ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Neural Architecture Search of SPD Manifold Nets and its End to End Extension

Research in Data Science

Rhea Sukthanker, Department of Computer Science

July 10, 2020

Advisors: Dr. Zhiwu Huang, Dr. Suryansh Kumar, Prof. Dr. Luc Van Gool

CVL Lab: Department of Information Technology and Electrical Engineering, ETH Zürich

Acknowledgements

I would like to express my deep gratitude to Dr. Zhiwu Huang and Dr. Suryansh Kumar for their patient guidance and constructive suggestions throughout this semester project. I would like to offer my special thanks to Prof. Dr. Luc Van Gool and the CVL lab.

Abstract

Neural Architecture Search has gained popularity in Deep Learning. This is mainly because with the abundant number of architecture designs available for tasks like image classification eg: VGGNet, Resnet etc., it becomes increasingly important to automate the procedure for architecture search on neural networks. The most promising of such architecture search methods is DARTS which performs a differential architecture search on given candidate operations which compose a neural network. DARTS is primarily designed to operate on Euclidean data eg: Images. However it is well known in literature that operating on the SPD Manifolds can prove to be very beneficial for applications like MRI, Drone Recognition etc. In this project we propose Neural Architecture Search on SPD Manifold Networks **(NeurIPS 2020 submission)**. In addition we propose an end to end approach to extract the corresponding SPD matrix automatically from the video frames for more effective learning on the AFEW dataset for Emotion Recognition.

Contents

Chapter 1

Introduction

The foundation of deep learning and its success in areas like image recognition, video classification, and other areas in natural language understanding has been design of very carefully handcrafted architectures for eg: VGGNet [51], AlexNet [34], the Transformer architecture [56] etc. Thus though neural networks are automatic feature extractors the architecture of the network is still handcrafted by experts. Thus it becomes very much necessary to automate the procedure of architecture design by searching over a set of architectures. Recently there has been a lot of progress along this direction of research broadly known as the area of AutoML [65][47] [38][39][4][10][7][20] [12][46][44][32].

However these **Neural Architecture Search (NAS)** algorithms are very often proposed just to operate on Euclidean data as an input. This is clearly not desirable as it is well-known that the datasets typically obtained in computer vision lie on a much richer geometries (eg: SPD geometries, Grassmanian geometries etc). Manifold-valued data representation such as symmetric positive definite (SPD) matrices have shown overwhelming accomplishments in many real-world applications in areas such as pedestrian detection [54] [55], magnetic resonance imaging analysis [45], action recognition [26], face recognition [29] [30], brain-computer interfaces [6] structure from motion [36] [35], and many more. This has further lead to development of many networks like SPDNet [27], LieNet [28], DCNN [62]etc. which directly operate on manifold value data and obtain an improved performance.

This thesis is aimed at alleviating the network engineering effort for networks operating on SPD Manifold data. Here we propose a new NAS problem on SPD matrices. We base off the definition of our new NAS problem on the DARTS framework for architecture search. This precisely requires a redefinition of the computational cell. A **computational cell** can be seen as the most basic unit composing the SPD network. Particularly the design of NAS for SPD data must ensure that all the **operations(edges)** of the DARTS

[40] DAG (Directed Acyclic graph) preserve the SPD geometry of the input. Thus each edge must correspond to a valid candidate operation on the manifold. Further it becomes necessary to also ensure that the mixture of these operations (as proposed in DARTS) also lies on the SPD Manifold.

To address the above problem, this thesis exploits a SuperNet search strategy to model the architecture search problem as a one-shot training process of a supernet that comprises of a mixture of SPD Neural Networks. Our approach performs a differentiable architecture search on a continuous relaxation of SPD neural architecture search space and can be efficiently optimized using an alternating optimization scheme based on the standard gradient descent algorithm. The proposed method is able to reliably design efficient neural networks from scratch and gives a significant performance improvement on our benchmark datasets.

In this thesis we also go a step further and propose an end to end training methodology for our SPDNetNAS. A major design flaw in SPDNets is that the input matrices are pre-constructed using some standard computer vision algorithms like co-variance pooling [2] on frame-wise features. The motivation for this idea is to facilitate automatic feature extraction and covariance pooling to generate SPD matrices on the fly as a part of the optimization procedure. The proposed idea uses standard convolutional networks to extract the necessary features on the fly and pool them into a covariance matrix. The weights of these convolutional feature extractors are optimized in unison with the other weights during the training phase. We observe that the benefit of this end-to-end learning is two fold. **First** it reduces the dependency on handcrafted SPD matrices and **second** it achieves a comparable performance to the proposed SPDNetNAS while relying on much fewer number of parameters.

To summarize the major contributions made in this project are as follows:

- We introduce a new NAS problem for SPD manifold networks that opens up a new direction for research in the deep learning domain.
- We propose a supernet search method to address the new problem in a one-shot learning manner. The proposed method incorporates SPD Riemannian metric into a deep neural architecture framework that utilizes the appropriate SPD representation and operations
- In addition to the existing operations for SPD networks[27][11], we introduce a new notion of skip connection, none operation, max pooling, averaging pooling, and weighted Riemannian pooling for SPD valued data. By doing so, we enlarge the operation search space for SPD network NAS.
- We provide an exhaustive analysis of our methods on different datasets. We also provide results of transferring architectures from one dataset to another to verify the generalization capacity of architectures obtained

- Further we propose an end to end training scheme for SPDNetNAS, to automatically extract the necessary SPD matrix as a part of the optimization.

In this thesis we start by providing the reader with a detailed overview of the different properties of the SPD manifold data, SPD neural networks and the NAS problem.Here we also define the mathematical terminology used throughout this thesis. Next we proceed to our problem definition and provide motivation on the importance of this problem. This is followed by description of our proposed solution to tackle this issue. We finally conclude with an exhaustive discussions on the evaluation of the proposed method and the extension to an end-to-end model.

Chapter 2

Background and Related work

In this section of the thesis we provide the necessary background on SPD Manifold geometry along with the notations used throughout the thesis. We also survey the SPD Manifold Nets proposed in the literature and the analog of operations on Euclidean data (eg: convolutional neural network) to operations on SPD Manifolds. Finally we provide an overview of the existing literature on NAS and SuperNets.

2.1 Riemannian geometry and Operations of the SPD manifold

In differential geometry a Riemannian manifold or Riemannian space ($\mathcal{M}$,q) is a real, smooth manifold $\mathcal{M}$ equipped with a positive-definite inner product g_p on the tangent space $T_p\mathcal{M}$ at each point p eg: the Euclidean manifold. The SPD matrices of a fixed dimension say m, is a convex smooth sub-manifold of the Euclidean space The inherited Euclidean metric further turns the SPD manifold into a Riemannian manifold. However this classic metric is not adequate in many applications due to the unique properties of the SPD matrices [37]. The notion of distance, origin, mean or average on the SPD Manifold is very different from their corresponding notion in the Euclidean Manifold. Hence it becomes necessary to re-define such metrics for the SPD Manifold. In this section of this report we provide a brief recap of some preliminaries of the SPD geometry

We denote the space of $n \times n$ real SPD as $\mathcal{S}^n_{++}$. Any real $n \times n$ SPD matrix $\boldsymbol{X} \in \mathcal{S}^n_{++}$ satisfies the property that for any non-zero $z \in \mathbb{R}^n$, $z^T \boldsymbol{X} z > 0$. The space of $\mathcal{S}^n_{++}$ forms the interior of the cone of $n(n+1)/2$ dimensional Euclidean space and is often studied with equipped affine-invariant Riemannian metric [45] [25].

2.1.1 The distance metric

Let $\mathcal{T}_X\mathcal{M}$ denote the tangent space of the manifold $\mathcal{M}$ at $\boldsymbol{X} \in \mathcal{S}^n_{++}$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{T}_X\mathcal{S}^n_{++}$, then affine-invariant Riemannian metric is defined as

$$< \boldsymbol{x}, \boldsymbol{y} >_X = \mathbf{Tr}(\boldsymbol{X}^{-1}\boldsymbol{x}\boldsymbol{X}^{-1}\boldsymbol{y}) \tag{2.1}$$

The above metric helps to define correct geometric distance between two points on the manifold. Let $\boldsymbol{X}_1, \boldsymbol{X}_2$ be the two points on the manifold, then the distance between the two point is given by

$$\delta_{\mathcal{M}}(\boldsymbol{X}_1, \boldsymbol{X}_2) = \frac{1}{2}\| \log(\boldsymbol{X}_1^{-\frac{1}{2}}\boldsymbol{X}_2\boldsymbol{X}_1^{-\frac{1}{2}})\|_F \tag{2.2}$$

There are other computationally efficient ways to compute the approximate distance between two points on the SPD manifold, such as the usage of Bregman divergence [5] [9] [50], Fisher-Bures metric [53], and optimal transport [41], however, their discussion is beyond the scope of our work.

