}_{\mathcal{M}}) = \underset{\boldsymbol{X}^{\mu}_{\mathcal{M}}}{\operatorname{argmin}} \sum_{k=1}^{N_e} \tilde{\alpha}^k \delta^2_{\mathcal{M}} \Big(\mathscr{O}^{(k)}_{\mathcal{M}}(\boldsymbol{X}_{\mathcal{M}}), \boldsymbol{X}^{\mu}_{\mathcal{M}} \Big); \quad \text{subject to: } \mathbf{1}^T \tilde{\alpha} = 1,\ 0 \leq \tilde{\alpha} \leq 1, \tag{7}$$

where $\mathscr{O}^k_{\mathcal{M}}$ is the k^{th} candidate operation between nodes, $\boldsymbol{X}_\mu$ is the intermediate SPD manifold mean (Eq.4) and, N_e denotes number of edges. We can compute wFM solution either using Karcher flow [Karcher, 1977] or recursive geodesic mean [Chakraborty *et al.*, 2020] algorithm. Nonetheless, we adhere to Karcher flow algorithm as it is widely used to calculate wFM. To impose the explicit convex constraint on $\tilde{\alpha}$, we project the solution onto the probability simplex as

$$\underset{\alpha}{\operatorname{minimize}}\ \|\alpha - \tilde{\alpha}\|_2^2; \ \text{subject to: } \mathbf{1}^T\alpha = 1,\ 0 \leq \alpha \leq 1. \tag{8}$$

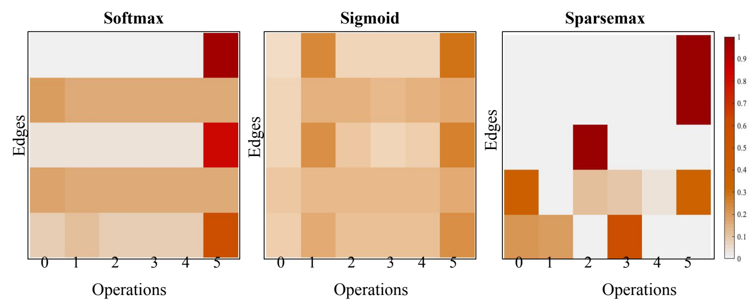
[2]Appendix can be found at https://arxiv.org/pdf/2010.14535.pdf

Eq:(8) enforces the explicit constraint on the weights to supply α for our task and can easily be added as a convex layer in the framework [Agrawal *et al.*, 2019]. This projection is likely to reach the boundary of the simplex, in which case α becomes sparse [Martins and Astudillo, 2016]. Optionally, softmax, sigmoid and other regularization methods can be employed to satisfy the convex constraint. However, [Chu *et al.*, 2020] has observed that the use of softmax can cause performance collapse and may lead to aggregation of skip connections. While [Chu *et al.*, 2020] suggested sigmoid can overcome the unfairness problem with softmax, it may output smoothly changed values which is hard to threshold for dropping redundant operations with non-marginal contributions to the supernet. Also, the regularization in [Chu *et al.*, 2020], may not preserve the summation equal to 1 constraint. Besides, [Chakraborty *et al.*, 2020] proposes recursive statistical approach to solve wFM with convex constraint, however, the definition proposed do not explicitly preserve the equality constraint and it requires re-normalization of the solution. In contrast, our approach composes of the sparsemax transformation for convex Fréchet mixture of SPD operations with the following two advantages: 1) It can preserve most of the important properties of softmax such as, it is simple to evaluate, cheaper to differentiate [Martins and Astudillo, 2016]. 2) It is able to produce **sparse distributions** such that the best operation associated with each edge is likely to make more dominant contributions to the supernet, and thus better architectures can be derived (refer to Fig 2(a),2(b) and §4).

