Nonlinear Classifier Stacking on Riemannian and Grassmann Manifolds with Application to Video Analysis

Vitaliy Tayanov

A Thesis
In the Department of Computer Science
and Software Engineering

Presented in Partial Fulfilment of the Requirements for the Degree of
Doctor of Philosophy (Computer Science and Software Engineering) at
Concordia University
Montreal, Quebec, Canada

March 2022

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Vitaliy Tayanov

Entitled: Nonlinear Classifier Stacking on Riemannian and Grassmann Manifolds

and submitted in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy Program Computer Science

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Bruno Lee

_____ Thesis Supervisor
Dr. Adam Krzyzak

_____ Thesis Supervisor
Dr. Ching Y. Suen

_____ Thesis Supervisor
Dr.

_____ Examiner
Dr. Thomas Fevens

_____ Examiner
Dr. Charalambos Poullis

_____ Examiner
Dr. Nawwaf Kharma

_____ Examiner
Dr.

_____ External Examiner
Dr. Michal Wozniak

Approved by

_____
Dr. Leila Kosseim , Graduate Program Director

March 30, 2022

_____
Dr. Mourad Debbabi , Dean

Abstract

Nonlinear Classifier Stacking on Riemannian and Grassmann Manifolds with Application to Video Analysis

Vitaliy Tayanov, Ph.D.
Concordia Univeresity, 2022

This research is devoted to the problem of overfitting in Machine Learning and Pattern Recognition. It should lead to improving the generalisation ability and accuracy boosting in the case of small and/or difficult classification datasets. The aforementioned two problems have been solved in two different ways: by splitting the entire datasets into functional groups depending on the classification difficulty using consensus of classifiers, and by embedding the data obtained during classifier stacking into nonlinear spaces i.e. Riemannian and Grassmann manifolds. These two techniques are the main contributions of the thesis. The insight behind the first approach is that we are not going to use the entire training subset to train our classifiers but some part of it in order to approximate the true geometry and properties of classes. In terms of Data Science, this process can also be understood as Data Cleaning. According to the first approach, instances with high positive (easy) and negative (misclassified) margins are not considered for training as those that do not improve (or even worsen) the evaluation of the true geometry of classes. The main goal of using Riemannian geometry consists of embedding our classes in nonlinear spaces where the geometry of classes in terms of easier classification has to be obtained. Before embedding our classes on Riemannian and Grassmann manifolds we do several Data Transformations using different variants of Classifier Stacking. Riemannian manifolds of Symmetric Positive Definite matrices are created using the classifier interactions while Grassmann manifolds are built based on Decision Profiles. The purpose of the two aforementioned approaches is Data Complexity reduction. There is a consensus among researchers, that Data Complexity reduction should lead to an overfitting decrease as well as to classification accuracy enhancement.

We carried out our experiments on various datasets from the UCI Machine Learning repository. We also tested our approaches on two datasets related to the Video Analysis problem. The first dataset is a Phase Gesture Segmentation dataset taken from the UCI Machine Learning repository. The second one is the Deep Fake detection Challenge dataset. In order to apply our approach to solve the second problem, some image processing has been carried out. Numerous experiments on datasets of general character and those related to Video Analysis problems show the consistency and efficiency of the proposed techniques. We also compared our techniques with the state-of-the-art techniques. The obtained results show the superiority of our approaches for most of the cases. The significance of carried out research and obtained results manifests in better representation and evaluation of the geometry of classes which may overlap only in feature space due to some improper measurements, errors, noises, or by selecting features that do not represent well our classes. Carried out research is a pioneering in terms of Data Cleaning and Classifier Ensemble Learning in Riemannian geometry.

This thesis is dedicated to my parents. Their endless encouragement and support allowed me to complete my thesis and finish my graduate studies. I appreciate everything that you have done for me.

Acknowledgement

I wish to express my gratitude to my advisors Drs. Ching Y. Suen and Adam Krzyżak for their continuous support in different ways during my PhD studies. Besides my advisors, I would like to thank the rest of my thesis committee for their valuable comments and discussions. My particular recognition is to our laboratory manager Nicola Nobile for his technical support and help. I want to express my gratitude and appreciation to Compute Canada for offering powerful computing resources. It allowed me to carry out numerous experiments and obtain results that have a particular value, which would not be possible with only computing resources from our laboratory at Concordia University. Finally, I wish to acknowledge the Natural Sciences and Engineering Research Council of Canada for their support.

Contents

List of Figures

List of Tables

Acronyms

AdaBoost Adaptive Boosting viii, 4, 39, 40, 41, 44

ANN Artificial Neural Networks 30

BKS Behaviour Knowledge Space 7

CCE Cascades of Classifier Ensembles vi, 1, 2, 4, 5, 14, 48, 49, 54, 59, 72, 84, 85

CGRBM Convolutional Gated Restricted Boltzmann Machines 22

CNN Convolutional Neural Networks viii, ix, 2, 3, 4, 8, 20, 21, 22, 23, 25, 26, 27, 31, 33, 50, 51, 52, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 70, 71, 72, 77, 82, 84, 85

CPPM Classification Prediction Pairwise Matrix vi, 4, 49, 50, 56, 59, 63, 70

CRF Cascades of Random Forests ix, 1, 5, 9, 14, 48, 59, 69, 70, 71, 72, 73, 74, 75, 76, 79, 80, 81, 82, 84, 85

CS Christoffel Symbols 50, 51, 59, 85, 94

CV Computer Vision 2, 19, 20, 28, 30, 31

DCGAN Deep Convolutional Generative Adversarial Network 85

DCNN Deep Convolutional Neural Networks 14, 84

DF Deep Forensics 30, 32, 52

DFC Deep Forensics Challenge 30

DFDC Deep Fake Detection Challenge xi, 30, 31, 32, 33, 78, 81, 82, 84, 85

DL Deep Learning 2, 20, 21, 27, 28, 63, 69

DLA Deep Learning Architecture 1, 2, 5, 9, 14, 20, 21, 28, 31, 51, 52, 79, 84

DNF Deep Neural Forests 1, 5, 56, 57, 58, 62, 70, 72, 84, 85

DNN Deep Neural Networks 1, 2, 14

DNRF Deep Neural Random Forests vi, 16, 17, 18

DP Decision Profile 4, 5, 48, 52, 54, 55, 56, 59, 63, 64, 69, 85

DRF Deep Random Forests viii, 14, 15, 17

DT Decision Trees viii, ix, x, 2, 14, 16, 17, 55, 56, 57, 59, 60, 61, 70, 71, 72, 73, 74, 75, 76, 77, 80, 81, 84, 85

DTL Decision Template 52, 53, 54

E-and-R-metrics Euclidean-and-Riemannian-metrics 49

E-and-R-geometry Euclidean-and-Riemannian-geometry 2, 59

ELCS Efective Local Shatter Coefficient 11

E-subspace Euclidean Subspace 2

E-space Euclidean Space 2, 4, 5, 18, 19, 51, 52, 56, 57, 59, 83, 85

E-geometry Euclidean Geometry 2, 52, 54, 58, 72

ET Extra Trees ix, x, 2, 3, 8, 52, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 70, 71, 72, 75, 76, 77, 80, 81, 84

FSGAN Facial Swapping Generative Adversarial Networks 32

GAN Generative Adversarial Network 27, 32, 78

GC Grassmann Cascades 60, 63, 64, 69, 70, 71

G-manifold Grassmann Manifold ix, 2, 4, 20, 48, 53, 54, 55, 56, 57, 59, 63, 68, 69, 85, 86

HoF Histogram of Optical Flow 2, 20, 26

HoG Histogram of Oriented Gradients 2, 20, 26

HQ High Quality 28, 30

ISA Independent Subspace Analysis 22

k-NN K-Nearest Neighbors Algorithm 8, 52, 54, 55, 56, 57, 58, 61, 62, 63, 85

LQ Low Quality 28, 30

LR Logistic Regression 39, 40

LSTM Long Short-Term Memory viii, ix, 2, 3, 5, 26, 27, 62, 79, 80, 81, 82, 84

MBH Motion Boundary Histogram 2, 20, 26

ML Machine Learning 1, 31, 35, 83

MLP Multilayer Perceptron 8, 52, 56, 57, 58, 62, 63

MTCNN Multi-task Cascaded Convolutional Networks 31, 33, 78

MV Majority Voting 2, 4, 7, 8, 9, 48, 70

NB Naive Bayes 39, 40

NN Neural Networks 1, 2, 5, 9, 16, 28, 29, 31, 62, 69, 85

NTH Natural Talking Heads 32

PCA Principal Component Analysis 51, 52, 59, 85

PFM Pair-Wise Fusion Matrix 9

PR Pattern Recognition 35

ReLU Rectified Linear Unit 29

R-metric Riemannian metric 18, 19

R-geometry Riemannian Geometry 2, 48

R-and-G-manifolds Riemannian and Grassmann manifolds 2, 3, 4, 19, 54, 56, 62, 84, 85

RF Random Forests ix, x, 2, 3, 8, 14, 15, 18, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 70, 71, 72, 73, 74, 77, 80, 84

R-manifold Riemannian Manifold viii, 2, 3, 4, 5, 18, 19, 20, 48, 50, 51, 52, 54, 56, 57, 59, 62, 70, 72, 80, 84, 85

RO Random Oracle 39, 40

RS Random Subspace 39, 40

SGD Stochastic Gradient Descent 17, 18, 23

SPD Symmetric Positive Definite viii, 2, 3, 4, 5, 19, 48, 50, 51, 52, 54, 56, 57, 59, 62, 63, 70, 84, 85, 94, 95

SVD Singular Value Decomposition 53, 55, 85, 86, 97

SVM Support Vector Machines 2, 8, 20, 21, 23, 25, 26, 52, 56, 57, 58, 62, 63, 70, 71, 84

UCI University of California, Irvine 3, 54, 60, 70, 84, 85

VAE Variational Autoencoder 85

VC Vapnik-Chervonenkis 9, 10, 11, 12, 83

VLAD Vector of Locally Aggregated Descriptors 21

List of Principal Symbols

ϵ_t Total error of classifier consensus

γ_a Portion of ambigous data in a dataset

γ_m Portion of misclassified patterns in a dataset

λ_i Eigenvalues of a square matrix

$\mathbb{R}^n$ Euclidean n-space

A^T Matrix transpose

$A^\ell(x)$ Classification prediction pairwise matrix

A^k_{uv} Co-association matrix for algorithms u and v and pattern k

$d(x, y)$ Distance function between objects x and y

$diag(A)$ Matrix diagonalization

$DP(x)$ Decision profile for pattern x

DT_k Decision template for class k

F Set of strong classifiers

$f \circ g$ Function composition

g Riemannian metric

G^m_D Grassmann manifold

H Homotopy between two functions or topological spaces

L Number of classes

$p(y = C_\ell|X)$ Prediction probability for class C_ℓ having feature vector X

$Proj(A)$ Matrix projection

S^+_d Symmetric positive definite matrix

T Number of individual classifiers in a classifier ensemble

$T_x\mathcal{M}$ Tangent space at point x on a manifold $\mathcal{M}$

$u \equiv v$ Operation of taking consensus between classifiers u and v

$u(x, y)$ Conformal factor depending on local coordinates

$vec(A)$ Matrix vectorizing

X	Feature space
Y	Orthonormal basis matrix
y	Class label
Z	Prediction space

1 Introduction

1.1 Motivation and intuition

There are two principal approaches to enhance classification accuracy: 1) use more advanced and complex classifiers and architectures (classifier ensembles and deep learning architectures (DLA)) or 2) use data transformation (kernel-based algorithms and nonlinear manifolds). However, sometimes it is hard to distinguish between these two categories because classifier stacking [88] performs data transformation and there are multiple data transformations when learning optimal features in DLA. Recently, in [89], it was proposed to use cascades of classifier ensembles (CCE) such as cascades of random forests (CRF) as an alternative to DLA based on neural networks (NN). Again, in this situation, the approach can be classified either as data transformation or by using advanced complex architectures. There are also combined approaches such as deep neural forests (DNF) [44] where random forests which are classifier ensembles are trained using deep neural networks (DNN). One can create architectures that use all the aforementioned techniques. Focusing principally on geometric-based transformations to classifier ensemble learning, we involve all four techniques to build different algorithms, and during the experiments compared them.

There is another big problem in machine learning (ML) known as overfitting. We propose to solve it in two different ways: by splitting data into functional groups and by applying nonlinear geometry to a classifier stacking. Those approaches allow boosting the classification accuracy as well.

Splitting data into functional groups depending on classification difficulty by classifier consensus allows obtaining a subset where consensus between classifiers is not achieved. Such a group of ambiguity patterns is then reclassified by some other classifier or classifier ensemble. This allows to break a difficult classification problem into parts and solve them separately. Generally, we have three subsets of patterns after applying consensus of classifiers: subset of easy patterns (where consensus is achieved, and it is correct), subset of ambiguous patterns (where consensus is not achieved) and subset of misclassified patterns (where consensus is achieved, and it is incorrect). The last subset is irreducible and will determine the lower bound of the classification error. When consensus is achieved, then classification is accomplished by consensus automatically. One of the reasons for overfitting is applying a rather complex classifier to a relatively small dataset. Less complex classifiers may lead to an underfitting problem. This is one of the reasons why very advanced DLA based on NN fail in case of datasets hard for classification when those datasets are relatively small. We can define misclassified patterns as those that can not be classified correctly by most of the existing advanced algorithms. If a dataset is small, then it will be difficult for any classifier to focus itself on those three types of patterns. In particular, partial focusing on misclassified patterns can lead to change the separation hyperplane in a way that may lead to wrong classification of some part of easy and ambiguous patterns. This is where the overfitting issue arises. Consensus of classifiers disregards misclassified patterns as those that are impossible to classify correctly both during testing phase by most or all of the existing advanced classifiers. Thus, they have no any influence on the classifier trained only on the part of datasets consisting of ambiguous patterns only. This allows to avoid the influence of misclassified patterns on overfitting. If easy patterns have some influence on a classifier building, disregarding them

on the second phase of classification where the classifier is trained purely on the ambiguous patterns helps to avoid overfitting caused by this portion of patterns.

Another way to deal with the overfitting problem is to use classifier stacking. Classifier stacking performs data transformation by using predictions from base classifiers to be used as new features to train another classifier, known as meta-classifier or meta-learner. It is known to be superior to classifier ensembles using majority voting (MV) as a decision rule [88]. There is a recursive classifier stacking built as CCE. The principal objective of introduction of CCE is to create an architecture that involves fewer parameters and has overall less computational load than DNN. On the other hand, CCE is a DLA where every cascade performs the role similar to hidden or convolutional layers of DNN. Usually, as an ensemble of classifiers for every cascade, random forests (RF) or extra trees (ET) are taken. It is known that for many applications, RF or ET outperform NN with one hidden layer. Applying cascades with shallow decision trees (DT) allows us to avoid the overfitting issue. If it is true for every cascade (assume for simplicity that our cascades are identical) then applying multiple cascades will not lead to overfitting. Different data transformation techniques such as R-and-G-manifolds targeting the reduction of data classification complexity by projecting data in nonlinear spaces will additionally reduce the overfitting both for simple classifier stacking or recursive classifier stacking using CCE.

In some applications of Computer Vision (CV) such as video analysis, our data can lie on a nonlinear manifold or classes can not be separated effectively using classification algorithms in Euclidean geometry (E-geometry). Thus, all well-known classifiers, including classifier stacking and recursive classifier stacking using, CCE cannot be applied directly. In this situation, it is crucial to develop classifier stacking that can be applied in nonlinear geometry. That is why in this work we implemented classifier stacking including recursive classifier stacking in Riemannian geometry (R-geometry).

To apply standard classification algorithms in case of usage of Riemannian manifold (R-manifold), one needs to linearize a manifold. Linearization is possible because nonlinear manifolds are homeomorphic to E-space. R-manifold of symmetric positive definite (SPD) matrices and G-manifold which is a manifold of E-subspaces have a broad application in CV. Using linearization of R-manifold is computationally expensive. That is why it is important to know which algorithms can handle nonlinear problems close enough to those working in R-geometry. On the other hand, it is interesting to know which classification algorithm can be applied to learn best on R-manifold. G-manifold assume application of some particular distances that count for the nonlinearity of those manifolds.

Standard algorithms applied to perform video analysis use handcrafted features such as Histogram of Oriented Gradients (HoG), Histogram of Optical Flow (HoF), Motion Boundary Histogram (MBH) and others. Then features are fed into SVM to perform the classification task. However, automatically selected robust features by DNN have different levels of abstraction should lead to better performance and feature engineering problem will be less costly in terms of time complexity. Supervised deep learning (DL) video classification can be divided into image-based video classification, end-to-end CNN architectures and long-term temporal dynamics modelling. We are going to apply CNN-based architecture to select features from every frame and then LSTM with attention for temporal analysis of features selected by CNN. We then compare this pipeline with another one where feature transformation is carried out using classifier stacking in both E-and-R-geometry.

1.2 Thesis organization

The thesis is organized as follows:

- **Chapter 2** is devoted to the review of methods related to classifier ensembles and their cascades. The overfitting problem specific for the ensembles of classifiers and classifier selection problem has also been considered. Some introduction to nonlinear manifolds has been provided. An overview of principal methods related to video analysis has been carried out.

- In **Chapter 3** we consider the overfitting phenomenon as some function of data complexity. One points out three categories of instances depending on their classification difficulty: easy, ambiguous and misclassified. To detect easy and ambiguous patterns, we apply a consensus of classifiers. Easy and ambiguous patterns are those where consensus is reached or not. Misclassified instances are among the easy ones when consensus is erroneous. We established upper and lower bounds for classification errors based on ambiguous and misclassified patterns. Using ambiguous instances only instead of all of them to train the final classifier allows to significantly reduce the overfitting and boost the classification accuracy. Some experiments show the advantage of the proposed approach.

- **Chapter 4** deals with different types of data transformation, including cascades of classifier ensembles and embedding to two different nonlinear manifolds. Homotopy diagrams of all data transformations using R-and-G-manifolds have been designed and presented. For the experiments, we have used 12 datasets from the UCI repository [22], 11 of which are general and 1 dataset related to the gesture segmentation problem. As seen from the experiments, in the prevailing number of cases, the most informative patterns are those created using classifier interactions. These patterns are created based on prediction probabilities for every classifier in the pair of classifiers. Data obtained as classifier interactions consist of objects lying on the R-manifold of SPD matrices.

- **Chapter 5** considers the application of some proposed techniques of data transformation to solve face forgery detection problems which belong to video analysis problems in general. We compared it to the architecture which does not use additional data transformation. We used cascades of RF and ET to transform initial features obtained using VGG 16 CNN. The experimental results show that using additional data transformation leads to "almost no overfitting" when using LSTM to catch the temporal dynamics.

- We conclude and summarize in **Chapter 6**. We have also mentioned what can be done as future research. Among principal conclusions, we can mention the proof of two different ideas that aimed to reduce the data complexity needed for training the classifier or classifier ensembles. As a consequence, this leads to overfitting reduction. Data complexity reduction has been accomplished in two different ways: by splitting data into functional groups or categories depending on the classification difficulty and by data transformation involving embedding our data on nonlinear manifolds. Numerous

experiments show the consistency of the proposed ideas. As principal research directions, we can mention building cascades of neural random forests and back-propagate through them. We are also going to build a geometric version of CCE. As one more direction, we propose to use generative models to learn a mapping between different manifolds.

1.3 Principal novel contributions

We are providing the principal contributions below.

In Chapter 3 we propose to consider misclassified patterns (those who have large negative margins) as the main reason for overfitting. We use classifier consensus to detect misclassified and easy classification patterns and exclude them from the training set. Consequently, we use ambiguous instances (those where a consensus of classifiers is not reached) as a new training set. Unlike the Adaptive Boosting (AdaBoost), [23] algorithm we are focusing only on ambiguous patterns which we consider as true difficult ones instead of misclassified and ambiguous together as in the case of AdaBoost. According to our belief, one of the main factors that cause AdaBoost overfitting is too much focusing on misclassified patterns. Eliminating them from the training set is the principal contribution and novelty of our approach. We also developed an algebraic approach based on abstract algebra of how to measure a distance or dissimilarity between classifier ensembles in case of crisp labels. To the best of our knowledge, the concept of distance between classifier ensembles does not appear in the literature. Using individual base classifiers of a classifier ensemble, the distance between classifier ensembles can be measured for every instance. To measure a dissimilarity between ensembles using the final decision made by using MV, for instance, one needs an entire training set.

Tayanov, V., Krzyżak, A. and Suen, C.Y.: Classification Boosting by Data Decomposition Using Consensus-Based Combination of Classifiers. *In: Proceedings of the International Conference on Image Analysis and Recognition (ICIAR)*, Montreal, Canada, July 5-7, 2017, pp. 408–415

Tayanov, V., Krzyżak, A. and Suen, C.Y.: Some Properties of Consensus-Based Classification. *In: Proceedings of the 10th International Conference on Computer Recognition (CORES'2017)*, Polanica-Zdrój, Poland, May 22-24, 2017, Springer Lecture Notes on Advances in Intelligent Systems and Computing, vol. 578, pp. 276–285.

Chapter 4 is devoted to data transformation and embedding transformed data on nonlinear manifolds, such as R-and-G-manifolds. The main novelty and contribution is in application of differential geometry and nonlinear manifolds to build classifier ensembles in nonlinear spaces which are R-and-G-manifolds. Using interactions between classifiers presented as CPPM allow creating patterns that are objects of the R-manifold of SPD matrices. We then linearize those manifolds, obtaining vectors that are objects of the Euclidean space (E-space). In E-space, we use a set of generally known classifiers. We can feed CNN with SPD matrices to learn from them. SPD matrices fed into CNN can be linearized or not. Both linearized and nonlinearized versions of SPD matrices can be vectorized as well. G-manifold is built from DP [48]. We can consider such G-manifold

as a nonlinear version of DP. As an additional classifier, we implemented DNF applied to SPD matrices directly. All general classifiers used in our experiments have linearized and nonlinearized versions. As an alternative to DLA based on NN, we implemented CCE which are CRF. We build the dependencies of the classifier accuracy as a function of the number of cascades in CRF. From the obtained results, we can conclude that CRF that operate on nonlinear manifolds need fewer cascades than those which operate in E-space. CRF applied on R-manifold are characterized by higher accuracy. Numerous experiments show the consistency of all aforementioned techniques.

Tayanov, V., Krzyżak, A. and Suen, C.Y.: Analysis of Different Deep Learning Architectures to Learn Generalised Classifier Stacking on Riemannian and Grassmann Manifolds. *In: Proceedings of the IEEE International Conference on Pattern Recognition (ICPR)*, Montreal, Canada, August 21-25, 2022 (accepted).

Tayanov, V., Krzyżak, A. and Suen, C.Y.: Ensemble learning using matrices of classifier interactions and decision profiles on Riemannian and Grassmann manifolds. *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, Vol. 35, No 12, (2021) 2160011, 22 p.

Tayanov, V., Krzyżak, A. and Suen, C.Y.: Comparison of Stacking-based Classifier Ensembles using Euclidean and Riemannian Geometries. *In: Proceedings of the IEEE International Conference on Pattern Recognition (ICPR)*, Milan, Italy, January 10-15, 2021, pp. 10359–10366.

Tayanov, V., Krzyżak, A. and Suen, C.Y.: Manifold-based classifier ensembles. *In: Proceedings of the International Conference on Pattern Recognition and Artificial Intelligence (ICPRAI)*, Zhongshan, China, October 19-23, 2020, pp. 293–305.

Tayanov, V., Krzyżak, A. and Suen, C.Y.: Learning from classifier predictions in Euclidean and Riemannian geometries.Pattern Recognition (submitted).

Tayanov, V., Krzyżak, A. and Suen, C.Y.: Prediction-based classification using learning on Riemannian manifolds. *In: Proceedings of the IEEE International Conference on Pattern Recognition (ICPR)*, Beijing, China, August 20-24, 2018, pp. 591–596.

Tayanov, V., Krzyżak, A. and Suen, C.Y.: Learning classifier predictions: is this advantageous? *In: Proceedings of the International Conference on Pattern Recognition and Artificial Intelligence (ICPRAI)*, Montreal, Canada, May 14-17, 2018, pp. 541–544.

We applied some proposed techniques to face forgery detection problem considered in Chapter 5. Indeed, we used CRF to perform a data transformation before feeding it to LSTM. Experimental results show that our pipeline does not overfit, unlike the one that does not use such data transformation. From the previous experiments on the Gesture Phase Segmentation dataset, we have seen that CRF applied in E-space can approximate classification results obtained using R-manifold. Taking into account that face forgery detection problem may require very large datasets, to transform and to embed data on Riemannian or Grassmann manifolds is unjustified.

1.4 Author's contribution

The contributions from each author for every of the published or submitted papers are presented below.

1. **Some Properties of Consensus-Based Classification**
2. **Classification Boosting by Data Decomposition Using Consensus-Based Combination of Classifiers**
3. **Learning classifier predictions: is this advantageous?**
4. **Prediction-based classification using learning on Riemannian manifolds**
5. **Learning from classifier predictions in Euclidean and Riemannian geometries**
6. **Manifold-based classifier ensembles**
7. **Comparison of Stacking-based Classifier Ensembles using Euclidean and Riemannian Geometries**
8. **Ensemble learning using matrices of classifier interactions and decision profiles on Riemannian and Grassmann manifolds**
9. **Analysis of Different Deep Learning Architectures to Learn Generalised Classifier Stacking on Riemannian and Grassmann Manifolds**

- Vitaliy Tayanov: Ideas and concepts generation, creating mathematical models and algorithms, planning the experiments, interpretation results, conclusions, creation of the first draft and editing
- Adam Krzyżak: Supervision of the research, funding, editing, discussions
- Ching Y. Suen: Funding, editing, literature support, connection with other researchers, discussions

2 Background

2.1 Introduction

Generally, advanced classifier fusion, classifier fusion learning or optimal fusion learning are the most successful techniques for classifier combining [88, 47, 57]. This includes classifier stacking, behaviour knowledge space (BKS) [30], decision templates and profiles [48], pairwise matrix fusion [43], Bayesian inference [13] and others. Deterministic classifier fusion schemes require more accurate and more diverse individual classifiers. Kappa-error diagram [47] shows the relation between the diversity of a pair of classifiers and the error made by the ensemble of those classifiers. According to the analysis using the aforementioned diagrams, the most accurate classifiers are not the most diverse. Some analysis of different diversity measures and their relations with the classification accuracy of the MV has been carried out in [49]. For analysis, authors took four averaged pairwise measures and six non-pairwise measures. It has been claimed that there is a strong correlation between different diversity measures. During all the experiments, all base classifier in created ensembles had approximately same accuracy. The results of the carried out experiments show that there is no pronounced relationship between the diversity and the averaged accuracy of individual classifiers. Notwithstanding, authors think that in different situations, having real-world problems, diversity and accuracy can exhibit a stronger relationship. As a conclusion, authors agree that the general motivation for designing diverse classifiers is correct, but the problem of measuring this diversity and to effectively use it for building better classifier ensembles is still remains open. In [9] an interesting analysis on classifier diversity has been carried out. Authors propose to split the general classifier diversity into two parts: 'good' diversity and 'bad' diversity. Here, a classifier 'diversity' measures the average squared loss of the individual base classifiers from the ensemble prediction. The 'good' diversity measures the disagreement on datapoints where the ensemble is correct–and due to the negative sign, any disagreement on these points increases the gain relative to the average individual error. The 'bad' diversity measures the disagreement on datapoints where the ensemble is incorrect–here, the diversity term has a positive sign, so any disagreement reduces the gain relative to individual error. Also, the classification error of the MV combiner can be decomposed into three terms: individual accuracy, 'good' diversity, and 'bad' diversity. It has been noticed that diversity measure should be naturally derived as a direct consequence of two factors: the loss function of interest, and the combiner function. During the experiments, it was established that the diversity terms tend to exhibit a large variance in smaller ensembles, and stabilize with very large ensembles. Comparing ensembles with weak base classifiers having the accuracy slightly better than a random choice and with selected individual classifiers having a relatively high accuracy, the following observation can be made. Ensembles composed with weak classifiers have better improvement in comparison with ensembles built by using selected classifiers. Both ensembles have lower error rate when the number of individual classifiers grows.

2.2 Classifier selection and classifier fusion problems

Learning from classifier predictions does not necessarily require classifiers to be accurate. At least, there is no straightforward dependency between those two characteristics of classifiers. The only requirement for them is to be diverse. As it will be shown during the experiments, more diverse classifiers can initiate manifolds with a higher nonlinearity. The classifier predictions can be learnt by a meta-classifier such as CNN, Multilayer Perceptron (MLP), SVM, k-nearest neighbours classifier (k-NN) and others (in contrast to deterministic MV as for regular RF or ET). For the weighted fusion, the weights for classifier predictions can be estimated using the aforementioned learning algorithms as well.

