CVL - Computer Vision Lab

Semester Project

# Neural Architecture Search of SPD Manifold Nets: A Geometric Perspective

**Author**
Erik Goron

**Supervisors**
Zhiwu Huang
Suryansh Kumar
Luc Van Gool

Spring 2020

## Abstract

Symmetric positive definite (SPD) matrix deep learning methods have become a popular choice in many visual classification tasks. They learn and deal with descriptive and compact statistical representations of data that respects the underlying geometric structure of the Riemannian SPD manifold. Yet, these Riemannian networks called SPDNets [13] require lots of expertise to design optimal architectures.
In recent years, large amounts of algorithms for Neural Architecture Searh (NAS) on euclidean deep learning networks have emerged. Despite the great success of these architecture search methods, to the best of our knowledge, none of them deal with Riemannian manifold networks.
To circumvent the aforementioned issue, this thesis proposes a new differentiable architecture search method of SPD manifold networks. This novel NAS method introduces a new computational cell where a search space of SPD architectures is considered and optimized based on the validation performance, we call it SPDNetNAS.
Instead of searching for discrete architectures, we relax the search space to be continuous and differentiable. We then introduce an efficient bi-level optimization problem to jointly search and train the best SPD architectures. Experiments on RADAR [8], HDM05 [24] and AFEW [9] show that our method outperforms previous SPD networks and opens a new direction for researchers interested in the world of geometric deep learning and AutoML [1].

# Contents

## 1 Introduction

Over the last decade, deep learning has experienced great success. On one hand, due to the increase in performance of computational resources as well as the availability of large amounts of data. On the other hand, its excelling advantage compared to other classical machine learning techniques by automating the process of feature engineering. This automation eliminates the demand of domain expertise when it comes to manually extracting features. Unfortunately, a new challenge appears when dealing with deep neural networks, that is, the need to manually design its underlying architecture. Designing better and more complex networks requires expertise and is often done through trial and error. To alleviate this issue, researchers over recent years have been working towards automating architecture engineering. The rise of a new topic now called Neural Architecture Search (NAS) is thus evident and is a logical step towards machine learning automation (AutoML). Several NAS methods have already outperformed hand crafted architectures on several computer vision tasks [35] [36] [22] [21] [20], opening a new promising path for new applications. Despite the great success of most NAS methods, a fundamental challenge of scalability and efficiency remains. Dominant approaches based on RL [35] or Bayesian optimization [16] treat the architecture search problem over a discrete domain where extensive amounts of evaluations are required.
Furthermore, to the best of our knowledge, no research has been done on NAS methods that deal with non-Euclidean data.
Throughout the recent years, many scientific researchers have studied data with underlying non-Euclidean or manifold structure. An interest on building deep neural networks models working with manifold-valued data has grown and has achieved outstanding success in a broad range of applications. In this paper, we focus our attention on data lying in the manifold of Symmetric Positive Definite (SPD) matrices. SPD matrices are capable of representing compact high order statistical representations and have been successfully used for a variety of visual tasks such as medical imaging analysis [25], pedestrian detection [29],[28], face recognition [31], [14], action recognition [11], [27], structure from motion [19],[18] and many more. However, as in euclidean neural networks, SPD networks architectures depend on complex handcrafted designs and requires lots of domain knowledge.
In this work, we propose to combine both NAS and SPD neural networks to remove consuming manual network engineering. We call our network : SPDNetNAS. We define a new building block cell that respects the underlying manifold structure of our data and where all Riemannian computations happen. This cell, called SPD cell, is represented by a directed acyclic graph (DAG) where intermediate nodes are latent SPD representations and edges are manifold constrained operations. Our method aims at searching for an optimal architecture as a one-shot training process of a supernet formed by a mixture of SPD architectures. To do so, we first continuously relax our search space to enable differential architecture search as in [22]. Then, a complex bi-level optimization procedure provides the optimal SPD neural architecture based on its validation performance. Finally, the optimal selected architecture is trained from scratch and test performance is reported.
We demonstrate that this novel approach outperforms the current baselines on three different datasets : Drone recognition dataset RADAR [8], action recognition dataset HDM05 [24] and emotion recognition dataset AFEW [9].
In this work we make the following contributions :

- A new differentiable architecture search method for SPD manifold networks.

- A one-shot optimization procedure to jointly train and search for the best possible SPD neural architecture.
- We introduce new operations well suited for SPD networks : Two Riemannian pooling operations (max and average), a Riemannian skip connection and a weighted Riemannian pooling operation based on the Fréchet mean.
- An analysis on different iterative Fréchet mean solvers.
- Description and detailed analysis on the architecture search optimization.
- Statistical performance of SPDNetNAS on several benchmark datasets and comparison with existing SPD networks. [2],[13].

## 2 Background on Riemannian geometry

We provide a brief summary of notions on Riemannian geometry and the manifold of SPD matrices, useful for further understanding.

### 2.1 Riemannian manifold

A Riemannian manifold is a smooth and differentiable manifold, locally similar to the Euclidean space, and equipped with a positive definite inner-product on the tangent space of each point. Given a m-dimensional Riemmanian submanifold of some Euclidean space $\mathbb{R}^N$, noted $\mathcal{M}$, we define the tangent space at point $\mathbf{X} \in \mathcal{M}$ as $\mathcal{T}_X\mathcal{M}$. The tangent space is an m-dimensional subspace of $\mathbb{R}^N$ formed by all tangent vectors passing throught $\mathbf{X}$. In this work, we focus our attention on a specific Riemannian manifold proven to yield promising results in several visual recognition tasks, the manifold of SPD matrices.

### 2.2 Manifold of SPD matrices

#### 2.2.1 Riemannian metric and tangent space

We define the manifold of SPD matrices $\mathcal{S}_{++}^n$ and two points $\mathbf{X}, \mathbf{Y} \in \mathcal{S}_{++}^n$, for simplicity, we define the tangent space of a SPD manifold at point $\mathbf{X}$ as $\mathcal{T}_X$ When dealing with manifolds, we need a way of defining lengths and angles. Analyzing these notions requires the definition of a suitable Riemannian metric, considered as the family of inner products on all tangent spaces. Several metrics have been studied on SPD manifolds. The Euclidean classic metric has shown to yield poor results and can lead to the swelling effect, well known in the context of diffusion tensor imaging (DTI) [25], where the Euclidean average leads to inadequate results. Several metrics have emerged to avoid the issues of the Euclidean metric on SPD matrices [23] [30].
A popular choice is the Affine Invariant Riemannian Metric (AIRM) [25], which has several interesting properties such as invariance under affine transformations, closed form solutions for parallel transport along geodesics and provides an easy way to compute exponential and logarithmic maps. When it comes to its downsides, the AIRM suffers from slow computating derivation when the dimensions of the manifold increases.

Given $\mathbf{X} \in \mathcal{S}_{++}^n$ and $\mathbf{x},\mathbf{y} \in \mathcal{T}_X$. The AIRM is defined as

$$\begin{aligned} \langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\boldsymbol{X}} &\triangleq \left\langle \boldsymbol{X}^{-1/2} \boldsymbol{x} \boldsymbol{X}^{-1/2}, \boldsymbol{X}^{-1/2} \boldsymbol{y} \boldsymbol{X}^{-1/2} \right\rangle \\ &= \operatorname{Tr}\left( \boldsymbol{X}^{-1} \boldsymbol{x} \boldsymbol{X}^{-1} \boldsymbol{y} \right) \end{aligned} \tag{1}$$

This metric induces a geodesic curve connecting two points on the manifold, the respective distance between the points is defined as

$$\delta_R(\boldsymbol{X}, \boldsymbol{Y}) = \frac{1}{2} \left\| \log \left( \boldsymbol{X}^{-1/2} \boldsymbol{Y} \boldsymbol{X}^{-1/2} \right) \right\|_F \tag{2}$$

The Euclidean tangent space $\mathcal{T}_X$ at each point of the manifold, varying smoothly between points, gives us the opportunity to define natural mappings with the manifold. We define the exponential map $\exp_{\mathbf{X}} : T_{\mathbf{X}} \mapsto \mathcal{S}_{++}^n$ as

$$\exp_{\mathbf{X}}(\mathbf{y}) = \mathbf{X}^{\frac{1}{2}} \exp\left( \mathbf{X}^{-\frac{1}{2}} \mathbf{y} \mathbf{X}^{-\frac{1}{2}} \right) \mathbf{X}^{\frac{1}{2}} \tag{3}$$

The logarithm is then uniquely defined $\log_{\mathbf{X}} : \mathcal{S}_{++}^n \mapsto T_{\mathbf{X}}$ as

$$\log_{\mathbf{X}}(\mathbf{Y}) = \mathbf{X}^{\frac{1}{2}} \log\left( \mathbf{X}^{-\frac{1}{2}} \mathbf{Y} \mathbf{X}^{-\frac{1}{2}} \right) \mathbf{X}^{\frac{1}{2}} \tag{4}$$

The exponential $\exp(\,\cdot\,)$ and logarithm $\log(\,\cdot\,)$ matrix operators are easily computed by eigenvalue decomposition or singular value decomposition (SVD) since we are dealing with symmetric matrices with positive eigenvalues.