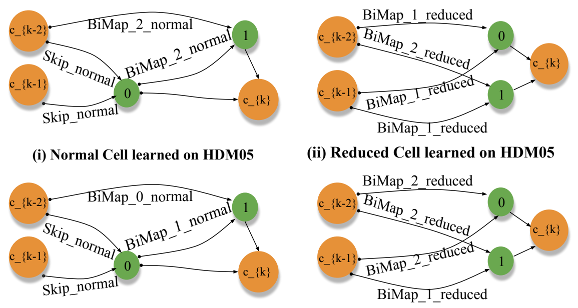
From Eq:(7–8), the **mixing of operations** between nodes is determined by the weighted combination of alpha's (α^k) and the set of operations ($\mathscr{O}^k_{\mathcal{M}}$). This relaxation makes the search space continuous and therefore, architecture search can be achieved by learning a set of alpha ($\alpha = \{\alpha^k, \forall\ k\ \in N_e\}$). To achieve our goal, we simultaneously learn the contributions (i.e., the architecture parameterization) α of all the candidate operations and their corresponding network weights (w). Consequently, for a given w, we optimize α and vice-versa resulting in the following bi-level optimization problem.

$$\underset{\alpha}{\min.}\ \boldsymbol{E}^U_{val}\big(w^{opt}(\alpha), \alpha\big); \ \text{s.t. } w^{opt}(\alpha) = \underset{w}{\operatorname{argmin}}\ \boldsymbol{E}^L_{train}(w, \alpha), \tag{9}$$

where the lower-level optimization ($\boldsymbol{E}^L_{train}$) corresponds to the optimal weight variable learned for a given α *i.e.*, $w^{opt}(\alpha)$ using a training loss. The upper-level optimization ($\boldsymbol{E}^U_{val}$) solves for the variable α given the optimal w using a validation loss. This bi-level search method gives an optimal mixture of multiple small architectures. To derive each node in the discrete architecture, we maintain top-k operations *i.e.*, the k^{th} highest weight among all the candidate operations associated with all the previous nodes.



(a)



(b)

Figure 2: (a) Distribution of edge weights for operation selection using softmax, sigmoid, and sparsemax on the Fréchet mixture of SPD operations. (b) Derived sparsemax architecture by the proposed SPDNetNAS. The better sparsity leads to less skips and poolings compared to those of other NAS solutions shown in our Appendix[2].

Bi-level Optimization. The bi-level optimization problem proposed in Eq:(9) is difficult to solve. On one hand, the α can be interpreted as hyper-parameter but it's not a scalar and hence, harder to optimize. On the other hand, the lower optimization is computationally expensive. Following [Liu *et al.*, 2018b] work, we approximate $w^{opt}(\alpha)$ in the upper-optimization problem to skip inner-optimization as follows:

$$\nabla_\alpha \boldsymbol{E}_{val}^{U}\big(w^{opt}(\alpha),\alpha\big) \approx \nabla_\alpha \boldsymbol{E}_{val}^{U}\big(w-\eta\nabla_w \boldsymbol{E}_{train}^{L}(w,\alpha),\alpha\big), \tag{10}$$

Here, η is the learning rate and ∇ is the gradient operator. Note that the gradient based optimization for w must follow the geometry of SPD manifolds to update the structured connection weight, and its corresponding SPD matrix data. Applying the chain rule to Eq:(10) gives

$$\overbrace{\nabla_\alpha \boldsymbol{E}_{val}^{U}(\tilde{w},\alpha)}^{\text{first term}} - \overbrace{\eta\nabla^2_{\alpha,w}\boldsymbol{E}_{train}^{L}(w,\alpha)\nabla_{\tilde{w}}\boldsymbol{E}_{val}^{U}(\tilde{w},\alpha)}^{\text{second term}}, \tag{11}$$

where, $\tilde{w} = \boldsymbol{\Psi}_{\mathbf{r}}\big(w-\eta\tilde{\nabla}_w \boldsymbol{E}_{train}^{L}(w,\alpha)\big)$ denotes the weight update on the SPD manifold for the forward model. $\tilde{\nabla}_w$, $\boldsymbol{\Psi}_{\mathbf{r}}$ symbolizes the Riemannian gradient and the retraction operator respectively. The second term in the Eq:(11) involves second order differentials with very high computational complexity, hence, using the finite approximation method the second term of Eq:(11) reduces to:

$$\frac{\nabla_\alpha \boldsymbol{E}_{train}^{L}(w^{+},\alpha) - \nabla_\alpha \boldsymbol{E}_{train}^{L}(w^{-},\alpha)}{2\delta} \tag{12}$$

where, $w^{\pm} = \boldsymbol{\Psi}_{\mathbf{r}}(w \pm \delta\tilde{\nabla}_{\tilde{w}}\boldsymbol{E}_{val}^{U}(\tilde{w},\alpha))$ and δ is a small number set to $0.01/\|\nabla_{\tilde{w}}\boldsymbol{E}_{val}^{U}(\tilde{w},\alpha)\|_2$.