2.1.2 Exponential and Logarithmic Map

Another matter of importance is the definition of the natural mappings to and from the manifold and its tangent bundle, which groups the tangent Euclidean spaces at each point in the manifold. The **exponential map** is a map from a subset of a tangent space $\mathcal{T}$ of a Riemannian manifold (or pseudo-Riemannian manifold) $\mathcal{M}$ to the manifold $\mathcal{M}$ itself. While the **logarithmic map** is the map from the manifold $\mathcal{M}$ itself to its tangent space $\mathcal{T}$. Note that this is not to be confused with matrix exp and log operations. Further the tangent space of a manifold is assumed to be locally euclidean. Next we clarify these definitions using precise formulas.

One more important property of the Riemannian manifold is that the geodesics has a local diffeomorphism which is a bijection from the point on the tangent space of the manifold to the manifold. To define such notions, let $\boldsymbol{X} \in \mathcal{S}^n_{++}$ symbolises the reference point and, $\boldsymbol{Y} \in \mathcal{T}_X\mathcal{S}^n_{++}$, then Eq:(2.3) associates each tangent vector $\boldsymbol{Y} \in \mathcal{T}_X\mathcal{S}^n_{++}$ to a point on the manifold.

$$\exp_X(\boldsymbol{Y}) = \boldsymbol{X}^{\frac{1}{2}} \exp(\boldsymbol{X}^{-\frac{1}{2}}\boldsymbol{Y}\boldsymbol{X}^{-\frac{1}{2}})\boldsymbol{X}^{\frac{1}{2}} \in \mathcal{S}^n_{++} \;\; \forall\, \boldsymbol{Y} \in \mathcal{T}_X \tag{2.3}$$

since this diffeomorphism is global [45] a unique inverse logarithm mapping can be defined as

$$\log_X(\boldsymbol{Z}) = \boldsymbol{X}^{\frac{1}{2}} \log(\boldsymbol{X}^{-\frac{1}{2}}\boldsymbol{Z}\boldsymbol{X}^{-\frac{1}{2}})\boldsymbol{X}^{\frac{1}{2}} \in \mathcal{T}_X \;\; \forall\, \boldsymbol{Z} \in \mathcal{S}^n_{++} \tag{2.4}$$

The above defined exponential chart allow us to develop the manifold in the tangent space along the geodesics and, provides a one to one mapping to and from SPD manifold to its tangent bundle.

2.1.3 Parallel Transport

Given two SPD matrix $\boldsymbol{X}_1, \boldsymbol{X}_2$ and a vector $\boldsymbol{Y} \in \mathcal{T}_{\boldsymbol{X}_1}$, parallel transport $\mathscr{P}_{\boldsymbol{X}_1 \to \boldsymbol{X}_2}(\boldsymbol{Y})$ defines the path from $\boldsymbol{X}_1$ to $\boldsymbol{X}_2$ such that $\boldsymbol{Y}$ remains parallel to itself in the tangent space along the path. Mathematically, it is defined as:

$$\mathscr{P}_{\boldsymbol{X}_1 \to \boldsymbol{X}_2}(\boldsymbol{Y}) = (\boldsymbol{X}_2 \boldsymbol{X}_1^{-1})^{\frac{1}{2}} \boldsymbol{Y} (\boldsymbol{X}_2 \boldsymbol{X}_1^{-1})^{\frac{1}{2}} \in \mathcal{T}_{\boldsymbol{X}_2}, \ \forall \ \boldsymbol{Y} \in \mathcal{T}_{\boldsymbol{X}_1} \tag{2.5}$$

Now, to transport a point on the SPD manifold instead of vector, we may choose to first map the data point to the tangent space using logarithm operation, and apply parallel transport to it, which is then mapped back to the manifold using exponential operation. However, it turns out the above relation holds even for SPD transport [11]

2.1.4 Computing batch mean on the SPD Manifold

a) Batch mean: Given a batch of N SPD matrices $\{\boldsymbol{X}_i\}_{i=1}^N$, we can compute its geometric mean or Riemannian barycenter ($\mathfrak{G}$) as

$$\mathfrak{G} = \underset{\boldsymbol{X}_\mu \in \mathcal{S}_{++}^n}{\operatorname{argmin}} \sum_{i=1}^{N} \delta_{\mathcal{M}}^2(\boldsymbol{X}_i, \boldsymbol{X}_\mu) \tag{2.6}$$

This mean is known as Fréchet mean if it is a global minimizer to the above equation. The above relation of geometric mean is given by Moakher [42] and Bhatia et.al. [8]. This definition is trivially extended to compute the weighted Riemannian Barycenter or popularly termed as weighted Fréchet Mean (wFM).

$$\mathfrak{G} = \underset{\boldsymbol{X}_\mu \in \mathcal{S}_{++}^n}{\operatorname{argmin}} \sum_{i=1}^{N} w_i \delta_{\mathcal{M}}^2(\boldsymbol{X}_i, \boldsymbol{X}_\mu); \ \text{ s.t. } \ w_i \geq 0 \text{ and } \sum_{i=1}^{N} w_i = 1 \tag{2.7}$$

Eq:(2.7) has a closed form solution for $N = 2$, however, for $N > 2$ we can compute an approximate solution using **Karcher flow** algorithm [33] [60]. The **karcher flow** algorithm is as described below:

Algorithm 1 Karcher flow [2] to compute the Riemannian mean of N SPD matrices

Require: data points $\{P_i\}_{i \leq N}$, iterations K, step α

1: $\mathfrak{G} \leftarrow \sum_{i \leq N} P_i$
2: **for** $k \leq K$ **do**
3: $\quad G \leftarrow \frac{1}{N} \sum_{i \leq N} Log_{\mathfrak{G}}(P_i)$
4: $\quad \mathfrak{G} \leftarrow Exp_{\mathfrak{G}}(\alpha G)$
5: **end for**
6: **return** $\mathfrak{G}$

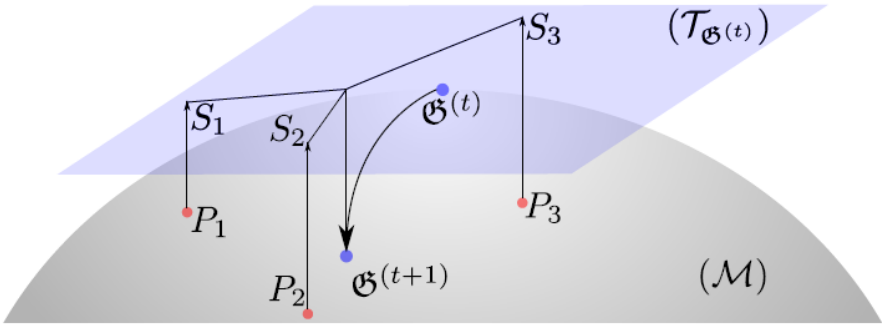


Figure 2.1: Illustration of one iteration of the Karcher flow [11]. Data points P_i are logarithmically mapped to the S_i on the tangent space at $\mathfrak{G}(t)$. The S_i are then arithmetically averaged, the result of which is exponentially mapped back to the manifold, yielding $\mathfrak{G}(t+1)$.

The second possible algorithm to approximate the batch fréchet mean uses an **online recursive strategy**. Let $\{\tilde{X}_i\}_{i=1}^{N}$ be samples drawn from $\tilde{p_{\mathcal{X}}}$.Given, $\{X_i\}_{i=1}^{N} \subset U$ and $\{w_i := w(X_i)\}_{i=1}^{N}$ such that $\forall i, w_i > 0$, the n^{th} estimate, M_n of $wFM(\{X_i\}, \{w_i\})$ is given by the following recursion:

$$M_1 = X_1$$
$$Mn = \tau_{M_{n1}}^{X_n}\left(\frac{w_n}{\sum_{j=1}^{N}(w_j)}\right)$$

In the above equation, $\tau_X^Y : [0,1] \rightarrow U$ is the shortest geodesic curve from X to Y . This recursive geodesic mean computation procedure is proved to be consistent. Throughout the remaining part of this thesis we have opted for the karcher flow algorithm for mean approximation and hence the recursive approach will not be discussed further. But it should be noted that at any

point in this thesis this recursive procedure can be effectively be used to replace the karcher flow algorithm.