### 2.2.2 Geometric Riemannian mean : Fréchet mean

We will now define a generalisation of the weighted Euclidean mean to Riemannian manifolds. It is a weighted averaging operation defined in all Riemannian manifolds called the weighted Fréchét mean [10]. This operation has been proven to be useful for several Riemannian manifold learning purposes such as batch normalization on SPDnet [2] or as a convolution operation in [4].
Given a set $\mathcal{X}$ of N samples in a Riemannian manifold $\{P_i\}_{i=1\ldots N} \in \mathcal{S}_{++}^n$ and a set of weights $\{w_i\}_{i=1}^N$ satisfying the following convexity constraints : $w_i \geq 0$ and $\sum_{i \leq N} w_i = 1$, the weighted Fréchet mean, which we note for simplicity $\boldsymbol{\Psi}_{\mathbf{w}}(\mathcal{X})$, is defined as

$$\boldsymbol{\Psi}_{\mathbf{w}}(\mathcal{X}) = \mathbf{wFM}\left( \{P_i\}_{i \leq N}, \{w_i\}_{i \leq N} \right) = \underset{\mathbf{G} \in \mathcal{S}_{++}^n}{\operatorname{argmin}} \sum_{i=1}^{N} w_i \delta_{\mathcal{R}}^2 \left( P_i, \mathbf{G} \right) \tag{5}$$

A special case of this operation arises when the weights are equal, this yields to a simple geometric mean, simply called Fréchet mean [10] or Riemannian Barycenter. For the same previous set of samples $\mathcal{X}$, we note this operation $\boldsymbol{\Psi}(\mathcal{X})$ and define it as

$$\boldsymbol{\Psi}(\mathcal{X}) = \mathbf{FM}\left( \{P_i\}_{i \leq N} \right) = \underset{\mathbf{G} \in \mathcal{S}_{++}^n}{\operatorname{argmin}} \sum_{i=1}^{N} \delta_{\mathcal{R}}^2 \left( P_i, \mathbf{G} \right) \tag{6}$$

Both versions are repeatedly used throughout our study. Since they are based on argmin operations, most methods for solving the optimization problem of (5) and (6) rely on

iterative solvers.
We consider two iterative solvers and will compare both performances on future sections. The first and most well known solver is called the Karcher flow algorithm [17] [32]. We provide a detailed pseudo code of the method in **Algorithm 1**. Briefly, at iteration step k, data samples $P_i$ are projected into the tangent space of $\mathfrak{G}^{(k)}$. Then, the projections are averaged based on the Euclidean mean and the result is mapped back to the manifold, giving $\mathfrak{G}^{(k+1)}$.

**Algorithm 1** Karcher flow [17] to compute the Fréchet mean of N SPD matrices

```
 Require: N data samples $\{P_i\}_{i\leq N}$ and weights $\{w_i\}_{i\leq N}$, step $\alpha$, iterations $K$
1 : Initialize : $\mathfrak{G}^{(0)} \leftarrow \sum_{i\leq N} P_i$
2 : for $k \leq K$ do
3 :     $G \leftarrow \frac{1}{\sum_{j\leq N} w_j} \sum_{i\leq N} w_i \log_{\mathfrak{G}^{(k)}}(P_i)$
4 :     $\mathfrak{G}^{(k+1)} \leftarrow \exp_{\mathfrak{G}^{(k)}}(\alpha G)$
5 : end for
6: return $\mathfrak{G}$
```

(7)

It is important to note that convergence is guaranteed on constant negative curvature manifolds, such as our SPD manifold. Also, by choosing K = 1 and $\alpha = 1$ in the algorithm, we are expected to get a good approximation of the true mean in a considerably fast manner.

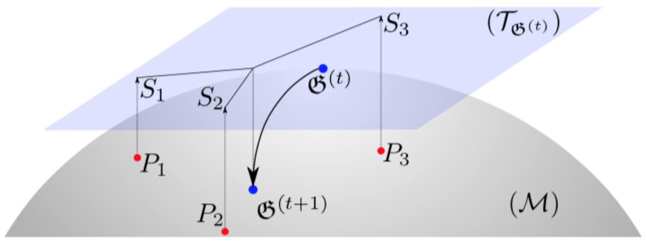


Figure 1: One iteration of the Karcher flow [2]

The second and less known Fréchet mean solver is an online algorithm introduced by Chakraborty et al. in [7],[4] called the inductive FM estimator (iFME). Considering the same set up as before, we define the $n^{th}$ Fréchet expectation estimator $\mathfrak{G}_n$ by the following recursion :

$$\mathfrak{G}_1 = P_1 \tag{8a}$$

$$\mathfrak{G}_n = \Gamma_{\mathfrak{G}_{n-1}}^{P_n}\left(\frac{w_n}{\sum_{i=1}^{n} w_i}\right) \tag{8b}$$

where $\Gamma_{P_1}^{P_2} : [0,1] \to \mathcal{S}_{++}^n$ is the geodesic curve from $P_1$ to $P_2$ induced by the AIRM. This means that the $n^{th}$ estimator lies between the $n^{th}$ estimator and the $n+1^{th}$ SPD sample in the SPD manifold.

This recursive algorithm takes advantage of the existence of a unique closed form solution to compute the wFM between two points in the manifold. In [7], it is stated that this intrinsic recursive solver demonstrates high computational advantages while achieving good performance in comparison with projection based algorithms.

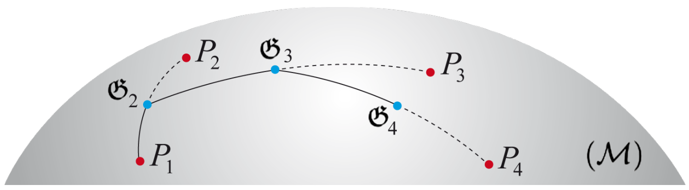


Figure 2: iFME algorithm of four SPD samples

#### 2.2.3 Parallel transport

One last important notion when talking about manifolds and used throughout this study is the notion of parallel transport. This operation gives us the opportunity to transport vectors belonging to a tangent plane to another, by moving them along a curve without changing them. Let's define two points $P_1, P_2 \in \mathcal{S}_{++}^n$ and $S \in \mathcal{T}_{P_1}$, the parallel transport of $S$ is defined as

$$\forall S \in \mathcal{T}_{P_1}, \Gamma_{P_1 \to P_2}(S) = \left(P_2 P_1^{-1}\right)^{\frac{1}{2}} S \left(P_2 P_1^{-1}\right)^{\frac{1}{2}} \in \mathcal{T}_{P_2} \tag{9}$$

In most cases, we want to directly transport $P_1 \in \mathcal{S}_{++}^n$ to $P_2 \in \mathcal{S}_{++}^n$. For this matter, we map $P_1$ into its tangent bundle using equation (4) and resulting in $S \in \mathcal{T}_{P_1}$. We then transport this vector using equation (9) and finally map back to the manifold using the exponential mapping, equation (3). Surprisingly, this last special case of parallel transport is exactly the same as in equation (9).

## 3 Operations on SPDNetNAS

In this section we present classical operations and layers from SPDnets [13], the recently proposed batch normalization on SPDnets [2] and we finally introduce additional operations that will enrich the complexity of SPDNetNAS . Since we are dealing with manifold valued data (SPD matrices), these operations must preserve the underlying geometric structure of our data.

### 3.1 Standard operations on SPDnets

#### 3.1.1 BiMap layer

Analogously to the fully connected layer on classical deep neural networks, Huang et. al. 2017 [13] presented the BiMap layer. The BiMap layer generates more compact and discriminative samples from a layer of input samples $\mathbf{X}_{k-1} \in \mathcal{S}_{++}^{d_{k-1}}$, using learned transformation matrices (weights), noted $\boldsymbol{W}_k \in \mathbb{R}_*^{d_k \times d_{k-1}}$, that lie on a compact Stiefeld manifold. The transformation outputs $\mathbf{X_k}$ as

$$\mathbf{X_k} = \boldsymbol{W}_k \boldsymbol{X}_{k-1} \boldsymbol{W}_k^T \in \mathcal{S}_{++}^{d_k} \tag{10}$$

In general, for dimensionality reduction purposes $d_k \leq d_{k-1}$.

### 3.1.2 ReEig layer

This layer, presented in [13], acts as a non-linear/activation operation in the manifold of SPD matrices, enabling higher discrimination between samples. The idea is simple, small positive eigenvalues are tuned up and we prevent them from being close to non positive ones.

$$\boldsymbol{X}_k = \boldsymbol{U}_{k-1} \max\left(\epsilon \boldsymbol{I}, \boldsymbol{\Sigma}_{k-1}\right) \boldsymbol{U}_{k-1}^T \tag{11}$$

We can solve this operation by using singular value decomposition (SVD) or eigenvalue decomposition (EIG)

### 3.1.3 LogEig and ExpEig layers

The LogEig layer will always be our last layer of the network and is necessary to reduce our samples to a flat space, allowing final classification with a traditional Euclidean dense layer. This is possible by projecting our samples with the matrix logarithm operation (note the difference from equation 4). We define the LogEig operation as

$$\boldsymbol{X}_k = \log\left(\boldsymbol{X}_{k-1}\right) = \boldsymbol{U}_{k-1} \log\left(\boldsymbol{\Sigma}_{k-1}\right) \boldsymbol{U}_{k-1}^T \tag{12}$$

where $\boldsymbol{X}_{k-1} = \boldsymbol{U}_{k-1}\boldsymbol{\Sigma}_{k-1}\boldsymbol{U}_{k-1}^T$.
Similarly and as a way to project flattened samples back to the manifold of SPD matrices, we define the ExpEig operation as

$$\boldsymbol{X}_k = \exp\left(\boldsymbol{X}_{k-1}\right) = \boldsymbol{U}_{k-1} \exp\left(\boldsymbol{\Sigma}_{k-1}\right) \boldsymbol{U}_{k-1}^T \tag{13}$$

where $\boldsymbol{X}_{k-1} = \boldsymbol{U}_{k-1}\boldsymbol{\Sigma}_{k-1}\boldsymbol{U}_{k-1}^T$.

## 3.2 Batch normalization for SPDnets

Brooks et. al. propose with their SPDNetBN [2] a new building block to improve Huang et. al. SPDNet [13]. This new layer, inspired by the famous batch normalization [15], considers a specific definition of Gaussians distributions on SPD matrices, where the only notion taken into account is the Riemannian mean and the notion of variance is excluded. Hence, to compute the proposed normalization, we begin by computing the Fr échet mean or Riemannian barycenter [10], $\mathfrak{G}_B$, of a given batch of SPD samples, $\{X_i\}_{i \leq N}$. Furthermore, we update the running mean noted $\mathfrak{G}_S$ and then iteratively transport each sample from $\mathfrak{G}_B$ to the origin of the manifold (Identity matrix), we consider this the centering step. Finally, analogously to the batch biasing, we transport our centered samples towards a learned SPD matrix $G$. When testing, we proceed similarly but consider the final training running mean as batch mean. Please refer to **Algorithm 2** and Brooks et.

al. [2] for further understanding.