Though the optimization follows the suggested structure in [Liu *et al.*, 2018b], there are some key differences. Firstly, the updates on the manifold-valued kernel weights are constrained on manifolds, which ensures that the feature maps at every intermediate layer are SPDs. For concrete derivations on back-propagation for SPD network layers, refer to [Huang and Van Gool, 2017] work. Secondly, the update on the aggregation weights of the involved SPD operations needs to satisfy an additional strict convex constraint, which is enforced as a part of the optimization problem Eq:(8). The pseudo code of our method is outlined in **Algorithm** 1.

Algorithm 1 The proposed SPDNetNAS

```
while not converged do
    Step1: Update the architecture parameter $\alpha$ using Eq:(9) solution by satisfying an additional strict convex constraint. Note that updates on those manifold parameter $w$ of the network follow the gradient descent on SPD manifolds;
    Step2: Update $w$ by solving $\nabla_w \boldsymbol{E}_{train}(w,\alpha)$; Enforce the SPD manifold gradients to update those structured $w$ [Huang and Van Gool, 2017; Brooks et al., 2019];
end
Ensure: Final architecture based on $\alpha$. Decide the operation at an edge $k$ using $\underset{o\in\mathcal{O}_{\mathcal{M}}}{\text{argmax}}\{\alpha_o^k\}$
```

4 Experiments and Results

To keep the experimental evaluation consistent with the existing SPD networks [Huang and Van Gool, 2017; Brooks *et al.*, 2019], we follow them to use RADAR [Chen *et al.*, 2006], HDM05 [Müller *et al.*, 2007], and AFEW [Dhall *et al.*, 2014] datasets. For our SPDNetNAS[3], we first optimize the supernet on the training/validation sets and then prune it with the best operation for each edge. Finally, we train the optimized architecture from scratch to document the results. For both these stages, we consider the same normal and reduction cells. A cell receives preprocessed inputs using fixed BiMap_2 to make the input of the same initial dimension. All architectures are trained with a batch size of 30. Learning rate (η) for RADAR, HDM05, and AFEW dataset is set to 0.025, 0.025 and 0.05 respectively. Besides, we conducted experiments where we select architecture using a random search path (SPDNetNAS (R)) to justify whether our search space with the introduced SPD operations can derive meaningful architectures.

We compare SPDNet [Huang and Van Gool, 2017], SPDNetBN [Brooks *et al.*, 2019], and ManifoldNet [Chakraborty *et al.*, 2020] that are handcrafted SPD networks. SPDNet and SPDNetBN are evaluated using their original implementations and default setup, and ManifoldNet is evaluated following the video classification setup of [Chakraborty *et al.*, 2020]. It is non-trivial to adapt ManifoldNet to RADAR and

[3]Source code link: https://github.com/rheasukthanker/spdnetnas.

Dataset	DARTS	FairDARTS	SPDNet	SPDNetBN	SPDNetNAS (R)	SPDNetNAS
RADAR	98.21%± 0.23	**98.51% ± 0.09**	93.21% ± 0.39	92.13% ± 0.77	95.49% ± 0.08	97.75% ± 0.30
#RADAR	2.6383 MB	2.6614 MB	0.0014 MB	0.0018 MB	0.0185 MB	0.0184 MB
HDM05	54.27% ± 0.92	58.29% ± 0.86	61.60% ± 1.35	65.20% ± 1.15	66.92% ± 0.72	**69.87% ± 0.31**
#HDM05	3.6800MB	5.1353 MB	0.1082 MB	0.1091 MB	1.0557 MB	1.064MB MB

Table 2: Performance comparison of our SPDNetNAS against existing SPDNets and traditional NAS on drone and action recognition. SPDNetNAS (R): randomly select architecure from our search space, DARTS/FairDARTS: accepts logarithm forms of SPDs. The search time of our method on RADAR and HDM05 is noted to be 1 CPU days and 3 CPU days respectively. And the search cost of DARTS and FairDARTS on RADAR and HDM05 are about 8 GPU hours. #RADAR and #HDM05 show the model sizes on the respective dataset.