2.1.5 Batch centering and biasing

Equipped with the notion of Riemmannian barycenter (fréchet mean) and parallel transport, we can define Batch centering and Biasing as follows:

$$\textbf{Batch centering}: \text{ Centering the} \mathfrak{G}: \boldsymbol{X}_i^c = \mathscr{P}_{\mathfrak{G} \to I}(\boldsymbol{X}_i) = G^{-\frac{1}{2}} \boldsymbol{X}_i G^{-\frac{1}{2}} \quad (2.8)$$

$$\textbf{Baising the batch}: \text{ Bias towards } \boldsymbol{G}: \boldsymbol{X}_i^b = \mathscr{P}_{I \to \boldsymbol{G}}(\boldsymbol{X}_i^c) = \boldsymbol{G}^{\frac{1}{2}} \boldsymbol{X}_i^c \boldsymbol{G}^{\frac{1}{2}} \quad (2.9)$$

2.2 SPD Neural Networks

Standard deep learning architectures like convolutional networks and LSTMs are primarily designed to process euclidean input data. Unfortunately these operations cannot be applied directly to SPD manifold valued data as at every stage of the network we need to ensure that the input matrix is a valid SPD matrix. This problem has motivated many researchers to propose specific architectures tailored to deal specifically with SPD valued data. In this section of the report we provide a review of such proposed architectures.

2.2.1 SPDNet

The first attempt at defining a neural network to process an SPD matrix as an input was made by [27]. They aimed at proposing a Riemannian network architecture to open up a new direction of SPD matrix non-linear learning in a deep model. Precisely the main contributions of this work were as follows:

- Define a **BiMap** layer which is considered to be an analog of a conv layer is traditional network
- Define **non-linerality** in the network to facilitate better learning on the SPD matrices
- Propose a new backpropagation with a variant of stochastic gradient descent on Stiefel manifolds to update the structured connection weights in the BiMap layer.

Assuming $\boldsymbol{X}_{k-1} \in \mathcal{S}_{++}^n$ be the input SPD matix to the k^{th} layer, the SPD network layers are defined as follows:

- **BiMap layer**: The BiMap layer corresponds to a dense layer for SPD data. The BiMap layer mainly reduces the dimension of the input SPD matrix via a transformation matrix $\boldsymbol{W}_k$

$$\boldsymbol{X}_k = \boldsymbol{W}_k \boldsymbol{X}_{k-1} \boldsymbol{W}_k^T \quad (2.10)$$

To ensure the matrix $\boldsymbol{X}_k$ to be a valid SPD matrix, the $\boldsymbol{W}_k$ matrix must be of full row-rank (lie on a Stiefel Manifold).

- **ReEig layer**: The ReEig layer is analogous to ReLU like layers present in the classical convolutional networks. It aims to introduce non-linearity to SPD network. The ReEig for the k^{th} layer is defined as

$$\boldsymbol{X}_k = \boldsymbol{U}_{k-1} \max(\varepsilon \boldsymbol{I}, \Sigma_{k-1}) \boldsymbol{U}_{k-1}^T \text{ where, } \boldsymbol{X}_{k-1} = \boldsymbol{U}_{k-1} \Sigma_{k-1} \boldsymbol{U}_{k-1}^T \tag{2.11}$$

  Here, $\boldsymbol{I}$ is the identity matrix and ε is a rectification threshold value. U_{k-1}, Σ_{k-1} are the orthonormal matrix and singular-value matrix respectively, which are obtained via matrix factorization of $\boldsymbol{X}_{k-1}$.Note that the Relu operation here can be effectively replaced by any of the other non-linearities eg:sigmoid or tanh.
- **LogEig layer**: To map the manifold representation of SPD to flat space so that a Euclidean operation can be performed, LogEig layer is introduced. The LogEig layer is defined as follows

$$\boldsymbol{X}_k = \boldsymbol{U}_{k-1} \log(\Sigma_{k-1}) \boldsymbol{U}_{k-1}^T \text{ where, } \boldsymbol{X}_{k-1} = \boldsymbol{U}_{k-1} \Sigma_{k-1} \boldsymbol{U}_{k-1}^T \tag{2.12}$$

  The output of LogEig layer is used with fully connected layers to solve tasks with SPD representation.

- **ExpEig layer**: ExpEig layer is defined to map the abstract SPD representation from flat space back to SPD manifold space. It is converge to ExpEig layer i.e.,

$$\boldsymbol{X}_k = \boldsymbol{U}_{k-1} \exp(\Sigma_{k-1}) \boldsymbol{U}_{k-1}^T \text{ where, } \boldsymbol{X}_{k-1} = \boldsymbol{U}_{k-1} \Sigma_{k-1} \boldsymbol{U}_{k-1}^T \tag{2.13}$$

- **Pooling Operations**: The paper also proposed that standard types of poolings which are used in a convolutional neural network eg: max and average pooling can be applied after a **LogEig** Map as after the LogEig map we are operating in the Euclidean space.

It is important to note that the weight parameter $\boldsymbol{W}_k^T$ is orthogonal here so the optimization procedure (backpropagation) needs to ensure that it resides on a compact Stiefel manifold $S(d_k, d_{k1})$ to adhere to the SPD geometry.

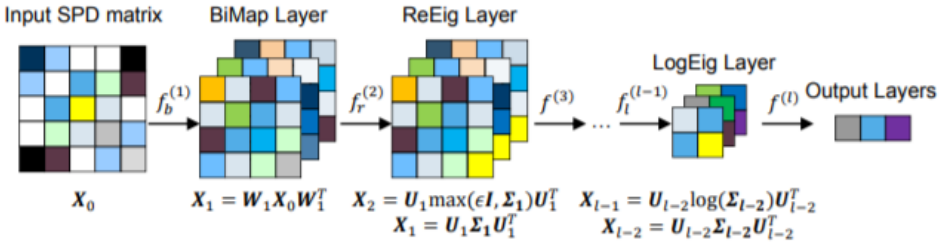


Figure 2.2: SPDNet Architecture [27]

2.2.2 SPDNetBN

Training deep neural networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization. Batch normalization [31] is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks. The standard batch normalization algorithm is as defined in the figure below:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 2.3: Batch Normalization Algorithm [31]

Given the benefits of Batch Normalization for standard deep networks operating on Euclidean data it becomes necessary to define an analog of Batch normalization for SPDNet. The work of [11] aimed at mainly solving this issue and defining a procedure for batch normalization of SPD matrices. They observed a significant performance gain from incorporating of define batch normalization layer into the SPDNet. The main contribution of this work was to define a procedure for batch normalization on SPD valued input data. They perform batch normalization on SPD data as follows:

SPD Batch Normalization layer: To perform the batch normalization after each BiMap layer, they compute the Riemannian barycenter of the batch of SPD matrices, followed by a running mean update step which is the Riemannian weighted average between the batch mean and the current running mean, with the weights $(1-\theta)$ and (θ) respectively. Once the mean is computed, they centralize and add bias to each SPD sample of the batch using Eq:(2.8). An interested reader should note that couple of other works eg:[13] also exist to perform SPD batch normalization, but in this project we adhere to the work of [11].

2.2.3 Other variants of SPD Neural Networks

In this thesis we focus mainly on the operations defined in SPDNet and SPDNetBN. However a plethora of other research works also exist which define several other layer operations eg: residual connections, dilations for SPD matrices. **ManifoldNet** [14] proposed recursive convergent computation of weighted fréchet mean as an analog of convolutions with non-linearity. **Manifold DCNN** [62] proposed an end to end approach to extract features from video frames to learn the input SPD matrix. In addition they also propose a generalization of dilated convolution, residual connection, weight normalization, ReLU and Dropout, to directly operate on SPD manifold valued data. Discussion on these architectures and operations is beyond the scope of this report, but these works can be an important potential future extension on the architecture search space of the proposed NAS framework for SPD valued data.

2.3 Neural Architecture Search

Recently given the amount of fine-tuning required to design a suitable network for a particular task, we have seen a surge of interest in the area of automated architecture search for deep networks. Also the relatively easy availability and access to computational resources has encouraged

researchers to tackle this computationally expensive but important problem effectively. Most of the earlier NAS approaches were based on reinforcement learning strategies and hence their computation time was thousands of GPU days. This drawback lead to development of less computationally expensive methods. Most of these recent approaches involve training a **SuperNet** that incorporates many candidate sub-networks. These can be further classified into two categories, those that decouple search and training within one stage and others who decouple them into two stages wherein the training and where the trained SuperNet is treated as an evaluation for final searching. Furthermore, there are mainly two types of one-shot NAS methods based on the architecture modeling [21] i.e, parameterized architecture [40] [63] [59], and sampled architecture [17] [16]. In this paper, we adhere to the parametric modeling mainly due to its promising results on conventional neural architectures [40]. A majority of the previous work on NAS with continuous search space fine-tunes the explicit feature of specific architecture [48] [57] [3] [49]. On the contrary, [40] provides architectural diversity for NAS with comparable performance.