**Algorithm 2** Riemannian batch normalization on a batch of SPD matrices

```
Require: batch of $N$ SPD matrices $\{X_i\}_{i \leq N}$, running mean $\mathfrak{G}_{\mathcal{S}}$, bias $G$, momentum $\eta$
1 : $\mathfrak{G}_{\mathcal{B}} \leftarrow \mathbf{FM}\left(\{X_i\}_{i \leq N}\right)$    (Compute batch mean)
2 : $\mathfrak{G}_{\mathcal{S}} \leftarrow \mathbf{wFM}\left(\{\mathfrak{G}_{\mathcal{S}}, \mathfrak{G}_{\mathcal{B}}\}, \eta\right)$    (Running mean)
3 : for $i \leq N$ do
4 :     $\bar{X}_i \leftarrow \Gamma_{\mathfrak{G}_{\mathcal{B}} \to I_d}(X_i)$    (Center batch)
5 :     $\tilde{X}_i \leftarrow \Gamma_{I_d \to G}\left(\bar{X}_i\right)$    (Bias batch)
6 : end for
Return normalized batch $\left\{\tilde{X}_i\right\}_{i \leq N}$
```

(14)

Additional and extensive study on Riemannian batch normalization techniques can be found in Chakraborty work [3].

## 3.3 New proposed operations

We introduce several new operations that will enrich the complexities of any SPD based neural network and which end up being crucial for our own.

### 3.3.1 Operations inspired by ManifoldNet [4][34]

Chakraborty et. al. presented in [4] a convolution type operation on Riemannian manifolds, considering the weighted Fréchet mean [10] as the averaging operation to be passed over a moving window of SPD samples. It is stated that this new manifold valued convolution operation is equivariant to the group action, allowing to share weigths within layers and following closely the classical equivariance to translation property from euclidean convolutions. The proposed manifold convolution operation is well suited for medical-imaging applications, where we deal with large amounts of small tensors, lying on Riemannian manifolds, such as the SPD manifold [5]. In our application case (action and drone recognition), we consider a small amount of SPD samples per layer and higher dimensions. New operations are needed to deal with this issue. We propose a similar operation to Chakraborty et. al. manifold valued convolution, called Weighted Riemannian pooling. For this matter, we consider a dense input layer of SPD matrices, fully connected to an output layer. The output layer of SPD matrices are derived by computing the weighted Fréchet mean of the input layer as many times as the desired dimension of our output layer, considering the weights as parameters to be learned by the optimization procedure. For clear intuition, refer to figure 11.
Analogously to the well-known skip connection introduced by [12], indispensable when dealing with very deep neural networks, Chakraborty et. al. presented in [6] a resiudal connection for the Riemannian case. This intuitive and simple operation maps an input layer of SPD matrices to a new one, while preserving its input representation.

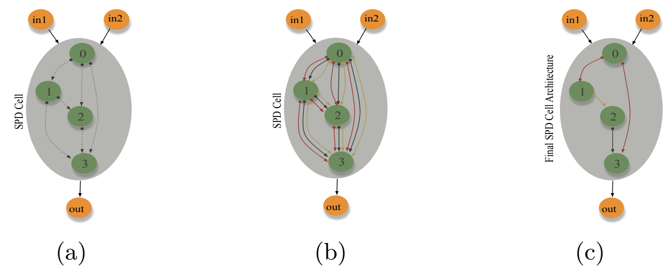


Figure 3: (a) A SPD cell structure composed of 4 SPD nodes, 2 input node and 1 output node. Initially the edges are unknown (b) Mixture of candidate SPD operations between nodes (c) Optimal cell architecture obtained after solving the relaxed continuous search space under a bi-level optimization routine.

### 3.3.2 Max and Average Riemannian pooling

Huang et. al. work [33] introduces a Grassmanian approach to the classical max and mean euclidean poolings operations. It is represented by the ProjMap layer. This layer acts as a parameter-free feature extractor and reduces the complexity of the model by reducing the dimensions of data. Similarly, we adapt this layer and propose a max and average pooling on the manifold of SPD matrices. We begin by projecting our samples to a flat space using the LogEig operation presented in section 3.1.3. Next, we compute max or mean classical pooling operations, valid operations since we currently lie on the Euclidean space. Finally, we project the reduced channels back to the manifold using ExpEig operations. Refer to figure 12 for intuition.

# 4 Architecture search of SPDNetNAS

Our approach aims at finding and training optimal architectures for SPD networks as a one-shot process, given a set of manually designed feasible architectures. As discussed in the introduction, the best existing algorithms that have achieved state-of-the-art results on traditional tasks [36] [26] suffer from heavy computational demand. Our method, inspired by Liu et. al. DARTS [22], rely on a relaxation of the discrete set of architectures so that a continuous optimization problem can be solved in a substantially more efficient manner. Our greatest challenge is to implement such method in the space of SPDNets, that is, preserving the manifold structure of our flowing data. We first introduce the building block of our network: the SPD cell. This cell, represented as a directed acyclic graph (DAG), follows an adaptation of DARTS [22] computational cell for the Riemannian SPD case. We then present the manually designed search space of valid SPD operations. Finally, we offer a detailed explanation on the continuous relaxation of the network and the bi-level optimization procedure which jointly learns both optimal architectures and its respective weights.

## 4.1 SPD cell : Building block of SPDNetNAS

Following [22][36][26], the building block of our network is a cell where all the computations happen, we call it SPD cell. The learned cell can then be stacked with others to

Table 1: Operation Search Space for SPDNetNAS.

| Operation | Definition |
|---|---|
| **BiMap_0** | BiMap → SPD Batch Normalization |
| **BiMap_1** | BiMap → SPD Batch Normalization → ReEig |
| **BiMap_2** | ReEig → BiMap → SPD Batch Normalization |
| **Skip_normal** | Identity transformation (Output same as input) |
| **None_normal** | Return Identity Matrix i.e, notion of origin on SPD |
| **WeightedRiemannianPooling** | Fully connected Weighted Fréchet Mean (section 3.3.1). |
| **AveragePooling_reduced** | LogEig → AveragePooling → ExpEig (section 3.3.2) |
| **MaxPooling_reduced** | LogEig → MaxPooling → ExpEig (section 3.3.2) |
| **Skip_reduced** | $\{C_i = \text{BiMap}(X_i),\ [U_i D_i\ ] = \text{svd}(C_i); i = 1, 2\}$, $C_{out} = U_b D_b U_b^T$, where, $U_b = \text{diag}(U_1, U_2)$ and$D_b = \text{diag}(D_1, D_2)$ |

form deeper and more complex networks. An SPD cell is a directed acyclic graph (DAG) represented by an ordered sequence of nodes and edges. Our SPD cell is assumed to have two inputs nodes, corresponding to the outputs of the previous two cells, two intermediate nodes, and a single output node, computed by applying a concatenation to all intermediate nodes. Since our main goal when training SPDNets is to pursue a reduction mapping from high dimensional manifold valued data to low dimension and more discriminative manifold samples, we need to provide dimensionality reduction operations within our network. In order to fulfill this need, we design two types of cells, a reduction cell and a normal cell. The reduction cell is responsible for reducing the dimensions of our SPD matrices and the normal cell contributes to a deeper and more complex network, without reducing its dimensionality. The following descriptions hold for both cell types.
Let $\mathcal{M} = \mathcal{S}^n_{++}$. Each node $\boldsymbol{X}^{(j)}_{\mathcal{M}}$ is a latent representation (feature map) formed of SPD matrices and each edge $(i, j)$ represents an SPD manifold constrained operation, $O^{(i,j)}_{\mathcal{M}}$, refer to table 1. Since we are dealing with a DAG, intermediate nodes can and, in our case, will be connected to only two previous nodes in the topological order. Considering this, we have to define a way to compute this intermediate transformation while remaining manifold consistent. We derive the intermediate node $\boldsymbol{X}^{(j)}_{\mathcal{M}}$, considering its $N$ predecessors $\mathcal{X}_N = \{\boldsymbol{X}^{(1)}_{\mathcal{M}}, ..., \boldsymbol{X}^{(N)}_{\mathcal{M}}\}$ and $\mathcal{O}_N = \{O^{(1,j)}_{\mathcal{M}}(\boldsymbol{X}^{(1)}_{\mathcal{M}}), ..., O^{(N,j)}_{\mathcal{M}}(\boldsymbol{X}^{(N)}_{\mathcal{M}})\}$, as a simple Fréchet mean :

$$\boldsymbol{X}^{(j)}_{\mathcal{M}} = \boldsymbol{\Psi}(\mathcal{O}_N) = \underset{\boldsymbol{X}^{(j)}_{\mathcal{M}}}{\text{argmin}} \sum_{i \le N} \delta^2_{\mathcal{R}} \left( O^{(i,j)}_{\mathcal{M}} \left( \boldsymbol{X}^{(i)}_{\mathcal{M}} \right), \boldsymbol{X}^{(j)}_{\mathcal{M}} \right) \tag{15}$$

This operation ensures that at each intermediate transformation, our computational graph preserves the geometric structure of the SPD manifold. Asides from this crucial fact, taking the Fréchet mean makes sure that intermediate SPD samples $\boldsymbol{X}^{(i)}_{\mathcal{M}}$ are close to $O^{(i,j)}_{\mathcal{M}}(\boldsymbol{X}^{(i)}_{\mathcal{M}})$ in the manifold $\mathcal{M} = \mathcal{S}^n_{++}$, for all predecessor nodes $i$.