Conclusions made in [47] about the importance of individual classifier accuracy to build a good ensemble are mostly based on the following statement. Assume that we have an infinite number of voters and the probability that each of them makes correct decisions is slightly better than 0.5. Then the probability of correct decision of the whole ensemble of voters or individual classifiers is equal to 1 [57]. Thus, if we have n independent binary classifiers each characterized by an error, ϵ then the classification error of the entire ensemble can be written formally as follows [88]:

$$\epsilon_{ensemble}(n) = \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} \binom{n}{i} \epsilon^i (1-\epsilon)^{n-i}. \tag{1}$$

Putting $n \to \infty$ and $\epsilon < 0.5$ we can see that

$$\epsilon_{ensemble}(\infty) = \lim_{n \to \infty} \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} \binom{n}{i} \epsilon^i (1-\epsilon)^{n-i} = 0. \tag{2}$$

However, in practice, the aforementioned conditions are very hard to achieve. There are numerous approaches on how to create independent classifiers [57] but none of them allows generating perfectly independent ones. Moreover, there can be samples in a dataset for which a probability of correct classification is less than 0.5. This probability can be evaluated by putting this sample into different testing subsets and compute the relative number of times when it was classified correctly. In [69] authors give an example when individual classifiers have better accuracy than MV decision-making. They also underlined that these observations are relevant only if applied to jointly independent (not just uncorrelated, or equivalently, pairwise independent) errors and not to classifiers. On the other hand, splitting individual base classifiers by error rate leads to decrease of the probability of overfitting [74], while diversity of them leads to its increase. So, as we see, there are problems that do not require base classifier in a classifier ensemble to be diverse in case of MV decision-making. There is a review of the state-of-the-field with regard to explanations of what the term 'diversity of errors' means, and why it can lead to improved ensemble performance [8]. Although, it has no any stated conclusions, it provides the example when diversity can be harmful. Combining with the research made in [74] it is possible to make a conclusion that diversity may be beneficial only for accurate classifiers. Splitting classifiers by error rate when individual classifiers produce large error values can drastically increase the error of the entire ensemble using MV as a decision-making rule.

Learning from classifier predictions in opposite to MV decision-making does not require the individual accuracy of classifiers in an ensemble, thus allowing to use of simpler classifiers, which lead to less overfitting and less computational load for the classifier ensemble generating the predictions. Optimal ways of how to build an ensemble of classifiers that are less overfitted are discussed in [74]. According to the paper, to make an ensemble of classifiers less overfitted, we have to put the classifiers in a chain and split them by error rate. A chain of classifiers means that the Hamming distance between the two consecutive classifiers in an ensemble is equal to 1. So by combining chaining and splits we can have chains, non-chains, splits and non-splits. The minimal value of overfitting is achieved for the chain-split combination. However, it is noticed also that splitting is much more important to reduce the overfitting for the easiest problems, where the effect of splitting declines with the growth of the problem complexity. This is also one of the reasons why we have to reduce the data classification complexity.

Another way to reduce an overfitting and classifier complexity is to divide a total dataset into functional groups depending on the type of patterns: easy for classification patterns that yield sufficiently large positive margin and ambiguous patterns which yield low absolute value of positive or negative margin [64]. The easy instances are classified automatically, while ambiguous ones should be reclassified using some other classifier. Because the post classifier should focus itself on the certain type of patterns (ambiguous) only, it can reduce overfitting and use a simpler algorithm. The approach with some experiments showing the advantages of the proposed idea is given in [64].

The idea of using classifier interaction to obtain better fusion of classifiers has been explored in [43]. For that purpose, a pair-wise fusion matrix (PFM) is introduced. Then MV decision-making rule is applied for this fusion matrix. It was however shown that using PFM-MV is advantageous in comparison to the standard MV. It should be noticed that idea of using interactions between different instances was proposed in [37, 59]. In those works, probabilistic models based on distributions of pairwise distances between different samples were built for very small training sets.

One can also combine original features and those obtained from some other classifier (for instance Restricted Boltzmann machines) and then use both of them during the second stage of ensemble stacking or meta-learning. This idea is presented in [84].

There are some interesting recent approaches dedicated to CRF [89, 52]. The principal idea is to create a DLA using classifier ensembles instead of NN. In [89] every next cascade uses data coming from the previous cascade of DF, however, in [52] outputs from every decision tree are used. For both approaches, one stops adding more cascades when training classification accuracy is not improved. The experimental results on different datasets show the successful applications of such architectures.

2.3 Probability of overfitting of classifier ensemble

In machine learning, there is a very important problem of improving Vapnik-Chervonenkis (VC) bounds. Interesting related works devoted to improvement of overrated VC bounds are [74] and [73]. However, here we are focusing on the analysis of the first paper.

In [74] it was shown how diversity between classifiers influences the probability of overfitting. By overfitting, one understands the difference between error rates during testing and

training. Having low overfitting is much more important than obtaining a low error rate during training. Low overfitting means that we have a good generalization ability of our algorithm. For ensembles of classifiers, the overfitting is much more complicated because each algorithm has its overfitting and error rate. Also taking into account diversity between classifiers and splitting them by error rate has a big influence on the improvement of VC bounds which means the better estimation of the probability of overfitting.

The principal challenge of creating a very accurate ensemble consists in the following dilemma. On one hand, to have a very accurate ensemble we need to have diverse accurate classifiers [47]. If they are diverse and have a large error rate, the accuracy drops dramatically. At the same time, diversity of classifiers leads to overfitting [74]. So the challenge is how to have a very accurate ensemble of classifiers that have low overfitting.

Thus, in [74] it has been analyzed the influence of splitting classifiers by error rate and the diversity of classifiers on the probability of overfitting of an ensemble of T classifiers. For diversity measurement, Hamming distance has been used. The author claims that some factors influence the probability of overfitting, however, they are negligible in comparison with the two mentioned above. Because all the experiments should be performed on the finite set of objects, the notation of week axiomatic in probability theory has been introduced. Week axiomatic does not require the existence of any probability density (density might not exist in some cases) and does not operate on probabilities that require having infinite sets of objects. In reality, we do not have infinite samples hence probability density function and probability function can not be evaluated, and we always operate on a limited number of realizations. Instead of the probabilities, we can use just estimates which are built based on a purely combinatorial way by executing all possible splits of a generalized set on subsets with lower cardinality and averaging all particular estimates obtained on these particular subsets. This might be a very good approximation of unknown probabilities.

First, some definitions about what is a generalized sample, objects, classifiers learning algorithm, learning function, risk minimization problem, deviation of error rates, weak axiomatic and some others have been provided. Mainly it was claimed that analogues of some theorems in probability theory can be achieved in weak axiomatic, particularly strong Kolmogorov axiom. Then the notations of the error vector, shatter coefficient and shatter profile that come from VC theory have been introduced. Error vector has a size equal to the number of samples and is a binary one where 1 or 0 show if the algorithm made an error or not on a particular object correspondingly. The shatter coefficient shows the number of different error vectors. There is also a shatter profile that shows the number of different error vectors for each error rate. If we generate a very large pool of classifiers, then only a relatively small part of classifiers can produce sufficiently small errors. If we build a distribution of shatter coefficient, it will be very peaky (near 50% of made errors). If we want to generate a pool of diverse classifiers that have small errors, this is very difficult to achieve. It means that we can have alternatively good classifiers, which might mean the high complexity of a data manifold. It is very interesting to discover how an ensemble of classifiers depends on the shape of a manifold: how the shatter coefficient and the shatter profile depend on the topological aspects of a manifold.

There are some works on the improvement of VC bound. One approach generates a random subset of classifiers by the Monte-Carlo algorithm. However, this approach does not give any improvement for VC bound and estimates are cumbersome by form and computation.

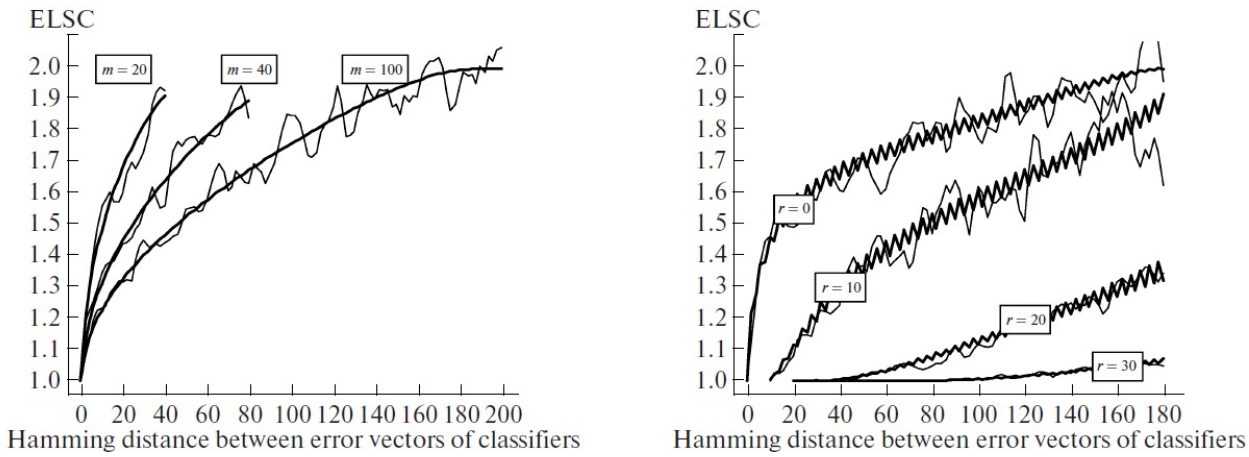


Figure 1: Influence of split by error rate on the rate of overfitting

Another approach uses the luckiness function, ordering all classifiers from the set by their preference for the particular object. Then union bound from classical VC theory applied to estimate covering numbers. However, this approach leads to overfitting. The effect of the similarity of classifiers on the probability of overfitting is rather badly studied. In those studies where this effect has been taken into account, significant improvements were not achieved. Another attempt to improve the bounds is to consider a connectedness of a parametric set of classifiers. The idea of this approach consists of the following assumption. Continuous changing of one of the coordinates of a multidimensional parameter γ leads to variations in error vector only by one element (classifiers differ on one object). Thus, error vectors of all classifiers almost always form a connected graph whose edges connect the nodes corresponding to vectors that differ by one element. The corresponding version of this estimate can be obtained in weak axiomatic. Notwithstanding, this approach also does not give considerable improvements of VC bound.

In a simple example considering an ensemble of two classifiers, it was shown that it is possible to obtain exact bounds of overfitting based on weak axiomatic using only combinatorial arguments. It was shown also that overfitting arises even in a simple case, and taking into account the similarity of classifiers and splitting them by error rates helps to improve VC bounds. For the experiments, it is more convenient to use an effective local shatter coefficient (ELCS). By this, one means the ratio between the probability of overfitting for an ensemble of classifiers and the best classifier (by error rate) in the set. The maximum value of this coefficient for an ensemble of two classifiers is equal to 2 (see Figure 1). Using the results of these experiments, the following conclusions can be made. Overfitting is induced by incompleteness in data used to train classifiers in ensemble. If classifiers are different and make the same amount of errors (no splitting by error rates) then the maximum value of ELCS is nearly attained (ELCS value of 2) meaning the maximum of overfitting. If classifiers are similar, then ELCS approaches 1. In this case, the ensemble behaviours as one single classifier. If classifiers in an ensemble are split by error rate ELCS also does not attain the VC bound.

Consequently, more experiments were carried out for ensembles with arbitrary number of classifiers. To this end, the chain of classifiers is introduced (see some experiments on Figure

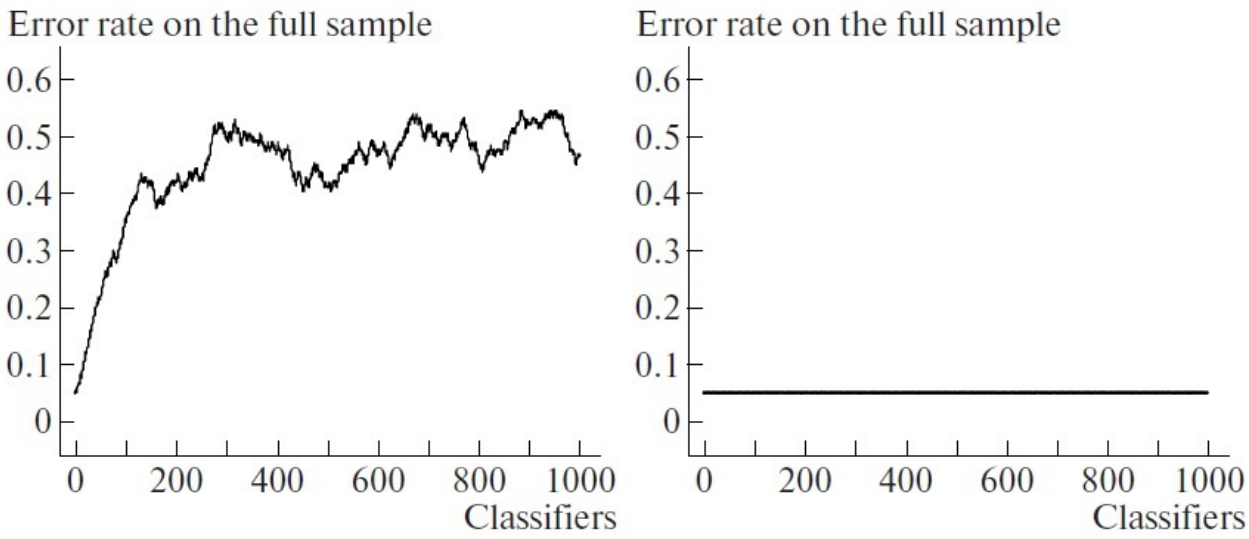


Figure 2: Classifier chains with splitting (left) and with no splitting (right)

2). By the chain of classifiers, one understands a sequence of classifiers where Hamming distance between two neighbour classifiers in a chain is equal to 1. Then four notations are introduced: chains, nonchains, splits and nonsplits. We have splits if classifier set produces different errors and nonsplit–when errors alternate between m and $m + 1$. According to carried out experiments, the following conclusions can be made (see Figure 3):

1. in case of chains of classifiers, overfitting has asymptotic character and after some moment does not grow with T. According to VC-theory, the overfitting has a linear dependence on T. This shows that chains allow to slow down the overfitting when T is growing. This gives a possibility to use a larger number of classifiers in an ensemble.

2. Connectedness between classifiers allows having much more classifiers in the ensemble than what was predicted by VC theory.

3. The splits of classifiers by error rate allow obtaining lower overfitting than just for chains. In this case, the probability of overfitting does not achieve 1 even for very large T.

4. For relatively easy problems when the error rate of some classifier in the ensemble might be low, splitting classifiers by error rate allows reducing dramatically the probability of overfitting. With growing the complexity of a problem, the effect of splitting decreases.

5. For a more complex problem when one needs to have a large value of T both splitting and chains are required.

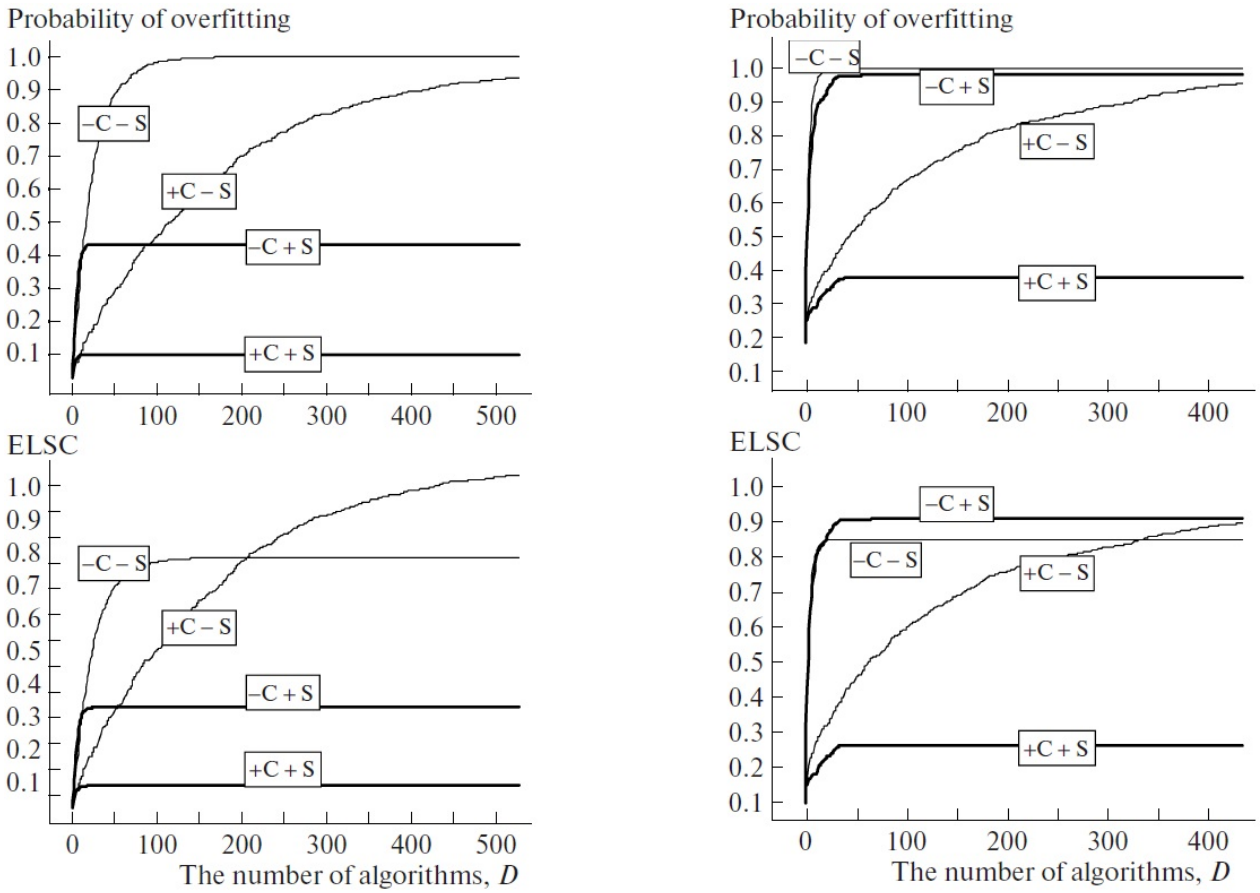


Figure 3: Probability of overfitting and overfitting rate as a function of the number of algorithms for easy (two left figures) and difficult (two right figures) cases

2.4 Classifier stacking in general

Stacking is a sort of meta-learning procedure where global learners should learn how to combine the results coming from individual learners. Thus, there are two levels of learners: the first level (individual learners) and the second level (meta-learner). This may also be called prediction-based learning [65]. From every individual learner, we have a vector of predictions of size L, where L is a number of classes. If the number of classifiers is equal to T, then we end up with the matrix of size $T \times L$. This is known as a decision profile [88]. So for every instance x to be classified, we will have a decision profile. Information containing in a decision profile is used for the classification of the instance x. The idea of stacking and prediction-based learning, in general, is not to use the information containing in the decision profile to make a final decision but to use it to compare with other decision profiles during meta-learning or use this information similarly to the vector of features for individual learners. This distinguishes classifier stacking from deterministic fusion schemes [88, 46]. It will be later shown that decision templates can be considered as points on the Grassmannian manifold. Thus, we need to use particular measures on this manifold to classify the instances.

The classical stacking approach [75] uses different training subsets for the individual learners and the meta-learner. The main reason for this is to avoid overfitting. If taking k-fold cross-validation that the initial training data X is stochastically split into $\{X_j\}_{j=1}^{k}$

parts and $X_j \cap X_{(-j)} = \emptyset$, where $X_{(-j)} = X \setminus X_j$ are non-overlapped training and test sets. Thus, an individual learner $h_t^{(-j)}$ is trained on the data $X_{(-j)}$ and then applied for every test instance x_i from the test set X_j. Hence, the transformed dataset generated during the entire cross-validation process will be

$$X^{'} = \{(z_{i1}, ..., z_{iT}, y_i)\}_{i=1}^{m}, \tag{3}$$

where z_{it} is the output of the learner $h_t^{(-j)}$. The meta-learner $h^{'}$ is then the function of the new feature vector $(z_1, ..., z_T)$ with some label y.

The training process in the case of a classifier stacking has two phases. In the first stage, we have to train base classifiers on some part of the entire dataset. In the second stage, a meta-learner should be trained on the unseen for the training part of the entire dataset during the first stage. This unseen part of the entire dataset is converted to the dataset of classifier predictions using base classifiers. Then this new dataset is used to train the meta-classifier. The main drawback, in this case, is that the meta classifier is trained only on some part of the initial dataset which can be critical for sufficiently small datasets.

2.5 Classifier stacking using CCE

In [89] one proposes an approach aiming to use deep random forests (DRF) instead of DNN. This is done by organizing trees via cascades. Feature vectors can be augmented using the multi-grained scanning procedure if there is a spatial and sequential relationship between them.

The principal motivation of deep forests exploration is to create a DLA other than DNN. For instance, DCNN uses learning by backpropagation when gradients are computed many times. If the architecture is very deep then either gradient vanishing or explosion may happen. Another drawback is that DNN may require many parameters to train and memorize. Also, there is no much theoretical development to analyze DNN. So, the idea is to create a deep DLA that does not suffer from all those drawbacks. Although DT are characterized by depth, DRF uses CRF which are ensembles of DT. The entire architecture can be interpreted as an ensemble of ensembles. Such an architecture would have fewer parameters to train, is very good for parallelization and has a good theoretical development.

First, one needs to receive an augmented feature vector. This is done using the so-called multi-grained procedure. In Figure 4 the multi-grained scanning for sequentially and spatially structured patterns is presented. For sequentially-based patterns, the 1D sliding window of size 100 is used. For spatially oriented features, the 2D sliding window of size 10×10 has been implemented. The initial feature vector for sequence-shaped data is 400-dimensional and for spatially-shaped data, the scanning field has a size 20×20. Moving the sliding window for sequentially shaped data, one obtains 301 instances with dimensions equal to 100. For spatially-shaped data, one has 121 instances shaped as 10×10 window. Then all these instances go to two RF: RF (A) and completely RF (B). In the case of a 3-class problem, the output of A or B RF for every single instance x is 3-dimensional. By having 301 instances from one initially shaped as 400-dimensional and assuming that every forest produces a 3-dimensional prediction vector, one obtains 903-dimensional output from either

A or B RF. By concatenating them we end up with an 1806-dimensional feature vector. In the case of spatially shaped initial data, one ends up with two 363-dimensional vectors produced by each forest. Concatenation of these two vectors gives an entire feature vector of 726 dimensions.

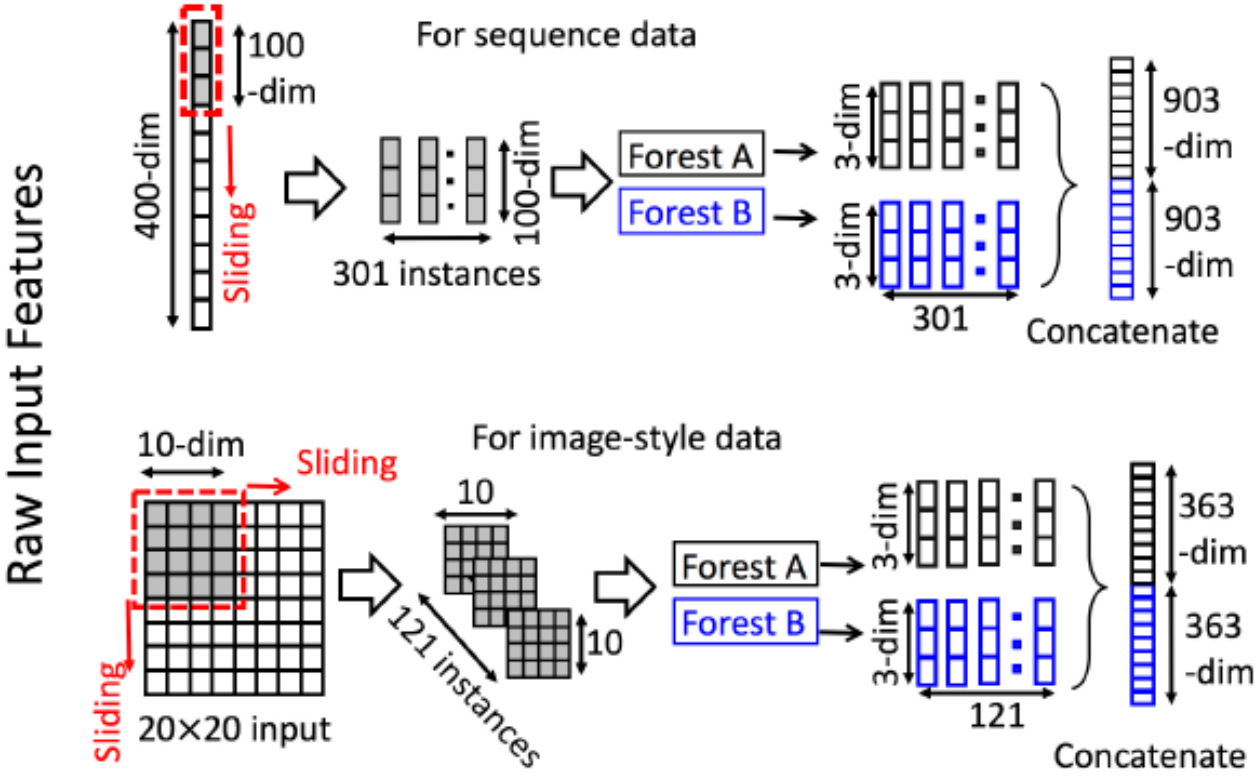


Figure 4: Multi-grained scanning

In Figure 5 the full diagram of the gcForest procedure is given. The multi-grained scanning procedure discussed before is on the left side of the figure. By having different sizes of the scanning window, we can obtain different levels of feature representation. Since there are four RF (two for random tree forest and two for completely RF) in the example, the total size of the concatenated prediction vector will be equal to $3 \times 4 = 12$. Then this vector is concatenated with the augmented feature vector, and the procedure of adding cascades will last until no improvements are obtained during a training phase. It is suggested to split the training set into two parts: one is used to train the cascades and the other one is for the improvement evaluation.

One can concatenate all features obtained from multiple grains into one vector and use it for further multi-cascaded processing. The procedure is illustrated in Figure 6.

One more version of DRF has been presented in [52]. In the proposed architecture, each layer consists of a forest filled with random and extra random trees (see Figure 7). Having T trees in a layer and L class problem, one gets output of each layer having the shape $T \times L$. In gcForest, each layer outputs a stack of prediction probabilities produced by four RF. In the proposed approach, predictions of individual trees are used. Also, the current approach requires fewer trees as in the case of [89] (2000 vs 4000) and uses entropy gain instead of the Gini coefficient to compute the splits on decision nodes.

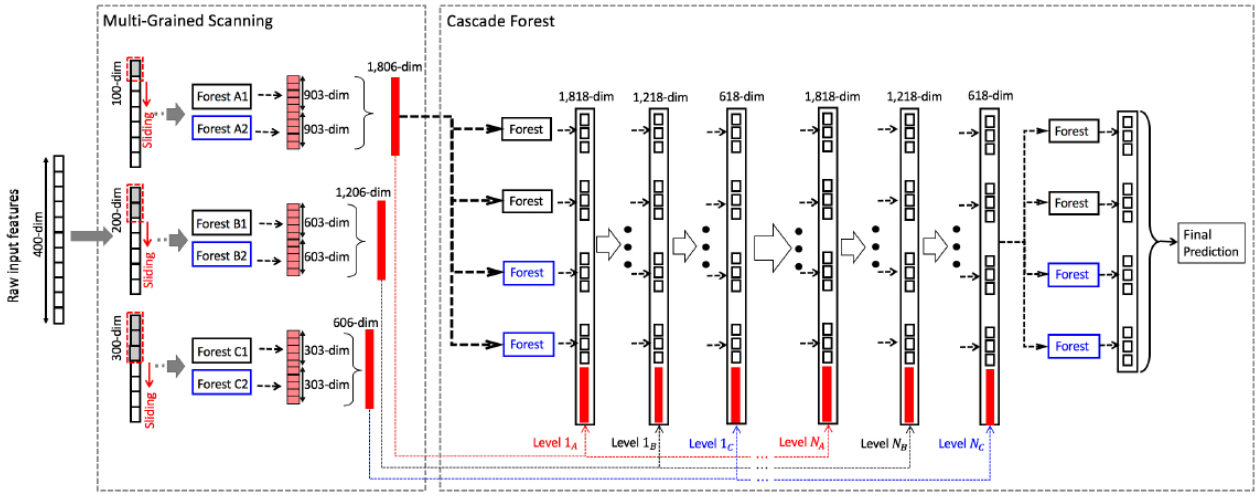


Figure 5: The overall diagram of gcForest algorithm

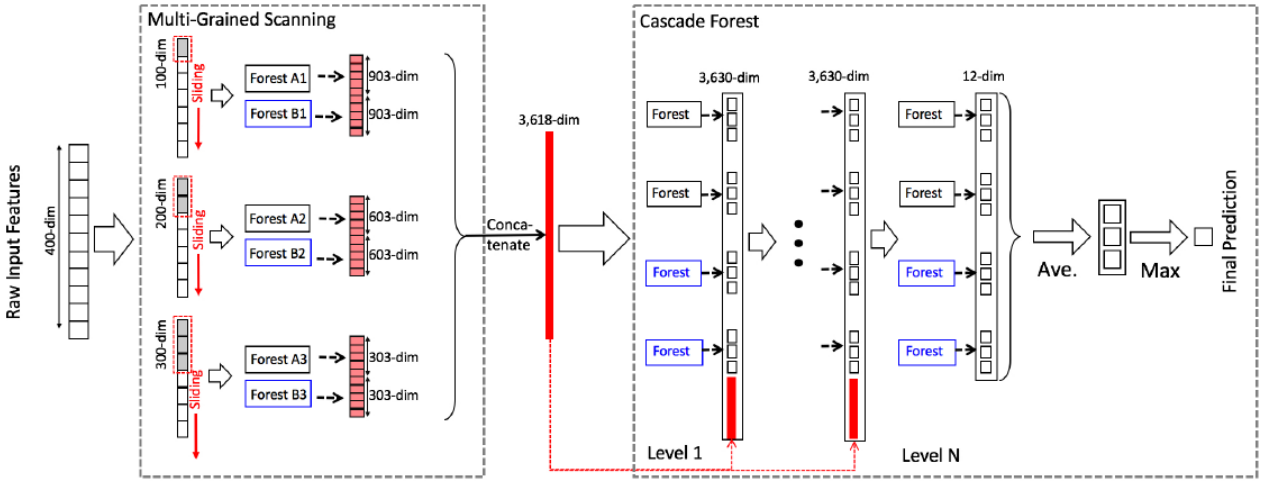


Figure 6: The version of $gcForest_{conc}$ which concatenates features from multiple grains

For the MNIST dataset, which is a dataset of binary images of handwritten digits, two data augmentation procedures have been proposed: single-pixel wiggle and multi-grained scanning. The experimental results on the MNIST dataset using the two aforementioned data augmentation procedures show similar performance to gcForests having much fewer trees in each processing layer.