In our project we focus on a particular SuperNet strategy proposed by Differentiable Architecture Search (DARTS), which combines the SuperNet training and search as a bi-level optimization where each operation is associated with a weight or coefficient of importance. All sub-networks in DARTS are initialized to be equally important and the optimization procedure gradually introduces bias towards particular architectures. Next we provide a brief overview of DARTS.

2.3.1 DARTS

DARTS models the NAS problem using DAGS (Directed Acyclic Graphs). Each **cell** in DARTS is a directed acyclic graph consisting of an ordered sequence of N nodes. Each node x is a latent representation (e.g. a feature map in convolutional networks) and each directed **edge** (i, j) is associated with some operation $o^{(i,j)}$ that transforms $x^{(i)}$. Cells are assumed to have two input nodes and a single output node. The output of the cell is obtained by applying a reduction operation (e.g. concatenation) to all the intermediate nodes.

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)}) \tag{2.14}$$

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{exp(\alpha_o^{(i,j))}}{\sum_{o' \in \mathcal{O}} exp(\alpha_{o'}^{(i,j)})} o(x) \tag{2.15}$$

DARTS poses architecture search as a bi-level optimization problem. The weights for operations on the edges i.e α's are updated using the validation

set while the weights of the architecture w's are updated using a training set. These two optimization steps are taken alternatively to converge to an architecture that corresponds to a good local minimum. This is the first stage i.e. the **search stage** of DARTS. Once the search phase is complete, the best architecture (corresponding to operations with highest weights) is **retrained** on the training dataset from scratch to obtain the final performance of the model. The DARTS algorithm is as described below:

Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge (i, j)
while *not converged* **do**
1. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$ ($\xi = 0$ if using first-order approximation)
2. Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$

Derive the final architecture based on the learned α.

Figure 2.4: DARTS optimization algorithm

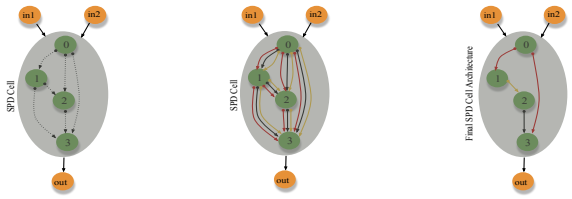
Despite its popularity DARTS faces many issues. A major issue which has been addressed by multiple extensions to DARTS is that DARTS can often get stuck in a local minimum. Also it is a well known issue that DARTS has affinity towards operations with fewer parameters like MaxPooling and residual connections and may possibly miss out on a more complex and better architecture [61]. Discussion of these extensions to DARTS is beyond the scope of this thesis.

Chapter 3

Problem Definition

In this thesis we aim at extending the DARTS framework for neural architecture search to process SPD valued data. The suggested problem aims at neural architecture search for SPD networks (SPDNetNAS). To address this problem, there are a few key issues that need to be resolved. Firstly, the SPD network neural architecture search problem needs a new definition for *computation cell*. In contrast to the traditional computational cell, this novel cell must incorporate the notions of SPD manifold geometry while performing any operation. Moreover, analogous to traditional NAS problem, this brand-new computation cell can either be a normal cell that returns SPD feature maps of the same dimension (width and height) or, a reduction cell in which the SPD feature maps are reduced by a certain factor in width and height. Secondly, the SPDNetNAS problem demands an introduction to appropriate SPD *search space* §4.1 that can help an *architecture search method* §3.1 to provide valid SPD cells, which can then be stacked and trained to build an efficient SPD network architecture. Another important issue to deal with is to perform appropriate mixing of output SPDs. It is known that the Euclidean notion of mean cannot be extented to the SPD geometry. Hence we also need to re-define the mixing operator in DARTS

Following the above procedure to tackle the SPDNetNAS problem, we first define the basic computational cell called **SPD cell**. Concretely, a SPD cell is a directed acyclic graph (DAG) composed of nodes and edges, where each *node* is a *latent representation* of the SPD manifold valued data and, each *edge* corresponds to a *valid candidate operation* on SPD manifold (see Fig. 3.1). 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:



(a) (b) (c)

Figure 3.1: (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 relaxed continuous search space under a bi-level optimization routine.

$$\boldsymbol{X}_{\mathcal{M}}^{(j)} = \underset{\boldsymbol{X}_{\mathcal{M}}^{(j)}}{\operatorname{argmin}} \sum_{i<j} \delta_{\mathcal{M}}^{2}\left(\mathscr{O}_{\mathcal{M}}^{(i,j)}(\boldsymbol{X}_{\mathcal{M}}^{(i)}), \boldsymbol{X}_{\mathcal{M}}^{(j)}\right), \tag{3.1}$$

where $\delta_{\mathcal{M}}$ denotes the geodesic distance Eq:(2.2). Generally, the above 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 guarantees that each computational graph preserves the appropriate geometric structure of the SPD manifold. Equipped with notion of SPD cell and its intermediate transformation, we are prepared to propose our solution to the SPDNetNAS problem and its corresponding search space.

3.1 Proposed Method

To solve the suggested SPDNetNAS problem, one of the most promising NAS methodologies is SuperNet modeling. While we can resort to some other NAS methodologies to solve the problem like reinforcement learning based method [64] or evolution based algorithm [47], in general, the SuperNet method models the architecture search problem as a one-shot training process of one single SuperNet that consists of all architectures. Based on the SuperNet modeling, we can search for the optimal SPD neural architecture either using parameterization of architectures or sampling of single-path architectures. In this paper, we focus on the parameterization approach that is based on the continuous relaxation of the SPD neural architecture representation. Such an approach allows for an efficient search of the architecture using gradient descent approach. In the following chapters, we first introduce the discrete SPD neural architecture search space which is then followed by our SuperNet search method that utilizes the continuous relaxation of the search space. Lastly, we provide a solution to the proposed bi-level optimization problem.

Chapter 4

NAS for SPD Manifold Nets

Equipped with the necessary background and the problem definition we now proceed to a more detailed description of the proposed method. We start out by description of the operations composing our search space. Next we define our SuperNet search methodology and the description of our bi-level optimization problem.

4.1 Search Space

Our search space consists of a set of valid SPD network operations which is defined for the SuperNet search. Inspired by the traditional NAS methods like [40], we apply the SPD batch normalization to every SPD convolution operation (i.e., BiMap), and design three various convolution blocks including the one without activation (i.e., ReEig), the one using post-activation and the one using pre-activation. These three operations are illustrated in Table (4.1) with the basic BiMap, SPD batch normalization and, ReEig operation. In addition to those candidate operations, we introduce some new operations analogous to [40] to enrich the search space in the context of SPD networks. They are skip normal, none normal, weighted Riemannian pooling, average pooling, max pooling and skip reduced. Most of these operations are not fully explored for SPD networks. The definition of these new SPD network operations are as follows:

a) Skip normal: This operation preserves the input representation and is similar to skip connection.
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

Table 4.1: Operation Search Space for the proposed SPD architecture search method.

Operation	Definition
BiMap_0	{BiMap, SPD Batch Normalization}
BiMap_1	{BiMap, SPD Batch Normalization, ReEig}
BiMap_2	{ReEig, BiMap, SPD Batch Normalization}
Skip_normal	Output same as input.
None_normal	Return Identity Matrix i.e, notion of origin on SPD
WeightedReimannainPooling_normal	weighted Fréchet Mean on the input SPD multiple times.
AveragePooling_reduced	{LogEig, AveragePooling, ExpEig}
MaxPooling_reduced	{LogEig, MaxPooling, ExpEig}
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$D_b = \text{diag}(D_1, D_2)$

is used to map the sample 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 SPD manifold, an ExpEig operation is used. In Figure 4.1 we show our average and max pooling operations. 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. The diagram shown in Figure 4.1 is inspired by [27].