## 4.2 Search space of manifold operations

Based on the basic SPD operations presented in section 3, we define several final operations that will constitute our search space, table 1. The first three operations inspired by [2] and [13], are based on a sequence of BiMap followed by SPD batch normalization layers. We consider three cases. Two cases where a ReEig activation operation is positioned at the front and at the end of the aforementioned two layers, and one where no activation is considered. Analogously to [22], we define a residual connection called Skip_normal that performs an identity transformation over the inputted SPD samples. Its corresponding operation in the reduction cell case follows a different procedure : It begins by decomposing input SPDs into smaller ones using BiMap operations and then creates the output SPDs using a diagonal block form of their singular value decompositions (SVD). We also define a special operation that returns the identity (origin in the SPD mainfold) as output, called None_normal. This operation is included to mimic a lack of connection between two nodes. Finally, we use the basic SPD pooling operations presented in section 3, refer to table 1.

## 4.3 The search procedure : Relaxation and optimization

### 4.3.1 Continuous relaxation and Weighted Fréchet mean

During the search procedure, connections between nodes occur as a weighted mixture of all candidate operations, i.e. several edges are considered between two subsequent nodes and respective outputs are mixed to provide one single latent representation. Once the search phase is over, a single discrete architecture can be obtained by replacing each mixture of operations with the preferred one by our optimization procedure. An essential part of our method is making the search space continuous so that the architecture can be optimized by gradient descent. We use a weighted Fréchet mean over all candidate operations, $\mathcal{O}_{N_e} = \{O_{\mathcal{M}}^{(1)}(\boldsymbol{X}_{\mathcal{M}}), ..., O_{\mathcal{M}}^{(N_e)}(\boldsymbol{X}_{\mathcal{M}})\}$, to perform such relaxation :

$$\begin{aligned}\bar{O}_{\mathcal{M}}\left(\boldsymbol{X}_{\mathcal{M}}\right) &= \boldsymbol{\Psi}_{\alpha}(\mathcal{O}_{N_e}) \\ &= \underset{\boldsymbol{X}_{\mathcal{M}}^{\mu}}{\operatorname{argmin}} \sum_{k=1}^{N_e} \alpha^k \delta_{\mathcal{M}}^2 \left(O_{\mathcal{M}}^{(k)}\left(\boldsymbol{X}_{\mathcal{M}}\right), \boldsymbol{X}_{\mathcal{M}}^{\mu}\right)\end{aligned} \tag{16}$$

where $N_e$ is the number of operations/edges, $\boldsymbol{\alpha} = \{\alpha^1, ..., \alpha^{N_e}\}$ and $\alpha^k$ is the weight of the $k^{th}$ operation, optimized by gradient descent. Since Equation (16) is a weighted Fréchet mean, it can be solved by the Karcher flow algorithm, Algorithm 1 or the inductive FM estimator, equation (8). Both approaches will be studied and compared in further sections. After relaxation, our search space is continuous and architecture weights $\boldsymbol{\alpha}$ can be learned so that higher weights imply better operations.
An important detail to be aware when computing and back-propagating through (16) is the convexity constrain over $\boldsymbol{\alpha}$, where $\alpha^k \geq 0$ and $\sum_{k \leq N_e} \alpha^k = 1$ have to be satisfied.

### 4.3.2 Satisfying the convexity constrain over $\boldsymbol{\alpha}$

In the first place, following DARTS [22], we satisfy this constrain applying a softmax over all values in $\boldsymbol{\alpha} = \{\alpha_1, ..., \alpha_{N_e}\}$ :

$$\alpha_i^{'} = \frac{\exp\left(\alpha_i\right)}{\sum_{k \leq N_e} \exp\left(\alpha_k\right)} \tag{17}$$

where $\alpha_i^{'}$ is the weight of the $i^{th}$ edge after applying a softmax function.
We then provide a study using ManifoldNet [4] approach to deal with the convexity constrain. For this matter, we first use a simple weight normalization to ensure that the weights are within $[0, 1]$. Then, to satisfy the unit summation constrain, we add a weight penalty, $l_{\boldsymbol{\alpha}}$, to the loss function for all weighted mixtures of our network. We define :

$$l_{\boldsymbol{\alpha}} = \sum_{k \leq N_e} \alpha^k - 1 \tag{18}$$

### 4.3.3 Bi-level optimization

For simplicity, in the following sections we will refer to $\alpha$ as the encoding of the architecture. SPDNetNAS architecture search aims at jointly learning the optimal architecture $\alpha^*$ and its respective optimal network weights $w^*$. To do so, we rely on a bi-level optimization problem. The lower-level part corresponds to the minimization of the network weights $w$ with respect to the training set $\mathcal{L}_{train}(w, \alpha)$. The upper-level part aims at minimizing the validation loss $\mathcal{L}_{val}(w^*, \alpha)$ associated with the lower-level optimal weights $w^*$.
Mathematically,

$$\begin{array}{ll} \min_\alpha & \boldsymbol{\mathcal{L}}_{val}\left(w^*(\alpha), \alpha\right) \\ \text{subject to} & w^*(\alpha) = \operatorname{argmin}_w \boldsymbol{\mathcal{L}}_{train}(w, \alpha) \end{array} \tag{19}$$

Once this complex bi-level optimization solved, we can select the best architecture by maintaining the operations with the highest alphas between nodes. The second stage of our work consists in training the optimal selected architecture from scratch. Before that, let's dive into the details of the proposed optimization and how we can adapt it for the Riemannian case.
Due to the high complexity of the proposed optimization, solving (19) with exact gradients results in extremely expensive computations. Following [22], we propose an efficient and reliable approximation scheme.
We define the following gradient approximations :

$$\nabla_\alpha \boldsymbol{\mathcal{L}}_{\text{val}}\left(w^*(\alpha), \alpha\right) \tag{20a}$$
$$\approx \nabla_\alpha \boldsymbol{\mathcal{L}}_{\text{var}}\left(w - \eta \nabla_{\text{w}} \boldsymbol{\mathcal{L}}_{\text{train}}(w, \alpha), \alpha\right) \tag{20b}$$
$$= \nabla_\alpha \boldsymbol{\mathcal{L}}_{val}(\tilde{w}, \alpha) - \eta \nabla^2_{\alpha, w} \boldsymbol{\mathcal{L}}_{train}(w, \alpha) \nabla_{\tilde{w}} \boldsymbol{\mathcal{L}}_{val}(\tilde{w}, \alpha) \quad \textit{(chain rule)} \tag{20c}$$

where $\eta$ is the inner optimization learning rate and $\tilde{w} = \Re\left(w - \eta \tilde{\nabla}_w \boldsymbol{\mathcal{L}}_{train}(w, \alpha)\right)$, considering $\Re$ and $\tilde{\nabla}_w$ as a Riemannian retraction operator and a Riemannian gradient respectively.
A single step is performed on the lower-level part of equation (19) to find an approximation of $w^*$ and derive equation (20a). Then, we derive equation (20b) from (20a) applying the famous chain rule. Second term of this last equation relies on second order differentials, practically too expensive to compute. We are able to reduce considerably the high complexity of the aforementioned term by using a finite difference approximation :

$$\nabla^2_{\alpha, w} \boldsymbol{\mathcal{L}}_{train}(w, \alpha) \nabla_{\tilde{w}} \boldsymbol{\mathcal{L}}_{val}(\tilde{w}, \alpha) = \left(\nabla_\alpha \boldsymbol{\mathcal{L}}_{train}\left(w^+, \alpha\right) - \nabla_\alpha \boldsymbol{\mathcal{L}}_{train}\left(w^-, \alpha\right)\right) / 2\delta \tag{21}$$

where $\delta$ [1] is a small scalar and $w^{\pm} = \Re\left(w \pm \delta \tilde{\nabla}_{\tilde{w}} \boldsymbol{\mathcal{L}}_{val}(\tilde{w}, \alpha)\right)$.
It is important to note that gradients with respect to $w$ must respect the geometry of SPD manifold when the respective weight is a SPD matrix. As discussed in section 4, our network is expected to deal with both scalar and SPD matrices as parameters. To remain geometrically consistent throughout the learning process, we consider a mix optimizer. For further understanding on how to optimize through SPDNets kindly refer to Huang et al. and Brooks work [13],[2].

[1]We set $\delta = 0.01 / \left\|\nabla_{\tilde{w}} \mathcal{L}_{val}\left(\tilde{w}, \alpha\right)\right\|_2$ as in [22]

# 5 Experiments and Results

To validate the effectiveness of SPDNetNAS, we discuss its performance on three synthetic datasets, RADAR [8], HDM05 [24] and AFEW [9] datasets. We compare our performance to previous baselines and provide additional interesting experiments.

## 5.1 Datasets

### 5.1.1 RADAR [8]: A drone recognition dataset

RADAR dataset [8] is based on drone micro-Doppler radar classification. A wave is emitted and reflected on a desired target, which results in a radar signal sample. This signal is represented by a time-series signal that can be split in windows of length $n = 20$. From these windows, we can pool a single covariance matrix of size $20 * 20$ that has to be classified by our SPD network. The original dataset called NATO features 10 classes of drones, unfortunately, we can only consider a similar synthetic form (due to confidentiality) representing only 3 different classes. The synthetic dataset RADAR has 1000 samples per class, from which we consider 50 % for training and two times 25% for validation and test set. Our network (precisely a reduction cell) will perform a unique dimensionality reduction from input SPD matrices of size $20 * 20$ to matrices of size $12 * 12$.

### 5.1.2 HDM05 [24] : An action recognition dataset

HDM05 [24] is one of the largest-scale skeleton-based human action recognition datasets, which contains 130 action classes. For fair comparison with the baselines, we extract a subset of 2083 equally distributed data samples among 117 classes. As done in [11], from each sequence we pool a covariance descriptor of size $93 * 93$ that lies in the SPD manifold. Similarly to RADAR set up, we split the dataset into 50%, 25 % and 25% for train, validation and test set respectively. In this scenario, our reduction cell is constructed to reduce dimensions from 93 to 30 as in [2].