2.6 Deep neural random forests (DNRF)

The main idea of the approach proposed in [44] is to use representation learning for DT for end-to-end training. The classical DT use the divide-and-conquer principle with the entropy and Gini coefficient as measures of the pureness of decision nodes [6]. So authors propose to reorganize classical DT in such a way that it will be possible to train them using backpropagation similarly to NN.

The tree to be learnt consists of two types of nodes: decision nodes where all splits should be accomplished and prediction nodes or termination nodes where probability distribution

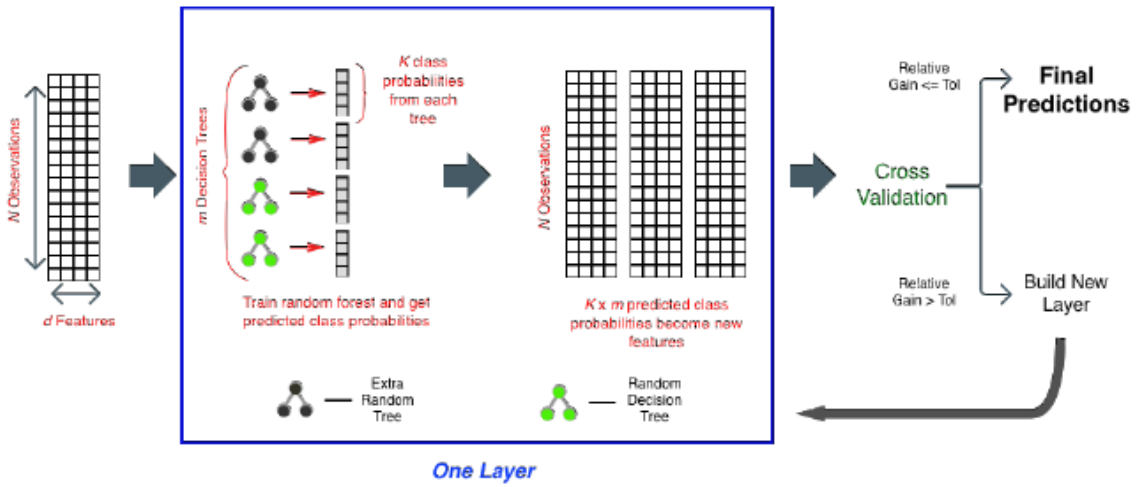


Figure 7: Forward thinking DRF architecture: a diagram of the algorithm

over labels is evaluated. In that case, we have a stochastic routing where the final prediction is done by averaging all the probabilities of taking a certain path for every pattern until it terminates on a leaf. DNRF in that case might be obtained by averaging all predictions made by DT. Other functions on predictions are also possible but the averaging is more robust to outliers.

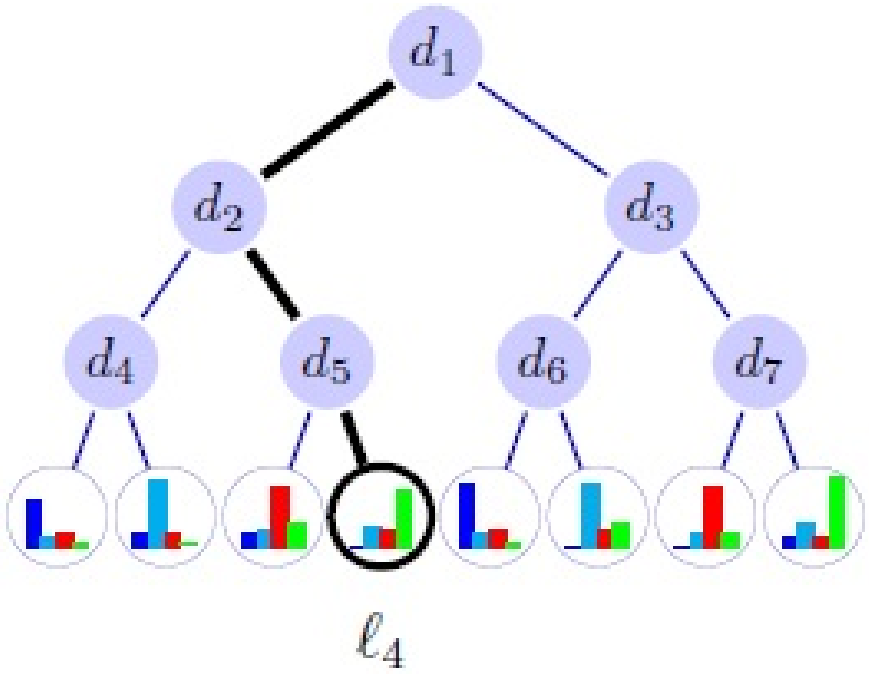


Figure 8: DT structure: test and prediction nodes

The learning procedure is made using the Stochastic Gradient Descent (SGD) method. SGD is applied to optimize the parameter that determines the splits on nodes. The complete algorithm of the learning splitting parameter is divided into two stages. First, we iterate over the prediction probability, and then we use mini-batches to optimize the splitting parameter

with SGD. We optimize over trees first and then average over them to have a forest prediction.

To compare the proposed architecture with the state-of-the-art ones, it was analyzed against the most advanced classical RF architectures using four datasets, including MNIST. For DNRF only linear parametrized function has been used. Results show that DNRFs are better on all of these datasets.

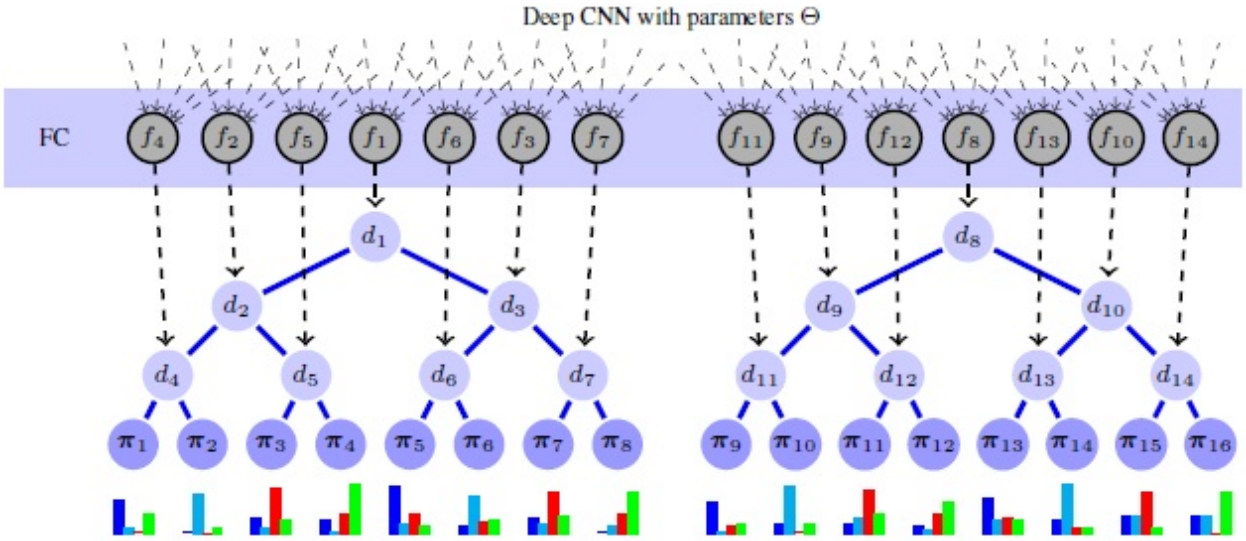


Figure 9: Learning decision tree test and prediction nodes

Some complementary analysis of the entropy on splitting nodes was carried out. It shows that with augmenting the number of training epochs the entropy in all leafs declines and becomes very picky meaning big progress in learning splits on decision nodes.

2.7 Nonlinear smooth manifolds

Smooth manifolds are topological differential manifolds and are de facto nonlinear spaces that have a structure locally similar to E-space. Smooth mappings between different manifolds (including E-space) and smooth manifolds themselves are objects of Lie algebra and Lie groups [50]. Differential manifolds assume infinitely differential functions (C^∞) that map the local neighbourhood of a point x on a manifold to E-space $\mathbb{R}$. Assume that $\mathcal{M}$ is a d-dimensional differential manifold with boundary $\partial\mathcal{M}$. Is said that a manifold is homeomorphic to a d-dimensional E-space within a boundary $\partial\mathcal{M}$. In this case, E-space is a tangent space denoted as $T_x\mathcal{M}$ around point $x \in \mathcal{M}$. The R-manifold is a manifold equipped with an R-metric g. The R-metric g on a manifold $\mathcal{M}$ $(\mathcal{M}, g)$ is an inner product $< \cdot, \cdot >_{T_x\mathcal{M}}$: $T_x\mathcal{M} \times T_x\mathcal{M} \to \mathbb{R}$. A smooth mapping of a neighbourhood $\partial\mathcal{M}$ of any point $x \in \mathcal{M}$ to Euclidean linear space $\mathbb{R}$ is a space of functions $f : \mathcal{M} \to \mathbb{R}$.

Mappings between manifolds as well as between manifolds and E-space can be presented via conformal factor [82]. Two R-metrics on a manifold $\mathcal{M}$ is said to be conformal when there exist a map $u : \mathcal{M} \to \mathbb{R}$ such that $\overline{g} = e^{2u(x,y)}g$, where $u(x, y)$ is a conformal factor depending on local coordinates. A smooth flow on a R-manifold can be expressed using a conformal factor. Such important characteristics of a manifold as curvature and Ricci flow can also be expressed in terms of conformal factor [82]. Conformal factor guarantees and

makes it possible a smooth mapping of one smooth manifold to another one when they are isomorphic. Because they are smooth they are diffeomorphic as well.

2.8 Riemannian manifolds

The R-manifold (M, g) equipped with a metric g is a differentiable manifold with slightly varying inner products of vectors of the tangent space. The R-metric is a family of inner products on all tangent spaces of the manifold. Among other properties of the manifold such as the angle between two curves, the R-metric allows computing the shortest distance between two points on the manifold, which is known as a geodesic's length.

There are many works devoted to using R-and-G-manifolds for the problem of CV [34, 68, 25]. Further, we are going to consider an R-manifold of symmetric positive definite (SPD) matrices Sym_+^d.

The space of $d \times d$ SPD matrices Sym_+^d is an open convex cone (see Figure 18)

$$Sym_+^d = \bigcap_{x \in R^d} \{\mathbf{P} \in Sym^d : \mathbf{x}^T \mathbf{P} \mathbf{x} > 0\} \tag{4}$$

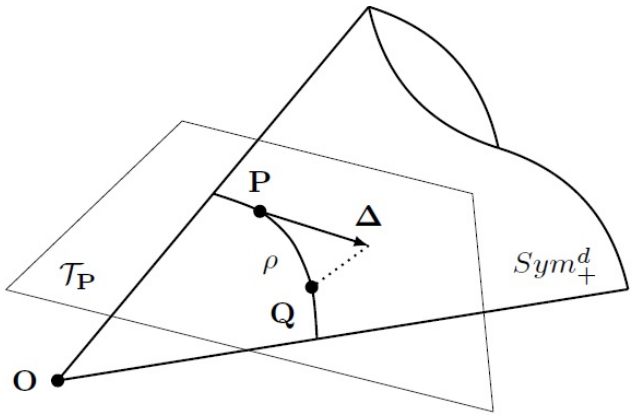


Figure 10: Geometric visualization of R-manifold of SPD matrices

Assume that we need to compute a distance ρ (geodesic's length) between two points $\mathbf{P}$ and $\mathbf{Q}$ on a manifold $\mathcal{M}$. We admit also that those points are sufficiently close, and our manifold is smooth, as it is one of the properties of the R-manifold. It is already known that points on the aforementioned manifold are SPD matrices. Because our manifold is a nonlinear space, we can not apply any linear norms that are usually applied in the case of linear spaces to compute a distance between those two matrices. If a point $\mathbf{Q}$ is in some neighbourhood ϵ of a point, $\mathbf{P}$ then we can compute distance ρ approximately. We also remind that our manifold is not given in a parametric form, so all distances can be computed only locally and approximately. Approximate computation of distances in a neighbourhood ϵ requires a definite procedure. First, we build a tangent space $\tau_\mathbf{P}$ at the point $\mathbf{P}$. Then we project our point $\mathbf{Q}$ into that space and find the projection vector Δ. Finally, using this vector and a logarithmic map, we find a distance in E-space which approximates the distance ρ. We can map a neighbourhood at some point on a manifold to E-space and vice versa. In case if we want to project some neighbourhood of E-space into a manifold, we need to use exponential

mapping. Details about exponential and logarithmic maps as well as two different metrics usually applied on Riemannian manifolds are provided in A.2.

2.9 Grassmann manifolds

The G-manifold $\mathcal{G}_D^m$ is the set of all D dimensional linear subspaces of the $\mathbb{R}^m$ and is $D(m-D)$ dimensional compact R-manifold [31]. So every point on $\mathcal{G}_D^m$ is $span(\mathbf{Y})$, where $\mathbf{Y}$ is an orthonormal basis matrix with D columns and m rows such that $\mathbf{Y}\mathbf{Y}^T = \mathbf{I}$. The size of the identity matrix $\mathbf{I}$ is $D \times D$. To learn on G-manifolds we need to compute a distance between two points lying on it. A distance on a G-manifold $\mathcal{G}$ is called a geodesic's length.

Some profound analysis of G-manifolds as well as some characteristics and properties of those manifolds together with particular cases of submanifolds can be found in [4]. In the aforementioned paper, topology of G-manifolds as well as geometrical properties of those manifolds are considered. Principal distance measures on a G-manifold are given in A.3.

2.10 Video analysis problems

There are many problems in CV related to video analysis in general. One can mention video classification and video content manipulation detection. While the first problem requires understanding what is happening on the scene of the video, the second one requires detection if the information presented on the video is true or fake. Among different applications of video classification problems, one can mention human behaviour recognition, video content classification, movie classification by genre and others. A special genre of recording motion on videos is Motion Capture. It can be used for different video analysis problems in different areas such as military, entertainment, sports, medical applications, and validation of CV and robotics. It is also often used in filmmaking and video games, where it is known as motion tracking.

Because our research is principally focused on the classification module of the video classification pipeline, we carried out our experiments on the Gesture Phase Segmentation dataset, which belongs to a video classification problem and where features of gestures have been already obtained. Nevertheless, we provide some overview of video classification architectures to familiarize a reader with this field of research.

2.10.1 Video classification methods

The broad applications of DLAs to video classification problems became possible due to the big enough video datasets. Algorithms applied beforehand used handcrafted features such as HoG, HoF, MBH and others. Then features are fed into SVM to perform the classification task. However, automatically selected robust features having different levels of abstraction should lead to better performance and feature engineering problems will be less costly in terms of time consumption.

Supervised DL video classification can be divided into image-based video classification, end-to-end CNN architectures and long-term temporal dynamics modelling. We will consider all three groups of approaches. In our research, we focus on the first and the third architectures.

2.10.1.1 Image-based video classification

These approaches are motivated by the great success of CNN feature engineering for image classification problems. There are some very successful DLAs [45, 63, 27]. The feature representation is derived by feed-forward pass until it reaches a fully connected layer with the aforementioned pre-trained deep models. Then for each frame features are averaged and put as inputs into some standard classifier, for instance, SVM. In [83] authors studied the possibility of using multiple SVM kernels for features that come from different layers of abstraction. They showed that descent performance can be obtained. In [60] it was proposed to use a vector of locally aggregated descriptors (VLAD) for video level representation. This leads to better performance than using pool averaging. In [56] it was introduced Fisher vector encoding integrated with variational autoencoder (FV-VAE) to make quantization of local activations in convolutional layers. This allows learning visual representations, leading to better generalization.

2.10.1.2 Video classification using different fusion models of two spatial-based CNNs

There are lots of successful applications of DL mechanisms (CNNs) for pattern analysis s.t. image classification [45, 63, 27]. Here one focuses on a more complicated problem presented in [39]–real-world video classification challenge.

It is known that DL architectures and particularly CNNs are very powerful tools for image analysis and recognition. Every DLA with millions of parameters can be very accurate when a training set is of sufficiently large scale. All these encouraging factors give the insight to use these architectures for video classification. Then instead of using only spatial information as for a single image, we should take into account the complex temporal evolution.

To make the experiments, a sufficiently large dataset should be collected. So it was created Sports-1M dataset containing 1 million YouTube videos divided into 487 classes of sport. Now, this dataset is publicly available. During the modelling stage, two questions should be addressed: what is the best architecture for video recognition and how temporal information can help to improve the prediction.

Usually, CNNs require a sufficiently long period to train the model. In the case of video analysis, the situation is even worse. To address this problem, it has been proposed to use two streams: a context stream learning features from low-resolution frames and a fovea stream operating on the middle part of the high-resolution frames. This allows reducing the computational load from 2 to 4 times.

One more question left is to consider the possibility to use some part of the model corresponding to low-level features trained on Sport-1M for another video dataset, such as UCF-101. This type of learning is known as transfer learning. Further, experiments show the improvement in accuracy from 41.3% for the entire network trained on UCF-101 up to 65.4% when transfer learning using Sport-1M has been applied. The main obtained contributions are all about the questions stated above.

Conventional approaches to video classification deal with visual feature detection locally or globally. Then vector quantization using k-means was applied to obtain the so-called "bag of words". Next SVM was applied for training. CNNs are networks that use convolutional filters to obtain features from images. They are characterized by internal connectivity and

millions of parameters to be trained. This allows obtaining a functionality similar to the human brain. There are several successful applications of CNNs on image recognition, object detection, scene labelling, image segmentation and inpainting, and many others. To this end, there are lots of publicly available datasets. There are not, so many works dedicated to video classification based on CNNs. One of the reasons is that there are no large enough publicly available video datasets. Regardless of this fact, some advances were made for the video recognition problem. One of the approaches uses the convolution both in time and space. Other approaches use unsupervised learning (Convolutional Gated Restricted Boltzmann Machines (CGRBM)) and Independent Subspace Analysis (ISA).

To have a possibility to use spatio-temporal relations, we need to build an architecture that would have connectivity both in time and space. This means that one fixed architecture cannot handle this. To model the connectivity in a temporal domain the three schemes have been proposed (early fusion, late fusion, and slow fusion).

Time information fusion can be done in different ways. It can be done early when the first convolutional layer is extended to handle time dependencies. In opposite, late fusion uses two networks shifted in time and then fuse their outputs. In single-frame fusion, we consider each frame as a separate image in a static way (time is not taken into account). It is considered a particular architecture of the ImageNet challenge winning model. In early fusion, convolutional filters operate on 4D tensors, including 3 spatial dimensions and the time window T. A typical value of the time window is equal to 10. So, in this case, fusion is done immediately across the entire frame. Late fusion is organized as two single-frame networks, and they are shifted in time. First, a fully connected layer makes fusion from two streams. Slow fusion or progressive fusion is a balance between early and late fusion. In the first layer time extent is used ($T = 4$) which produces 4 outputs for the size of a clip equal to 10. Then, second and third layers iterate over time with a time frame window $T = 2$. The third layer "sees" information from all 10 frames (see Figure 11).

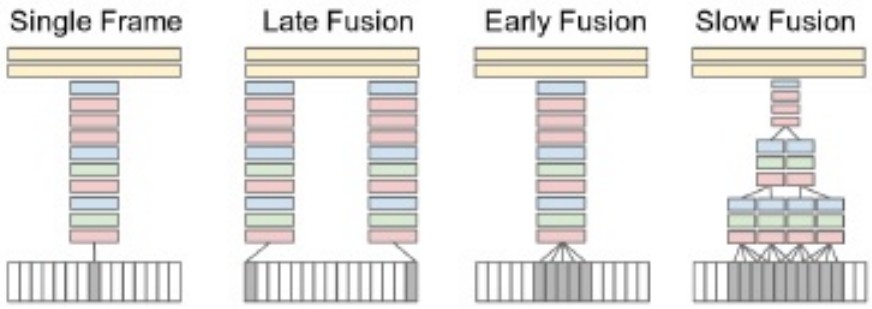


Figure 11: Explored approaches for fusing information over temporal dimension through the network. Red, green and blue boxes indicate convolutional, normalization and pooling layers respectively. In the Slow Fusion model, the depicted columns share parameters.

Large-scale datasets require a very long time to be trained. Even with the fastest GPUs, it can take up to several weeks. One way is to reduce the number of layers, but this leads to underfitting and resulting in relatively poor performance. To keep the same architecture, it is proposed a network with two streams. One stream use entire frames with two times the lower resolution (context stream) and the second stream uses only the central part with the same resolution but two times smaller (fovea stream). We have to take out the last pooling layer because the initial resolution was reduced two times (see Figure 12).

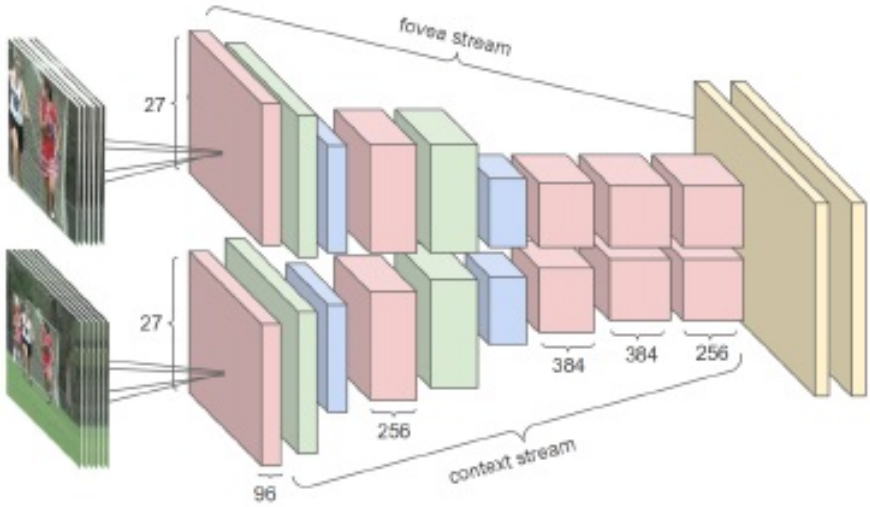


Figure 12: Multiresolution CNN architecture. Input frames are fed into two separate streams of processing: a context stream that models low-resolution image and a fovea stream that processes high-resolution center crop. Both streams consist of alternating convolution (red), normalization (green) and pooling (blue) layers. Both streams converge to two fully connected layers (yellow).

The learning process uses Downpour SGD with multiple replicas. The model uses mini-batches of size 32 and momentum 0.9 with a learning rate of 10^{-3}. Data augmentation is done by using image cropping and then random downsampling. Finally, horizontal random flipping with 0.5 probability has been applied. The value of 117 is subtracted from all the pixel values, which is approximately the mean value of all images.

The carried out study shows the superiority of CNN over the existing feature-based approaches. Slow fusion architecture is the most accurate one. Using mixed-texture resolution speeds up the learning process. Learned video features are generic and can be used for other datasets.

2.10.1.3 Spatiotemporal CNNs for the action classification

In [62] it is proposed to use two-stream CNNs to classify spatial and temporal components of video and fuse them at the softmax layer (see Figure 13). Then all scores obtained from both CNNs are averaged or used as features to train the (SVM). To classify objects in the scene, one uses spatial information from the video. To learn this information some CNN architectures can be used or even some pre-trained models. In [62] CNN-M-2048 has been used to classify the spatial component of a video stream. For spatial classification as input data, one uses a sequence of static frames (images). For temporal or motion classification the similar structure of CNN is used, however, the input information is different.

Temporal data is obtained as follows. First, optical flow is applied to obtain motion vectors at every pixel (see Figure 14). This part of the pipeline for motion classification is known as handcrafted motion features. If one uses a stack of frames of size L to classify the motion information and the field of size $w \times h$ of the initial frame then the input tensor will be of size $w \times h \times 2L$ (both horizontal and vertical components of a motion vector have been used). If we use a track for motion classification purposes then instead of fields of size $w \times h$ one will use only points in a track each of which is characterized by horizontal and vertical

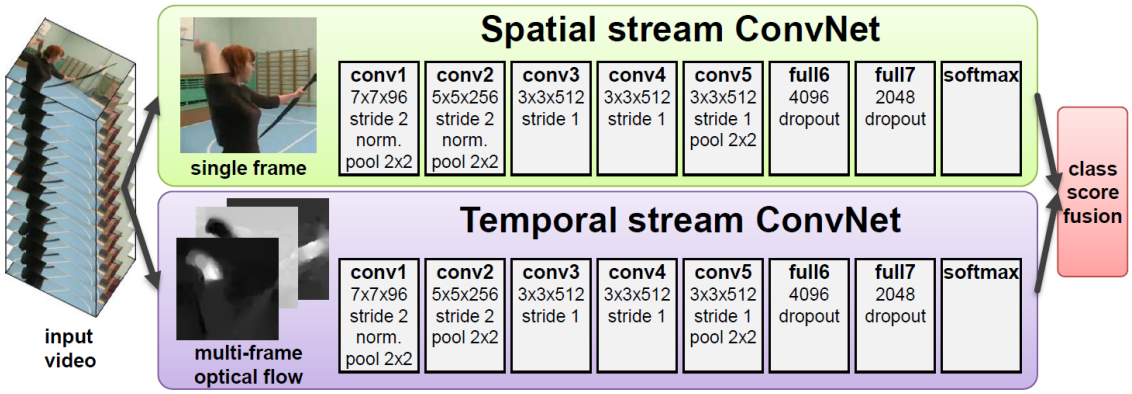


Figure 13: Two-stream architecture for video classification

motion components. Both versions of ConvNet input derivation are presented in Figure 15.

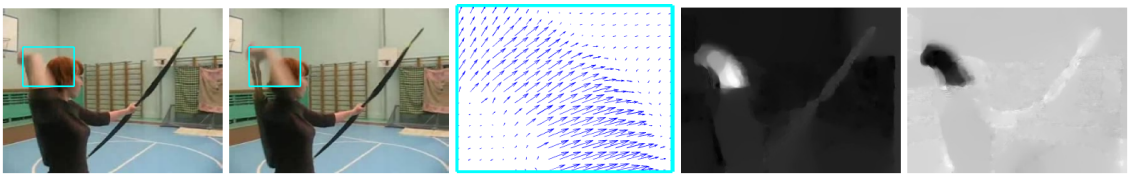


Figure 14: Optical flow computing. From the left to the right: two consecutive frames, optical flow vectors, horizontal and vertical components of the optical flow rescaled to $[0, 255]$ gray levels.

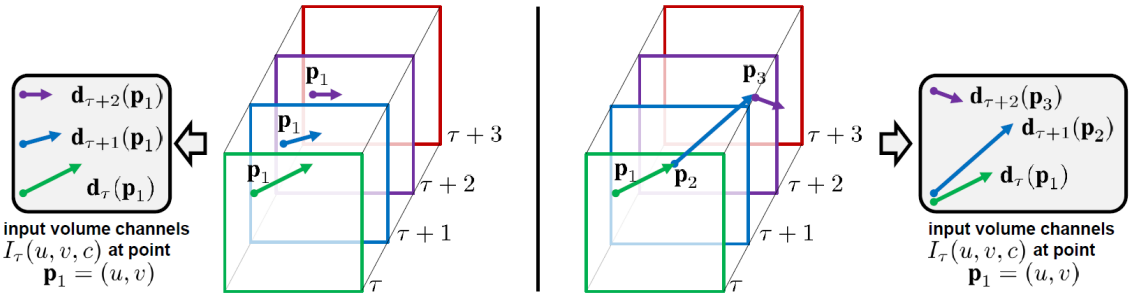


Figure 15: Two versions of ConvNet input derivation. Left: optical flow stacking, right: trajectory stacking.

One of the main problems in video classification is camera motion compensation. Assuming that the motions in arbitrary directions have no preference or are uniformly distributed, one can use a very simple approach by subtracting the average value from all motion vectors. However, in general, this problem is sufficiently difficult. Another problem can be the lack of training data and its storage. There is one very big labelled dataset Sport-1M for video classification problems but it needs a very big space for storing which is not always available. Also, this dataset has many similar videos and categories which makes it an additional problem to use it. Training using this dataset also needs exceptionally powerful computers with very powerful GPUs. Using much smaller datasets such as UCF-101 and HMDB-51 needs to use pretraining for spatial classification (using for example ImageNet dataset). For temporal classification, multi-task learning is applied.

During the experiments, it was determined that the optimal size of the frame stack is equal to 10. The best training strategies are those which use pretraining with dropout as

a regularizer. Multi-task learning gives the best results. Optical flow stacking is slightly better than trajectory stacking. Using SVM to learn data coming from the softmax layer is advantageous in comparison to averaging.

2.10.1.4 Different fusion schemes in two-stream CNN video classification

One more architecture proposed in [21]. This is a two-stream CNN with fusion in different layers. Principally fusion is considered in two types of layers: convolutional layers and prediction layers. According to the proposed approach fusion can be done in different layers. To make a fusion of spatial and temporal information in convolutional layers one needs to integrate two feature maps. There are the following schemes for feature maps fusion: sum fusion, max fusion, concatenation fusion, convolutional fusion, and bilinear fusion. Sum and max fusion need the same number of parameters, convolutional fusion requires slightly more parameters and concatenation fusion requires approximately two times more parameters. Bilinear fusion requires much more parameters, that is why it is not a very popular fusion scheme.