DARTS	FairDARTS	ManifoldNet	SPDNet	SPDNetBN	SPDNetNAS (RADAR)	SPDNetNAS (HDM05)
26.88 %	22.31%	28.84%	34.06%	37.80%	**40.80%**	**40.64%**

Table 3: Performance comparison of our SPDNetNAS transferred architectures on AFEW against handcrafted SPDNets and Euclidean NAS. SPDNetNAS(RADAR/HDM05): architectures searched on RADAR and HDM05 respectively.

HDM05, as ManifoldNet requires SPD features with multiple channels, and both of the two datasets can hardly obtain them. For comparing against Euclidean NAS methods, we used DARTS [Liu *et al.*, 2018b], and FairDARTS [Chu *et al.*, 2020] by treating SPD's logarithm maps[4] as Euclidean data in their official implementation with the default setup.

a) Drone Recognition. For this task, we use the RADAR dataset from [Chen *et al.*, 2006]. This dataset's synthetic setting is composed of radar signals, where each signal is split into windows of length 20, resulting in a 20x20 covariance matrix for each window (one radar data point). The synthesized dataset consists of 1000 data points per class. Given 20×20 input covariance matrices, our reduction cell reduces them to 10×10 matrices followed by the normal cell to provide a higher complexity to our network. Following [Brooks *et al.*, 2019], we assign 50%, 25%, and 25% of the dataset for training, validation, and test set, respectively. The Euclidean NAS algorithms are evaluated on the Euclidean map of the input. For direct SPD input, DARTS performance (95.86%) and FairDarts (92.26%) are worse as expected. For this dataset, our algorithm takes 1 CPU day of search time to provide the SPD architecture. Training and validation take 9 CPU hours for 200 epochs. Test results on this dataset are provided in Table (2), which clearly shows our method's benefit. The statistical performance shows that our NAS method provides an architecture with much fewer parameters (more than 140 times) than well-known Euclidean NAS on this task. The normal and reduction cells obtained on this dataset are shown in Fig. 2(b). The rich search space of our algorithm allows inclusion of skips and poolings (unlike traditional SPDNets) in our architectures thus improving the performance.

b) Action Recognition. For this task, we use the HDM05 dataset [Müller *et al.*, 2007] which contains 130 action classes, yet, for consistency with previous work [Brooks *et al.*, 2019], we used 117 class for performance comparison. This dataset has 3D coordinates of 31 joints per frame. Following the earlier works [Harandi *et al.*, 2017], we model action for a sequence using 93×93 joint covariance matrix. The dataset has 2083 SPD matrices distributed among all 117 classes. Similar to the previous task, we split the dataset into 50%, 25%, and 25% for training, validation, and testing. Here, our reduction cell is designed to reduce the matrices dimensions from 93 to 30 for legitimate comparison against [Brooks *et al.*, 2019]. To search for the best architecture, we ran our algorithm for 50 epoch (3 CPU days). Figure 2(b) shows the final cell architecture that got selected based on the validation performance. The optimal architecture is trained from scratch for 100 epochs, which took approximately 16 CPU hours. The test accuracy achieved on this dataset is provided in Table (2). Statistics clearly show that our models perform better despite being lighter than the NAS models and the handcrafted SPDNets. The Euclidean NAS models' inferior results show the use of SPD geometric constraint for SPD valued data.

c) Emotion Recognition. We use the AFEW dataset [Dhall *et al.*, 2014] to evaluate the transferability of our searched architecture for emotion recognition. This dataset has 1345 videos of facial expressions classified into 7 distinct classes. To train on the video frames directly, we stack all the handcrafted SPDNets, and our searched SPDNet on top of a convolutional network [Meng *et al.*, 2019] with its official implementation. For ManifoldNet, we compute a 64×64 spatial covariance matrix for each frame on the intermediate ConvNet features of $64 \times 56 \times 56$ (channels, height, width). We follow the reported setup of [Chakraborty *et al.*, 2020] to first apply a single wFM layer with kernel size 5, stride 3, and 8 channels, followed by three temporal wFM layers of kernel size 3 and stride 2, with the channels being 1, 4, 8 respectively. We closely follow the official implementation of ManifoldNet for the wFM layers and adapt the code to our specific task. Since SPDNet, SPDNetBN, and our SPDNetNAS require a single channel SPD matrix as input, we use the final 512-dimensional vector extracted from the convolutional network, project it using a dense layer to a 100-dimensional feature vector and compute a 100×100 temporal covariance matrix. To study our algorithm's transferability, we evaluate its searched architecture on RADAR and HDM05. Also, we evaluate DARTS and FairDARTS directly on the video frames of AFEW. Table (3) reports the evaluations results. As we can observe, the transferred architectures can handle the new dataset quite convincingly, and their test accuracies are better than those of the state-of-the-art SPDNets and Euclidean NAS algorithms. In Appendix[2], we present the results of these competing methods and our searched models on the raw SPD features of the AFEW dataset.