### 5.1.3 AFEW [9] : Emotion recognition dataset

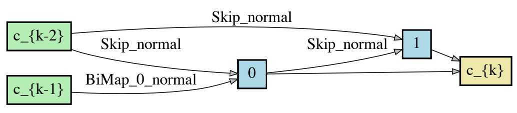
We finally use the famous Acted Facial Expression in the Wild (AFEW) [9] dataset for emotion recognition as our largest-scale experiment. This database gathers 1345 facial expressions videos from actors in real-life scenario movies. Each video is classified among 7 distinct classes. Following [13], each facial frame is normalized to an image of size $20 * 20$ and a covariance matrix of size $400 * 400$ is pooled out to represent each video. To report fair results, we split the dataset following [2] and [13] set up. Here, a single dimensionality reduction is performed by a reduction cell, where we go from matrices of size 400 to 50.

## 5.2 Training details

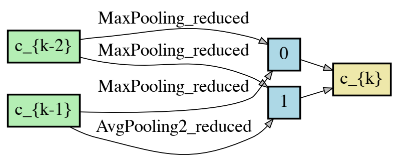
We recall the training procedure of SPDNetNAS. The first stage of our method, called the search phase, consists in finding the optimal architecture provided by our bi-level optimization, based on both validation and training sets. Once an optimal model is found, we re-train the model from scratch and report the statistical performance. We will name this final training procedure, the training phase. For both phases we consider the same kind of cells (normal and reduction), which includes 2 input nodes, 2 intermediate nodes and 1 output node. Input nodes of each cell correspond to the outputs of the previous two cells and are pre-processed by fixed BiMap operations. This pre-processing is necessary to work with identical channel-wise dimensions. All networks are trained with batch size of 30 and learning rate of 0.025, except for AFEW, where we increase the learning rate to 0.05. We choose, for all experiments and phases, to stack a normal cell on top of a reduction cell. This two-cell choice aroused as the optimal design after several observations. We believe that our model might not benefit from stacking more than two cells due to its demanding optimization problem and/or observed overfitting. For comparison and result documentation (section 5.3), we decided to rely on the Karcher flow algorithm as Fréchet mean solver and on a softmax over architecture weights to deal with the convexity constrain.

## 5.3 Results and comparison against baselines

In this section we document results found and several performance comparisons with two SPD networks.

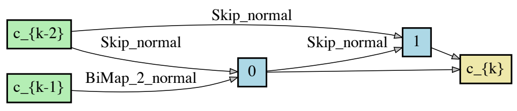


(a) Normal cell

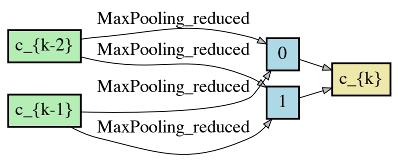


(b) Reduction cell

Figure 4: (a)-(b) Optimal Normal cell and Reduction cell for RADAR dataset



(a) Normal cell



(b) Reduction cell

Figure 5: (a)-(b) Optimal Normal cell and Reduction cell for HDM05 dataset

### 5.3.1 Search and training results

We first studied statistical performance on HDM05 and RADAR datasets since they are both considered as our small-scale datasets compared to AFEW. The first part of our training (the search phase) is the most computationally expensive one due to its complex

optimization problem. 1 CPU[2] day is required to train SPDNetNAS on RADAR [8] for 200 epochs and 3 CPU days on HDM05 [24] for 100 epochs. At the end of the search, the architecture obtained is expected to be the optimal. However, we noticed that different runs end up with different local minimum. It is crucial to repeat the search process with different seeds and select the best cells based on the validation performance.
We followed this procedure and selected the best cells for RADAR [8] and HDM05 [24]. Please refer to figure 4 and 5. For AFEW [9], running the search procedure happens to be extremely long due to its high dimensional inputs. Considering this, we decided to study the transferability of optimal architectures obtained from RADAR and HDM05 [24] to our large-scale dataset AFEW [9]. To do so, we selected several local minimum architecures from different runs for both RADAR [8] and HDM05 [24] and trained each model with our facial emotion recognition dataset. The transferred cell showing best accuracy results can be found in the Appendices, figure 10. The latter is an architecture coming from HDM05 training search.
We also provide a short convergence analysis on RADAR and HDM05 in figure 6. We observe overfitting for both HDM05 [24] and AFEW [9], illustrated by an important gap between training and validation loss functions. This observation has been found in Huang et. al [13] work and endorse the fact that both datasets might lack data. To the contrary, validation and training losses for RADAR [8] follow a similar convergence trend.

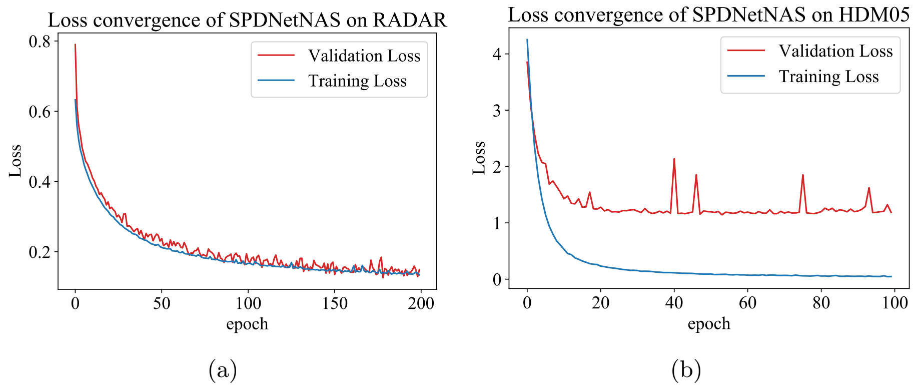


Figure 6: (a) Loss function curve showing the loss values over 200 epochs for the RADAR dataset [8], (b) Loss function curve showing the loss values over 100 epochs on the HDM05 dataset [24]

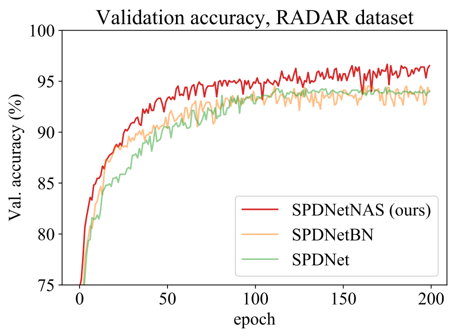
### 5.3.2 Comparison

We compare SPDNetNAS with two related SPD networks :

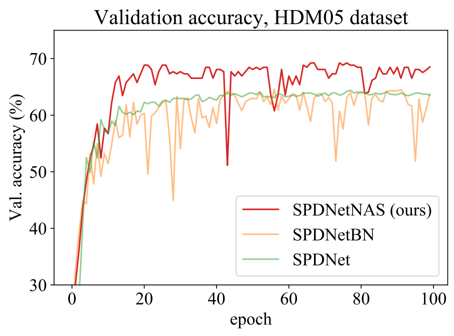
- The original and first deep learning network for geometric data lying on SPD manifolds proposed by Huang et al. [13] called SPDNet.
- A SPD network based on SPDNet [13] with the new Riemannian batch normalization layer proposed by Brooks et al. [2] that we call SPDNetBN.

[2]Our training does not benefit from GPU acceleration since most of our operations rely on singular value decomposition (SVD). Slow implementation of SVD and eigenvalue decomposition (EIG) is a current open problem on CUDA platforms, its CPU counterpart, far from ideal, speeds the process considerably when matrix dimensions are not substantial.

We follow SPDNetBN [2] experimental set up for all experiments for fair comparison. Table 2 shows test results for HDM05, RADAR and 10 % of RADAR training dataset. SPDNetNAS clearly outperforms all previous baseline methods. We also provide in table 2 test results with a random SPDNetNAS cell architecture (RAND) to show that the search optimization is working as expected. We finally report validation curves for all comparison methods, figure 7.



(a)



(b)

Figure 7: (a) Performance on the validation set for all baselines and SPDNetNAS, RADAR dataset [8] (b) Performance on the validation set for all baselines and SPDNetNAS, HDM05 dataset [24]

Table 2: Test accuracy comparison (%) of our method with existing SPDNets for RADAR and HDM05. RAND: random search, (*) Accuracy with 10% training data.

| Dataset ↓ / Method | SPDNet [13] | SPDNetBN [2] | SPDNetNAS (RAND) | SPDNetNAS | NAS Search Time |
|---|---|---|---|---|---|
| RADAR [8] | 93.21% ± 0.39 | 92.13% ± 0.77 | 95.49% ± 0.08 | **96.47% ± 0.11** | 1 CPU day |
| HDM05 [24] | 61.60% ± 1.35 | 65.20% ± 1.15 | 66.92% ± 0.72 | **68.74% ± 0.93** | 3 CPU days |
| 10% of RADAR | 82.73% ± 0.14 | 84.84% ± 0.73 | 85.90% ±0.30 | **87.57%± 0.08** | NA* |

Table 3: Performance of transferred SPDNetNAS Network architecture in comparison to existing SPD Networks on the AFEW dataset [9]. RAND symbolizes random architecture transfer from our search space.

| SPDNet [13] | SPDNetBN [2] | Ours (RAND) | Ours (RADAR) | Ours (HDM05) |
|---|---|---|---|---|
| 33.17% | 35.22% | 32.88% | **35.31** % | **38.01%** |

## 5.4 Additional experiments

In this section we present two additional experiments. The first one is a performance and efficiency comparison between the inductive FM estimator (iFME) and the Karcher flow algorithm. The second is about the weighted penalty as convexity constrain for the optimization.