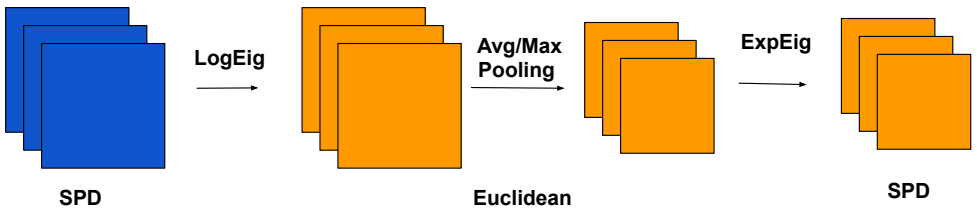


Figure 4.1: Avg/Max Pooling: Maps the SPD matrix to Euclidean space using LogEig mapping, does avg/max pooling followed by ExpEig map

e) Weighted Riemannian pooling: Our weighted Riemannian pooling uses ManifoldNet's suggested weighted Fréchet Mean [14]. Contrary to [14], we use Karcher flow method [33] to compute it.
Figure 4.2 provides an intuition behind the Weighted Riemannian Pooling operation. Here, w_11, w_21, etc., corresponds to the set of normalized weights for each channel (shown as two blue channels). The next channel —shown in orange, is then computed as weighted Fréchet mean over these two input channels. This procedure is repeated to achieve the desired number of output channels (here two), and finally all the output channels are

Algorithm 1: The proposed Neural Architecture Search of SPD Manifold Nets (SPDNetNAS)

Require: Mixed Operation $\bar{\mathcal{O}}_{\mathcal{M}}$ which is parameterized by α^k for each edge $k \in N_e$;
while *not converged* **do**
  Step1: Update α (architecture) using Eq:(4.3) solution. Note that updates on w and $\tilde{w}$ (Eq4.4, Eq4.5) should follow the gradient descent on SPD manifold;
  Step2: Update w by solving $\nabla_w \boldsymbol{E}_{train}(w, \alpha)$; Ensure SPD manifold gradient to update w [27][11];
end
Ensure: Final architecture based on α. Decide the operation at an edge k using $\underset{o \in \mathcal{O}_{\mathcal{M}}}{\text{argmax}}\{\alpha_o^k\}$

concatenated. The weights are learnt as a part of the optimization procedure and, we ensure that they are correctly normalized using a softmax.

Figure 4.2: Weighted Riemannian Pooling: Performs multiple weighted Fréchet means on the channels of the input SPD

f) Skip reduced: It is analogous to 'skip_normal' but in contrast it decomposes the input into small matrices to reduces the inter-dependency between channels. Our definition is in line with reduce operation in [40].

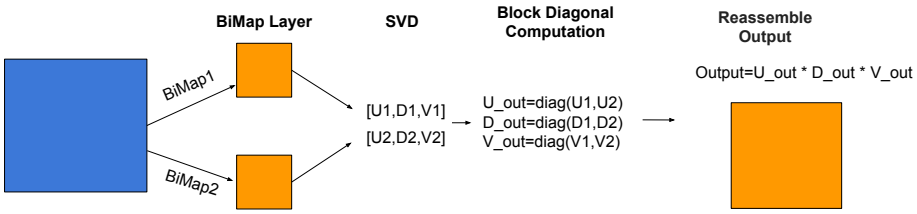


Figure 4.3: Skip Reduced:Maps input to two smaller matrices using BiMaps, followed by SVD decomposition on them and then computes the output using a block diagonal form of U's D's and V's

4.2 Supernet Search

To search for an optimal SPD architecture, 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 a weighted fréchet mean over all possible candidate operations. Mathematically,

$$\bar{\mathscr{O}}_{\mathcal{M}}(\boldsymbol{X}_{\mathcal{M}}) = \underset{\boldsymbol{X}^{\mu}_{\mathcal{M}}}{\operatorname{argmin}} \sum_{k=1}^{N_e} \alpha^k \delta^2_{\mathcal{M}}\Big(\mathscr{O}^{(k)}_{\mathcal{M}}(\boldsymbol{X}_{\mathcal{M}}), \boldsymbol{X}^{\mu}_{\mathcal{M}}\Big) = \exp_{\boldsymbol{X}_{\mu}}\Big[\big(\sum_{k=1}^{N_e} \alpha^k\big)^{-1} \sum_{k=1}^{N_e} \alpha^k \log_{\boldsymbol{X}_{\mu}}\big(\mathscr{O}^{k}_{\mathcal{M}}(\boldsymbol{X}_{\mathcal{M}})\big)\Big] \tag{4.1}$$

where, $\mathscr{O}^{k}_{\mathcal{M}}$ is the k^{th} candidate operation between nodes, $\boldsymbol{X}_{\mu}$ is the intermediate SPD manifold mean (Eq.2.7) and, N_e denotes number of edges. To be precise, we apply Karcher flow approach [33] that uses a logarithm-exponential mapping to achieve the weighted Fréchet mixture of operations. Also, following [14], we apply softmax to normalize the weights for the mixture of candidate operation. In Figure 4.4 we provide an intuition of the mixed operation we have proposed in the main paper. We consider a very simple base case of three nodes, two input nodes (1 and 2) and one output node (node 3). The goal is to compute the output node 3 from input nodes 1 and 2. We perform a candidate set of operations on the input node, which correspond to edges between the nodes (here two for simplicity). Each operation has a weight α_{i_j} where i corresponds to the node index and j is the candidate operation identifier. In Figure 4.4 below i and j $\in \{1,2\}$ and $\boldsymbol{\alpha_1} = \{\alpha_{1\_1}, \alpha_{1\_2}\}$, $\boldsymbol{\alpha_2} = \{\alpha_{2\_1}, \alpha_{2\_2}\}$. α's are optimized as a part of the bi-level optimization procedure proposed in the main paper. Using these alpha's, we perform a channel-wise weighted Fréchet mean (wFM) as depicted in the figure below. This effectively corresponds to a mixture of the candidate operations. Note that the alpha's corresponding to all channels of a single op are assumed to be the same. Once the weighted Fréchet means have been computed for nodes 1 and 2, we perform a channel-wise concatenation on the outputs of the two nodes, effectively doubling the number of channels in node 3.

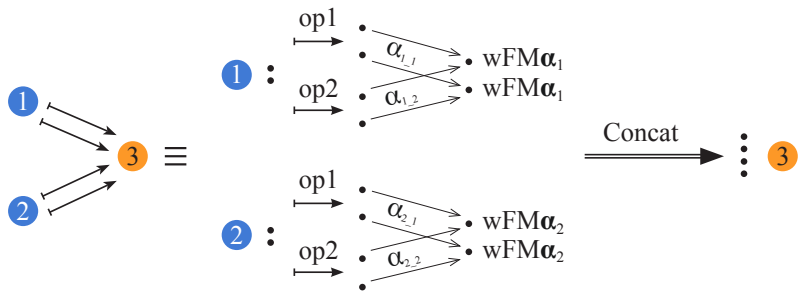


Figure 4.4: Detailed overview of mixed operations. We simplify the example by taking 3 nodes (two input nodes and one output node) and two candidate operations. Input nodes have two channels (SPD matrices), we perform channelwise weighted Fréchet mean between the result of each operation (edge) where weights α's are optimized during bi-level architecture search optimization. Output node 3 is formed by concatenating both mixed operation outputs, resulting in a four channel node.

From Eq:(4.1), the mixing of operations between nodes is determined by the weighted combination of alpha's (α^k) and the set of operations. 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 must simultaneously learn the contribution of several possible operation within all the mixed operations (w) and the corresponding architecture α. As a result, for a given w, we can find α and vice-versa, resulting in a bi-level optimization problem. The lower-level problem corresponds to the optimal weight variable learned for a given α i.e., $w^{opt}(\alpha)$ using a training loss ($\boldsymbol{E}^L_{train}$). The upper-level optimization solves for the variable α given the optimal w using a validation loss ($\boldsymbol{E}^U_{val}$).

$$\underset{\alpha}{\text{minimize}}\ \boldsymbol{E}^U_{val}\big(w^{opt}(\alpha), \alpha\big);\ \ \text{subject to: } w^{opt}(\alpha) = \underset{w}{\text{argmin}}\ \boldsymbol{E}^L_{train}(w, \alpha) \tag{4.2}$$

The bi-level search approach leads to a optimal mixture of multiple small architectures. To derive each node in the discrete architecture, we maintain the top-k strongest operations (i.e., with the k-th highest softmax weights) among all the candidate operations associated with all the previous nodes.