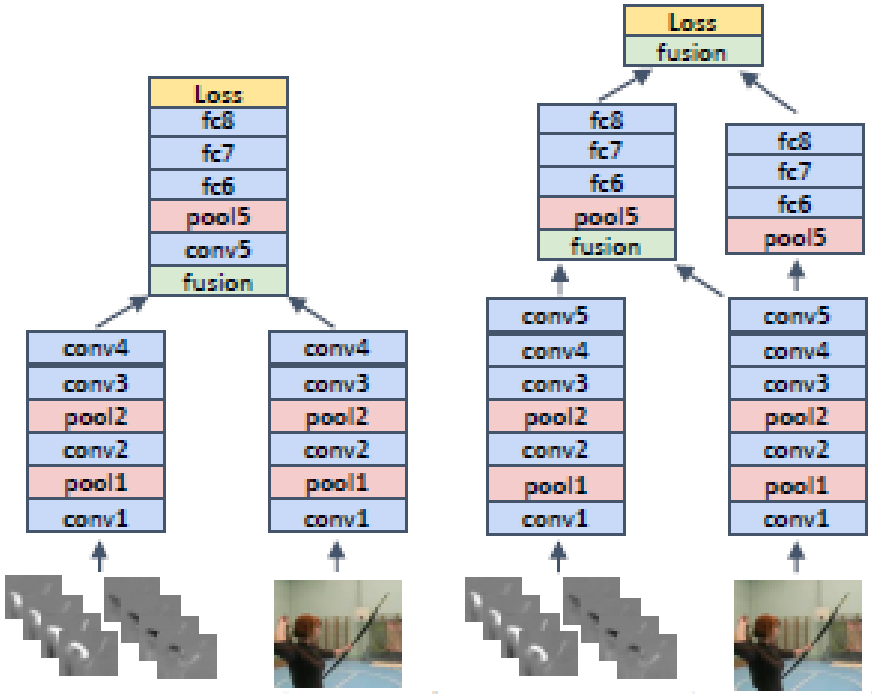


Figure 16: Two architectures showing where spatio-temporal fusion can be applied

Figure 16 demonstrates two fusion schemes using two-stream CNNs. The left scheme shows spatio-temporal fusion after the fourth convolutional layer followed by the fifth convolutional layer dealt with spatio-temporal information. The scheme on the right fuses the spatial and the temporal information after the fifth convolutional layer and preserving the tower for spatial information processing. Then one more fusion is accomplished in the prediction layer. In Figure 17 several schemes of temporal information fusion have been presented. The simplest scheme uses the 2D pooling of temporal information. This pooling is applied to separate feature maps corresponding to a particular frame. 3D pooling means that we pool

from the stack of feature maps organized in a "cuboid". The last scheme applies 3D convolution to a stack of feature maps and then use 3D pooling to the results of 3D convolution.

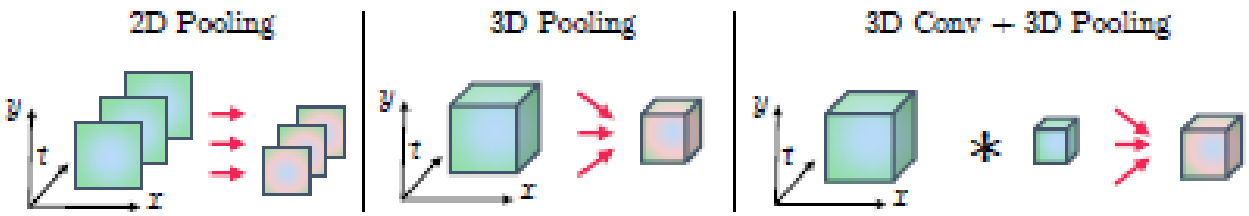


Figure 17: Several schemes of temporal fusion

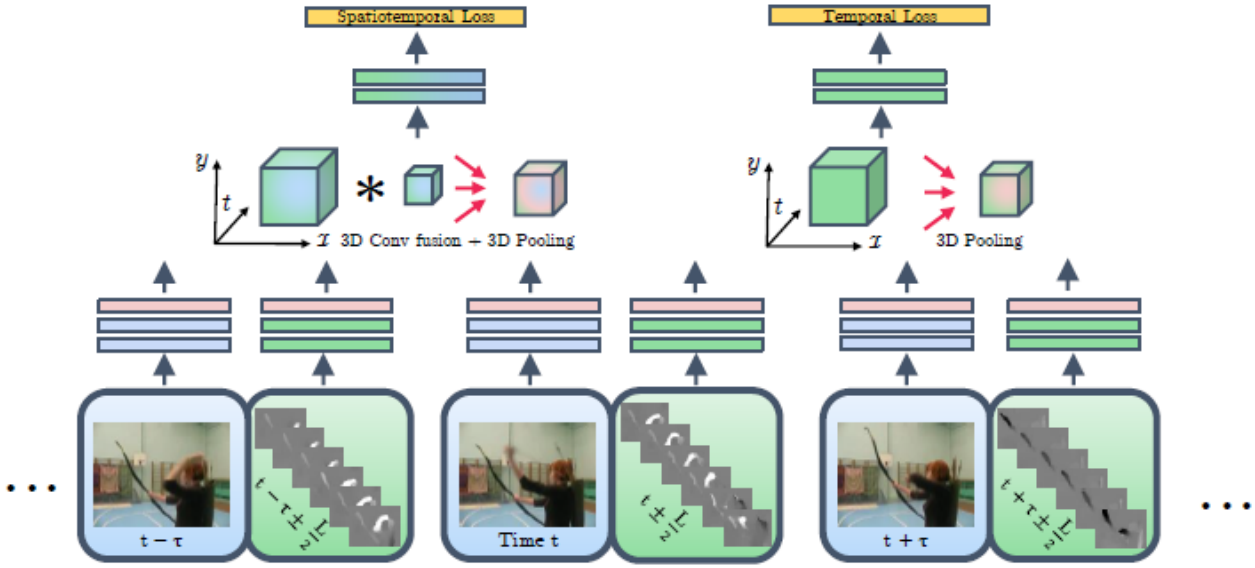


Figure 18: Spatiotemporal fusion in ConvNets. Temporal scale is $t \pm \frac{L}{2}$ applied to T temporal chunks that are τ frames apart.

Experimental results show that convolutional fusion with 3D pooling is the best. Keeping one tower after fusion in Relu5 gives only slightly lower accuracy than keeping two towers (spatio-temporal and spatial ones) having two times fewer parameters. Softmax averaging gives a very competitive accuracy. Having deeper architectures leads to significant improvement in performance. Keeping both temporal and spatial information processing towers deep using VGG-16 architecture (13 convolutional layers) gives the highest performance. Fusion handcrafted features with those obtained by spatio-temporal information fusion give the highest overall accuracy 93.5% and is the state-of-the-art video classification accuracy. This is done by combining scores from the SVM using FV-encoded IDT features (HoG,HoF,MBH) and those obtained before the softmax layer.

2.10.1.5 Modelling long-term temporal dynamics

There are situations when the motion is complicated and can be divided into several parts. In this particular situation, it is hard to use CNN's only. To catch long-term temporal dynamics LSTM is a very good model to apply. In [17] it was proposed to train two-layer LSTM by

using features from two-stream CNN (see Figure19). In [77] it has been proposed to fuse outputs of CNN and LSTM to model spatio-temporal clues. The authors conclude that both models are highly complementary. In [80] authors trained 5-layer LSTM. It was shown that the deep LSTM model performs similarly to single image CNN on the Sports-1M dataset. This may occur because this dataset is quite noisy. In [70] authors introduced differential gating LSTM to find the most important information that characterizes the motion in videos. In [76] it was proposed to use CNN for spectrogram analysis of soundtracks that accompany video streams. This allows complementing visual clues learned by CNN and LSTM.

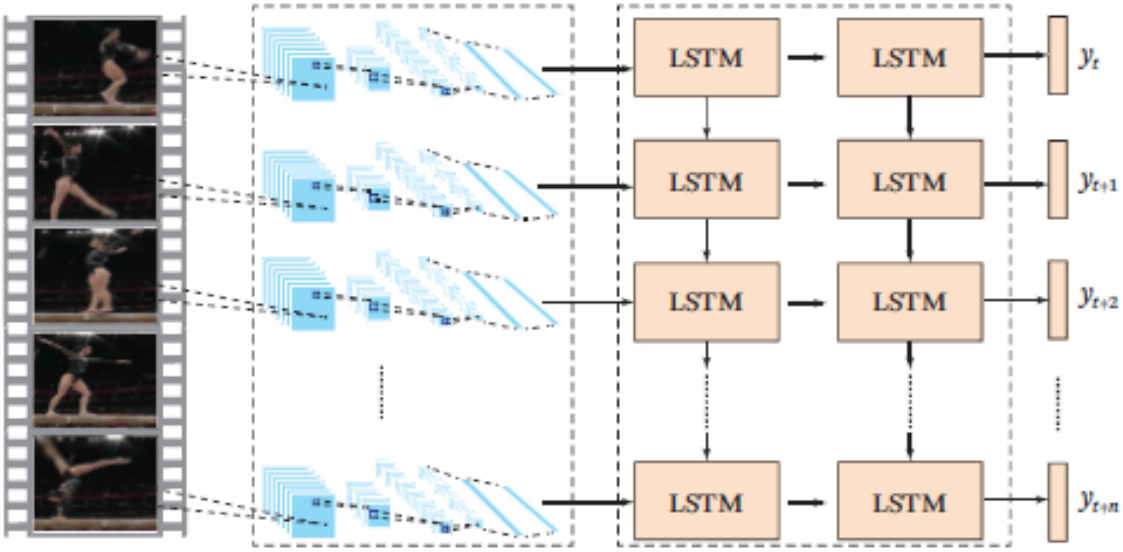


Figure 19: LSTM that is trained on features obtained from a two-stream CNN

2.10.2 Video face forgery detection

2.10.2.1 General aspects

Over the last decades, there are more and more digital content uploaded on the Internet. Many uploaded videos and images are manipulated or fake. That is why image/video forgery detection is of crucial importance. Among forgery detection problems, face forgery detection on videos is of particular interest. For example, fake news can have a huge impact on stock markets and politics. Known forensic techniques are not very suitable for these kinds of problems because they often require images of sufficiently good quality which often is not satisfied in the case of videos that are heavily compressed and hence degraded. During the last few years, DL architectures were successfully applied to detect facial forgeries such as double JPEG compression, image slicing, image general falsification, detect whether computer graphics have been applied. There are also manipulations related to physics-based artifacts that occur during image formation. For videos, one can mention dropped or duplicated frames, varying interpolation types, copy-move manipulations, or chroma-key compositions. Generative adversarial networks (GANs) are widely applied for the high-quality synthesis of faces and different face manipulations.

There are two and four types of forgeries considered in [1] and [58] correspondingly. In [1] only two datasets with $Face2Face$ and $DeepFake$ manipulations were used to carry out the experiments. In [58] except of those forgeries $FaceSwap$ and $NeuralTextures$ have been added. $DeepFake$ and $NeuralTextures$ are created using DLAs. $Face2Face$ and $FaceSwap$ are conventional real-time techniques where computer graphics and CV approaches have been used. All four methods require source and target actor video pairs as input. The initial face is often called as "target" and the face using to swap this face as "source" or "swapped" identity [16].

A DL tool known as $DeepFake$ has been designed to capture faces on videos and reenact them. Deepfake is a technique that allows swapping one face into another. Deep fake has two implementations $FaceApp$ and $FaceSwap$. The idea behind $DeepFake$ consists in training two parallel autoencoders. Thus, one should train two autoencoders (both encoder and decoder) for images A and B independently. Then we share the weights of both encoders and the image A encoded by the shared encoder is then reconstructed by a decoder trained on the image B. After applying such a technique, the general information about illumination, position and expression is preserved, but the morphological details are swapped. The decoder reconstitutes the constant characteristic shape and facial details. It should be noticed that the image details are not well-reconstructed and many artifacts can appear. Among them, we can mention large blurry areas in case of facial occlusions or doubled facial contours. Some networks can deal sufficiently well with these kinds of issues, but overall, autoencoders poorly reconstruct image details. Having a larger compression layer leads to transferring too much morphology from the "target" image.

The most advanced technique that transfers expression from one face to another is known as $Face2Face$ [67]. It allows the production of photorealistic markerless forged images in real-time. To perform such a facial reenactment, one first needs to train facial model reconstruction on a prerecorded video of the "target" face. Then the algorithm tracks facial expressions on both "target" and "swapping" images, followed by overlaying the target face with a morphed facial blendshape to transfer the facial expression from the source image to a target one.

$NeuralTextures$-based rendering approach has been introduced in [66]. Neural texture and rendering networks are learnt using original video data. Training is performed by combining photometric and adversarial losses.

There are some architectures for face forgery detection mentioned in [58]. Notwithstanding, we are going to consider two of them: MesoNet and FaceForensics++ based on Xception architecture pre-trained on ImageNet dataset. The reason for that is that they are performing the best for the compressed high quality (HQ) and low-quality (LQ) videos [58].

2.10.2.2 Base models

MesoNet. In [1] two architectures that work on the mesoscopic level of analysis have been proposed. Both types of face forgeries can be handled by only one network. Going to the microscopic level is rather impossible due to the degradation of noise and the fact that the human eye cannot distinguish forged details on that level. To this end, one proposes to use NNs with a small number of layers. These NN are based on well-known architectures which perform quite well for image classification [45]. So the first architecture named Meso-4

consists of four convolutional and pooling layers followed by dense fully-connected NN with softmax classifier. For nonlinear transformation, the rectified linear unit ReLU function has been applied. Batch Normalization has been applied to avoid the effect of gradient vanishing and dropout has been used for regularization.

One more architecture has been proposed, known as MesoInception-4. In this architecture, two first layers were substituted by a variant of inception module introduced in [63]. The idea consists in stacking outputs from several convolutional layers with different kernels. This allows augmenting the function space to be learnt. Instead of using 5×5 convolutional kernels, it was proposed to use 3×3 dialed filters. Replacing more than two convolutional layers by inception modules does not lead to improvement of classification results.

To create a dataset, 175 forged videos have been collected from different platforms. Those videos were compressed by H.264 with different compression rates. The resolution of videos is 854×480 pixels. Detection of faces has been performed using Viola-Jones face detector [72]. Faces were aligned using a face landmark detector [42]. Around 50 images of the face were extracted per scene. The same distribution of images with the higher and lower resolution has been preserved for training and testing. The whole dataset has been doubled with real-world face images. Misalignment and wrong face detection has been removed.

The classification results are obtained for the $Deepfake$ dataset using both architectures. The accuracy is around 90% for both of them. For the $FaceForensic$ dataset, one obtains 96 % of accuracy in case of 0 compression rate and 93.5 % in case of compression rate equal to 23. The classification accuracy is lower in comparison with the original paper where authors use Xception model [58].

To boost the classification accuracy, image aggregation has been applied. This is implemented as averaging prediction scores obtained for a sequence of face images over the scene. This allows making predictions robust in case of noise and facial occlusions. Classification accuracy using image aggregation in the case of $Face2Face$ is 0.953 for both architectures. For the $Deepfake$ dataset, corresponding accuracies are 0.969 and 0.984 for Meso-4 and MesoInception-4 correspondingly. There is one more way of doing image aggregation which consists in using only intra-frames i.e. images that were not interpolated. Experiments on the $Deepfake$ dataset show a slight decrease of classification accuracy by 0.037 and 0.025 in the case of Meso-4 and MesoInception-4 correspondingly.

Classification abilities of the aforementioned networks have been analyzed using feature maps obtained as outputs of convolutional layers. In particular, a mean value of such outputs from the last convolutional layer shows that for real images regions of eyes fire, and it can be seen by plotting those feature maps. On the other hand, for the forged images facial regions have low values on those feature maps. Positively activated neurons are typical for the regions of eyes, mouths and noses while negatively activated ones display strong details of the background regions and not for the smooth facial regions which are typical for the forged images. Thus, facial details are key aspects in distinguishing forged and real faces. Notwithstanding, more sophisticated and advanced techniques for facial tempering will require more profound and detailed analysis.

FaceForensics++ The objective of the research proposed in [58] is to create a benchmark for testing different kinds of algorithms, as well as to establish the dependency between the quality of video and the ability of the algorithm to distinguish between real and forged faces on these videos. For these purposes, a large dataset of fake and real videos has been

created where all four types of forgeries are presented. There are versions of this dataset with different levels of video compression. Raw videos are not compressed, HQ videos are compressed with a factor equal to 23 and LQ videos are compressed with a factor of 40. Also, some dependencies between the size of the dataset and the classification accuracy of two types of videos have been shown in their work. To make their experiments more representative, they've also involved humans in the classification process. That way, they wanted to show if Artificial Neural Networks (ANN) can outperform humans in their ability to correctly classify those videos. The architecture proposed in [58] is based on ANN architecture known as XcepetionNet. Unlike the naive approach of using XceptionNet [11] which produces relatively low classification accuracy and has been applied to the entire video, authors applied it to a tracked sequence of faces with some contextual information around them. This allowed making them robust. The comparison with other architectures including Mesonet shows that this architecture outperforms all others including humans with a significant margin. This happens especially for HQ and LQ videos. Humans show very low detection ability for LQ videos, which is not surprising. One more comparison based on the ability to detect a particular forgery has been carried out. According to it, the worst detection is observed for real images. Then in ascending order are the following forgeries: $NeuralTextures$, $Face2face$, $FaceSwaps$ and $DeepFakes$. XceptionNet outperforms all other architectures by a large margin for each of the aforementioned subsets.

In CV community transfer learning with fine-tuning is a very popular technique. That is why creating such datasets of a large scale and pre-training models on them are of very importance.

2.10.2.3 Face forgery detection challenges

We are going to consider two challenges aimed to engage as many teams and developers as possible. First one is the DeeperForensics Challenge (DFC) 2020 [35] and Deep Fake Detection Challenge (DFDC) [15] organized by Kaggle.

DeeperForensics Challenge 2020. The DFC 2020 offers a DF-1.0 dataset [36] which contains $60,000$ videos having 17.6 a million of frames in total. This dataset is characterized by three properties: good quality, high diversity and large scale. Full HD videos from 100 paid actors having 26 skin tones. The dataset has been augmented with 53 facial expressions and using seven real-world perturbations.

During a development phase, for evaluation purposes, $test - dev$ a hidden test set containing $1,000$ videos has been created. This dataset reflects the main properties of the full hidden test set. This dataset is used to determine a public leaderboard. The final test set contains 3000 videos also including $test - dev$ videos.

Because face forgery detection is a binary classification problem, binary cross-entropy loss as an evaluation metric should be applied:

$$BCE_{loss} = -\frac{1}{N}\sum_{i=1}^{N}\Big[y_i \log(p(y_i)) + (1-y_i)/log(1-p(y_i))\Big], \tag{5}$$

where N is the number of videos of the hidden test set, y_i is a ground truth label (in our

case 0 or 1), and $p(y_i)$ is a predicted probability that a video is fake.

After evaluation, leader board is determined. We are going to consider approaches provided by the first three leaders.

First-place approach Detection of faces has been performed using Multi-task Cascaded Convolutional Networks MTCNN face detector [86]. Then face region was expanded by a factor of 1.2. During the classification process, they applied an ensemble of three EfficientNet models: EfficinetNet-B0, EfficinetNet-B1 and EfficinetNet-B2. The general prediction that the face is fake is accomplished based on individual predictions for every detected face. In addition to the provided dataset they have used some other public datasets: UADFV [79], Deep Fake Detection [19], FaceForensics++ [58], Celeb-DF [51], and DFDC Preview [16]. To augment the initial dataset, they applied different kinds of image distortions. All images were resized to 224×224. Their model is the fastest one due to a very fast MTCNN face detector and an ensemble of relatively simple models having together fewer parameters than many complex DLAs. In particular, in [71] it has been mentioned that DNN can be presented as an ensemble of shallow NNs. In this case, we may also expect that ensemble of shallow architectures is less complex than the initial deep NN.

Second-place approach The authors propose to use an attention mechanism to fuse spatial and temporal information. They use RetinaFace [14] to detect faces. For both spatial and temporal domains they use EfficinetNet-B5. In the case of image-based EfficinetNet-B5, they feed it frame by frame and average the predictions obtained for every frame, while for a video-based model they use the set of frames and apply the attention model to fuse the temporal information. Then the average predictions were obtained for both models. In addition to the existing, three external datasets were used by them: FaceForensics++ [58], Celeb-DF [51], and Diverse Fake Face Dataset [12]. Also, different data augmentation techniques have been applied.

Third-place approach. As the key point to detect fake images, authors used poor temporal consistency. For this purpose, they have used a 3D convolutional neural network (3DCNN). Different data augmentation techniques were applied as in previous cases. All detected face images using the MTCNN face detector were then used to form a video clip which was resized to $64 \times 112 \times 112$ or $64 \times 224 \times 224$ and is the input to the 3DCNN. They examine different 3DCNN architectures which were pre-trained on action recognition datasets, i.e. kinetics [41]. In addition, they have used external face manipulation datasets such as the DFDC dataset [15], Deep Fake Detection [19], and Face-Forensics++ [58].

Deep Fake Detection Challenge The DFDC organized by Kaggle aims to engage specialists that work in ML and CV to develop algorithms and pipelines that would be able to detect tampered face images on videos with a sufficiently high level of reliability. To this end, the DFDC dataset containing $100K$ videos involving $3,426$ paid actors has been created. This dataset is of a very large scale and is the largest created so far, where 8 different types of facial manipulations have been applied.

All previous datasets can be divided into two categories: the first and the second generation. Generally, the next generation has more videos by an order of magnitude than the previous one. Notwithstanding both generations are characterized by a relatively small number of manipulated videos and are very unbalanced. The datasets from the first generation usually contain less than 1000 videos and 1 a million of frames while the second generation contains between 1 and 10 thousands of videos (1 and 10 a million of frames). Videos of the

second generation are characterized by overall better quality.

The third generation of datasets includes the DFDC dataset and DF-1.0 known also as DeepFake datasets. Although these two datasets belong to the same generation, there are significant differences between them. First, the DFDC dataset is much larger. Also, the DFDC dataset was created using paid actors, while DF-1.0 was created using videos from the Internet. The latter one contains only $1,000$ unique fake videos, while the DFDC dataset has $100,000$ unique facial swaps. DFDC dataset contains more diverse videos including indoor and outdoor scenes where lighting conditions are very different which may often lead to very low contrast in video frames. A public test set contains $4,000$ videos and is used to determine a leader board. The hidden test set has $10,000$ videos and is used for the final test to determine the winners. Both public and hidden test sets have equal numbers of real and tempered videos. In the case of DF-1.0, the hidden test set contains 400 videos and the augmented one has 1000 videos. However, it is not clear which types of face forgeries have been applied to faces in this dataset.

There are 8 methods applied to create a face forgery dataset. These methods can be combined to create even more diverse forged videos. These methods are DF-128, DF-256, MM/NN, Natural Talking Heads (NTH), Facial Swapping Generative Adversarial Networks (FSGAN), StyleGAN, refinement, and audio swaps. These methods produce videos of different quality. The first two methods that use autoencoders produce videos of higher quality than GAN-like methods.

The first two methods use a pair of autoencoders and were described before. In our case they produce facial regions with two different input/output resolutions 128×128 and 256×256. MM/NN face swaps performed with a custom frame-based morphable-mask model [29]. NTH model [81] generates realistic people talking heads in few- and one-shot learning settings. FSGAN method [54] allows performing facial swaps accounting for pose and expression variations. The StyleGAN [40] produces a face swap between a given fixed identity descriptor onto a video by projecting this descriptor on the latent face space. Refinement belongs to a post-processing method when the perceptual image quality is enhanced using a sharpening filter. Audio swapping is accomplished on some video clips using the TTS Skins Voice Conversion Method [55].

There are two types of augmentations applied using augmenters and distractors. Distractors overlay various kinds of objects onto a video while augmenters apply geometric and colour transforms, frame rate changes and others into the video. There are about 70% of videos influenced by augmenters and 30% of videos with distractors.

For the model's performance, two metrics are applied: weighted precision and recall. If the ratio of unaltered videos to tempered ones is $\alpha = \frac{x}{y}$, the weighted precision wP and recall R can be presented as

$$wP = \frac{TP}{TP + \alpha FP}, R = \frac{TP}{TP + FN}, \tag{6}$$

where TP, FP, and FN are true positives, false positives, and false negatives. To rank participants one uses $log - loss$ measure as weighted PR metric is often small and somewhat noisy.

There are five winners in the DFDC challenge. We are going to briefly consider their

approaches. The first solution proposes to use MTCNN face detector and EfficientNet-B7 for feature encoding. The second solution uses XceptionNet [11] for frame by frame feature extraction and WS-DAN [28] model for augmentation. In the third approach, ensemble of EfficientNets with *mixup* [85] as data augmentation is used. The fourth model uses ensemble of frame and video models such as EfficientNet, Xception, ResNet [27] and SlowFast [20]. Finally, the fifth approach also uses a MTCNN face detector and an ensemble of 7 models. Among these models, there are 3 3DCNNs, including the I3D model [10].

For future research, the creators of the DFDC dataset plan to implement the original raw dataset with around 50K of one-minute videos with annotations. This allows the creation of even much larger datasets containing fake videos in the future.

2.11 Ranking statistics of algorithm's performance

There are three principal ranking methods: average ranks (AR), success rate ratios (SRR) and significant wins (SW) [5]. The first method, AR, uses individual rankings to derive an overall ranking. The second method, SRR, ranks algorithms according to the relative advantage/disadvantage they have over the other algorithms. The third method, SW, is based on pairwise comparisons of the algorithms using statistical tests. This kind of tests is often used in comparative studies of classification algorithms.

2.11.1 Average Ranks Ranking Method

Let r_j^i be the rank of algorithm j on dataset i. The rank of the algorithms is determined by its error rate: the less is the error rate, the higher is the rank. We then calculate the AR for each algorithm:

$$\bar{r}_j = (\sum_i r_j^i)/n, \tag{7}$$

where n is the number of datasets.

2.11.2 Success Rate Ratios Ranking Method

This method employs ratios of success rates between pairs of algorithms. One creates a *success rate ratio table* for each of the datasets:

$$SSR_{jk}^i = (1 - E_j^i)(1 - E_k^i), \tag{8}$$

where ER_j^i is the measured error rate of algorithm j on dataset i.

Next, one computes a *pairwise mean success rate* ratio for each pair of algorithms j and k:

$$SSR_{jk} = \Big(\sum_i SSR_{jk}^i\Big)/n, \tag{9}$$

where n is the number of datasets.

Finally, we derive the overall *mean success rate ratio* for each algorithm, $SRR_j =$

$$SRR_j = \Big(\sum_k SRR_{jk} \Big) / (m-1), \tag{10}$$

where m is the number of algorithms. The ranking is derived directly from this measure and often is similar to AR.

2.11.3 Significant Wins Ranking Method

This method builds a ranking on the basis of results of pairwise hypothesis tests concerning the performance of pairs of algorithms. First, we construct a *win table* for each of the datasets as follows. The value of each cell W^i_{jk} indicates whether algorithm j wins over algorithm k on a dataset i at a given significance level and is determined in the following way:

$$W^i_{jk} = \begin{cases} 1 & \text{if } E^i_j \ll E^i_k \ , \\ -1 & \text{if } E^i_k \ll E^i_j \ , \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

It is obvious that $W^i_{jk} = -W^i_{kj}$. Then, we have to calculate the pairwise estimate of the probability of winning for each pair of algorithms pw_{jk}. This value estimates the probability that algorithm j is significantly better than algorithm k. Finally, we calculate the overall estimate of the probability of winning for each algorithm:

$$pw_j = \Big(\sum_k pw_{jk} \Big) / (m-1), \tag{12}$$

where m is the number of algorithms.

3 Splitting data into functional groups to reduce overfitting using co-association-based combined pattern classifiers

3.1 Introduction

Analysis of data complexity is an important subject in pattern recognition (PR) and ML [2]. Data complexity is one of the reasons for a very negative effect known in ML as overfitting. In the general case, our data consist of three types of objects in terms of classification difficulty: misclassified, ambiguous and easy (see Figure 20). Misclassified objects together with ambiguous ones can be also considered as difficult objects. Misclassified objects have a large negative margin, ambiguous patterns have small positive or negative margins, and easy ones have a large positive margin. By the margin, we understand how far objects are located from the separation hyperplane. A large negative margin means that objects from other than the target class are deep inside this class or can be located close to the centroid of this class. Ambiguous objects are located close to the separation hyperplane on both sides, and the sign of the small margin depends on whether they are from a target class or not. For those objects, using different, even relatively accurate classifiers can lead to different classification results. Easy objects are typical patterns from a target class, and their centroids are usually far from the separation hyperplane. All those three types of patterns are reasons for the complexity in data due to their diversity and different nature. That is why the idea to analyze them separately is a promising one.

Due to the nature of the aforementioned objects, we can separate only ambiguous patterns from others. We are rather unable to separate misclassified patterns because they are similar to the typical objects of the target class. To separate ambiguous objects from others, we need at least two dissimilar enough classifiers. We can involve more classifiers to have a bigger pool of those objects. Such a separation of objects allows avoiding the influence of other than ambiguous objects on the separation hyperplane, which should simplify the classification task in general. To this end, we need to select two dissimilar and accurate enough classifiers from the pool of generated classifiers. Then we apply a consensus or agreement of those classifiers for every object. If a consensus is achieved then we admit that those objects are easy (or misclassified), otherwise, they are ambiguous.