### 5.4.1 Iterative Fréchet mean solvers

Our network dependent highly on Fréchet Mean computations. From the weighted mixture of operations between nodes to the derivation of intermediate nodes, both compute the Fréchet mean of a set of points on the SPD manifold. As mentioned in 2.2.2, there is no closed form solution when the number of input samples is bigger than 2. We can only compute an approximation using the famous Karcher flow algorithm or the aforementioned inductive FM estimator (iFME). Previous experiments and documented results have been

achieved using Karcher flow, we will now focus on the iFME. Ideally, for fair comparison between both solvers, we should search for a new optimal architecture using the iFME and train the latter from scratch. Unfortunetely, while looking for new architectures with the iFME, our optimization was not working as expected and architecture weights $\boldsymbol{\alpha}$ are not correctly learned. We explain this poor learning ability by the fact that architecture weights are hardly varying after several epochs.
Due to lack of any optimal architecture, we transfer selected architectures from Karcher flow experiments (RADAR [8] and HDM05 [24]) and evaluate iFME performance with these. We present in table 4 results using the iFME algorithm.

Table 4: Test and Training Performance of SPDNetNAS using the Karcher flow algorithm [17] and the iFME [7] to compute Fréchet means.

| Dataset / Method | Test/Train accuracy | Karcher flow [17] | iFME [7] [2] |
|---|---|---|---|
| RADAR [8] | test acc. | **96.47% ± 0.08** | 68.13% ± 0.64 |
| | train acc. | **100%** | 65.36% ± 1.21 |
| HDM05 [24] | test acc. | **68.74% ± 0.93** | 56.85% ± 0.17 |
| | train acc. | **100%** | 81.25% ± 0.86 |

### 5.4.2 The convexity constrain on the Fréchet mean

Finally, as already mentioned in section 4.3.2, a convexity constrain over architecture weights $\boldsymbol{\alpha}$ is crucial since our weighted mixture of operations rely on a weighted Fréchet mean. For each mixture of operations during the search phase, $\boldsymbol{\alpha} = \{\alpha^1, ..., \alpha^{N_e}\}$ must satisfy the following conditions :

$$\alpha^k \geq 0 \tag{22a}$$
$$\sum_{k \leq N_e} \alpha^k = 1 \tag{22b}$$

This ensures that we are properly computing a weighed Fréchet mean and the most powerful operation, defined by the highest weight value, lie on the $[0, 1]$ interval.
Previous experiments were done considering a mixture of softmaxes as in DARTS [22]. We also searched for optimal architectures using a weighted penalty over all architecture weights following ManifoldNet [4] (section 4.3.2) and found the exact same optimal architecture previously found, figure 4.
To provide more details on how architecture weights are being optimized during the search phase of SPDNetNAS, we studied their behaviour for RADAR dataset [8] and HDM05 [24]. For simplicity, we just show how operation weights connecting the first input node ($c_{k-1}$) to the first (node 0) and the second (node 1) intermediate node of our reduction and normal SPD cell, change. Please refer to figures 15, 16 and 17.
As expected, all weights lie between $[0, 1]$ and their respective summation in a single cell are equal to 1. One clear observation is how the weighted softmaxes (figure 15) vary in a smoother way compared to the weighted penalty approach. Other than that, both methods end up with the same selected top operations. In the next section we discuss some insights extracted from these experiments.

# 6 Discussion

This section focuses on the discussion and analysis of previous experiments and proposed methods. Possible improvements and future work are also considered.

## 6.1 Search and Optimization

The most important part of our method is the architecture search process. This part relies on a complex bi-level optimization that selects best architectures based on its validation performance. How can we ensure that selected architectures are the optimal ones? Are $\boldsymbol{\alpha}$ weights correctly optimized by the Riemannian optimizer?
First important observation noticed for all considered datasets is the fact that architectures with low number of parameters are preferred. These kind of architectures, with mostly skip connections or pooling layers, are selected since they give high validation performance at an early stage of the training. Changes only happen after several epochs, when the validation performance starts to stabilize and BiMap layers emerge occasionally.
From figure 15, 16 and 17, we also notice that for normal cells, skip connections are mostly selected and their respective weights converge rapidly to 1. Remaining operation weights saturate to 0, which disables the opportunity for any change. On the other hand, as observed in the second connection of all normal cells, weights from none operations quickly gain strength and remaining weights converge to values close to 0. Since none operations are excluded when retaining the top operation, an issue emerges since we end up retaining operations whose weights are almost null. In this last case, selected top operations are chosen in a quasi random way, which makes the search process unsuccessful. Surprisingly, we notice the exact same phenomena for both RADAR and HDM05 normal cells. In both cases, all selected operations are skip connections except the second one, where none operations are preferred and BiMaps end up being selected. Reduction cells don't suffer from this anomaly. For reduction cells, all operations weights are in the interval $[0.1, 0.2]$, which means that they all have some importance to improve the validation performance of the model. For HDM05 case, figure 16, we can see that weights in the reduction cell quickly converge and Max poolings are preferred. However, in the RADAR dataset case, and for both types of constrains (figure 15 and 17), we notice that some weights might need more time to achieve convergence. Unfortunately, search training is really expensive for both datasets.
Further study on the bi-level optimization needs to be done to deal with the previously mentioned optimization issues.

## 6.2 SPD Batch Normalization

Brooks et al. in [2] proposed a new Riemannian batch normalization layer for SPD networks. The proposed new operation is supposed to provide stability in the learning process and improvement in performance. Additionally, it is stated that adding these kind of operations to SPDNet [13] provides better robustness to lack of data.
We reproduced results of SPDNetBN [2] for all considered datasets (figure 7) and noticed that adding a batch normalization layer did not improve the performance in most cases and the training time increased. As stated in table 2, only for HDM05 and 10% of RADAR, SPDNetBN outperformed the previous SPDNet from Huang et al. [13]. We also observe in figure 7 that adding a riemannian batch normalization layer augments considerably the

variance of our validation performance.
For future improvements, we suggest to analyze and review the Riemannian batch normalization layer proposed by [2]. Chakraborty et al. proposed in [3] several interesting Riemannian normalization techniques for manifold valued data.
**Algorithm 3** describes the Riemannian batch normalization for SPD matrices proposed by [3].

**Algorithm 3** Riemannian batch normalization on a batch of SPD matrices from [3]

```
**Input**: batch of $N$ SPD matrices $\{X_i\}_{i\leq N}$, running mean $\mathfrak{G}_{\mathcal{S}}$, bias $G \in GL(n)$,
positive diagonal scaling matrix $S \in R^{m*m}$
1 : $\mathfrak{G}_{\mathcal{B}} \leftarrow \mathbf{FM}\left(\{X_i\}_{i\leq N}\right)$        (Compute batch mean)
2 : $\mathfrak{G}_{\mathcal{S}} \leftarrow \mathbf{FM}\left(\{\mathfrak{G}_{\mathcal{S}}, \mathfrak{G}_{\mathcal{B}}\}\right)$        (Running mean)
3 : for $i \leq N$ do
4 :     $\bar{X}_i \leftarrow \Gamma_{\mathfrak{G}_{\mathcal{B}} \to I_d}(X_i)$        (Center batch)
5 :     $\tilde{X}_i \leftarrow Exp(\kappa^{-1}(S\kappa(\boldsymbol{x}_i)))$        (Scale)        where $\boldsymbol{x}_i = Log_I(\tilde{X}_i)$
6 :     $\tilde{X}_i \leftarrow G\bar{X}_iG^T$        (Group action - Bias batch)
7 : end for
**Return** normalized batch $\left\{\tilde{X}_i\right\}_{i\leq N}$
```

(23)

Where $\forall M \in \mathcal{M}$, $\kappa : \mathcal{T}_M\mathcal{M} \to \mathbb{R}^m$ is an isomorphism from the tangent space at $M$ to the Euclidean space $R^m$.

Several differences between **Algorithm 1** and **3** can be observed. Third algorithm leverages the use of a scaling matrix that enables to scale the variance without changing the batch mean from $I$. The origin of this difference lies at the considered Gaussian distributions used for both cases. In [2] and for **Algorithm 1**, they adopt a Gaussian density derived from the definition of maximum entropy on exponential families using the geometric information on the cone of SPD matrices. In this setting, the notion of variance is excluded from the definition and a single bias parameter is needed. On the other hand, in [3] and **Algorithm 3**, the considered distribution follows [25] Gaussian distribution, where both normalization constants and covariance matrices are defined.
No experiments have been done using **Algorithm 3**. We suggest to analyze and implement this new normalization technique to further improve our method.

## 6.3 Karcher flow or iFME?

We provide in section 5.4.1 experiments with two different Fréchet mean solvers. The first and most famous is the Karcher flow algorithm [17]. With this last approach, we outperformed the current baseline methods on all studied datasets. The second iterative solver called the inductive Fréchet mean estimator (iFME) [7] had issues learning the appropriate Fréchet weights from all mixture of operations while searching for architectures. Test accuracies for both RADAR [8] and HDM05 [24] are reported in table 4. Unfortunately, these are still quite poor, even compared to the baselines. We also notice that the model is not able to overfit the train set as training accuracies remain quite small compared to Karcher flow ones.
Poor performance on both training and test set is mostly due to the model's inability to correctly optimize through the iFME. In [7], it is also stated that as the number of SPD samples to compute the Fréchet mean goes to infinity, the iFME converges to the Fréchet

expectation. In our case, we are computing Riemannian averages of a maximum of 6 SPD inputs. This number might be too small to ensure a small error between the true Fréchet mean and the iFME. Finally, a last possible influencing factor to take into consideration is the size of our SPD samples. In most studied applications that leverage the use of the iFME, [5];[4], the dimensions of the SPD samples are considerably small (n=3 for [5]) compared to our case, where the minimum dimension is of size 20 for RADAR [8]).

## 7 Conclusion

This study proposes a novel Differentiable Neural Architecture Search (NAS) method for networks classifying SPD manifold-valued data. We propose a new computational SPD cell where a search space of Riemannian operations are continuously relaxed to enable an efficient architecture search procedure. This procedure leverages the use of a bi-level Riemannian optimizer to search and train for the best possible network architecture. We evaluate our method on three classification tasks and show performance improvements for all of them. In addition, we provide a discussion on the downsides and future possible improvements to make our model more robust.
Other interesting future directions would be to adapt this method for end-to-end training or to consider different Riemannian manifolds such as the Grassmann manifold.