Bi-level Optimization: The proposed bi-level optimization problem is hard 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. To solve it in an efficient way, we

adhere to approximate $w^{opt}(\alpha)$ using a single training step and skip to solve the inner optimization completely until convergence [22] [40]. By applying approximations, Eq:(4.2) reduces to

$$\nabla_\alpha \boldsymbol{E}^U_{val}\big(w^{opt}(\alpha),\alpha\big) \approx \nabla_\alpha \boldsymbol{E}^U_{val}\big(w-\eta\nabla_w \boldsymbol{E}^L_{train}(w,\alpha),\alpha\big) \tag{4.3}$$

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 manifold to update the structured connection weight, and its corresponding SPD matrix data [27] [1]. Applying the chain rule to Eq:(4.3) gives

$$\underbrace{\nabla_\alpha \boldsymbol{E}^U_{val}\big(\tilde{w},\alpha\big)}_{\text{first term}} - \underbrace{\eta\nabla^2_{\alpha,w}\boldsymbol{E}^L_{train}(w,\alpha)\nabla_{\tilde{w}}\boldsymbol{E}^U_{val}(\tilde{w},\alpha)}_{\text{second term}} \tag{4.4}$$

where, $\tilde{w} = \Psi_r\big(w-\eta\tilde{\nabla}_w \boldsymbol{E}^L_{train}(w,\alpha)\big)$ denotes the weight update on the SPD manifold for the forward model. $\tilde{\nabla}_w$, Ψ_r symbolizes the Riemannian gradient and the retraction operator respectively. The second term in the Eq(4.4) involves second order differentials with very high computational complexity, hence, using the finite approximation method the second term of Eq(4.4) reduces to:

$$\nabla^2_{\alpha,w}\boldsymbol{E}^L_{train}(w,\alpha)\nabla_{\tilde{w}}\boldsymbol{E}^U_{val}(\tilde{w},\alpha) = \big(\nabla_\alpha \boldsymbol{E}^L_{train}(w^+,\alpha) - \nabla_\alpha \boldsymbol{E}^L_{train}(w^-,\alpha)\big)/2\delta \tag{4.5}$$

where, $w^\pm = \Psi_r(w \pm \delta\tilde{\nabla}_{\tilde{w}}\boldsymbol{E}^U_{val}(\tilde{w},\alpha))$ and δ is a small number set to $0.01/\|\nabla_{\tilde{w}}\boldsymbol{E}^U_{val}(\tilde{w},\alpha)\|_2$ [40].

For training a SPD network, one must use the back-propagation in the context of Riemannian geometry of the SPD matrices. For concrete mathematical derivations on backprogation for SPD network layers, kindly refer to [27][11]. The pseudo code of our approach is outlined in **Algorithm(1)**.

Chapter 5

Experimental Results

5.1 Dataset description

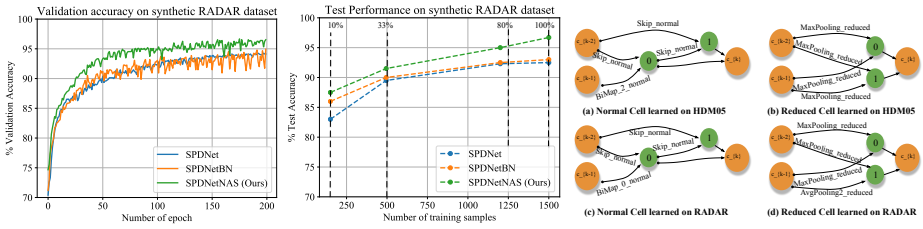
- **RADAR (Drone Recognition)** For this task, we considered a synthetic setting of a confidential dataset from NATO organization [15]. This dataset is composed of radar signals, where each signal is split into windows of length 20 resulting in a 20x20 covariance matrix for each window (one radar data point). The synthesized dataset consists of 1000 data points per class [11]. Given 20×20 input covariance matrices, our reduction cell reduces them to 10×10 matrices followed by normal cell to provide complexity to our network.
- **HDM05 (Action Recognition)** For this task, we used the HDM05 dataset [43] which contains 130 action classes, yet, for consistency with previous work[11], we used 117 class for our performance comparison. This dataset has 3D coordinates of 31 joints per frame. Following the previous works [25] [27], we describe an action for a sequence using 93×93 joint covariance matrix. The dataset is composed of 2083 SPD matrices distributed among all 117 classes. For this dataset, our reduction cell is designed to reduce the matrices dimensions from 93 to 30 for legitimate comparison against [11].
- **AFEW 2014 (Emotion Recognition):** Lastly, we used AFEW dataset [18] to evaluate the performance of our algorithm for emotion recognition. This dataset has 1345 videos of facial expressions classified into 7 distinct classes. For evaluation, we normalized each image to 20×20 and then represent each video using a 400×400 covariance matrix [58][27]. To keep our evaluation consistent with the previous work, we split the dataset equally for training and test sets. We used this dataset to study the generalization of the SPD network architecture supplied by our SPDNetNAS approach. For this, we transferred the optimal architecture obtained from RADAR and HDM05 experiments

and observed its performance behaviour. We reduce the 400×400 dimensional SPD matrix to 100×100 dim SPD matrices.

5.2 Using CPU v/s GPU for training our network

SPDNet and SPDNetBN both of which are the foundation of our model definition use many computationally expensive operations like eigen decomposition and SVD computation. Unfortunately these operations cannot be parallelized effectively and hence cannot exploit the computational speedup offered by a GPU unless the dimensionality of the matrices is very small. In our study, we analyzed SPD matrices with the Affine Invariant Riemannian Metric (AIRM), this induces operations heavily dependent on singular value decomposition (SVD) or eigen-decomposition (EIG). Both decompositions suffer from weak support on GPU platforms [1]. Hence, our training did not benefit from GPU acceleration and we decided to train on CPU. As a future work, we will speedup our GPU implementation by optimizing the SVD Householder bidiagonalization process as studied in some existing works like [19, 23].

5.3 Evaluation on RADAR and HDM05 datasets



(a) (b) (c)

Figure 5.1: (a) Validation accuracy of our method in comparison to the SPDNet [27] and SPDNetBN [11] on RADAR dataset [15]. Clearly, our SPDNetNAS algorithm show a steeper validation accuracy curve. (b) Test accuracy on 10%, 33%, 80%, 100% of the total data sample. It can be observed that our method exhibit superior performance. (c) Normal cell and Reduced cell learned on HDM05 [43] and RADAR dataset[15].

Following SPDNetBN[11] experimental setup for the **RADAR** dataset, we assign 50%, 25%, and 25% of the dataset for training, validation, and test

[1] https://discuss.pytorch.org/t/torch-svd-is-slow-in-gpu-compared-to-cpu/10770

Table 5.1: Performance comparison (%) of our method with existing SPDNets for drone and action recognition. RAND: random search, Last Col: SPDNetNAS search time. (*) Accuracy comparison with 10% training data.

Dataset ↓ / Method	SPDNet [27]	SPDNetBN [11]	SPDNetNAS (RAND)	SPDNetNAS	NAS Search Time
RADAR [15]	93.21% ± 0.39	92.13% ± 0.77	95.49% ± 0.08	**96.47% ± 0.11**	1 CPU day
HDM05 [43]	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*

set respectively. Our algorithm takes 1 CPU day of search time to provide the SPD architecture on this dataset. Training and validation take 9 CPU hours for 200 epochs. Figure 5.1(a) shows the validation curve which almost saturates at 200 epoch demonstrating the stability of our training process. Test results on this dataset are provided in Table (5.1) which clearly shows the benefit of our method. Further, we study the robustness of SPDNetNAS architecture by taking 10%, 33%, 80% of the data for training (see Figure 5.1(b)). Last row of Table (5.1) show the test accuracy comparison when only 10% of the data is used for training our architecture. Statistical results under these setups show our algorithm supply the best possible architecture. The normal and reduction cells obtained on this dataset are shown in Figure(5.1(c)).
Similar to RADAR for **HDM05** we split the dataset into 50%, 25%, and 25% for training, validation, and testing. For this dataset, our reduction cell is designed to reduce the matrices dimensions from 93 to 30 for legitimate comparison against [11]. To search for the best architecture, we ran our algorithm for 50 epoch (3 CPU days). Figure 5.1(c) show 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 under a similar setup as the previous experiment is documented in Table (5.1). Statistics clearly support the efficacy of our approach.

5.4 Transferring architectures to AFEW Dataset

To keep our evaluation consistent with the previous work, we split the dataset equally into training and test sets. We used the AFEW dataset to study the generalization of the SPD network architecture supplied by our SPDNetNAS approach. For this, we transferred the optimal architecture obtained from RADAR and HDM05 experiments and observed its performance behaviour. Each evaluation takes around 3 CPU days for 50 epochs. Table (5.2) results demonstrate that the transferred architecture can handle the new dataset quite convincingly. The test accuracy is comparable to the best SPD network method for RADAR model transfer. For HDM05 model transfer, the test accuracy is much better than the existing SPD networks. The statistics shown in Table (5.2) for SPDNet and SPDNetBN are reproduced using their original implementations.

Table 5.2: Performance of transferred SPDNetNAS Network architecture in comparison to existing SPD Networks on the AFEW dataset [18]. RAND symbolizes random architecture transfer from our search space.