d) Comparison under similar model complexities. We compare the statistical performance of our method against the other competing methods under similar model sizes. Table (4) shows the results obtained on the RADAR dataset. One key point to note here is that when we increase the number of parameters in SPDNet and SPDNetBN, we observe a very severe

[4]Feeding raw SPDs generally results in performance degradation.

Dataset	Manifoldnet	SPDNet	SPDNetBN	SPDNetNAS
RADAR	NA	73.066%	87.866%	**97.75%**
#RADAR	NA	0.01838 MB	0.01838 MB	0.01840 MB
AFEW	25.8%	34.06%	37.80%	**40.64%**
#AFEW	11.6476 MB	11.2626 MB	11.2651 MB	11.7601 MB

Table 4: Performance of our SPDNetNAS against existing SPDNets with comparable model sizes on RADAR and AFEW.

Dataset	softmax	sigmoid	sparsemax
RADAR	96.47% ± 0.10	97.70% ± 0.23	**97.75% ± 0.30**
HDM05	68.74% ± 0.93	68.64% ± 0.09	**69.87% ± 0.31**

Table 5: Ablations study on different solutions to our suggested Fréchet mixture of SPD operations within SPDNetNAS.

degradation in the performance accuracy —mainly because the network starts overfitting rapidly. The performance degradation is far more severe for the HDM05 dataset with SPDNet (1.047MB) performing 0.7619% and SPDNetBN (1.082MB) performing 1.45% and hence, is not reported in Table (4). That further indicates the ability of SPDNetNAS to generalize better and avoid overfitting despite the larger model size.

e) Ablation study. Lastly, we conducted an ablation study to realize the effect of probability simplex constraint (sparsemax) on our suggested Fréchet mixture of SPD operations. Although in Fig. 2(a) we show better probability weight distribution with sparsemax, Table(5) shows that it performs better empirically as well on both RADAR and HDM05 compared to the softmax and sigmoid cases. Therefore, SPD architectures derived using the sparsemax is observed to be better.

5 Conclusion and Future Direction

We presented a neural architecture search problem of SPD manifold networks. To address it, we proposed a new differentiable NAS algorithm that consists of sparsemax-based Fréchet relaxation of search space and manifold update-based bi-level optimization. Evaluation on several benchmark datasets showed the clear superiority of the proposed NAS method over handcrafted SPD networks and Euclidean NAS algorithms.

As a future work, it would be interesting to generalize our NAS algorithm to cope with other manifold valued data (e.g., [Huang *et al.*, 2017; Huang *et al.*, 2018; Chakraborty *et al.*, 2018; Kumar *et al.*, 2018; Zhen *et al.*, 2019; Kumar, 2019; Kumar *et al.*, 2020]) and manifold poolings (e.g., [Wang *et al.*, 2017; Engin *et al.*, 2018; Wang *et al.*, 2019]), which are generally valuable for visual recognition, structure from motion, medical imaging, radar imaging, forensics, appearance tracking to name a few.

Acknowledgments

This work was supported in part by the ETH Zürich Fund (OK), an Amazon AWS grant, and an Nvidia GPU grant. Suryansh Kumar's project is supported by "ETH Zürich Foundation and Google, Project Number: 2019-HE-323" for bringing together best academic and industrial research. The authors would like to thank Siwei Zhang from ETH Zürich for evaluating our code on AWS GPUs.

References

[Agrawal *et al.*, 2019] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J.Z. Kolter. Differentiable convex optimization layers. In *NeurIPS*, 2019.

[Ahmed and Torresani, 2017] K. Ahmed and L. Torresani. Connectivity learning in multi-branch networks. *arXiv:1709.09582*, 2017.

[Bhatia and Holbrook, 2006] R. Bhatia and J. Holbrook. Riemannian geometry and matrix geometric means. *Linear algebra and its applications*, 2006.