During a training phase, we select two or more classifiers that are dissimilar and accurate enough. Also, by applying consensus, we separate ambiguous objects from all others. Then we use those ambiguous objects to train a separate classifier, which can be heterogeneous to the classifiers used to find an ambiguous set of objects. During a validation or testing phase, we apply those already found two or more classifiers to build a consensus that allows us to detect ambiguous objects in the validation or testing dataset. Then we apply a classifier trained on ambiguous objects from the training dataset to classify ambiguous objects from the validation or testing datasets.

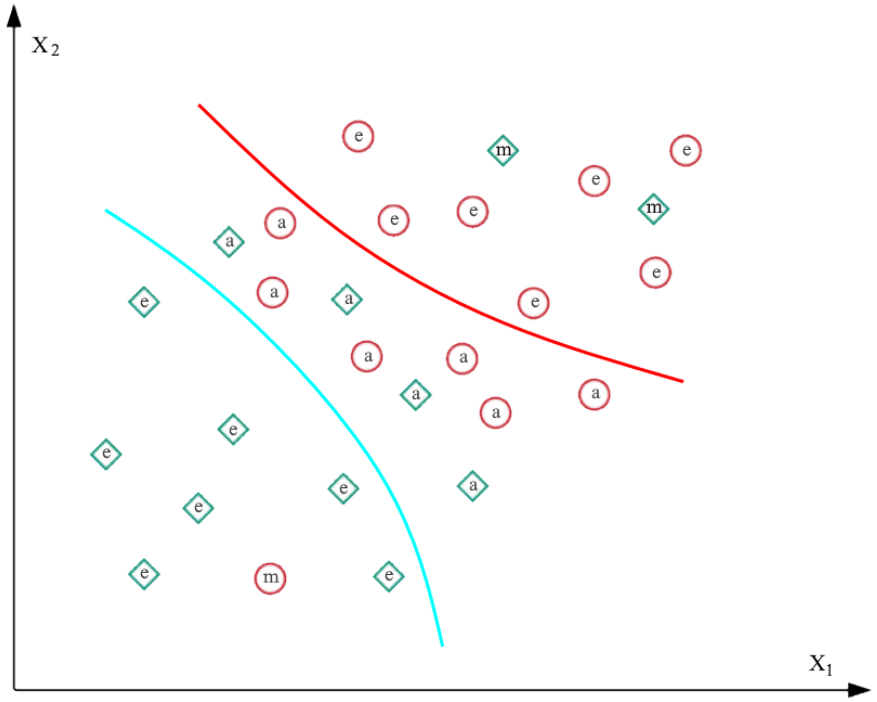


Figure 20: Different types of objects in terms of classification difficulty and two separation hyperplanes needed to build a consensus. Objects are marked as follows: "e"– easy classification objects; "a" – ambiguous objects; "m"–misclassified objects

3.2 Theoretical aspects of classifier consensus

Let us consider the consensus of classifiers using abstract algebra. Because algebraic operations are applied to classification results, they are defined on a lattice with three elements: 1 (positive decision), 0 (negative decision) and Δ (no decision is made). Such abstract algebraic operations were first considered in [90] where it was introduced an algebraic operation of addition of binary classification results ($\oplus$ algebraic operation). A consensus of classifiers corresponds to an abstract multiplication operation of classification results ($\otimes$ algebraic operation). We can unite both operations using the composition operation $\circ$. One demands a set of three algebraic laws both for addition and multiplication operations to be respected:

- preserving of element law: $a \circ a = a$;
- preserving of the set law: $a \circ \Delta \in \{a, \Delta\}$;
- commutative law.

Let us give general rules on how to apply consensus operation on the lattice with three elements a, b and Δ such that $a \neq b$ and $c = a \circ b$. So the result of consensus operation on those three elements belongs to a set $\{a, b, \Delta\,\{a, \Delta\}, \{b, \Delta\}\}$:

$$
\begin{aligned}
\otimes(a,a) = a \circ a = a;\\
\otimes(a,b) = a \circ b = b \circ a = \Delta;\\
\otimes(\Delta,\Delta) = \Delta \circ \Delta = \Delta;\\
\otimes(a,\Delta) = a \circ \Delta = \Delta \circ a = \{a,\Delta\};\\
\otimes(a,a,b) = (a \circ a) \circ b = \Delta \neq\\
(a \circ b) \circ a = a \circ \Delta = \{a,\Delta\};\\
\otimes(a,a,b,b) = (a \circ a) \circ (b \circ b) =\\
(a \circ b) \circ (a \circ b) = (a \circ b) = \Delta;\\
\otimes(a,a,a,b) = (a \circ a) \circ (a \circ b) = a \circ \Delta = \{a,\Delta\}.
\end{aligned} \tag{13}
$$

Due to the symmetry of algebraic operations, we can change a on b everywhere and the expressions remain true.

Let us consider in detail the algorithm that allows us to split the data into functional groups, depending on their complexity. Assume that there is a set of strong classifiers F, so that any strong classifier belongs to this set, e.g. $f_i \in F, i = 1, ...n$, where n is a number of strong classifiers in a set so that $|F| = n$. Here one considers classification into N classes. Let I be the number of objects in the set and P be the number of classification results.

Every classification p ($p = 1, ..., P$) associates observation k from the set with one and only one class. Elementary co-association matrix $\mathbf{A}^k$ contains the information according to which algorithms u and v have consensus or not with respect to some class label i for an object k:

$$
a^k_{u,v} = \begin{cases} 1, & \text{for } u \equiv v \\ 0, & \text{otherwise} \end{cases} \tag{14}
$$

where $\equiv$ denotes consensus between classifiers f_u and f_v. Because $u \equiv v$ and $v \equiv u$, $\mathbf{A}^k$ is a $n \times n$ symmetric binary matrix. The number of different consensuses that could be created are equal to $P = \binom{n}{2} = \frac{n(n-1)}{2}$.

From the entire set of classifiers equal to n one needs to select two of them that are maximally dissimilar in terms of Hamming distance between binary results of classification for objects $k \in I$ of the training part of a set $\mathcal{X}$. Finding maximal Hamming distance between all the classifiers in the generated pool of classifiers spans the entire classifier set F. This allows to consider all possible combinations between classifiers and use all possible Hamming distances between them, which is important during optimization of consensus. Results of classification are represented as a binary vector $\mathbf{v}^p$ that consists of zeros and ones. The elements of vector p are taken from the set of matrices $\mathbf{A}^k$:

$$
v_k = a^k_{u,v}. \tag{15}
$$

The size of the vector $\mathbf{v}^p$ is equal to the number of objects, I and the number of those vectors is equal to $\binom{n}{2}$. Then tensor $\mathbf{A}$ has size $n \times n \times I$. Now we need to determine the pair of classifiers that yield the maximal Hamming distance. Determination of appropriate indices could be done by the following procedure

$$\left\{i, j\right\} = \arg\min_{u,v} \sum_{k} A^{k}_{u,v}. \tag{16}$$

To estimate the pair of the most dissimilar classifiers, one needs to use some training set. That is why solution (16) is approximate.

It is possible to estimate an upper bound of the total error of the consensus-based approach based on misclassified and ambiguous partitions of objects. We assume that the worst-case scenario is when we classify ambiguous objects correctly in 50% of cases when we have a binary classification problem. For simplicity, assume that our dataset is balanced. Then if the portions of misclassified and ambiguous objects are γ_m and γ_a the upper bound of the total error is:

$$\epsilon_t = \gamma_m + 0.5\gamma_a. \tag{17}$$

The upper bound of the total error allows us to evaluate how far is the actual error produced by a certain classifier is below this bound. For a binary classification problem, this error can be always achieved. The strategy applied should reduce the value of γ_m moving object to the ambiguity set of objects because the maximum error of reclassification of ambiguity objects will not be higher than 0.5. However, in this case, we can move easy for classification objects to ambiguous sets as well. So it should be some trade-off between those sets of objects. On the other hand, relatively strong classifiers should reclassify the ambiguous portion of objects with an error probability less or much less than 0.5. Assuming that the error caused by a misclassified portion of objects is irreducible, we can write the bounds for the classification error:

$$\gamma_m \leq \epsilon_t \leq \gamma_m + 0.5\gamma_a. \tag{18}$$

If the classification error of any classifier on a certain dataset is greater than the upper bound of (18) then such a classifier should not be used for this dataset.

3.2.1 Experiments

We have carried out some experiments to justify the necessity of splitting initial datasets into functional groups and performing a corresponding training. We used 5 datasets from UCI repository [22] as indicated in Table 1. Three of them ('pima', 'bupa' and 'heart') belong to a medical domain and two others ('german' and 'sonar') are not medical datasets. They are all binary, however they belong to the most difficult (in terms of classification) problems from the UCI repository used for different tests of machine learning algorithms.

Table 1: Summary of characteristics of used datasets

Method/Task	**'pima'**	**'bupa'**	**'heart'**	**'german'**	**'sonar'**
Size	768	345	270	1000	208
Features	8	6	13	20	60
Classes	2	2	2	2	2

Table 2: Accuracy of classification for different combining algorithms: decision trees

Method/Task	'pima'	'bupa'	'heart'	'german'	'sonar'
Bagging	75.4	68.7	78.7	75.6	79.3
AdaBoost	69.8	65.8	78.1	73.3	79.8
RS	73.6	65.8	80.5	73.9	80.3
Rotation Forest	75.4	69.5	79.4	77.0	81.8
RO	77.2	69.3	78.1	74.4	78.4
Proposed (NB)	78.6	69.6	83.6	71.2	73.7
Proposed (LR)	78.6	60.3	76.9	68.0	82.2
Proposed (AdaBoost)	**86.0**	**84.3**	**88.2**	**81.3**	**92.4**
Accuracy bounds (Max/Min)	86.0/75.2	87.3/66.0	89.8/73.7	85.0/68.0	92.4/75.4

Table 3: Accuracy of classification for different combining algorithms: linear classifiers

Method/Task	'pima'	'bupa'	'heart'	'german'	'sonar'
Bagging	76.8	67.8	65.3	77.7	74.5
AdaBoost	77.3	68.6	69.4	76.7	73.6
RS	75.9	59.7	80.1	74.1	78.0
Rotation Forest	63.4	67.8	63.6	76.9	73.6
RO	76.0	73.0	67.7	76.4	78.9
Proposed (NB)	78.6	69.6	83.6	71.2	73.7
Proposed (LR)	78.6	60.3	76.9	68.0	82.2
Proposed (AdaBoost)	**86.0**	**84.3**	**88.2**	**81.3**	**92.4**
Accuracy bounds (Max/Min)	86.0/75.2	87.3/66.0	89.8/73.7	85.0/68.0	92.4/75.4

To build a consensus, one uses a set of k-NN classifiers with different L_p norms as dissimilarity functions. We changed p in a norm L_p from 0.1 to 5 with a step of 0.1. The value of k was changed from 1 to 30 using only odd values of k nearest neighbours yielding a pool of 800 k-NN classifiers. The classification results were compared to those obtained by the most popular ensembles of classifiers or their modifications e.g. Bagging, Boosting, Random Forests, Random Subspace (RS) and Random Oracle (RO). In one scenario (Table 2) as a base classifier decision tree has been taken and in the second scenario (Table 3) we used linear classifiers. The accuracies of those five ensembles of classifiers were compared to three versions of the developed algorithm for each of the two scenarios. The difference between them is only in the reclassification of "ambiguous" patterns. So the first version uses Naive Bayes (NB) classifier to reclassify these objects, the second one uses Logistic Regression (LR) and the third one uses AdaBoost with decision stumps as weak classifiers. All three versions use 90% of "ambiguous" objects for training and 10%–for testing. In some situations when the portion of difficult data is small in comparison with all available data, the ratio between training and testing partitions should be reduced.

Comparing NB and LR classifiers based on Tabs. 2 and 3 one may conclude that NB outperforms LR on "difficult" data in the most of the cases. This means that a generative model might better fit "difficult" data than a discriminative one which is oriented on training data and completely learns from the data. Nevertheless, AdaBoost learns very well on entirely "difficult" patterns and does not overfit. This explains that by reclassifying "ambiguous" data

with boosting algorithms, one might obtain very small error rates even on testing examples. Such behaviour of AdaBoost can be explained by the fact that one reduced the hypothesis space by selecting two of the most dissimilar classifiers from the pool of generated classifiers. These classifiers determine a set of "ambiguous" data. If one selects other two classifiers from the pool they can detect different subsets. So appropriate selection of classifiers is a very important issue. Thus learning classifiers on a partition of "difficult" instances which are grouped near the separating hyperplane and are characterized by lower variance in comparison to the entire set allows us to reduce the set of all potential classification hypotheses. This leads to better learning, less error rate on the test data and hence less overfitting.

Comparing the accuracies for five combining algorithms using decision trees as base classifiers and considered in [47] with the lower bound of the accuracy which can be always achieved (see Tabs. 2 and 3) one can see that AdaBoost and RS algorithms produce accuracies which are lower than lower bound for 'pima' and 'bupa' datasets. In the case of linear classifiers as base ones this happens for Rotation Forests ('pima' dataset), RS ('bupa' dataset), Bagging, AdaBoost, Rotation Forest and RO ('heart' dataset), Bagging, AdaBoost and Rotation Forest ('sonar' dataset). The reason for that is the fact that those algorithms classify ambiguous data with errors of more than 0.5. That is why it is important to separate an ambiguous portion of data from the entire dataset and reclassify it afterwards.

In Table 5 the separation of data into functional groups is given. So, any object can be misclassified, ambiguous or correctly classified by consensus. The difficulty of the dataset can be estimated by a number of misclassified patterns. This is an irreducible error. To have lots of "ambiguous" patterns may not be very disadvantageous since they can be learnt effectively using for example AdaBoost. Table 4 shows the accuracy of classification for some classifiers in reclassifying "ambiguous" data. As seen, the best results of such reclassification are provided by AdaBoost. It outperforms NB and LR with a large margin. As seen from the last two tables, the error of reclassification of "ambiguous data" using NB and LR is high enough which means that selecting the most dissimilar classifiers is reasonable. However, if we need less dissimilar classifiers we can also find them in the entire pool of generated classifiers. Thus, selection of algorithms to build a classifier consensus has to lead to a total error minimization which is a trade-off between misclassification error and error caused by "ambiguous" data.

Table 4: Reclassification accuracy of ambiguous data %

Method/Task	**'pima'**	**'bupa'**	**'heart'**	**'german'**	**'sonar'**
NB	65.8	58.5	80.8	59.5	45.0
LR	65.7	36.7	60.0	50.0	70.0
AdaBoost	**100**	**93.0**	**95.0**	**89.0**	**100**

Figure 21 presents the performance results of our algorithm for the five datasets mentioned above. The first column shows different errors produced by consensus, and the second one is about how close the real consensus error might be to that one when having error independence. All dependencies are built as a function of normalized Hamming distance. From the results given in the first column, we see that total consensus error drops out having more dissimilar classifiers in consensus. The second column shows that in the case of more dissimilar classifiers real consensus error approaches error when assuming independence of

errors in consensus.

Table 5: Distribution of different types of data (%)

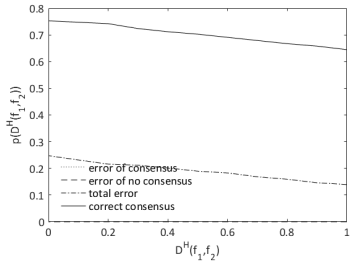
Method/Task	**'pima'**	**'bupa'**	**'heart'**	**'german'**	**'sonar'**
Misclassified	14.0	12.7	10.2	15.0	7.6
Ambiguous	21.6	42.7	32.2	34.0	34.0
Correctly classified	64.4	44.6	57.6	51.0	58.4

Figure 22 presents the local and global diversity scores for all five datasets. The first column corresponds to the local diversity scores, i.e., the diversity scores between any of the two classifiers as a function of the normalized Hamming distance. To evaluate that, one uses coefficients based on correlation and Q-statistics. The second column represents the global diversity score defined as entropy of generated and preselected classifiers as a function of the normalized Hamming distance. This shows how diverse is our subset of generated classifiers after the limitation of maximal error they can produce.

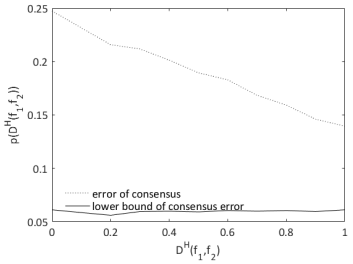
Figure 23 shows the results of reclassification of ambiguous data using AdaBoost with decision stumps for the same five problems from the UCI repository. The first column represents the training subset of ambiguous data and the second one – the testing subset. As seen from the presented results AdaBoost performs very similarly on the training and testing subsets of ambiguous data. For some problems, error approaches zero both during training and testing. This validates our hypothesis that splitting data into the difficult and easy part with the further reclassification of difficult one allows for radical improvement of learning in general. Learning only how to reclassify difficult parts of data needs lower size hypothesis space or lower capacity of our classification model which means better learning and less overfitting as observed in a number of results.

As we see, data decomposition is very important for training our classification models. Training separate models on some parts of data depending on their classification complexity is crucially important. Doing that, we receive much better performance and much less overfitting. Training our models on different subsets of training data depending on their complexity in terms of classification allows us to obtain independent models where patterns from one subset do not influence another one. This allows us to squeeze the hypothesis space and reduce the capacity of classification models which is required for such particular classifications on parts of the dataset.

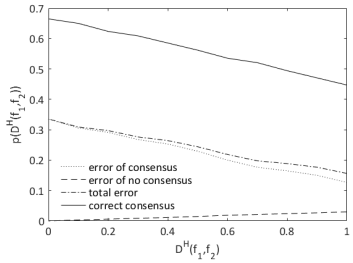
If we train our model on an entire amount of data we need to pass over a long training process (also depends on how much data we have) which might lead to overfitting of our model. Data decomposition allows us to obtain fewer data for each training and also each part of our data is less complex due to decomposition. All this leads to faster training, allowing to obtain fewer complex models and thus better generalization and less overfitting.



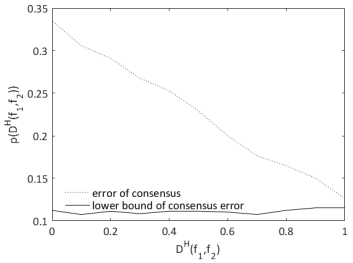
(a) Task "pima"



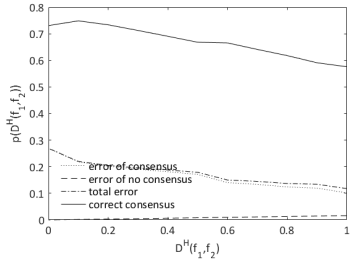
(b) Task "pima"



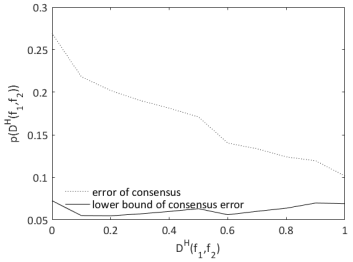
(c) Task "bupa"



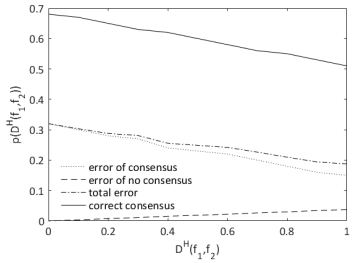
(d) Task "bupa"



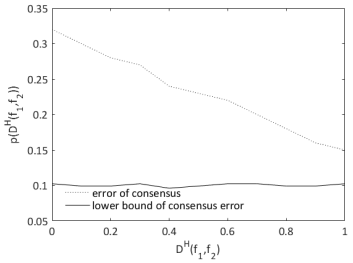
(e) Task "heart"



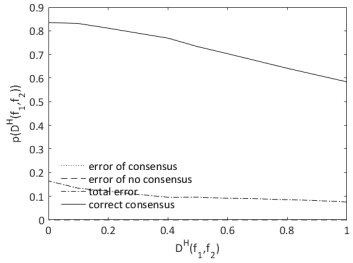
(f) Task "heart"



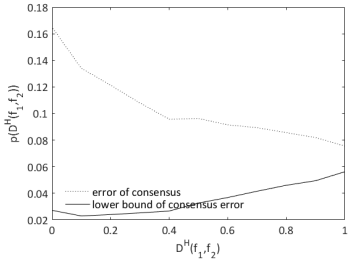
(g) Task "german"



(h) Task "german"

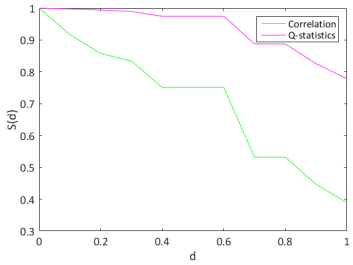


(i) Task "sonar"

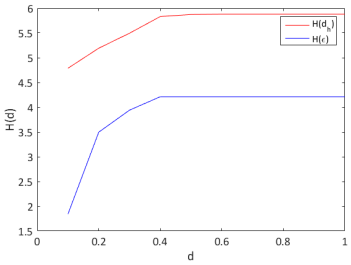


(j) Task "sonar"

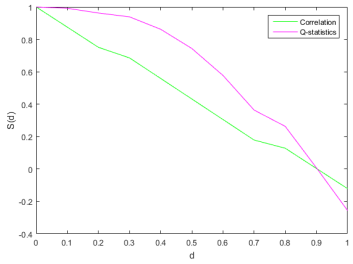
Figure 21: Performance of consensus: different errors during consensus and lower bound of consensus error



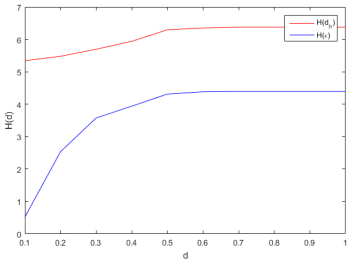
(a) Task "pima"



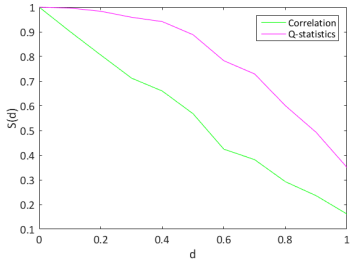
(b) Task "pima"



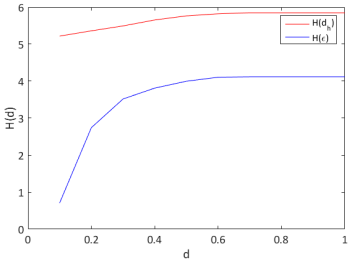
(c) Task "bupa"



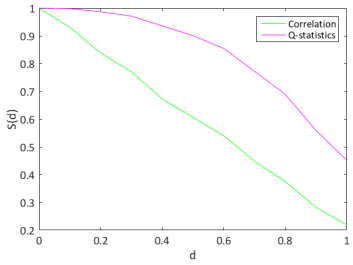
(d) Task "bupa"



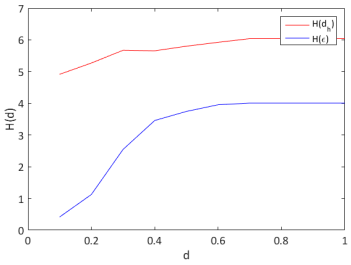
(e) Task "heart"



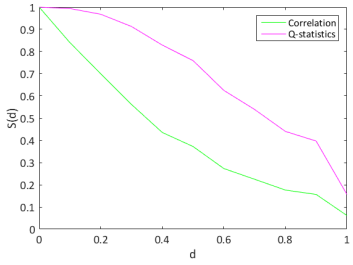
(f) Task "heart"



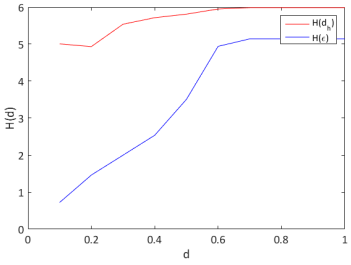
(g) Task "german"



(h) Task "german"

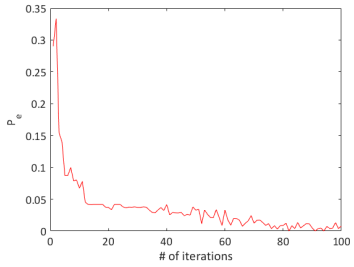


(i) Task "sonar"

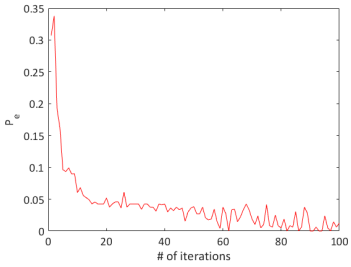


(j) Task "sonar"

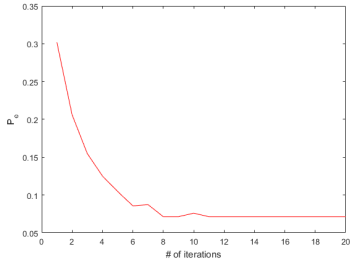
Figure 22: Local and global diversity scores



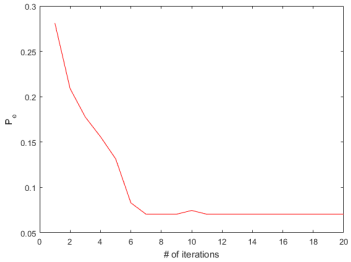
(a) Task "pima" (training)



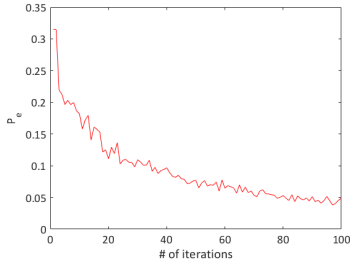
(b) Task "pima" (testing)



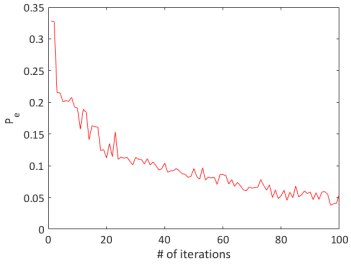
(c) Task "bupa" (training)



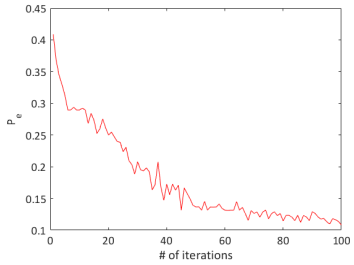
(d) Task "bupa" (testing)



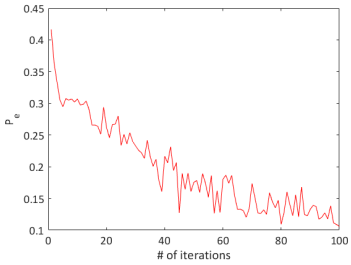
(e) Task "heart" (training)



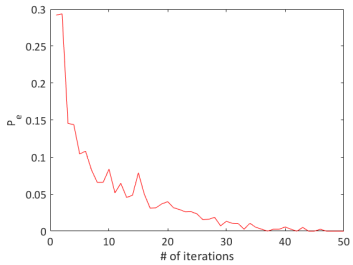
(f) Task "heart" (testing)



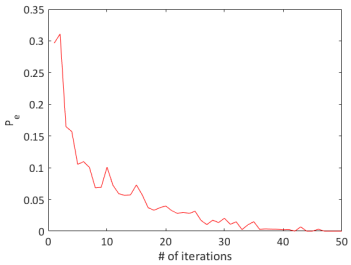
(g) Task "german" (training)



(h) Task "german" (testing)



(i) Task "sonar" (training)



(j) Task "sonar" (testing)

Figure 23: Reclassification of ambiguous data with decision stumps trained by AdaBoost

3.3 Algebraic approach to measure a distance between classifier subensembles

Let us analyze small ensembles of classifiers from two up to four. The set of possible decisions producing by an ensemble of two classifiers is a, b, Δ:

$$\otimes(a, a) = (a \circ a) = a;$$
$$\otimes(a, b) = (a \circ b) = \Delta.$$

If one adds one more classifier in ensemble, then the set of decisions will be $\{a, b, \Delta, \{a, \Delta\}, \{b, \Delta\}\}$. This is possible because associative law is not satisfied. Thus, one can end up with the subset $a, b, \{a, \Delta\}, \{b, \Delta\}$ or a, b, Δ: $\{a, b, \{a, \Delta\}, \{b, \Delta\}\} \bigcup \{a, b, \Delta\} = \{a, b, \Delta, \{a, \Delta\}, \{b, \Delta\}\}$. This can be illustrated as follows

$$\otimes(a, a, a) = (a \circ a) \circ a = a;$$
$$\otimes(a, a, b) = (a \circ a) \circ b = \Delta \neq (a \circ b) \circ a = a \circ \Delta = \{a, \Delta\}.$$

Using four classifiers, one can obtain the complete set of five possible decisions

$$\otimes(a, a, a, a) = (a \circ a) \circ (a \circ a) = a;$$
$$\otimes(a, a, a, b) = (a \circ a) \circ (a \circ b) = a \circ \Delta = \{a, \Delta\};$$
$$\otimes(a, a, b, b) = (a \circ a) \circ (b \circ b) = a \circ b = (a \circ b) \circ (a \circ b) = \Delta.$$

Due to the algebraic symmetry, one can obtain the same expressions for decision b by switching between a and b. Thus we obtain the final set of five possible decisions.