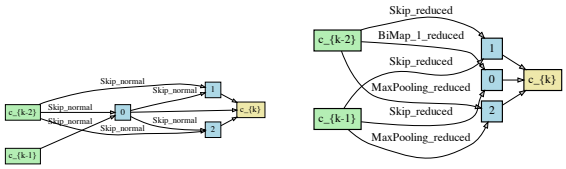
SPDNet [27]	SPDNetBN [11]	Ours (RAND)	Ours (RADAR)	Ours (HDM05)
33.17%	35.22%	32.88%	**35.31** %	**38.01%**

5.5 Experiments with node stacking

Experiments presented in the main paper consist of $N = 5$ nodes per cell which includes two input nodes, one output node, and two intermediate nodes. In these set of experiments we aimed at analyzing if adding more nodes to the SPD computational cell helps the accuracy of the network. We added one extra intermediate node ($N = 6$) to the cell design. We observe that we converge towards an architecture design that is very much similar in terms of operations (see Figure 5.2). The evaluation results shown in Table (5.3) suggest that the gain in the accuracy is not very significant and it is quite comparable to the accuracy of the original design. Hence we deduce that adding nodes further to the computational cell may not be very beneficial.

Table 5.3: Results for multi-node experiments on HDM05

Number of nodes	SPDNetNAS	Search time
5	**68.74% ± 0.93**	3 CPU days
6	67.96% ± 0.67	6 CPU days



(a) Normal cell (b) Reduction cell

Figure 5.2: (a)-(b) Optimal Normal cell and Reduction cell with 6 nodes on the HDM05 dataset

5.6 Experiments with multiple cell stacking

For more extensive analysis of the proposed method we conducted training experiments by stacking multiple cells in-line with the experiments conducted by [40]. We then transfer the optimized architectures from the singe cell search directly to the multi-cell architectures for training. Hence the

search time for all our experiments is same as for a single cell search i.e. 3 CPU days. Results for this experiment are provided in Table 5.4. The first row in the table shows the performance for single cell model, while the second and third rows show the performance with multi-cell stacking. *Remarkably, by stacking multiple cells our proposed SPDNetNAS outperforms SPDNetBN [11] by a large margin (about **8%**, i.e., about **12%** for the relative improvement).*

Table 5.4: Results for multiple cell search and training experiments on HDM05: reduction corresponds to reduction cell and normal corresponds to the normal cell.

Dim reduction in cells	Cell type sequence	SPDNetNAS	Search Time
$93 \rightarrow 46$	reduction-normal	68.74% $\pm$ 0.93	3 CPU days
$93 \rightarrow 46$	normal-reduction-normal	**71.48%** $\pm$ **0.42**	3 CPU days
$93 \rightarrow 46 \rightarrow 22$	reduction-normal-reduction-normal	**73.59 %** $\pm$ **0.33**	3 CPU days

5.7 Key observations and Takeaways

Through our experiments we make the following key observations:

- We obtain a statistically significant performance improvement on all the three datasets (all of which belong to different domains). Hence establishing the efficiency of the proposed approach.
- Through the search phase for DARTS we observe that very often the search culminates into an architecture (possibly a local minima) which predominantly architectures containing the operations with fewer number of parameters like pooling and skip operations. This is a well known issue in DARTS and solutions to resolve this many attempts have been made [61].
- Through our search on **HDM05** we pick the best three minimas as use them to train the architecture for **AFEW**. Interestingly the best result obtained for AFEW is from an architecture which is different from the architecture (amongst the 3 minimas) which gives the best performance for HDM05.
- A main disadvantage of our method is that it has not been extended to be trained in an end-to-end manner to use the features extracted to construct the SPD matrix as a part of the optimization problem (c.f. the next chapter)

Chapter 6

Extension to End-to-End training

In this section of the thesis we discuss extension of the SPDNetNAS proposed by us to end to end training.

6.1 Introduction to End to End Learning

Earlier approaches to machine translation involved different black box components chained together, each optimized separately. End-to-end learning process in deep learning is a methodology by which all of the parameters are trained jointly, rather than step by step. Precisely the end-to-end learning philosophy aims at carefully ensuring that all modules of a learning systems are differentiable with respect to all adjustable parameters (weights) and training this system as a whole are lifted to the status of principles. In the case of SPDNets this could be a network which directly takes multiple video frames as input and uses them on the fly to learn to generate an SPD matrix.

End to end learning has many benefits. In case of a convolutional neural network it allows the model to learn potentially useful feature extractors (convolutional filters) which can be directly transferred to extract features for other domains. End-to-end networks are typically easier to reason with since there's only one set of inputs and one set of outputs and feature extraction and training are all happening as a part of a single neural network. Areas of Natural Language Understanding (NLU) like Machine Translation have benefited greatly from end-to-end learning [52]. Inspite of being a somewhat brute-force technique has been popularized in the context of deep learning and end-to-end learning for deep networks has its limitations [24].

6.2 Motivation for end-to-end learning

The SPDNetNAS proposed in this thesis has a disadvantage. The SPD matrix input to the network is often hand engineering with different handcrafted feature engineering methods. The major motivation for end to end training on the raw video frames is that it facilitates automatic and effective feature extraction using the learned filters of a convolutional network. The main advantage here is that the video frames can be input directly to the network to construct the necessary SPD matrix on the fly. Here we apply the optimal architectures obtained in the search phase from HDM05 to directly do end to end training on the raw video frames of the AFEW dataset. We mainly base off the idea from the work of [2] on covariance pooling.

6.3 End to End Architecture design

Our feature extraction architecture is composed of a convolutional layer. Each of the frames (t_0,t_1,...t_n) are passed through a convolutional layer which maps it to a lower dimensional features (f_1,f_2,...f_n) where n is the number of frames. Once the features for the individual frames are extracted they are used to construct a covariance (SPD) matrix which further serves as an input to DARTS. The figure below depicts our architecture design:

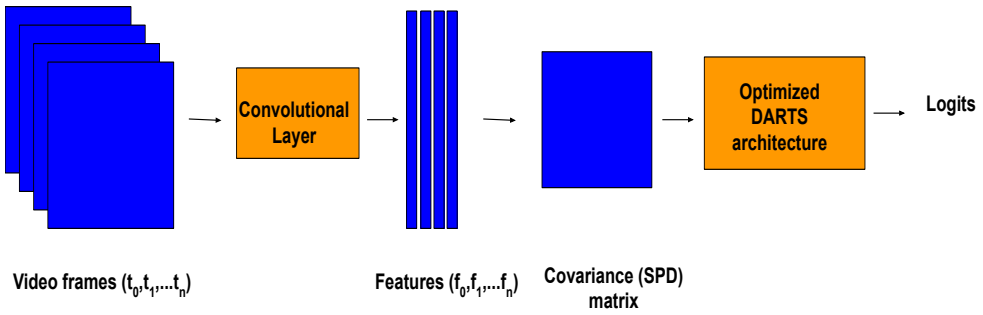


Figure 6.1: End to end training architecture

In the figure above the feature covariance matrix for n features is defined as :

$$C = \frac{1}{n-1}\sum_{i=1}^{n}(f_i - \tilde{f})(f_i - \tilde{f})^T \tag{6.1}$$

In the equation above $\tilde{f}$ is the mean of the set on features. We evaluate the network on the AFEW dataset using the optimal architecture obtained from search on HDM05 dataset.

6.4 Results

We explore two variants of the convolutional blocks.

- **Conv2D**: 2-D convolution is most popularly used for images. Its also called convolution over volume for eg: the input could be a 3-channel RGB image. In our case we could consider the video is composed of multiple images (frames) and obtain their feature mapping by passing them independently to the Conv2D layers.
- **Conv3D**: 3-D convolution is usually used for videos where we have a frame for each time span (the time axis). Conv3D layers thus have a higher number of parameters as compared to Conv2D layers. In our case we directly input the frames which compose the video into the Conv3D block to obtain the required number of features from the video.

In addition to this we tried stacking **multiple Conv2D and Conv3D** layers for hierarchical feature extraction. However they provided no further performance improvement and in some cases proved detrimental to the performance of the model.