[Bonnabel, 2013] S. Bonnabel. Stochastic gradient descent on riemannian manifolds. *T-AC*, 2013.

[Brooks *et al.*, 2019] D. Brooks, O. Schwander, F. Barbaresco, J. Schneider, and M. Cord. Riemannian batch normalization for spd neural networks. In *NeurIPS*, 2019.

[Chakraborty *et al.*, 2018] R. Chakraborty, C. Yang, X. Zhen, M. Banerjee, D. Archer, D. Vaillancourt, V. Singh, and B. Vemuri. A statistical recurrent model on the manifold of symmetric positive definite matrices. In *NeurIPS*, 2018.

[Chakraborty *et al.*, 2020] R. Chakraborty, J. Bouza, J. Manton, and B.C. Vemuri. Manifoldnet: A deep neural network for manifold-valued data with applications. *T-PAMI*, 2020.

[Chakraborty, 2020] R. Chakraborty. Manifoldnorm: Extending normalizations on riemannian manifolds. *arXiv:2003.13869*, 2020.

[Chen *et al.*, 2006] V.C. Chen, F. Li, S-S Ho, and H. Wechsler. Micro-doppler effect in radar: phenomenon, model, and simulation study. *T-AES*, 2006.

[Cheng *et al.*, 2016] G. Cheng, J. Ho, H. Salehian, and B.C. Vemuri. Recursive computation of the fréchet mean on non-positively curved riemannian manifolds with applications. In *Riemannian Computing in Computer Vision*. 2016.

[Chu *et al.*, 2019] X. Chu, B. Zhang, J. Li, Q. Li, and R. Xu. Scarletnas: Bridging the gap between scalability and fairness in neural architecture search. *arXiv:1908.06022*, 2019.

[Chu *et al.*, 2020] X. Chu, T. Zhou, B. Zhang, and J. Li. Fair DARTS: Eliminating Unfair Advantages in Differentiable Architecture Search. In *ECCV*, 2020.

[Deb *et al.*, 2002] K. Deb, A. Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *T-EC*, 2002.

[Dhall *et al.*, 2014] A. Dhall, R. Goecke, J. Joshi, K. Sikka, and T. Gedeon. Emotion recognition in the wild challenge 2014: Baseline, data and protocol. In *ICMI*, 2014.

[Dong *et al.*, 2017] Z. Dong, S. Jia, C. Zhang, M. Pei, and Y. Wu. Deep manifold learning of symmetric positive definite matrices with application to face recognition. In *AAAI*, 2017.

[Elsken *et al.*, 2018] T. Elsken, J.H. Metzen, and F. Hutter. Neural architecture search: A survey. *arXiv:1808.05377*, 2018.

[Engin *et al.*, 2018] M. Engin, L. Wang, L. Zhou, and X. Liu. Deepkspd: Learning kernel-matrix-based spd representation for fine-grained image recognition. In *ECCV*, 2018.

[Gao *et al.*, 2019] Z. Gao, Y. Wu, M.T. Harandi, and Y. Jia. A robust distance measure for similarity-based classification on the spd manifold. *TNNLS*, 2019.

[Gong *et al.*, 2019] X. Gong, S. Chang, Y. Jiang, and Z. Wang. Autogan: Neural architecture search for generative adversarial networks. In *ICCV*, 2019.

[Harandi *et al.*, 2017] M.T. Harandi, M. Salzmann, and R. Hartley. Dimensionality reduction on spd manifolds: The emergence of geometry-aware methods. *T-PAMI*, 2017.

[Huang and Van Gool, 2017] Z. Huang and L. Van Gool. A Riemannian network for SPD matrix learning. In *AAAI*, 2017.

[Huang *et al.*, 2017] Z. Huang, C. Wan, T. Probst, and L. Van Gool. Deep learning on Lie groups for skeleton-based action recognition. In *ICCV*, 2017.

[Huang *et al.*, 2018] Z. Huang, J. Wu, and L. Van Gool. Building deep networks on Grassmann manifolds. In *AAAI*, 2018.

[Karcher, 1977] H. Karcher. Riemannian center of mass and mollifier smoothing. *Communications on pure and applied mathematics*, 1977.

[Kumar *et al.*, 2018] S. Kumar, A. Cherian, Y. Dai, and H. Li. Scalable dense non-rigid structure-from-motion: A grassmannian perspective. In *CVPR*, 2018.