All aforementioned can be shown using the Cartesian product of sets. Each classifier output belongs to a set $\{a, b\}$. Output from two classifiers belongs to

$$\bigcup \otimes(\{a, b\} \times \{a, b\}) = \bigcup \otimes\{(a, a), (a, b), (b, a), (b, b)\} =$$
$$\bigcup\{\otimes(a, a), \otimes(a, b), \otimes(b, a), \otimes(b, b)\} = \bigcup\{a, \Delta, \Delta, b\} = \{a, \Delta, b\}. \quad (19)$$

Similarly, for the case of three classifiers, one can write

$$\bigcup \otimes(\{a, b, \Delta\} \times \{a, b\}) =$$
$$\bigcup \otimes\{(a, a), (a, b), (b, a), (b, b), (a, \Delta), (b, \Delta)\} =$$
$$\bigcup\{\otimes(a, a), \otimes(a, b), \otimes(b, a), \otimes(b, b), \otimes(a, \Delta), \otimes(b, \Delta\} =$$
$$\bigcup\{a, \Delta, \Delta, b, \{a, \Delta\}, \{b, \Delta\}\} = \{a, \Delta, b, \{a, \Delta\}, \{b, \Delta\}\}. \quad (20)$$

Adding more classifiers will not change the set of possible decisions. However, we have

to measure how close decisions are to either a or b. Even for an ensemble of three classifiers $\{a, \Delta\}$ means that decision is closer to a and for $\{b, \Delta\}$ – to b. So in the next part, we give an idea of how the distance between ensembles can be measured and also how to measure classification results produced by an ensemble using measure $\mu \in [0, 1]$.

To compare classification results produced by a set of two, three or four classifiers one needs some measure $\mu \in [0, 1]$. This can be arranged by introducing a distance between all five classifier decisions. Maximum of the distance (equal to 1) is achieved between two classifiers producing different binary decisions a and b: $d(a, b) = 1$. It is true for any number of classifiers. Minimum of the distance (equal to 0) between two single classifiers is achieved when decisions are identical: $d(a, a) = d(b, b) = 0$ for any number of classifiers as well. For an ensemble of two classifiers, one has three possible outcomes $\{a, b, \Delta\}$. The distance between a and Δ or b and Δ is equal to $\frac{1}{2}$ because Δ is neither a nor, b and is in the middle between a and b. If we put the origin in a ($a = 0$) then $b = 1$. Assuming that there are no weights for a and b, we come up with $\Delta = \frac{a+b}{2} = \frac{1}{2}$. Then $d(a, \Delta) = d(b, \Delta) = \frac{1}{2}$. Following the same principle for an ensemble of three classifiers, one obtains $\{a, \Delta\} = \frac{1}{3}$ and $\{b, \Delta\} = \frac{2}{3}$. Then for the case when the set of possible outcomes is $\{a, b, \{a, \Delta\}, \{b, \Delta\}\}$ one obtains the following distances

$$\begin{aligned} d(a, a) = d(b, b) &= 0; \\ d(a, b) &= 1; \\ d(a, \{a, \Delta\}) = d(b, \{b, \Delta\}) &= \frac{1}{3}; \\ d(a, \{b, \Delta\}) = d(b, \{a, \Delta\}) &= \frac{2}{3}. \end{aligned} \tag{21}$$

The case of possible outcomes $\{a, b, \Delta\}$ will not be considered since measures for Δ_a and Δ_b are different. Here Δ_a and Δ_b are unknown decisions when the majority of decisions is a and b correspondingly.

In the case of four classifiers in the ensemble, we receive the following distances

$$\begin{aligned} d(a, a) = d(b, b) = d(\Delta, \Delta) &= 0; \\ d(a, b) &= 1; \\ d(a, \Delta) = d(b, \Delta) &= \frac{1}{2}; \\ d(a, \{a, \Delta\}) = d(b, \{b, \Delta\}) &= \frac{1}{4}; \\ d(\Delta, \{a, \Delta\}) = d(\Delta, \{b, \Delta\}) &= \frac{1}{4}; \\ d(\{a, \Delta\}), \{b, \Delta\}) &= \frac{1}{2}; \\ d(a, \{b, \Delta\}) = d(b, \{a, \Delta\}) &= \frac{3}{4}. \end{aligned} \tag{22}$$

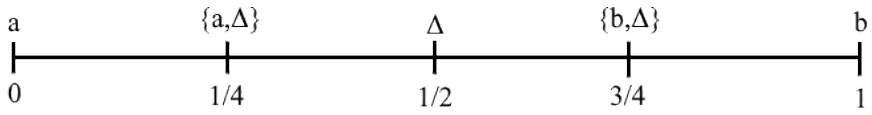


Figure 24: Illustration of how to measure the output of an ensemble of classifiers for binary classification problem and how to compute a distance between two ensembles having classification results of each classifier in ensemble (case of four classifiers in ensemble)

It is interesting to notice that if we switch between a and Δ or b and Δ (see distances 2 and 4 in (21)) we obtain the same distances. We can put the origin in b as well and all the distances will be preserved. This is because of the symmetry of algebraic operations, which is reflected by the measure of distances between corresponding operations.

Using more than four classifiers in an ensemble leads to more complex ensembles and often their number if we want to use all possible subsets of classifiers from the generalized initial set. Starting from five classifiers in the small ensemble, we have more than one $\{a, \Delta\})$ or $\{b, \Delta\})$ decisions which are equipped with different measures. To show that, let us consider two outcomes from an ensemble of five classifiers:

$$\begin{aligned} \otimes(a, a, a, a, b) &= (a \circ a) \circ (a \circ b) \circ a = \{a, \Delta\}; \\ \otimes(a, a, a, b, b) &= (a \circ b) \circ (a \circ b) \circ a = \{a, \Delta\}. \end{aligned} \tag{23}$$

In the first case one has $d(a, \{a, \Delta\}) = \frac{1}{5}$ and in the second case $d(a, \{a, \Delta\}) = \frac{2}{5}$. This is fine because the measure should reflect different classifier outputs even if the result of the algebraic operation is the same. However, as it was aforementioned, involving more classifiers leads to using more distances and overall complexity of the final complex classifier.

Instead of using every single classifier in the ensemble, we use small subensembles of two, three and four classifiers created as subsets of classifiers from the entire pool of generated classifiers. This gives a much larger variety of decisions if we consider every small ensemble as a single complex classifier that produces up to five different decisions instead of two for binary classification problems. Algebraic operations on classifier outputs applied in a form of a consensus operator allow enlarging the decision space. This should help to achieve better discriminative abilities for the entire algorithm, applied to classification problems with heavily overlapped classes.

4 Nonlinear classifier stacking in Riemannian geometry

4.1 Introduction

In this section, we are going to consider how R-geometry can be applied to build a nonlinear version of classifier stacking. It is known that classifier stacking is superior over the ensembles equipped with MV or other fusion algorithms [88]. We will show how to build a nonlinear classifier stacking in case of R-manifold of SPD matrices, as well as G-manifold built by using DPs. All data transformations needed to build both manifolds can be illustrated using homotopy diagrams. They can also be used to compute the computational complexity of any pipeline applied during the experiments. During the experimental part, we compared validation accuracies and log-losses for all stacking-based classification algorithms, as well as their complexities.

4.2 Topological and homotopy aspects of classifier stacking in Riemannian geometry

Stacking-based ensembles need data transformation. Depending on the geometry, such a transformation can be linear or nonlinear. Simple stacking requires transformation of the initial feature space to the prediction space, which is a space of new features. Formally, one has to accomplish two mappings: $f : X \to Z$ $(\mathbb{R}^m \to \mathbb{R}^k)$ and $g : Z \to Y$. In terms of homotopy this yields $H : X \times \{0, 1\} \to Y$, s.t. $H(x, 0) = f(x), H(x, 1) = g(x) = h(f(x)) = (h \circ f)(x)$.

Homotopy of data transformation using CCE or CRF can be expressed as a series of mappings $X \to Z_1 \to Z_2 \to Z_3, ..., \to Z_n$ $(\mathbb{R}^m \to \mathbb{R}^{k_1} \to \mathbb{R}^{k_2} \to \mathbb{R}^{k_3}, ..., \to \mathbb{R}^{k_n})$. Then classification is a mapping $Z_n \to Y$. Formally, it can be written as $H(x, 0) = f_1(x), H(x, 1) = (f_1 \circ f_2 \circ f_3, ..., \circ f_n)(x)$. Then any intermediate mapping can be formalized as $H(x, t) = (f_1 \circ f_2 \circ f_3, ..., \circ f_i)(x), i = 2, ..., n - 1, t \in (0, 1)$.

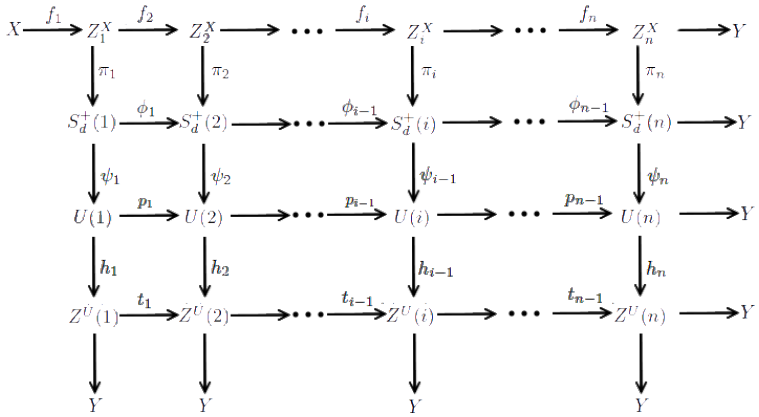


Figure 25: Homotopy diagram of data transformations to learn on R-manifolds

For a R-manifold of SPD matrices, $S_d^+ \subseteq \mathcal{M}$ we have the following mappings. In case of simple classifier stacking using S_d^+, we can write the following sequence of mappings $X \to Z \to S_d^+ \to Y$. In this case, we have $H(x, 0) = f_1(x)$, $H(x, 1) = g(x) = (f_1 \circ f_2 \circ f_3)(x)$. In case

of CCE, we can write $X \to Z_i \to S_d^+(i) \to Y, i = 1, ..., n$. On the other hand, we can consider homotopy between S_d^+ matrices of the following form $S_d^+(1) \to S_d^+(2) \to S_d^+(3), ..., \to S_d^+(n)$ in case of CCE. Every $S_d^+(i)$ matrix is computing using the output of the ith cascade Z_i. The homotopy between topological spaces of S_d^+ matrices and labels Y can be written as $(f_1 \circ f_2 \circ f_3, ..., \circ f_{n-1})(s)$ (see Figure 25). Using CCE requires vectorizing S_d^+ matrices with linearizing or not. We explain how to do matrix vectorizing properly in the upcoming subsection. Let us denote $vect(S_d^+)$ as U and $vect(\log(S_d^+))$ as V. After matrix vectorizing, we can apply simple classifier stacking or CCE to the new transformed data. In case of simple classifier stacking, the mapping will be $X \to Z^X \to S_d^+ \to U \to Z^U \to Y$ and $X \to Z^X \to S_+^d \to V \to Z^V \to Y$. Here Z^X and Z^U (Z^V) are outputs of base classifiers of a classifier ensemble applied to initial and transformed data correspondingly. In case of CCE the mapping will be $X \to Z^X(i) \to S_d^+(i) \to U(i) \to Z^U(i)(Z_1^U(i) \to Z_2^U(i), ..., \to Z_m^U(i)) \to Y, i = 1, ..., n$ or $X \to Z^X(i) \to S_d^+(i) \to V(i) \to Z^V(i)(Z_1^V(i) \to Z_2^V(i), ..., \to Z_m^V(i)) \to Y, i = 1, ..., n$. Here i is the number of a cascade.

For the experiments we have implemented and analyzed the following mappings: $X \to Z_1^X \to Y$, $X \to Z_n^X \to Y$, $X \to Z_1^X \to S_d^+(1) \to Y$ and $X \to Z_1^X \to S_d^+(1) \to U(1) \to Z_1^U \to Y$. The way of forming $S_d^+(1)$ will be considered in the next subsection.

4.3 Computing 3D and 4D tensors of stacked CPPM

The most common meta-learning strategy for a classifier ensemble is called classifier stacking. It belongs to generalized prediction-based learning. The only peculiarity of it is that it assumes to use different training subsets to train individual classifiers in ensemble and the meta-classifier. The series of mappings are the following $X \to Z \to Y$, where Y is the predicted label. Thus, instead of accomplishing a direct mapping, $X \to Y$ one uses additional data transformation $X \to Z$. Using notations of metrics, the chains of mappings can be interpreted as $g \to Y$ or $d \to g \to Y$, where d and g are Euclidean and Riemannian metrics (E-and-R-metrics) correspondingly.

Another interesting representation can be made using CPPM. Then the meta-classifier works with the new data presented as classifier interactions. Advantages of using pairwise interactions are shown in [43]. However, in this work, authors use deterministic rules to combine data from matrices of pair-wise interactions and not a fusion learning. So if we have L classes, then we have a vector of prediction probabilities for each classifier that the given pattern belongs to some of L classes. These predictions are conditional probabilities $p(y = C_\ell|X)$, where c_ℓ is the ℓth class, $\ell = 1, ..., L$. Let us assume that we have T classifiers in ensemble. Then we compose a tensor $\mathbf{T}$ of size $T \times T \times L$, where for each class $C_\ell, \ell = 1, ..., L$ we have a CPPM $\mathbf{A}^\ell(x)$ $T \times t$ with elements $a_{ij}^\ell, \{i, j\} = 1, ..., T$:

$$\begin{aligned} a_{ij}^\ell(x) = p_i(y = C_\ell|X)p_j(y = C_\ell|X) = h_i^\ell(x)h_j^\ell(x), i \neq j; \\ a_{ij}^\ell = p_i(y = C_k|X) = h_i(x), i = j, \end{aligned} \tag{24}$$

where $h_i^\ell(x) = p_i(y = C_\ell|X)$. Using matrix form, we can write $A^\ell(x)$ as

$$\mathbf{A}^{\ell}(x) = \begin{bmatrix} h_1^{\ell}(x) & \dots & h_1^{\ell}(x)h_j^{\ell}(x) & \dots & h_1^{\ell}(x)h_T^{\ell}(x) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ h_1^{\ell}(x)h_i^{\ell}(x) & \dots & h_i^{\ell}(x) & \dots & h_i^{\ell}(x)h_T^{\ell}(x) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ h_1^{\ell}(x)h_T^{\ell}(x) & \dots & h_T^{\ell}(x)h_j^{\ell}(x) & \dots & h_T^{\ell}(x) \end{bmatrix} \tag{25}$$

Thus, we can learn the information presented in these CPPMs which can be used for the final classification. Instead of building every separate CPPM for every class label ℓ [43] we stack all of them in a tensor $\mathbf{T}$. Then this tensor is used as an input to CNN for learning.

If we want to explore the interactions between three classifiers or to build subensembles involving three classifiers, then the learning pattern will be a $(0,4)$ tensor of size $T \times T \times T \times L$. The elements of this tensor can be computed as follows:

$$\begin{aligned} a_{ijk}^{\ell}(x) &= h_i^{\ell}(x), \quad i = j = k; \\ a_{ijk}^{\ell}(x) &= h_i^{\ell}(x)h_k^{\ell}(x), \quad i = j \quad \text{and} \quad i \neq k; \\ a_{ijk}^{\ell}(x) &= h_j^{\ell}(x)h_k^{\ell}(x), \quad i = k \quad \text{and} \quad j \neq k; \\ a_{ijk}^{\ell}(x) &= h_i^{\ell}(x)h_j^{\ell}(x), \quad j = k \quad \text{and} \quad i \neq j; \\ a_{ijk}^{\ell}(x) &= h_i^{\ell}(x)h_j^{\ell}(x), \quad i = j \quad \text{and} \quad j \neq k; \\ a_{ijk}^{\ell}(x) &= h_i^{\ell}(x)h_j^{\ell}(x)h_k^{\ell}(x), \quad i \neq j \neq k. \end{aligned} \tag{26}$$

As one can see, there are lots of same elements in the aforementioned tensor. At the same time, those matrices may have many small or even zero-valued elements. This means that SPD matrices as $(0,2)$ subtensors of a $(0,4)$ tensor, indexed by a pair of $\{i,j\}$ indexes, are SPD ones and singular with a very high probability. This is because we involved more classifiers in classifier subensembles and the probability that all of them have high values of prediction probabilities is less than in case of two classifiers. Involving more classifiers to build larger subensembles should even worsen the situation. This may be important if we want to compute some characteristics of a R-manifold such as Christoffel symbols (CS). CS can be used as a pattern because they describe a metric connection, which is a specialization of affine connection to surfaces. The latter straightforwardly leads to such concepts as parallel transport, covariant derivatives, geodesic, Ricci and curvature tensors and others. Torsion free connection or Levi-Civita connection can also be expressed in terms of CS.

The nonzero elements of CS can be exploited as some attributes or features of SPD matrices. For more details one can refer to [53], here we provide only some details. In our case, if we have a SPD matrix $A^{\ell}(x)$ depending on a pattern x, we first should compute a determinant ρ of this matrix $\rho(x) = \det(A^{\ell}(x))$. The adjoint matrix can be expressed as $\text{adj}(A^{\ell}(x)) = \rho(x)(A^{\ell}(x))^{-1}$. Then vectorizing the adjoint matrix in terms of (30) we get the nonzero element of CS. In our notations, we have:

$$s = [s^1, s^2, s^3, ..., s^n] = \text{vec}(\text{adj}(A^\ell(x))) = \text{vec}(\rho(x)(A^\ell(x))^{-1}) = \text{vec}(\det(A^\ell(x))A^\ell(x))^{-1}), \tag{27}$$

where $[s^1, s^2, s^3, ..., s^n]$ are nonzero elements of CS.

As some other kind of features, one can compute eigenvalues $\lambda_i, \quad i = 1, ..., T$ of SPD matrices $A^\ell(x)$. Because matrices $A^\ell(x)$ are symmetric and have nonnegative values, their eigenvalues will be also positive real values and ranked from the largest to the smallest one. We can also reduce a feature space when using $A^\ell(x)$ matrix vectorizing in terms of (30). This can be done in two ways: by applying PCA or using autoencoders. We are going to carry out the experiments using all aforementioned features.

4.4 Learning from classifier pairwise interactions on Riemannian manifolds

For each pattern X we have an appropriate tensor $\mathbf{T} : X \rightarrow \mathbf{T}$. A 3D tensor $\mathbf{T}$ consists of a stack of SPDs. This means that every instance can be presented as L connected R-manifold. Because predictions for every single class sum up to one, then we have $L-1$ independent predictions and $\{\mathcal{M}_\ell\}_{\ell=1}^{L-1}$ R-manifolds. The CNN can be applied to the tensor $\mathbf{T}$ to accomplish the classification. If we apply 2D convolutional filters, then we learn features independently for every $L-1$ 2D tensors or matrices. For larger datasets, one can use DLAs such as GoogleNet (2015) and ResNet (2016)) [63, 27].

A matrix $\mathbf{A}^\ell$ is a $(0, 2)$ metric tensor. This tensor has 0 contravariant and 2 covariant components. A $(0, 3)$ tensor $\mathbf{T}$ which has 3 covariant components is not a metric one, meaning that one can not use it to compute distances. For distance computation, we have to use $L-1$ metric tensors on $L-1$ R-manifolds. The final distance can be computed as the sum of distances on all $L-1$ independent R-manifolds:

$$d(x, y) = \sum_{i=1}^{L-1} d(x_i, y_i), \tag{28}$$

where $d(x_i, y_i)$ is a distance between two points on the $\mathcal{M}_i$ manifold.

For unlabeled patterns, we should associate it with some class based on $L-1$ manifolds which are learned by examples from the training set.

To project our SPD matrix to E-space, one needs to select an appropriate projection point. In a very general case, it can be done using the method proposed in [38]. For manifolds with quite a low curvature, one can use the identity matrix $T \times T$ as a projection point. Then the projection is computed as

$$Proj(\mathbf{A}^\ell) = \mathbf{U} \log(\mathbf{\Lambda}) \mathbf{U}^T, \tag{29}$$

where $\mathbf{U}$ is the matrix of eigenvectors of $\mathbf{A}^\ell$ and $\mathbf{\Lambda}$ is the diagonal matrix containing eigen-

values of $\mathbf{A}^{\ell}$. However, we need to compute eigenvectors and eigenvalues for $T \times T$ matrices $(L-1) \times N$ times, where N is the size of the dataset. In general, this is computationally expensive. That is why generative models may be useful to generate the projection matrices avoiding the computationally expensive eigenvalue computation routine. All learning algorithms such as SVM, k-NN, MLP, DF (ET) work in a linear vector space. That is why one needs to accomplish vectorizing of a metric tensor. Because $Proj(\mathbf{A}^{\ell})$ is a symmetric matrix too, it can be presented as a row vector containing elements under a principal diagonal together with the elements of the principal diagonal. The orthonormal coordinates of a tangent vector $\mathbf{y}$ in the tangent space at point $\mathbf{X}$ are given by

$$\text{vec}^{\ell}_{\mathbf{X}}(\mathbf{y}) = \text{vec}^{\ell}_{\mathbf{I}}\Big(\mathbf{A}^{\ell}\Big), \tag{30}$$

where $\text{vec}_{\mathbf{I}}(\mathbf{y}) = \Big[y_{1,1}, \sqrt{2}y_{1,2}, \sqrt{2}y_{1,3}, ...y_{2,2}, \sqrt{2}y_{2,3}, ...y_{d,d}\Big]^{T}$. Then we compute the coordinates in a flattened E-space by vectorizing the projection matrix:

$$\text{vec}^{\ell}_{\mathbf{X}}(\mathbf{y}) = \text{vec}^{\ell}_{\mathbf{I}}\Big(Proj(\mathbf{A}^{\ell})\Big) = \text{vec}^{\ell}_{\mathbf{I}}\Big(\mathbf{U}^{\ell}\log(\mathbf{S}^{\ell})(\mathbf{U}^{\ell})^{T}\Big). \tag{31}$$

.

Stacking all the $\text{vec}^{\ell}_{\mathbf{I}}$ vectors together, one obtains the final vector $\text{vec}_{\mathbf{I}}$ that is the input of one of the aforementioned classifiers. In opposite to existing learning algorithms, a DLA such as a CNN can learn the 3D tensor $\mathbf{A}$ directly. This is because it allows accomplishing end-to-end learning. A vectorizing process is performed inside the CNN before it goes to fully connected layers at the final stages of the CNN. Similarly to the previous, if one uses a metric tensor for learning directly, then the only interactions between the predictions are taken into account and E-geometry is applied. The nonlinearity of a manifold is exploited when we compute a projection matrix $Proj(\mathbf{A}^{\ell})$. Stacking vectorized metric tensors and their projections means that we use them independently and the entire information is summing up, which corresponds to the independent usage of metric tensors. For multi-class problems with many classes the resulting vectors can be very large which may require some dimensionality reduction routines such as PCA or autoencoders. Details of the algorithm showing how to build an R-manifold of SPD matrices from classifier predictions and the learning routine using different classifiers including CNN is shown in Appendix B.

4.5 Learning classifier ensembles by decision profiles on Grassmann manifolds

Let us consider a decision template (DTL) originally proposed in [48]. For a pattern x, predictions from classifiers in an ensemble can be written as DP in the matrix form

$$DP(x) = \begin{bmatrix} h_1^1(x) & \dots & h_1^j(x) & \dots & h_1^\ell(x) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ h_i^1(x) & \dots & h_i^j(x) & \dots & h_i^\ell(x) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ h_T^1(x) & \dots & h_T^j(x) & \dots & h_T^\ell(x) \end{bmatrix} \tag{32}$$

Instead of using $DP(x)$ to compute a DTL

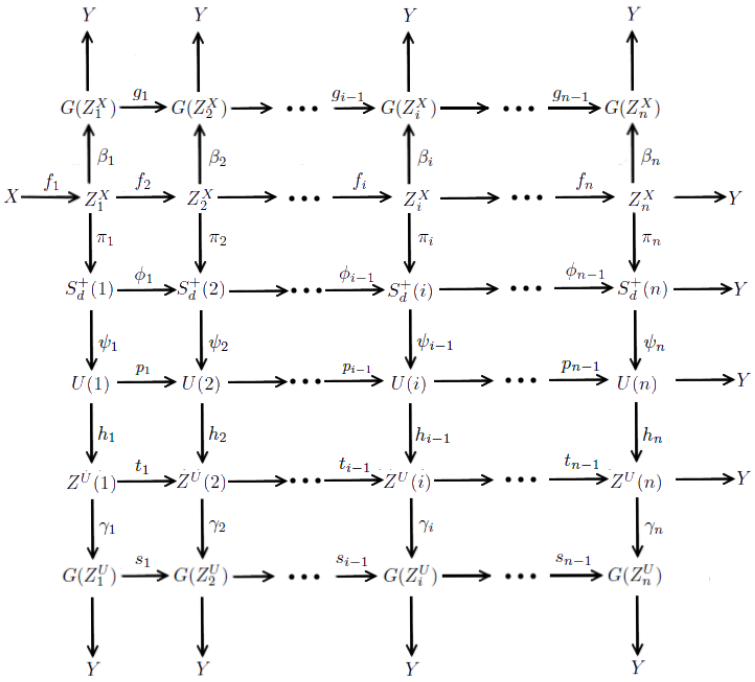


Figure 26: Homotopy diagram of data transformations to learn on G-manifolds

$$DT_k = \frac{1}{m_k} \sum_{i:y_i=c_k} DP(x_i), k = 1, ..., \ell, \tag{33}$$

where m_k is the number of instances belonging to class c_k, one can use them for learning. Since a number of columns represents a number of classes and assuming that $L < T$ one can consider a $DP(x)$ to compute an orthonormal basis using SVD:

$$DP(x) = U(x)S(x)U(x)^T \tag{34}$$

This is presented as an orthonormal matrix where the number of columns of $U(x)$ represents a size of subspace $\mathbb{R}^L$ of Euclidean space $\mathbb{R}^T$. To compute dissimilarity between $DP(x_1)$ and $DP(x_2)$ on a G-manifold we use orthonormal matrices $U(x_1)$ and $U(x_2)$ obtained as $DP(x_1) = U(x_1)S(x_1)U(x_1)^T$ and $DP(x_2) = U(x_2)S(x_2)U(x_2)^T$:

$$U(x_1)U(x_2)^T = V\Lambda V^T, \tag{35}$$

where Λ is the diagonal matrix of principal angles

$$\Lambda = diag(\cos(\theta_1), \cos(\theta_2), ..., \cos(\theta_n)). \tag{36}$$

Having principal angles we can use one of the metrics considered before. The classical way of using DP and DTL assumes to apply some similarity measure and k-NN-based algorithm to compute the proximity between decision profile initiated by instance x and DTL DT_k. For the references on how to build and apply G-manifolds as well as for nonlinear metrics on them see Appendix B.

Homotopy diagram showing all data transformations in the case of G-manifolds is shown in Figure 26. As seen from the diagram, G-manifolds have been applied to simple classifier stacking and CCE in both geometries. This means that G-manifolds are applying together with R-manifolds of SPD matrices. It is the most difficult case in terms of computational complexity.

For example, if we want to build a G-manifold for ith cascade in E-geometry from the initial data X, we can do that in two ways: $f_1 \circ f_2, ..., \circ f_i \circ \beta_i(X)$ or $f_1 \circ \beta_1 \circ g_1, ..., \circ g_{i-1}(X)$ (see Figure 26). Notwithstanding obtaining mappings g_i is not a trivial procedure. However they can be obtained approximately using learning algorithms. If we want to do the same in case of R-manifolds, we have the two following ways to do so: $f_1 \circ f_2, ..., \circ f_i \circ \pi_i \circ \psi_{i-1} \circ h_{i-1}\gamma_i(X)$ or $f_1 \circ \pi_1 \circ \psi_1 \circ h_1 \circ \gamma_1 \circ s_1, ..., \circ s_{i-1}(X)$. As seen, the chain of homotopies is longer in case of R-and-G-manifolds applied together and it is the longest overall.

4.6 Experiments

Table 6: Summary of characteristics of used UCI datasets

Dataset	**Size**	**Features**	**Classes**	**Tr, %**	**Ts, %**
balance	625	4	3	50	50
bupa	345	6	2	50	50
gamma	19200	10	2	50	50
german	1000	24	2	50	50
heart	270	13	2	50	50
mfeat-mor	2000	6	10	50	50
mfeat-zer	2000	47	10	50	50
pima	768	8	2	50	50
segment	2310	19	7	50	50
sonar	208	60	2	50	50
spambase	4601	57	2	50	50

To carry out the set of experiments, we used 11 datasets from UCI repository [22] (see Table 6). These datasets are of different scales where the minimum size is 208 and the maximum one is 19200. The instances in these datasets are equipped with binary and multi-class labels. They are also scaled by difficulty from the classification point of view: from very easy to very difficult. As seen, we selected as many datasets as possible. To make

experiments more challenging, we split all the datasets into two equal parts: for training and testing, as it was suggested in [43].