Table 6.1: Performance of transferred SPDNetNAS Network architecture on the AFEW dataset using end to end training

Input dimension	E2E accuracy	No of input frames	Convolution type
100	**37.20%**	32	Conv2D
100	**35.04%**	32	Conv3D
400	34.23%	32	Conv2D
400	33.96%	32	Conv3D

In the table above we provide the results for Conv2D and Conv3D models for two sets of dimentionalities (100,400). From the results below we observe that the best accuracy we obtain on the **AFEW** dataset is about 37.2%. Though this slightly outperforms SPDNetBN [11] it is still not able to outperform our original SPDNetNAS which achieves an accuracy of about 38%. This could be since the optimized architecture is obtained on the original SPDNetNAS hence giving it an unfair advantage over this end-to-end approach. Inspite of this it is interesting to note that the end to end approach achieves a competitive accuracy with much fewer parameters and much smaller dimensions of the input matrix hence has a computational advantage. In general we observe that Conv2D outperforms Conv3D in both the cases and in addition

a smaller dimensionality of 100 seems to perform better, thus reducing the size of the training parameters by a factor of approximately 5. Thus end to end training seems quite promising and may benefit from more optimal feature extraction and by conducting the search phase on this end to end framework to obtain a more optimal and promising architecture.

Chapter 7

Conclusion and future work

In this work, we present a new problem for the neural architecture search of SPD manifold networks. To solve this new problem, a rich SPD cell representation and corresponding candidate operation search space is introduced, which is then blended into a parameterized SuperNet search method. The SuperNet search method explores the relaxed continuous search space leading to a bi-level optimization problem for SPD network design. The solution to our proposed differentiable bi-level optimization energy function, using backpropagation, is carefully crafted so that the weight updates follow the geometry of the SPD manifold. Quantitative results on the benchmark dataset show a commendable performance gain over handcrafted SPD networks. Additionally, we demonstrate the learned SPD architecture is transferable to other datasets as well.

That said there are several directions for future work which we plan to explore in the future:

- Enrich the architecture search space with more operations like dilations, dropout, residual connections etc..
- Extend the current CPU version to GPU to decrease the search and training cost further
- Extend the end to end model with transfer learning. For example we could use some pre-trained model like resnet or vggnet to extract the features from the frames and fine tune it further.
- Extend the SPDNetNAS by robustifying the DARTS search stage [61]

Bibliography

[1] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

[2] Dinesh Acharya, Zhiwu Huang, Danda Pani Paudel, and Luc Van Gool. Covariance pooling for facial expression recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 367–374, 2018.

[3] Karim Ahmed and Lorenzo Torresani. Connectivity learning in multi-branch networks. *arXiv preprint arXiv:1709.09582*, 2017.

[4] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*, 2017.

[5] Arindam Banerjee, Srujana Merugu, Inderjit S Dhillon, and Joydeep Ghosh. Clustering with bregman divergences. *Journal of machine learning research*, 6(Oct):1705–1749, 2005.

[6] Alexandre Barachant, Stéphane Bonnet, Marco Congedo, and Christian Jutten. Multiclass brain–computer interface classification by riemannian geometry. *IEEE Transactions on Biomedical Engineering*, 59(4):920–928, 2011.

[7] Gabriel Bender. Understanding and simplifying one-shot architecture search. 2019.

[8] Rajendra Bhatia and John Holbrook. Riemannian geometry and matrix geometric means. *Linear algebra and its applications*, 413(2-3):594–618, 2006.

[9] Jean-Daniel Boissonnat, Frank Nielsen, and Richard Nock. Bregman voronoi diagrams. *Discrete & Computational Geometry*, 44(2):281–307, 2010.

[10] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

[11] Daniel Brooks, Olivier Schwander, Frédéric Barbaresco, Jean-Yves Schneider, and Matthieu Cord. Riemannian batch normalization for spd neural networks. In *Advances in Neural Information Processing Systems*, pages 15463–15474, 2019.

[12] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

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

[14] Rudrasis Chakraborty, Jose Bouza, Jonathan Manton, and Baba C Vemuri. Manifoldnet: A deep network framework for manifold-valued data. *arXiv preprint arXiv:1809.06211*, 2018.

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

[16] Xiangxiang Chu, Bo Zhang, Jixiang Li, Qingyuan Li, and Ruijun Xu. Scarletnas: Bridging the gap between scalability and fairness in neural architecture search. *arXiv preprint arXiv:1908.06022*, 2019.

[17] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[18] Abhinav Dhall, Roland Goecke, Simon Lucey, and Tom 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.

[19] Tingxing Dong, Azzam Haidar, Stanimire Tomov, and Jack J Dongarra. Optimizing the svd bidiagonalization process for a batch of small matrices. In *ICCS*, pages 1008–1018, 2017.

[20] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017.

[21] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[22] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[23] Mark Gates, Stanimire Tomov, and Jack Dongarra. Accelerating the svd two stage bidiagonal reduction and divide and conquer using gpus. *Parallel Computing*, 74:3–18, 2018.

[24] Tobias Glasmachers. Limits of end-to-end learning. *arXiv preprint arXiv:1704.08305*, 2017.

[25] Mehrtash Harandi, Mathieu Salzmann, and Richard Hartley. Dimensionality reduction on spd manifolds: The emergence of geometry-aware methods. *IEEE transactions on pattern analysis and machine intelligence*, 40(1):48–62, 2017.

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

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

[28] Zhiwu Huang, Chengde Wan, Thomas Probst, and Luc Van Gool. Deep learning on lie groups for skeleton-based action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6099–6108, 2017.

[29] Zhiwu Huang, Ruiping Wang, Shiguang Shan, and Xilin 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.

[30] Zhiwu Huang, Ruiping Wang, Shiguang Shan, Xianqiu Li, and Xilin Chen. Log-euclidean metric learning on symmetric positive definite manifold with application to image set classification. In *International conference on machine learning*, pages 720–729, 2015.

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

[32] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, pages 2016–2025, 2018.

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

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[35] Suryansh 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.

[36] Suryansh Kumar, Anoop Cherian, Yuchao Dai, and Hongdong 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.

[37] Zhenhua Lin. Riemannian geometry of symmetric positive definite matrices via cholesky decomposition. *SIAM Journal on Matrix Analysis and Applications*, 40(4):1353–1370, 2019.

[38] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[39] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

[40] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[41] SC Martin Arjovsky and Leon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34 th International Conference on Machine Learning, Sydney, Australia*, 2017.

[42] Maher 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.

[43] Meinard Müller, Tido Röder, Michael Clausen, Bernhard Eberhardt, Björn Krüger, and Andreas Weber. Documentation mocap database hdm05. 2007.

[44] Renato Negrinho and Geoff Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017.

[45] Xavier Pennec, Pierre Fillard, and Nicholas Ayache. A riemannian framework for tensor computing. *International Journal of computer vision*, 66(1):41–66, 2006.

[46] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[47] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[48] Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics. In *Advances in Neural Information Processing Systems*, pages 4053–4061, 2016.

[49] Richard Shin, Charles Packer, and Dawn Song. Differentiable neural network architecture search. 2018.

[50] Ali Siahkamari, Venkatesh Saligrama, David Castanon, and Brian Kulis. Learning bregman divergences. *arXiv preprint arXiv:1905.11545*, 2019.

[51] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[52] Siddhant Srivastava, Anupam Shukla, and Ritu Tiwari. Machine translation: From statistical to modern deep-learning practices. *arXiv preprint arXiv:1812.04238*, 2018.

[53] Ke Sun, Piotr Koniusz, and Jeff Wang. Fisher-bures adversary graph convolutional networks. *arXiv preprint arXiv:1903.04154*, 2019.

[54] Oncel Tuzel, Fatih Porikli, and Peter Meer. Region covariance: A fast descriptor for detection and classification. In *European conference on computer vision*, pages 589–600. Springer, 2006.

[55] Oncel Tuzel, Fatih Porikli, and Peter Meer. Pedestrian detection via classification on riemannian manifolds. *IEEE transactions on pattern analysis and machine intelligence*, 30(10):1713–1727, 2008.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[57] Tom Veniat and Ludovic Denoyer. Learning time/memory-efficient deep architectures with budgeted super networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3492–3500, 2018.

[58] Ruiping Wang, Huimin Guo, Larry S Davis, and Qionghai 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.

[59] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[60] Le Yang, Marc Arnaudon, and Frédéric Barbaresco. Riemannian median, geometry of covariance matrices and radar target detection. In *The 7th European Radar Conference*, pages 415–418. IEEE.

[61] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. *arXiv preprint arXiv:1909.09656*, 2019.

[62] Xingjian Zhen, Rudrasis Chakraborty, Nicholas Vogt, Barbara B Bendlin, and Vikas Singh. Dilated convolutional neural networks for sequential manifold-valued data. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 10621–10631, 2019.

[63] Xiawu Zheng, Rongrong Ji, Lang Tang, Baochang Zhang, Jianzhuang Liu, and Qi Tian. Multinomial distribution learning for effective neural architecture search. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1304–1313, 2019.

[64] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[65] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc 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.