## A List of figures

## B List of tables

## References

[1] Automl. https://www.automl.org/automl/literature-on-neural-architecture\-search/. Accessed: 30-05-2020.

[2] D. Brooks, O. Schwander, F. Barbaresco, J.-Y. Schneider, and M. Cord. Riemannian batch normalization for spd neural networks. In *Advances in Neural Information Processing Systems*, pages 15463–15474, 2019.

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

[4] R. Chakraborty, J. Bouza, J. Manton, and B. C. Vemuri. Manifoldnet: A deep network framework for manifold-valued data. *arXiv preprint arXiv:1809.06211*, 2018.

[5] R. Chakraborty, J. Bouza, J. Manton, and B. C. Vemuri. A deep neural network for manifold-valued data with applications to neuroimaging. In *Information Processing in Medical Imaging*, pages 112–124, 2019.

[6] R. Chakraborty, J. Bouza, J. Manton, and B. C. Vemuri. A deep neural network for manifold-valued data with applications to neuroimaging. In *International Conference on Information Processing in Medical Imaging*, pages 112–124. Springer, 2019.

[7] R. Chakraborty and B. C. Vemuri. Recursive fréchet mean computation on grassmannian and its applications to computer vision. In *ICCV*, 2015.

[8] V. C. Chen, F. Li, S.-S. Ho, and H. Wechsler. Micro-doppler effect in radar: phenomenon, model, and simulation study. *IEEE Transactions on Aerospace and electronic systems*, 42(1):2–21, 2006.

[9] A. Dhall, R. Goecke, S. Lucey, and T. Gedeon. Static facial expressions in tough conditions: Data, evaluation protocol and benchmark. In *1st IEEE International Workshop on Benchmarking Facial Image Analysis Technologies BeFIT, ICCV2011*, 2011.

[10] M. Fréchét. Les éléments aléatoires de nature quelconque dans un espace distancié. *IEEE Transactions on Biomedical Engineering*, 10(4):215–310, 1948.

[11] M. T. Harandi, M. Salzmann, and R. Hartley. From manifold to manifold: Geometry-aware dimensionality reduction for spd matrices. In *European conference on computer vision*, pages 17–32. Springer, 2014.

[12] K. He, H. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

[13] Z. Huang and L. Van Gool. A riemannian network for spd matrix learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[14] Z. Huang, R. Wang, S. Shan, and X. Chen. Learning euclidean-to-riemannian metric for point-to-set classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1677–1684, 2014.

[15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[16] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, pages 2016–2025, 2018.

[17] H. Karcher. Riemannian center of mass and mollifier smoothing. *Communications on pure and applied mathematics*, 30(5):509–541, 1977.

[18] S. Kumar. Jumping manifolds: Geometry aware dense non-rigid structure from motion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5346–5355, 2019.

[19] S. Kumar, A. Cherian, Y. Dai, and H. Li. Scalable dense non-rigid structure-from-motion: A grassmannian perspective. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 254–263, 2018.

[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 *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

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

[22] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[23] M. Moakher. A differential geometric approach to the geometric mean of symmetric positive-definite matrices. *SIAM Journal on Matrix Analysis and Applications*, 26(3):735–747, 2005.

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

[25] X. Pennec, P. Fillard, and N. Ayache. A riemannian framework for tensor computing. *International Journal of computer vision*, 66(1):41–66, 2006.

[26] 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*, volume 33, pages 4780–4789, 2019.

[27] A. Sanin, C. Sanderson, M. Harandi, and B. Lovell. Spatio-temporal covariance descriptors for action and gesture recognition. In *IEEE Workshop on Applications of Computer Vision (WACV)*, pages 103–110, 2013.

[28] D. Tosato, M. Farenzena, M. Spera, V. Murino, and M. Cristani. Multi-class classification on riemannian manifolds for video surveillance. In *European conference on computer vision*, pages 378–391. Springer, 2010.

[29] O. Tuzel, F. Porikli, and P. Meer. Pedestrian detection via classification on riemannian manifolds. *IEEE transactions on pattern analysis and machine intelligence*, 30(10):1713–1727, 2008.

[30] X. P. V. Arsigny, P. Fillard and N. Ayache. Geometric means in a novel vector space structure on symmetric positive-definite matrices. *SIAM Journal of Matrix Analysis and Applications*, 26:328–347, 2007.

[31] R. Wang, H. Guo, L. S. Davis, and Q. Dai. Covariance discriminative learning: A natural and efficient approach to image set classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2496–2503. IEEE, 2012.

[32] L. Yang, M. Arnaudon, and F. Barbaresco. Riemannian median, geometry of covariance matrices and radar target detection. In *The 7th European Radar Conference*, pages 415–418. IEEE.

[33] J. W. Z. Huang and L. V. Gool. Building deep networks on grassmann manifolds. *arXiv preprint arXiv:1611.05742*, 2016.

[34] X. Zhen, R. Chakraborty, N. Vogt, B. B. Bendlin, and V. Singh. Dilated convolutional neural networks for sequential manifold-valued data. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 10621–10631, 2019.

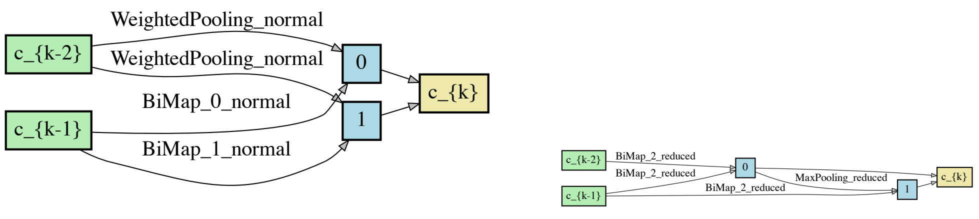
[35] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[36] 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*, pages 8697–8710, 2018.

## C Appendices

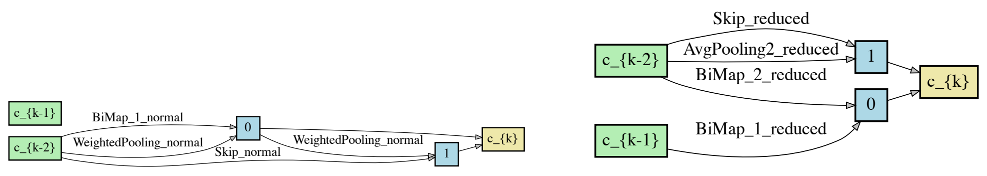
### C.1 Additional architectures

We add several plots of selected architectures from several experiments. Figure 10 corresponds to the best architecture selected from HDM05 search and transferred to AFEW dataset. Figure 8 and 9 are randomly selected architectures used to show that the optimization procedure is working well.



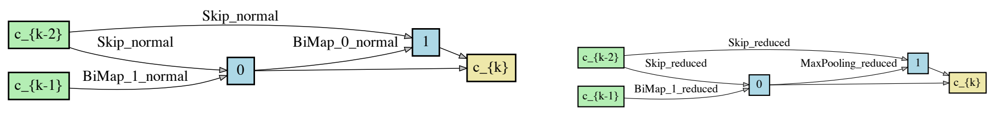
(a) Normal cell (b) Reduction cell

Figure 8: (a)-(b) RANDOM Normal cell and Reduction cell for HDM05 dataset



(a) Normal cell (b) Reduction cell

Figure 9: (a)-(b) RANDOM transferred Normal cell and Reduction cell for RADAR dataset



(a) Normal cell (b) Reduction cell

Figure 10: (a)-(b) Best transferred Normal cell and Reduction cell for AFEW dataset

## C.2 Detailed overview of SPD operations

In this section we provide a brief description of new introduced SPD operations and a detailed overview of how intermediate nodes and the mixture of operation are derived.

### C.2.1 Weighted Riemannian Pooling

Figure 11 provides an intuition behind the weighted Riemannian Pooling. This operation is similar to any euclidean fully connected layer but each latent neuron is derived as a weighted Fréchet mean of all previous input samples. For example, weights $w_{11}$ and $w_{12}$ correspond to the normalized weights (by softmax) for each input SPD matrix. The first SPD output neuron is then computed as a weighted Fréchet mixture of all SPD inputs with their respective weight. The weights are learned as part of the networks optimization problem.

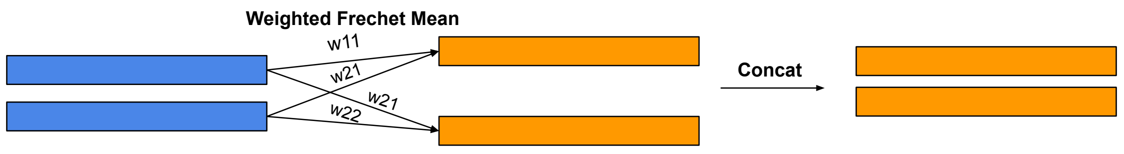


Figure 11: Weighted Riemannian Pooling

### C.2.2 Average and Max Pooling

Average and max pooling operations rely on Riemannian projections to their tangent bundle. We first perform a LogEig map on the SPD matrices to project them to the Euclidean space. Next, we perform average and max pooling on these Euclidean matrices similar to classical convolutional neural networks. We further perform an ExpEig map to project the Euclidean matrices back on the SPD manifold, please refer to figure 12.

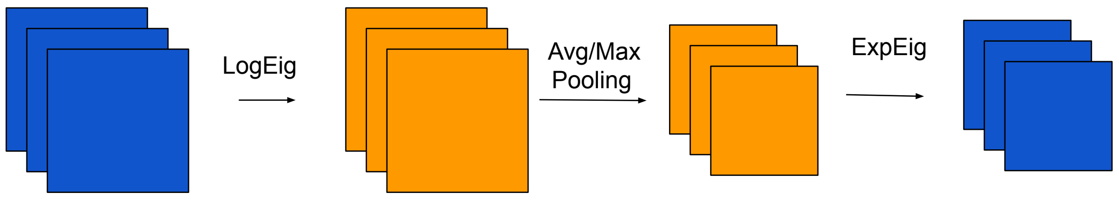


Figure 12: Avg/Max Riemannian Pooling

### C.2.3 Skip Reduced

Since normal Riemannian skip connections (1) can't reduce the dimensions of their respective SPD inputs we have to define an analogous operation for the reduction cell. We first use a BiMap layer to map the input channel to a reduced dimension SPD. Then we perform an SVD decomposition on the two SPDs followed by concatenating the Us, Vs and Ds obtained from SVD to block diagonal matrices. Finally, we compute the output by multiplying the block diagonal U, V and D computed before.