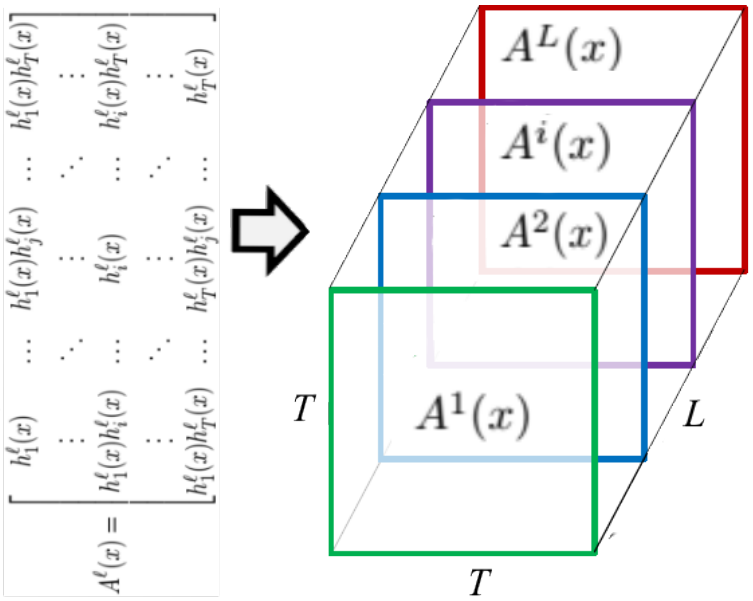


Figure 27: Tensor formation

For the first round of experiments, we used RFs and ETs to produce sets of classifier predictions. Then we used them to build matrices of pairwise interactions between the predictions coming from every DT either from RFs or ETs. We have as many of these matrices as the number of classes. After that, we compose a 3D tensor for every instance in both training and testing subsets and feed them to CNN (see Figure 27). The architecture of the CNN is presented in Figure 28 and the results of experiments related to this round are presented in Figures 29-32. The architecture presented in Figure 28 is the baseline architecture which we have used to carry out most of our experiments. We compared this architecture to such architectures as LeNet5, VGG16 and pyramidal one (it has a name CNN2 in Tables 7-10). To plot Figures 29-32 we selected four datasets from the Table 18. They are different by the number of classes and the size. The second round concerns the experiments on G-manifolds (see Figure 33). For that round, one uses DPs for every single instance. Then each DP is used to obtain an orthonormal basis using SVD. Obtained this way, an orthonormal basis is a point on a G-manifold. To measure a distance between two points on such a manifold, an appropriate measure should be applied. Finally, the k-NN classifier is applied to perform classification. The results given in Tabs. 7-10 are obtained in both rounds.

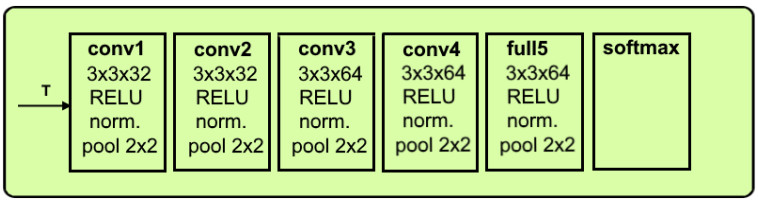


Figure 28: CNN for prediction tensor learning

Table 7: Learning classifier interactions (advanced experiments): means and standard deviations of prediction accuracy (shown in %) for each method on the benchmark datasets. A number of DTs in a RF is equal to 100 and its depth is equal to 2.

Method/Dataset	Balance	Bupa	Gamma	German	Heart	Mfeat-mor	Mfeat-zer	Pima	Segment	Sonar	Spambase
RF	80.35 ± 2.96	67.44 ± 3.47	75.88 ± 0.89	70.54 ± 1.65	79.10 ± 2.89	62.24 ± 2.87	58.06 ± 3.27	74.84 ± 1.44	75.55 ± 6.06	75.29 ± 3.65	89.05 ± 0.92
SVM	$\mathbf{89.29 \pm 0.96}$	59.07 ± 1.75	65.44 ± 0.40	70.10 ± 1.16	79.02 ± 2.61	19.73 ± 1.77	9.47 ± 2.21	65.76 ± 0.75	41.43 ± 3.13	54.04 ± 2.85	80.92 ± 0.45
SVM-stacking	82.12 ± 3.97	67.33 ± 4.33	81.28 ± 0.64	71.98 ± 2.20	80.30 ± 3.26	54.57 ± 1.75	48.19 ± 3.85	75.39 ± 1.19	71.03 ± 5.07	75.96 ± 3.89	92.58 ± 0.45
SVM-R-nonlinear	82.34 ± 3.93	67.97 ± 4.66	81.41 ± 0.65	73.50 ± 1.80	80.98 ± 3.46	44.77 ± 2.97	26.92 ± 2.84	75.52 ± 1.51	63.50 ± 5.01	76.44 ± 3.23	92.60 ± 0.60
SVM-R-linear	72.53 ± 5.42	60.70 ± 2.91	79.74 ± 0.51	70.00 ± 1.30	79.25 ± 2.86	67.01 ± 1.33	65.83 ± 1.56	67.42 ± 2.82	84.52 ± 4.85	73.94 ± 4.22	90.37 ± 1.14
k-NN	80.16 ± 1.80	63.90 ± 2.27	79.58 ± 0.35	67.20 ± 0.68	74.66 ± 2.56	39.02 ± 1.20	$\mathbf{80.82 \pm 0.88}$	71.07 ± 1.81	$\mathbf{91.72 \pm 0.84}$	75.87 ± 4.22	78.77 ± 0.71
k-NN-stacking	79.20 ± 1.85	66.40 ± 4.88	81.59 ± 0.66	71.74 ± 1.68	80.75 ± 2.95	58.75 ± 4.16	65.50 ± 1.50	73.49 ± 1.43	83.27 ± 6.49	76.44 ± 3.85	90.45 ± 2.33
k-NN-R-nonlinear	79.49 ± 1.17	67.15 ± 4.16	81.61 ± 1.37	71.94 ± 1.69	80.98 ± 2.67	58.56 ± 4.55	65.02 ± 1.63	73.18 ± 1.86	83.29 ± 6.61	76.73 ± 2.18	90.60 ± 1.48
k-NN-R-linear	78.94 ± 1.52	64.01 ± 2.74	80.82 ± 1.42	71.12 ± 1.63	79.25 ± 2.94	61.55 ± 4.38	65.20 ± 0.98	72.97 ± 1.24	87.75 ± 3.81	75.29 ± 4.21	92.30 ± 0.48
MLP	46.92 ± 2.46	50.81 ± 9.95	83.06 ± 0.67	71.60 ± 1.94	43.61 ± 23.31	18.05 ± 5.19	14.93 ± 1.75	72.81 ± 2.03	15.92 ± 2.53	47.21 ± 1.74	78.67 ± 4.22
MLP-stacking	45.90 ± 2.02	57.38 ± 9.06	83.66 ± 0.51	73.72 ± 1.46	78.65 ± 3.09	14.48 ± 4.09	15.19 ± 2.12	72.84 ± 2.52	15.90 ± 3.99	55.00 ± 7.47	82.85 ± 6.24
MLP-R-nonlinear	23.88 ± 5.90	50.11 ± 8.53	83.52 ± 0.19	73.32 ± 1.58	71.80 ± 6.08	14.20 ± 4.77	12.83 ± 1.00	74.66 ± 2.05	15.11 ± 1.15	51.63 ± 5.09	71.46 ± 8.35
MLP-R-linear	39.68 ± 9.31	42.67 ± 1.71	83.29 ± 0.43	56.24 ± 9.84	46.47 ± 21.60	15.94 ± 3.39	13.63 ± 1.56	66.54 ± 4.73	18.26 ± 5.63	60.77 ± 8.60	71.73 ± 8.39
DNF-nonlinear	82.54 ± 3.89	69.45 ± 4.37	65.11 ± 15.86	75.55 ± 1.88	$\underline{83.05 \pm 2.84}$	47.83 ± 12.83	63.38 ± 5.35	77.46 ± 2.37	70.78 ± 13.55	NAN	91.74 ± 0.82
DNF-linear	80.39 ± 11.37	66.72 ± 4.32	69.97 ± 6.94	74.53 ± 2.42	82.42 ± 2.27	38.93 ± 8.90	60.38 ± 15.75	76.48 ± 2.76	70.16 ± 6.67	NAN	91.70 ± 0.54
CNN1-R-nonlinear	85.61 ± 2.14	72.09 ± 3.81	84.10 ± 0.41	75.60 ± 0.88	82.71 ± 3.33	69.30 ± 1.30	70.58 ± 0.97	77.19 ± 1.13	87.63 ± 5.27	$\mathbf{78.65 \pm 2.71}$	93.14 ± 0.45
CNN1-R-linear	85.48 ± 1.98	68.55 ± 2.54	84.01 ± 0.33	73.72 ± 1.39	82.26 ± 2.41	$\underline{70.41 \pm 1.26}$	72.67 ± 1.28	75.39 ± 1.57	89.86 ± 2.74	76.44 ± 3.17	93.55 ± 0.47
CNN2-R-nonlinear	86.73 ± 1.88	$\underline{73.14 \pm 2.72}$	84.05 ± 0.35	$\mathbf{75.88 \pm 0.84}$	$\mathbf{83.38 \pm 2.09}$	70.30 ± 1.08	72.45 ± 0.98	$\underline{77.79 \pm 1.32}$	$\underline{90.30 \pm 2.11}$	$\underline{78.56 \pm 3.13}$	$\underline{93.77 \pm 0.26}$
CNN2-R-linear	86.60 ± 1.88	68.66 ± 4.01	$\underline{84.11 \pm 0.34}$	74.60 ± 1.13	82.86 ± 2.12	$\mathbf{70.50 \pm 1.22}$	$\underline{74.29 \pm 1.32}$	76.64 ± 1.54	90.27 ± 2.10	77.88 ± 4.15	$\mathbf{93.96 \pm 0.40}$
LeNet5-nonlinear	$\underline{86.96 \pm 1.85}$	$\mathbf{74.19 \pm 3.00}$	$\mathbf{84.14 \pm 0.33}$	$\underline{75.64 \pm 1.55}$	80.45 ± 2.89	70.17 ± 1.10	71.56 ± 1.19	$\mathbf{77.89 \pm 1.53}$	89.27 ± 1.90	77.69 ± 3.03	93.60 ± 0.37
LeNet5-linear	85.58 ± 2.15	70.29 ± 3.52	84.10 ± 0.33	75.22 ± 1.69	82.41 ± 2.08	70.41 ± 1.39	74.14 ± 1.28	77.06 ± 1.14	89.68 ± 1.93	76.44 ± 2.69	93.54 ± 0.38
VGG16-nonlinear	81.35 ± 4.93	63.95 ± 4.05	83.84 ± 0.36	74.18 ± 2.67	79.02 ± 2.61	35.83 ± 18.85	34.12 ± 22.58	77.27 ± 1.74	33.92 ± 14.53	69.23 ± 12.33	86.21 ± 12.96
VGG16-linear	67.98 ± 8.78	59.59 ± 3.28	83.38 ± 0.46	70.40 ± 1.31	79.02 ± 2.61	33.12 ± 20.84	13.05 ± 3.25	69.64 ± 4.62	23.75 ± 6.05	68.46 ± 5.92	73.43 ± 15.64
k-NN-G-geodesic	81.41 ± 1.60	65.25 ± 3.28	81.34 ± 1.52	72.45 ± 1.27	77.97 ± 3.89	56.77 ± 3.77	70.00 ± 1.18	72.92 ± 1.30	84.97 ± 4.53	74.81 ± 3.06	85.68 ± 2.17
k-NN-G-bc	80.99 ± 1.78	65.23 ± 2.75	81.08 ± 1.11	69.74 ± 2.04	77.37 ± 3.39	57.14 ± 3.26	69.22 ± 1.17	74.11 ± 1.52	85.10 ± 4.38	74.71 ± 3.77	86.25 ± 1.94
k-NN-G-projection	81.03 ± 1.34	65.52 ± 2.63	80.83 ± 1.31	72.00 ± 1.48	78.27 ± 2.50	56.15 ± 4.25	70.46 ± 1.61	72.47 ± 1.52	85.90 ± 4.63	74.80 ± 4.97	86.11 ± 2.32

To build all the tables one uses two types of ensemble classifiers: RFs and ETs as well as two different depths (two and five) of every DT in RFs and ETs. This all should bring a diversity between classifiers in an ensemble and non-linearity in the geometry of the data which lie on some manifold. Thus, the principal idea is to verify if created nonlinearity in data can simplify the separation of different instances that belong to different classes. The geometry of data is created in different ways: by using the original vectors of features, classifier predictions instead of original features, classifier interactions (CPPM matrices), DPs, different classification algorithms to generate predictions such as RFs or ETs. Additionally, data embedding on R-and-G-manifolds is provided. The linear and nonlinear versions of the classifier are created depending on if one exploited the nonlinearity of the created manifold of SPD matrices ("linear" versions) or simply the interactions between classifiers ("nonlinear" versions). Linearized versions exploit the nonlinear mapping from a manifold $\mathcal{M}$ to E-space $\mathbb{R}$ around some neighbourhood of a point on the manifold (because R-manifold of SPD matrices is locally homeomorphic to E-space). We always assume that mapping is infinitely differentiable $C^{\infty} : \mathcal{M} \to \mathbb{R}$ or $\mathcal{M} \to \mathcal{N}$.

So three different classifiers (SVM, k-NN and MLP) are compared against the original set of features, as meta-classifiers using predictions coming from RFs and ETs and as meta-classifiers using data as vectorized $(0, 2)$ metric tensor (linearized and nonlinearized). Then they are compared with linearized and nonlinearized versions of four CNN architectures that use stacked $(0, 2)$ metric tensors as an input. Next, everything is compared to nonlinear versions of decision profiles $DP(x)$ presented as points on G-manifolds with three different nonlinear measures. Finally, we present the results of classification by prediction generators (RFs and ETs).

As seen from Figures 30-32 as well as from Tables 7-10 using learning from classifier interactions in Riemannian geometry is advantageous. For most of the datasets classification accuracy is higher compared to other the most popular classifiers and classifier stacking when using R-manifolds. The proposed approach is not as much sensitive to the depth of trees

Table 8: Learning classifier interactions (advanced experiments): means and standard deviations of prediction accuracy (shown in %) for each method on the benchmark datasets. A number of DTs in RF is equal to 100 and its depth is equal to 5.

Method/Dataset	Balance	Bupa	Gamma	German	Heart	Mfeat-mor	Mfeat-zer	Pima	Segment	Sonar	Spambase
RF	85.83 ± 1.79	70.81 ± 3.58	83.80 ± 0.25	73.04 ± 1.13	81.05 ± 1.67	69.15 ± 0.97	73.97 ± 0.73	76.41 ± 1.61	92.40 ± 1.17	78.27 ± 4.33	92.76 ± 0.38
SVM	$\mathbf{89.29 \pm 0.96}$	59.07 ± 1.75	65.45 ± 0.40	70.10 ± 1.16	79.02 ± 2.61	19.73 ± 1.77	9.47 ± 2.20	65.76 ± 0.75	41.43 ± 3.13	54.04 ± 2.85	80.92 ± 0.45
SVM-stacking	84.94 ± 1.15	71.22 ± 3.14	85.86 ± 0.34	75.40 ± 1.70	80.90 ± 1.99	69.79 ± 0.86	73.55 ± 0.56	75.68 ± 1.35	93.60 ± 1.13	79.04 ± 5.01	93.56 ± 0.45
SVM-R-nonlinear	83.75 ± 1.46	71.16 ± 2.98	85.99 ± 0.31	75.56 ± 1.68	$80.08 \pm .76$	69.15 ± 0.64	62.34 ± 2.27	75.83 ± 1.23	92.70 ± 1.09	78.75 ± 4.78	93.53 ± 0.41
SVM-R-linear	83.49 ± 1.78	69.01 ± 4.63	83.72 ± 0.35	73.00 ± 1.37	80.38 ± 1.76	69.88 ± 0.81	76.28 ± 0.73	75.29 ± 2.02	94.09 ± 1.34	76.92 ± 3.16	92.88 ± 0.48
k-NN	80.16 ± 1.80	63.90 ± 2.27	79.58 ± 0.35	67.20 ± 0.68	74.66 ± 2.56	39.02 ± 1.20	$\mathbf{80.82 \pm 0.88}$	71.06 ± 1.81	91.72 ± 0.84	75.87 ± 4.22	78.77 ± 0.71
k-NN-stacking	80.67 ± 1.35	70.29 ± 3.27	85.12 ± 0.28	73.60 ± 2.04	81.13 ± 2.16	68.20 ± 1.04	74.59 ± 0.91	74.97 ± 2.27	95.01 ± 0.70	79.13 ± 3.87	93.60 ± 0.45
k-NN-R-nonlinear	80.32 ± 1.41	70.47 ± 3.41	85.08 ± 0.28	73.10 ± 1.83	80.98 ± 1.78	67.72 ± 1.09	68.62 ± 1.04	74.64 ± 1.54	94.98 ± 0.79	78.94 ± 4.46	93.38 ± 0.44
k-NN-R-linear	80.10 ± 1.34	66.45 ± 3.15	85.06 ± 0.27	71.08 ± 1.82	79.70 ± 1.81	68.42 ± 0.86	74.20 ± 0.64	73.80 ± 1.62	94.95 ± 0.88	75.96 ± 3.92	92.55 ± 0.41
MLP	47.47 ± 2.61	44.83 ± 8.44	83.10 ± 0.61	72.24 ± 1.47	45.49 ± 23.88	17.31 ± 4.50	10.59 ± 1.17	73.62 ± 2.40	15.74 ± 4.36	47.98 ± 3.11	77.59 ± 5.29
MLP-stacking	45.45 ± 2.69	51.28 ± 10.57	86.29 ± 0.37	74.76 ± 1.20	75.19 ± 4.28	11.07 ± 3.33	10.37 ± 0.60	74.66 ± 1.51	14.75 ± 0.59	62.40 ± 5.06	88.57 ± 5.12
MLP-R-nonlinear	25.93 ± 1.06	54.94 ± 11.63	86.29 ± 0.32	75.66 ± 1.07	72.26 ± 2.41	9.82 ± 2.88	10.26 ± 0.68	75.21 ± 1.19	15.90 ± 3.86	53.46 ± 3.45	79.74 ± 9.25
MLP-R-linear	46.28 ± 4.86	52.67 ± 11.82	85.93 ± 0.33	68.00 ± 3.94	71.88 ± 11.61	17.09 ± 3.27	10.86 ± 1.90	68.10 ± 2.53	18.42 ± 5.89	69.71 ± 8.34	81.85 ± 7.60
DNF-nonlinear	86.80 ± 1.96	71.48 ± 3.28	77.98 ± 8.57	75.83 ± 1.81	81.95 ± 1.90	61.96 ± 7.31	69.02 ± 5.54	$\mathbf{77.54 \pm 1.55}$	87.12 ± 8.71	NAN	93.28 ± 0.58
DNF-linear	85.90 ± 1.66	69.53 ± 3.70	68.49 ± 7.34	74.92 ± 1.26	82.42 ± 2.22	50.42 ± 14.88	55.33 ± 19.51	76.76 ± 1.47	88.85 ± 5.72	NAN	93.42 ± 0.50
CNN1-R-nonlinear	86.51 ± 1.79	72.85 ± 2.51	86.37 ± 0.28	76.04 ± 0.90	81.13 ± 2.14	72.06 ± 1.18	77.05 ± 1.16	77.01 ± 1.49	95.51 ± 0.84	79.52 ± 3.30	93.92 ± 0.39
CNN1-R-linear	86.92 ± 2.06	72.79 ± 2.72	86.15 ± 0.26	75.66 ± 1.07	80.90 ± 2.77	73.10 ± 0.77	77.83 ± 0.74	77.03 ± 1.03	96.08 ± 0.59	79.52 ± 4.24	94.08 ± 0.25
CNN2-R-nonlinear	86.79 ± 1.37	73.02 ± 2.98	86.41 ± 0.34	$\mathbf{76.40 \pm 0.85}$	81.95 ± 2.67	73.01 ± 0.66	76.92 ± 0.78	77.11 ± 1.17	96.20 ± 0.46	80.00 ± 3.41	94.26 ± 0.27
CNN2-R-linear	86.96 ± 1.85	72.73 ± 2.90	86.53 ± 0.27	74.12 ± 1.00	80.75 ± 3.62	73.16 ± 0.45	78.04 ± 0.68	76.74 ± 1.27	$\mathbf{96.26 \pm 0.60}$	$\mathbf{80.29 \pm 3.80}$	94.29 ± 0.44
LeNet5-nonlinear	85.13 ± 1.67	72.91 ± 3.05	86.73 ± 0.29	76.26 ± 1.02	82.03 ± 1.97	73.21 ± 0.93	76.64 ± 0.89	76.72 ± 1.30	96.16 ± 0.54	78.46 ± 2.89	93.97 ± 0.45
LeNet5-linear	86.44 ± 1.73	$\mathbf{73.60 \pm 2.98}$	$\mathbf{86.77 \pm 0.31}$	75.56 ± 0.91	$\mathbf{82.48 \pm 2.28}$	$\mathbf{73.61 \pm 0.65}$	77.26 ± 0.80	77.29 ± 1.16	96.16 ± 0.54	79.23 ± 3.78	$\mathbf{94.38 \pm 0.32}$
VGG16-nonlinear	82.40 ± 12.81	71.74 ± 5.06	85.85 ± 0.34	76.26 ± 1.07	80.45 ± 2.85	45.72 ± 24.27	55.19 ± 17.48	75.86 ± 3.29	86.77 ± 8.84	77.21 ± 9.24	90.19 ± 9.74
VGG16-linear	49.17 ± 3.20	61.92 ± 5.28	81.49 ± 8.47	70.02 ± 1.32	79.17 ± 2.56	25.32 ± 14.97	18.60 ± 10.74	73.05 ± 4.58	29.61 ± 18.86	65.38 ± 11.42	77.25 ± 16.88
k-NN-G-geodesic	79.01 ± 1.46	63.02 ± 3.46	84.29 ± 0.81	71.94 ± 1.73	76.54 ± 2.37	68.08 ± 1.08	73.63 ± 1.17	70.39 ± 1.79	94.73 ± 0.98	67.12 ± 3.74	90.90 ± 1.70
k-NN-G-bc	78.08 ± 1.59	62.73 ± 3.44	84.84 ± 0.77	71.38 ± 1.10	76.39 ± 1.58	68.65 ± 1.07	72.32 ± 0.82	71.09 ± 1.61	93.81 ± 0.78	68.65 ± 4.30	91.00 ± 0.66
k-NN-G-projection	79.01 ± 1.72	65.00 ± 3.51	84.00 ± 1.11	70.98 ± 1.67	76.77 ± 2.72	68.70 ± 1.02	73.84 ± 1.04	70.23 ± 1.75	94.15 ± 0.86	67.40 ± 6.19	90.78 ± 0.90

Table 9: Learning classifier interactions (advanced experiments): means and standard deviations of prediction accuracy (shown in %) for each method on the benchmark datasets. A number of trees in ETs is equal to 100 and its depth is equal to 2.

Method/Dataset	Balance	Bupa	Gamma	German	Heart	Mfeat-mor	Mfeat-zer	Pima	Segment	Sonar	Spambase
ETs	83.11 ± 2.28	58.84 ± 2.65	71.08 ± 0.39	70.54 ± 1.65	79.02 ± 2.61	65.49 ± 2.42	61.01 ± 3.26	66.43 ± 1.48	76.30 ± 6.07	75.10 ± 2.25	66.10 ± 1.84
SVM	$\mathbf{89.29 \pm 0.96}$	59.07 ± 1.75	65.45 ± 0.40	70.10 ± 1.16	79.02 ± 2.61	19.73 ± 1.77	9.47 ± 2.20	65.76 ± 0.75	41.43 ± 3.13	54.04 ± 2.85	80.92 ± 0.45
SVM-stacking	85.51 ± 1.66	58.26 ± 2.01	80.79 ± 0.89	71.98 ± 2.20	79.02 ± 2.61	57.98 ± 1.84	38.84 ± 3.95	75.78 ± 1.62	71.90 ± 5.64	75.87 ± 2.96	89.88 ± 0.64
SVM-R-nonlinear	85.77 ± 1.76	58.20 ± 1.97	81.01 ± 0.87	73.50 ± 1.80	79.02 ± 2.61	28.78 ± 5.64	14.66 ± 2.21	76.22 ± 1.54	65.00 ± 6.99	75.38 ± 2.48	90.40 ± 0.61
SVM-R-linear	66.35 ± 6.75	58.31 ± 1.95	77.07 ± 1.62	70.00 ± 1.30	79.02 ± 2.61	69.18 ± 0.79	70.51 ± 1.27	65.70 ± 0.74	84.21 ± 3.94	64.81 ± 4.35	79.25 ± 3.70
k-NN	80.16 ± 1.80	63.90 ± 2.27	79.58 ± 0.35	67.20 ± 0.68	74.66 ± 2.56	39.02 ± 1.20	$\mathbf{80.82 \pm 0.88}$	71.06 ± 1.81	91.72 ± 0.84	75.87 ± 4.22	78.77 ± 0.71
k-NN-stacking	79.52 ± 1.39	64.30 ± 4.97	81.27 ± 0.70	71.74 ± 1.68	77.97 ± 2.58	67.18 ± 1.55	71.76 ± 0.79	71.67 ± 1.98	82.57 ± 5.64	78.75 ± 2.67	86.55 ± 0.80
k-NN-R-nonlinear	79.78 ± 1.33	64.53 ± 4.97	81.30 ± 0.67	71.94 ± 1.69	78.57 ± 2.76	67.07 ± 1.41	70.33 ± 0.91	71.54 ± 1.83	82.35 ± 5.35	78.85 ± 2.72	87.48 ± 1.66
k-NN-R-linear	80.19 ± 1.40	63.08 ± 2.46	81.23 ± 0.72	71.12 ± 1.63	75.49 ± 2.74	66.69 ± 1.14	70.88 ± 1.54	72.11 ± 1.91	88.54 ± 1.95	73.75 ± 4.53	90.87 ± 0.86
MLP	47.47 ± 2.61	44.83 ± 8.44	83.10 ± 0.61	72.24 ± 1.47	45.49 ± 23.88	17.31 ± 4.50	10.59 ± 1.17	73.62 ± 2.40	15.74 ± 4.36	47.98 ± 3.11	77.59 ± 5.29
MLP-stacking	46.57 ± 1.67	44.30 ± 6.33	83.66 ± 0.47	73.72 ± 1.46	22.18 ± 3.85	11.42 ± 2.84	10.55 ± 0.60	73.44 ± 3.88	14.64 ± 0.40	48.75 ± 2.36	92.08 ± 0.56
MLP-R-nonlinear	46.31 ± 2.05	46.57 ± 9.34	83.54 ± 0.81	73.32 ± 1.58	27.14 ± 18.22	10.69 ± 1.48	10.43 ± 1.12	72.92 ± 5.07	18.62 ± 5.07	53.56 ± 10.19	91.38 ± 0.58
MLP-R-linear	45.93 ± 2.12	44.88 ± 7.46	83.58 ± 0.88	56.24 ± 9.84	23.53 ± 4.68	10.70 ± 1.02	10.55 ± 0.87	72.99 ± 3.20	20.31 ± 6.56	46.92 ± 1.96	91.80 ± 0.83
DNF-nonlinear	85.39 ± 3.08	65.47 ± 3.81	63.01 ± 9.79	76.02 ± 1.38	81.72 ± 2.99	52.37 ± 7.21	66.08 ± 3.98	76.02 ± 2.40	84.97 ± 4.18	NAN	83.73 ± 2.29
DNF-linear	68.36 ± 17.42	63.36 ± 4.63	63.43 ± 9.90	75.03 ± 1.82	80.47 ± 3.42	43.01 ± 13.07	53.18 ± 25.25	75.35 ± 2.25	87.65 ± 7.71	NAN	84.34 ± 2.47
CNN1-R-nonlinear	87.69 ± 1.24	67.09 ± 2.86	84.05 ± 0.75	75.60 ± 0.88	80.68 ± 2.99	71.79 ± 1.33	74.20 ± 1.25	77.32 ± 1.20	86.83 ± 5.70	78.17 ± 3.13	92.45 ± 0.68
CNN1-R-linear	84.97 ± 0.66	63.78 ± 3.13	84.17 ± 0.44	73.72 ± 1.39	80.98 ± 2.10	72.98 ± 0.75	76.45 ± 1.22	75.89 ± 1.12	90.21 ± 1.53	74.81 ± 3.24	92.60 ± 0.81
CNN2-R-nonlinear	88.88 ± 1.54	69.19 ± 3.99	84.33 ± 0.30	$\mathbf{76.30 \pm 1.29}$	82.26 ± 2.63	72.96 ± 1.05	74.83 ± 1.17	$\mathbf{77.63 \pm 1.42}$	95.65 ± 0.90	$\mathbf{80.10 \pm 1.97}$	$\mathbf{93.03 \pm 0.78}$
CNN2-R-linear	86.31 ± 0.88	65.87 ± 4.21	$\mathbf{84.40 \pm 0.46}$	75.00 ± 1.19	81.50 ± 2.21	$\mathbf{73.28 \pm 1.02}$	76.89 ± 1.11	76.28 ± 1.80	$\mathbf{95.80 \pm 0.78}$	74.04 ± 3.89	92.92 ± 1.03
LeNet5-nonlinear	89.17 ± 1.26	$\mathbf{70.70 \pm 3.07}$	84.33 ± 0.44	75.98 ± 2.01	80.00 ± 3.26	71.78 ± 0.99	74.22 ± 1.31	77.47 ± 2.31	94.40 ± 1.20	79.33 ± 1.94	89.41 ± 9.80
LeNet5-linear	86.12 ± 1.36	66.10 ± 4.09	83.93 ± 0.61	75.56 ± 1.23	$\mathbf{82.48 \pm 2.58}$	73.13 ± 1.03	76.65 ± 1.21	76.93 ± 1.68	95.06 ± 1.06	74.13 ± 2.83	92.55 ± 1.05
VGG16-nonlinear	82.79 ± 5.62	58.26 ± 2.01	83.49 ± 0.61	72.40 ± 2.77	79.02 ± 2.61	18.15 ± 3.43	20.64 ± 18.18	71.56 ± 4.46	51.06 ± 16.70	69.52 ± 9.26	85.95 ± 12.82
VGG16-linear	61.60 ± 11.37	58.26 ± 2.01	76.21 ± 8.83	70.18 ± 1.45	79.02 ± 2.61	35.45 ± 1.77	14.66 ± 4.45	68.39 ± 2.94	24.40 ± 12.22	59.52 ± 6.21	65.80 ± 8.17
k-NN-G-geodesic	81.44 ± 1.33	63.08 ± 3.82	81.16 ± 1.01	70.14 ± 1.49	77.67 ± 1.75	67.83 ± 1.51	75.45 ± 1.47	72.06 ± 1.55	94.36 ± 0.88	76.15 ± 3.03	88.20 ± 1.09
k-NN-G-bc	81.76 ± 1.20	62.67 ± 3.84	81.23 ± 0.80	70.64 ± 1.72	75.94 ± 4.15	68.24 ± 1.39	74.44 ± 1.48	71.93 ± 0.96	93.86 ± 1.77	74.52 ± 3.85	86.63 ± 1.64
k-NN-G-projection	80.83 ± 1.33	63.20 ± 3.92	80.97 ± 1.59	70.90 ± 1.91	78.12 ± 2.59	69.00 ± 1.46	75.85 ± 1.21	72.14 ± 1.46	94.53 ± 1.36	73.56 ± 3.20	86.80 ± 1.06

as RFs, ETs or classifier stacking when meta-learner is SVM (with a radial basis function (RBF)), kNN ($k = 3$) and MLP classifiers (MLP has a pyramidal structure with doubling the number of units on the next layer, the total number of layers is equal to 8). This confirms that the proposed approach counts for the geometry of data. The optimal depth of a DT in the case of G-manifolds is equal to 5. For R-manifolds using trees with a depth which is more than 10 is not reasonable. In the case of G-manifolds when depth is equal to 5 it is observed a quasi-monotonic growth of accuracy when augmenting the number of trees in the classifier ensemble both for RFs and ETs. For depth of DTs equal to 5 and multi-class classification problem G-manifolds perform similarly to R-manifolds of SPD matrices. Augmenting a depth of a tree leads to flattening both manifolds. This can be seen as diminishing the gap between dependencies that correspond to manifolds and those that correspond to E-space. For G-

Table 10: Learning classifier interactions (advanced experiments): means and standard deviations of prediction accuracy (shown in %) for each method on the benchmark datasets. A number of trees in ETs is equal to 100 and its depth is equal to 5.