[Kumar *et al.*, 2020] S. Kumar, L. Van Gool, C. Oliveira, A. Cherian, Y. Dai, and H. Li. Dense non-rigid structure from motion: A manifold viewpoint. *arXiv:2006.09197*, 2020.

[Kumar, 2019] S. Kumar. Jumping manifolds: Geometry aware dense non-rigid structure from motion. In *CVPR*, 2019.

[Lackenby, 2020] Marc Lackenby. Introductory chapter on riemannian manifolds. *Notes*, 2020.

[Liang *et al.*, 2019] H. Liang, S. Zhang, J. Sun, X. He, W. Huang, K. Zhuang, and Z. Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv:1909.06035*, 2019.

[Liu *et al.*, 2017] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv:1711.00436*, 2017.

[Liu *et al.*, 2018a] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *ECCV*, 2018.

[Liu *et al.*, 2018b] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *ICLR*, 2018.

[Martins and Astudillo, 2016] A. Martins and R. Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *ICML*, 2016.

[Meng *et al.*, 2019] D. Meng, X. Peng, K. Wang, and Y. Qiao. Frame attention networks for facial expression recognition in videos. In *ICIP*, 2019.

[Moakher, 2005] M. Moakher. A differential geometric approach to the geometric mean of symmetric positive-definite matrices. *SIAM*, 2005.

[Müller *et al.*, 2007] M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber. Documentation mocap database hdm05. 2007.

[Nguyen *et al.*, 2019] Xuan Son Nguyen, Luc Brun, Olivier Lézoray, and Sébastien Bougleux. A neural network based on spd manifold learning for skeleton-based hand gesture recognition. In *CVPR*, 2019.

[Pennec *et al.*, 2006] X. Pennec, P. Fillard, and N. Ayache. A riemannian framework for tensor computing. *IJCV*, 2006.

[Pennec, 2020] X. Pennec. Manifold-valued image processing with spd matrices. In *Riemannian Geometric Statistics in Medical Image Analysis*, pages 75–134. Elsevier, 2020.

[Real *et al.*, 2019] E. Real, A. Aggarwal, Y. Huang, and Q.V. Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.

[Saxena and Verbeek, 2016] S. Saxena and J. Verbeek. Convolutional neural fabrics. In *NeurIPS*, 2016.

[Shin *et al.*, 2018] R. Shin, C. Packer, and D. Song. Differentiable neural network architecture search. 2018.

[Tuzel *et al.*, 2008] O. Tuzel, F. Porikli, and P. Meer. Pedestrian detection via classification on riemannian manifolds. *T-PAMI*, 2008.

[Veniat and Denoyer, 2018] T. Veniat and L. Denoyer. Learning time/memory-efficient deep architectures with budgeted super networks. In *CVPR*, 2018.

[Wang *et al.*, 2017] Q. Wang, P. Li, and L. Zhang. G2denet: Global gaussian distribution embedding network and its application to visual recognition. In *CVPR*, 2017.

[Wang *et al.*, 2019] Q. Wang, J. Xie, W. Zuo, L. Zhang, and P. Li. Deep cnns meet global covariance pooling: Better representation and generalization. *arXiv:1904.06836*, 2019.

[Wu *et al.*, 2019] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019.

[Wu *et al.*, 2021] Y. Wu, A. Liu, Z. Huang, S. Zhang, and L. Van Gool. Neural architecture search as sparse supernet. In *AAAI*, 2021.

[Zhang *et al.*, 2020] X. Zhang, Z. Huang, N. Wang, S. XIANG, and C. Pan. You only search once: Single shot neural architecture search via direct sparse optimization. *T-PAMI*, 2020.

[Zhen *et al.*, 2019] X. Zhen, R. Chakraborty, N. Vogt, B.B. Bendlin, and V. Singh. Dilated convolutional neural networks for sequential manifold-valued data. In *ICCV*, 2019.

[Zheng *et al.*, 2019] X. Zheng, R. Ji, L. Tang, B. Zhang, J. Liu, and Q. Tian. Multinomial distribution learning for effective neural architecture search. In *ICCV*, 2019.

[Zhou *et al.*, 2019] H. Zhou, M. Yang, J. Wang, and W. Pan. Bayesnas: A bayesian approach for neural architecture search. In *ICML*, 2019.

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