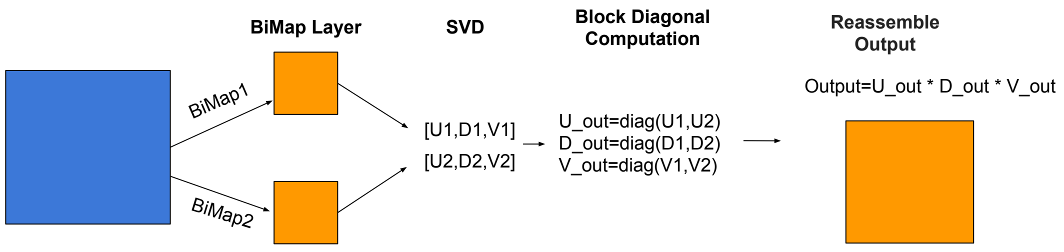


Figure 13: Skip Reduced connection

### C.2.4 Mixed Operation on SPDs

Figure 14 shows an intuition of the mixed operation repeatedly mentioned throughout our study. We simplify the example by taking 3 nodes (two input nodes and one output node) and two candidate operations. Input nodes have two channels (SPD matrices). We perform channel-wise weighted Fréchet mean between the result of each operation (edge), where weights $\alpha$'s are optimized by the bi-level architecture search optimization. Output node 3 is formed by concatenating both mixed operation outputs, resulting in a four channel node.

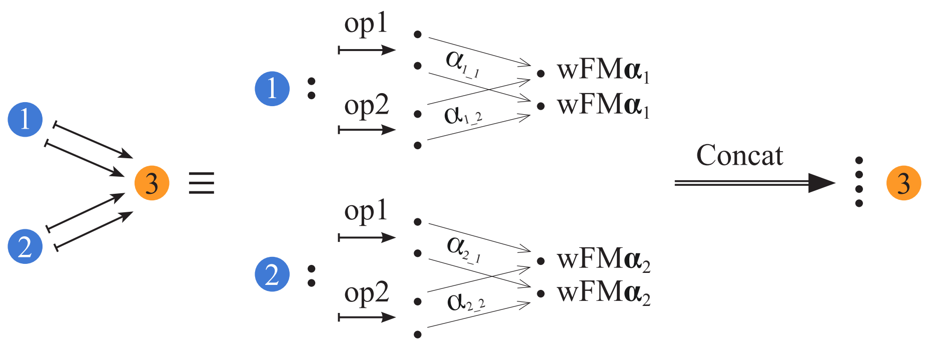

op1
op2
op1
op2
$\alpha_{1\_1}$
$\alpha_{1\_2}$
$\alpha_{2\_1}$
$\alpha_{2\_2}$
wFM$\boldsymbol{\alpha}_1$
wFM$\boldsymbol{\alpha}_1$
wFM$\boldsymbol{\alpha}_2$
wFM$\boldsymbol{\alpha}_2$
Concat
1
2
3
≡

Figure 14: Detailed overview of mixed operations.

## C.3 Analysis of architecture weights

Figure 15, 16 and 17 plot some architecture weights $\boldsymbol{\alpha}$ from normal and reduction cells optimized by the bi-level optimizer during the search phase. More specifically, we consider the first and second connection of the SPD cell, considered as the edges between first input node and first/second intermediate node. Figure 15 and 17 are from RADAR dataset and differ on the convexity constrain applied to the architecture weights. Figure 15 correspond to DARTS [22] softmax over weights and figure 17 correspond to ManifoldNet [4] constrain through a weighted penalty. Figure 16 correspond to softmaxes applied over HDM05 architecture weights.

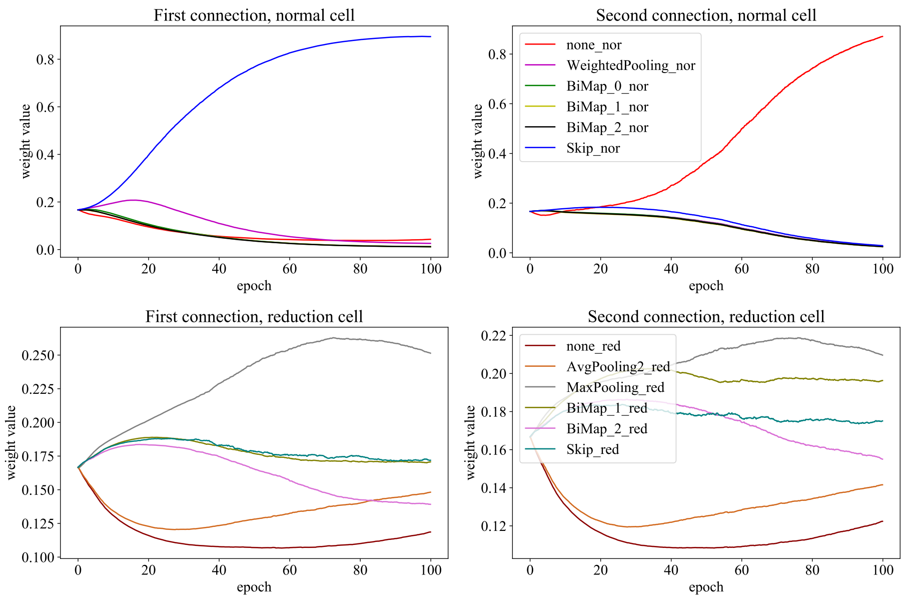


Figure 15: $\boldsymbol{\alpha}$ weights analysis from first connection for normal and reduction cell for RADAR [8]. Softmax applied to the weights.

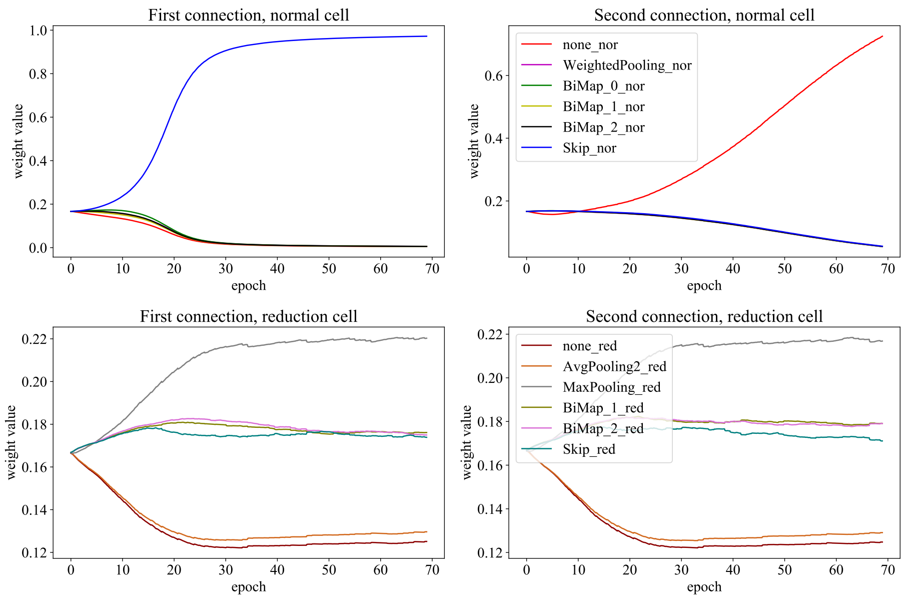


Figure 16: $\boldsymbol{\alpha}$ weights analysis from first connection for normal and reduction cell for HDM05 [24]. Softmax applied to the weights.

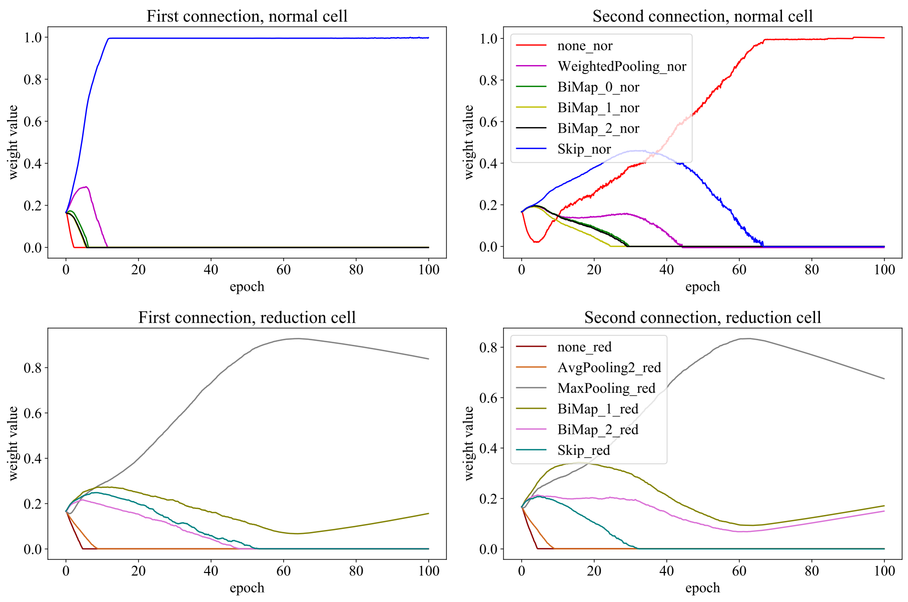


Figure 17: $\boldsymbol{\alpha}$ weights analysis from first connection for normal and reduction cell for RADAR [8], weights constrained by weighted penalty.