Method/Dataset	Balance	Bupa	Gamma	German	Heart	Mfeat-mor	Mfeat-zer	Pima	Segment	Sonar	Spambase
ETs	86.89 ± 2.08	63.60 ± 4.44	78.68 ± 0.50	72.34 ± 1.44	79.02 ± 2.80	69.35 ± 0.76	74.78 ± 1.66	74.14 ± 1.41	90.37 ± 1.54	79.42 ± 2.62	82.43 ± 1.33
SVM	**89.29 ± 0.96**	59.07 ± 1.75	65.45 ± 0.40	70.10 ± 1.16	79.02 ± 2.61	19.73 ± 1.77	9.47 ± 2.20	65.76 ± 0.75	41.43 ± 3.13	54.04 ± 2.85	80.92 ± 0.45
SVM-stacking	87.12 ± 1.65	68.55 ± 3.77	84.60 ± 0.53	74.70 ± 1.55	81.73 ± 2.33	69.58 ± 0.60	73.32 ± 0.80	76.28 ± 1.84	92.00 ± 1.27	82.12 ± 1.78	92.37 ± 0.45
SVM-R-nonlinear	86.70 ± 1.99	69.19 ± 3.86	79.32 ± 0.71	75.08 ± 1.48	81.80 ± 1.98	65.79 ± 2.33	39.13 ± 3.91	76.30 ± 1.58	90.20 ± 1.22	82.12 ± 1.98	92.46 ± 0.50
SVM-R-linear	86.70 ± 1.99	59.42 ± 3.84	84.31 ± 0.48	71.22 ± 1.41	79.55 ± 3.04	69.72 ± 0.62	78.19 ± 0.48	70.44 ± 1.23	93.14 ± 1.30	78.08 ± 2.71	85.94 ± 1.27
k-NN	80.16 ± 1.80	63.90 ± 2.27	79.58 ± 0.35	67.20 ± 0.68	74.66 ± 2.56	39.02 ± 1.20	**80.82 ± 0.88**	71.06 ± 1.81	91.72 ± 0.84	75.87 ± 4.22	78.77 ± 0.71
k-NN-stacking	81.06 ± 1.11	67.97 ± 3.27	83.74 ± 0.41	72.60 ± 1.11	80.08 ± 2.50	68.47 ± 1.02	75.95 ± 1.24	72.60 ± 0.99	95.61 ± 0.93	80.76 ± 2.94	91.01 ± 1.22
k-NN-R-nonlinear	81.19 ± 1.54	68.72 ± 3.09	83.66 ± 0.32	72.60 ± 1.58	81.50 ± 2.43	67.90 ± 1.10	70.61 ± 1.26	72.50 ± 1.35	95.29 ± 1.04	80.87 ± 2.56	91.68 ± 0.82
k-NN-R-linear	79.10 ± 1.91	64.13 ± 3.45	83.40 ± 0.46	69.62 ± 1.52	75.04 ± 2.12	67.74 ± 1.00	75.72 ± 0.74	68.57 ± 2.50	95.31 ± 0.97	79.33 ± 2.99	91.74 ± 0.56
MLP	47.47 ± 2.61	44.83 ± 8.44	83.10 ± 0.61	72.24 ± 1.47	45.49 ± 23.88	17.31 ± 4.50	10.59 ± 1.17	73.62 ± 2.40	15.74 ± 4.36	47.98 ± 3.11	77.59 ± 5.29
MLP-stacking	46.47 ± 1.84	44.13 ± 6.42	85.75 ± 0.41	75.00 ± 0.93	21.20 ± 2.77	10.27 ± 0.87	10.27 ± 0.68	75.78 ± 1.38	14.48 ± 0.47	55.38 ± 12.56	93.25 ± 0.59
MLP-R-nonlinear	46.76 ± 1.66	43.49 ± 4.22	85.40 ± 0.96	75.84 ± 1.30	29.92 ± 19.11	10.62 ± 1.10	11.43 ± 2.47	76.77 ± 0.98	15.83 ± 3.80	46.73 ± 1.62	93.21 ± 0.52
MLP-R-linear	46.67 ± 1.84	42.27 ± 2.60	85.58 ± 0.27	72.12 ± 1.98	28.27 ± 14.90	11.15 ± 1.73	10.53 ± 0.66	74.79 ± 1.41	16.45 ± 4.71	51.73 ± 9.68	93.00 ± 0.61
DNF-nonlinear	87.66 ± 1.27	70.31 ± 3.01	66.55 ± 12.51	**76.09 ± 1.25**	82.42 ± 3.32	57.60 ± 6.70	66.73 ± 5.88	**78.20 ± 1.61**	93.13 ± 4.27	*NAN*	88.93 ± 1.38
DNF-linear	86.95 ± 1.74	66.48 ± 3.05	70.38 ± 6.73	74.01 ± 1.93	82.11 ± 3.27	57.50 ± 6.83	55.77 ± 21.90	75.23 ± 1.76	90.52 ± 5.98	*NAN*	89.30 ± 1.36
CNN1-R-nonlinear	87.98 ± 2.31	**71.86 ± 3.01**	85.76 ± 0.41	74.74 ± 1.64	82.93 ± 2.65	72.18 ± 0.81	78.10 ± 0.74	77.11 ± 1.40	95.08 ± 0.87	**82.79 ± 1.94**	93.57 ± 0.54
CNN1-R-linear	87.44 ± 1.95	66.51 ± 2.62	85.52 ± 0.26	73.94 ± 1.08	81.88 ± 2.39	73.31 ± 0.62	78.71 ± 1.12	75.13 ± 1.70	96.76 ± 0.61	82.12 ± 1.83	93.38 ± 0.56
CNN2-R-nonlinear	87.98 ± 1.44	70.52 ± 2.80	85.90 ± 0.38	75.44 ± 0.74	**82.93 ± 2.38**	73.15 ± 0.65	78.48 ± 0.98	77.03 ± 1.62	96.21 ± 0.83	80.19 ± 3.11	**93.93 ± 0.53**
CNN2-R-linear	87.69 ± 1.54	66.98 ± 1.82	85.81 ± 0.28	74.28 ± 1.18	82.78 ± 2.78	73.49 ± 0.83	79.29 ± 0.86	75.29 ± 1.81	**97.11 ± 0.53**	81.06 ± 3.22	93.45 ± 0.39
LeNet5-nonlinear	87.24 ± 1.54	71.69 ± 3.53	**86.07 ± 0.29**	76.08 ± 1.11	80.00 ± 2.67	73.30 ± 0.76	78.31 ± 0.85	77.24 ± 1.65	96.20 ± 0.70	80.38 ± 3.42	93.58 ± 0.56
LeNet5-linear	86.92 ± 1.82	68.08 ± 2.33	86.07 ± 0.33	75.32 ± 1.02	82.48 ± 1.68	**73.61 ± 0.95**	78.63 ± 0.83	75.55 ± 1.33	96.84 ± 0.49	80.58 ± 2.31	93.46 ± 0.57
VGG16-nonlinear	80.71 ± 13.33	62.56 ± 5.70	84.72 ± 0.43	74.80 ± 2.62	79.62 ± 2.53	47.32 ± 16.31	53.29 ± 14.99	75.13 ± 4.87	68.49 ± 2.13	77.60 ± 9.13	80.27 ± 15.99
VGG16-linear	46.73 ± 3.10	58.26 ± 2.01	76.97 ± 9.44	70.28 ± 1.92	79.02 ± 2.61	19.55 ± 8.98	13.28 ± 4.80	67.89 ± 2.49	32.47 ± 22.34	71.25 ± 7.77	82.66 ± 14.56
k-NN-G-geodesic	79.48 ± 2.04	64.13 ± 2.65	83.18 ± 0.61	70.88 ± 1.21	75.34 ± 3.65	69.28 ± 0.68	77.31 ± 0.83	72.11 ± 2.23	95.75 ± 0.74	68.17 ± 4.09	90.80 ± 1.59
k-NN-G-bc	79.17 ± 1.71	63.55 ± 2.11	83.76 ± 0.96	70.54 ± 1.23	74.74 ± 3.23	68.80 ± 0.96	74.73 ± 0.63	72.89 ± 1.64	95.67 ± 0.85	65.67 ± 4.43	90.05 ± 1.51
k-NN-G-projection	80.03 ± 1.91	64.13 ± 2.01	83.27 ± 0.97	70.84 ± 1.32	74.29 ± 2.85	69.27 ± 0.57	77.06 ± 1.17	72.11 ± 1.85	95.45 ± 0.81	65.87 ± 4.68	89.45 ± 1.07

manifolds using deeper trees leads to a decrease in accuracy. That is why using trees with a depth of more than 5 is not recommended.

Using the results from Tabs. 7-10 we can conclude that exploiting the nonlinearity of manifolds gives a significant improvement in classification accuracy for four datasets: "bupa", "mfeat-mor", "mfeat-zer" and "segmentation" (except k-NN that uses an original set of features when predictions are generated by trees with a depth equal to two.). Using nonlinear manifolds and interactions between classifier predictions generated by RFs and ETs to classify instances from these datasets, gives better classification results (especially in the case of CNNs) than all other classifiers with rather a significant margin. This is especially noticeable for lower depths of trees in RFs and ETs. For "balance" and "mfeat-zer" datasets SVM and k-NN classifiers perform the best for the original sets of features. For all other datasets, nonlinear geometry creates a lower improvement margin or statistically less significant.

As additional classifiers we applied DNFs [44] to classify patterns presented as $(0, 3)$ tensors. In their work authors claim that their approach to train RFs gives very promising results. The results from Tables 7-10 show that for some datasets ("german", "heart" and "pima") DNFs produce comparable to CNNs results. For other datasets the training of DNFs is not stable. As a result for "sonar" dataset the algorithm does not work at all producing "NAN" values instead of prediction probabilities. We also noticed that it works the best for the size of $(0, 2)$ tensor equal to 28×28 as for grayscale images from MNIST dataset. This makes us thinking that the algorithm was optimised specifically for this dataset and does not work well for other datasets, e.g. for CIFAR-10 and CIFAR-100 datasets.

Analysing the results from Figures 30-32 allows to make a conclusion that E-geometry simply cannot handle very difficult classification problems which encourages to use a geometry other than Euclidean one. Classifier predictions are new data generated by a classifier ensemble. They can be used to compare different classifiers as well. Thus an initial dataset can be doubled or even tripled when using classifier interactions. This makes the entire set of experiments much more informative and richer. If the accuracy in E-geometry is similar

to the one obtained on nonlinear manifolds then it can mean that the data lie on a flat manifold. However, it is not known if classes can be separated better on nonlinear manifolds or embedded in E-space manifold is nonlinear enough. This all depends on different factors aforementioned before. Creating a manifold is an artificial process meaning some kind of data transformation and interpretation appropriately. Indeed, nonlinearity can be created due to some interaction between different instances. R-manifolds of SPD matrices are created using interactions between the predictions by every classifier (DT) in an ensemble for every class. G-manifolds obtained using DPs $DP(x)$ and can be interpreted as a nonlinear version of them. The approach can be extended to obtain a nonlinear version of decision templates [88] too.

We carried out some more experiments comparing different CNN architectures and different CRF both in E-and-R-geometry (see Tabs.11-12). As can be seen from aforementioned tables all CNN architectures except of VGG16 perform rather similar and the priority cannot be given to any of them. Moreover aforementioned architectures are computationally less consuming than VGG16. The difference varies from 3 to 6 times. Also, considering CRF in E-and-R-geometry one can notice that they perform similarly for some datasets as CNN architectures. Overall, using CCE leads to improving both accuracy and logarithmic loss for both geometries. This is important because there are some problems they require real-time processing and this would be possible only with RFs (ETs) and their cascades. It can be also seen that applying cascaded G-manifolds (see homotopy diagram in Figure 26) is advantageous for some datasets and leads to improving the classification results. In this case we used a geodesic's length as a distance measure between two points on a G-manifold. Because cascaded G-manifolds are the most computationally demanding architectures, we excluded two the largest datasets("spambase" and "gamma") from the experiments, so the results from these datasets are not in Tab. 11

Lastly, we wanted to know if involving more classifiers in classifier interactions can create better classification patterns. So we carried out some experiments on 4D tensors that correspond to interactions between 3 classifiers in classifier subensembles. To this end, we applied 3DCNNs. The experiments showed some degradation in classification accuracy. We think that this is because of the structure of $(0, 4)$ tensor that contains many zeros or very small values. Increasing the number of predictors or classifiers leads to larger CPPMs which are poor conditioned. For those matrices, the larger is the size the poorer is their conditioning. For that reason we did not go for higher dimensional tensors in our experiments. Using eigenvalues of CPPMs as new features and compressing vectors obtained after vectorizing original CPPMs and those, obtained using logarithmic maps by PCA and autoencoders also did not improve the classification results. For that reason we do not publish the results from all these experiments. We also cannot compute vector of nonzero elements of CS because SPD matrices become singular with growth their sizes.

In Table 13 there are some results of classification accuracy for different approaches to classifier ensemble learning. Among the 4 presented approaches, 3 of them are within 10 last years. All these algorithms were tested on different benchmark datasets, so for some algorithms, there are no results for certain datasets. As seen from the aforementioned Table for some algorithms no standard deviation is provided which is required by the experimental protocol for such kinds of tests. Also, there is no description of the experimental protocol, so it is rather hard to guess, how these results have been obtained. Due to the aforementioned

Table 11: Learning classifier predictions using different data transformation procedures: means and standard deviations of prediction accuracy (shown in %) for each method on the benchmark datasets.

Method/Dataset	Balance	Bupa	German	Heart	Mfeat-mor	Mfeat-zer	Pima	Segment	Sonar
A number of DTs in a RFs is equal to 100 and its depth is equal to 2									
RFs_1	80.35 ± 2.96	67.44 ± 3.47	70.54 ± 1.65	79.10 ± 2.89	62.24 ± 2.87	58.06 ± 3.27	74.84 ± 1.44	75.55 ± 6.06	75.29 ± 3.65
RFs_{max}	81.89 ± 1.88	68.90 ± 1.81	74.10 ± 1.84	80.53 ± 2.16	–	–	75.62 ± 1.24	–	76.35 ± 4.20
$RFs - nonlinear_{max}$	82.88 ± 2.06	70.17 ± 2.28	74.50 ± 1.95	81.43 ± 2.52	45.98 ± 5.81	48.00 ± 6.38	75.47 ± 1.23	63.54 ± 10.34	77.40 ± 3.05
$RFs - linear_{max}$	83.91 ± 1.58	70.12 ± 3.89	75.30 ± 1.43	80.98 ± 2.28	46.06 ± 5.96	48.40 ± 7.16	75.52 ± 1.32	68.97 ± 1.32	76.73 ± 3.72
$CNN - R - nonlinear$	$\mathbf{85.61 \pm 2.14}$	$\mathbf{72.09 \pm 3.81}$	$\mathbf{75.60 \pm 0.88}$	$\mathbf{82.71 \pm 3.33}$	69.30 ± 1.30	70.58 ± 0.97	$\mathbf{77.19 \pm 1.13}$	87.63 ± 5.27	$\mathbf{78.65 \pm 2.71}$
$CNN - R - linear$	85.48 ± 1.98	68.55 ± 2.54	73.72 ± 1.39	82.26 ± 2.41	$\mathbf{70.41 \pm 1.26}$	$\mathbf{72.67 \pm 1.28}$	75.39 ± 1.57	$\mathbf{89.86 \pm 2.74}$	76.44 ± 3.17
GC_{max}	80.74 ± 1.75	66.80 ± 2.18	72.02 ± 1.57	77.67 ± 3.01	58.90 ± 1.22	69.34 ± 1.27	73.05 ± 1.84	85.28 ± 3.96	74.04 ± 4.17
$GC - nonlinear_{max}$	80.45 ± 1.49	67.15 ± 4.56	72.60 ± 1.68	78.65 ± 3.66	37.04 ± 4.76	46.06 ± 9.64	73.46 ± 2.96	58.72 ± 4.35	69.33 ± 4.64
$GC - linear_{max}$	81.99 ± 1.63	67.91 ± 3.65	72.50 ± 1.26	79.32 ± 4.09	36.84 ± 4.83	48.72 ± 9.41	72.60 ± 1.35	64.55 ± 5.43	67.69 ± 4.53
Number of DTs in a RFs is equal to 100 and its depth is equal to 5									
RFs_1	85.83 ± 1.79	70.81 ± 3.58	73.04 ± 1.13	81.05 ± 1.67	69.15 ± 0.97	73.97 ± 0.73	76.41 ± 1.61	92.40 ± 1.17	78.27 ± 4.33
RFs_{max}	–	–	75.10 ± 0.89	–	–	–	–	92.88 ± 2.50	78.65 ± 4.16
$RFs - nonlinear_{max}$	82.63 ± 1.37	69.30 ± 3.03	75.42 ± 1.10	81.13 ± 2.34	67.78 ± 2.29	72.20 ± 1.72	75.91 ± 1.44	94.15 ± 1.52	79.52 ± 2.28
$RFs - linear_{max}$	81.76 ± 1.08	69.65 ± 2.83	74.76 ± 1.59	80.90 ± 2.02	67.88 ± 1.30	71.44 ± 2.51	75.31 ± 1.21	94.06 ± 1.50	$\mathbf{79.62 \pm 3.62}$
$CNN - R - nonlinear$	86.51 ± 1.79	$\mathbf{72.85 \pm 2.51}$	$\mathbf{76.04 \pm 0.90}$	$\mathbf{81.13 \pm 2.14}$	72.06 ± 1.18	77.05 ± 1.16	77.01 ± 1.49	95.51 ± 0.84	79.52 ± 3.30
$CNN - R - linear$	$\mathbf{86.92 \pm 2.06}$	72.79 ± 2.72	75.66 ± 1.07	80.90 ± 2.77	$\mathbf{73.10 \pm 0.77}$	$\mathbf{77.83 \pm 0.74}$	$\mathbf{77.03 \pm 1.03}$	$\mathbf{96.08 \pm 0.59}$	79.52 ± 4.24
GC_{max}	79.23 ± 2.22	64.48 ± 3.16	70.92 ± 1.39	76.39 ± 3.56	68.50 ± 1.05	74.30 ± 0.63	71.87 ± 2.25	94.21 ± 0.88	67.50 ± 5.10
$GC - nonlinear_{max}$	75.99 ± 2.45	64.48 ± 3.16	70.26 ± 2.18	78.05 ± 3.64	60.32 ± 2.06	66.00 ± 1.17	73.07 ± 1.81	92.63 ± 1.13	63.27 ± 5.30
$GC - linear_{max}$	73.56 ± 2.45	62.33 ± 4.53	70.56 ± 1.79	77.52 ± 3.97	61.36 ± 2.08	67.94 ± 2.27	70.03 ± 2.92	90.61 ± 4.54	67.21 ± 4.87
A number of DTs in a ETs is equal to 100 and its depth is equal to 2									
ETs_1	83.11 ± 2.28	58.84 ± 2.65	70.54 ± 1.65	79.02 ± 2.61	65.49 ± 2.42	61.01 ± 3.26	66.43 ± 1.48	76.30 ± 6.07	75.10 ± 2.25
ETs_{max}	–	65.81 ± 2.31	73.72 ± 1.68	–	–	–	75.52 ± 1.08	–	76.44 ± 4.43
$ETs - nonlinear_{max}$	84.26 ± 2.38	66.80 ± 2.89	74.20 ± 1.42	80.83 ± 3.21	31.52 ± 5.72	45.68 ± 8.96	76.04 ± 1.27	47.12 ± 7.42	76.92 ± 4.55
$ETs - linear_{max}$	82.60 ± 0.87	66.16 ± 2.62	74.46 ± 1.95	$\mathbf{81.35 \pm 1.95}$	31.74 ± 5.62	55.62 ± 4.28	75.89 ± 1.52	49.45 ± 7.11	77.12 ± 4.38
$CNN - R - nonlinear$	$\mathbf{87.69 \pm 1.24}$	$\mathbf{67.09 \pm 2.86}$	$\mathbf{75.60 \pm 0.88}$	80.68 ± 2.99	71.79 ± 1.33	74.20 ± 1.25	$\mathbf{77.32 \pm 1.20}$	86.83 ± 5.70	$\mathbf{78.17 \pm 3.13}$
$CNN - R - linear$	84.97 ± 0.66	63.78 ± 3.13	73.72 ± 1.39	80.98 ± 2.10	$\mathbf{72.98 \pm 0.75}$	76.45 ± 1.22	75.89 ± 1.12	90.21 ± 1.53	74.81 ± 3.24
GC_{max}	81.03 ± 1.72	65.00 ± 4.11	71.88 ± 2.844	77.07 ± 3.10	67.70 ± 1.33	75.58 ± 1.38	75.89 ± 1.52	$\mathbf{94.42 \pm 0.72}$	76.44 ± 4.33
$GC - nonlinear_{max}$	80.93 ± 2.98	65.23 ± 4.19	73.44 ± 1.25	79.17 ± 3.23	30.84 ± 4.75	56.80 ± 4.42	72.40 ± 3.60	50.70 ± 7.08	71.83 ± 5.20
$GC - linear_{max}$	81.28 ± 2.01	65.06 ± 1.85	72.34 ± 2.73	77.52 ± 3.34	32.76 ± 5.63	66.52 ± 0.62	73.07 ± 2.81	55.98 ± 9.16	73.65 ± 4.60
Number of DTs in a ETs is equal to 100 and its depth is equal to 5									
ETs_1	86.89 ± 2.08	63.60 ± 4.44	72.34 ± 1.44	79.02 ± 2.80	69.35 ± 0.76	74.78 ± 1.66	74.14 ± 1.41	90.37 ± 1.54	79.42 ± 2.62
ETs_{max}	–	68.78 ± 3.96	74.62 ± 1.26	81.05 ± 2.81	–	–	76.43 ± 1.61	91.85 ± 0.76	79.52 ± 2.89
$ETs - nonlinear_{max}$	83.08 ± 1.22	67.85 ± 4.04	74.64 ± 1.05	81.43 ± 2.63	63.46 ± 3.85	70.98 ± 1.17	75.52 ± 1.56	84.76 ± 2.47	80.58 ± 2.78
$ETs - linear_{max}$	83.62 ± 1.02	67.33 ± 4.15	73.60 ± 1.38	80.75 ± 2.92	63.86 ± 4.81	72.48 ± 0.99	76.20 ± 1.73	91.54 ± 1.22	79.42 ± 5.04
$CNN - R - nonlinear$	$\mathbf{87.98 \pm 2.31}$	$\mathbf{71.86 \pm 3.01}$	$\mathbf{74.74 \pm 1.64}$	$\mathbf{82.93 \pm 2.65}$	72.18 ± 0.81	78.10 ± 0.74	$\mathbf{77.11 \pm 1.40}$	95.08 ± 0.87	$\mathbf{82.79 \pm 1.94}$
$CNN - R - linear$	87.44 ± 1.95	66.51 ± 2.62	73.94 ± 1.08	81.88 ± 2.39	$\mathbf{73.31 \pm 0.62}$	$\mathbf{78.71 \pm 1.12}$	75.13 ± 1.70	$\mathbf{96.76 \pm 0.61}$	82.12 ± 1.83
GC_{max}	79.78 ± 1.53	64.13 ± 2.69	72.44 ± 1.62	76.84 ± 2.20	69.48 ± 1.18	76.48 ± 0.81	72.16 ± 1.86	95.54 ± 0.78	66.54 ± 4.53
$GC - nonlinear_{max}$	72.50 ± 2.39	61.05 ± 3.11	70.76 ± 2.09	76.24 ± 2.99	55.94 ± 3.00	68.16 ± 1.40	63.70 ± 2.28	89.89 ± 2.80	57.12 ± 5.00
$GC - linear_{max}$	64.39 ± 3.83	60.52 ± 3.42	70.74 ± 3.63	77.52 ± 3.67	63.68 ± 2.54	68.78 ± 1.64	65.68 ± 6.75	93.60 ± 1.10	57.79 ± 6.28

circumstances, comparing the results is rather unfair. However, even in these conditions, we can see that our best algorithm wins in half of the cases and in the prevailing number of cases it is a first-or-second place winner. Here we have very high competition between the algorithms. Every approach, including ours, involves at least 10 different algorithms. Also, in a competition many general well-known classifiers or classifier ensembles have been engaged. Thus, the total number of algorithms (including their modifications) involved in a competition is between 50 and 100. One of the principal objectives of this research is not to build the best overall classification algorithm, but to show and emphasize that for some classification problems applying geometric learning of classifier ensembles by nonlinear manifolds is necessary and justified. For our experiments, we have selected datasets from UCI repository [22] to be rather difficult, which means that applying nonlinear geometry might be potentially reasonable. Half of the problems are binary, and the other half are multi-class problems. Other researchers select the datasets for their experiments, having their own reasons. The results in Table 13 show also the overlaps between used datasets.

We are going to provide computational complexity for most of the algorithms that have been applied during the experiments. The complexities will be calculated both for training and testing. The space complexities will not be provided at this point. Below we explain the notations used to compute aforementioned complexities:

Table 12: Learning classifier predictions using different data transformation procedures: means and standard deviations of prediction of validation logarithmic loss for each method on the benchmark datasets.

Method/Dataset	Balance	Bupa	Gamma	German	Heart	Mfeat-mor	Mfeat-zer	Pima	Segment	Sonar	Spambase
A number of DTs in RF is equal to 100 and its depth is equal to 2											
RF_1	0.691 ± 0.036	0.613 ± 0.027	0.493 ± 0.005	0.555 ± 0.011	0.415 ± 0.027	1.267 ± 0.041	1.660 ± 0.022	0.527 ± 0.004	1.038 ± 0.042	0.534 ± 0.013	0.389 ± 0.006
RF_{min}	0.477 ± 0.049	–	0.434 ± 0.004	0.520 ± 0.016	0.404 ± 0.039	–	–	0.499 ± 0.011	1.032 ± 0.035	0.483 ± 0.065	0.237 ± 0.010
$RF - nonlinear_{min}$	0.421 ± 0.046	0.593 ± 0.032	0.409 ± 0.005	0.514 ± 0.014	0.407 ± 0.048	1.472 ± 0.068	1.727 ± 0.051	0.491 ± 0.012	1.030 ± 0.044	**0.458 ± 0.045**	0.219 ± 0.010
$RF - linear_{min}$	0.432 ± 0.059	0.601 ± 0.040	0.409 ± 0.005	0.515 ± 0.014	0.429 ± 0.052	1.472 ± 0.068	1.715 ± 0.046	0.497 ± 0.012	1.022 ± 0.042	0.475 ± 0.045	0.225 ± 0.013
$CNN1 - R - nonlinear$	0.398 ± 0.090	0.597 ± 0.030	0.380 ± 0.007	**0.504 ± 0.009**	0.431 ± 0.041	0.731 ± 0.034	0.853 ± 0.050	**0.488 ± 0.011**	0.263 ± 0.041	0.516 ± 0.040	0.202 ± 0.011
$CNN1 - R - linear$	0.426 ± 0.109	0.640 ± 0.019	0.382 ± 0.008	0.546 ± 0.006	0.411 ± 0.029	0.716 ± 0.029	0.753 ± 0.047	0.530 ± 0.014	**0.249 ± 0.039**	0.523 ± 0.024	0.189 ± 0.006
CNN2-R-nonlinear	0.378 ± 0.088	**0.589 ± 0.042**	0.378 ± 0.007	0.505 ± 0.013	0.407 ± 0.050	0.733 ± 0.026	0.849 ± 0.062	0.489 ± 0.012	0.273 ± 0.053	0.507 ± 0.041	0.190 ± 0.010
CNN2-R-linear	0.403 ± 0.068	0.636 ± 0.021	0.379 ± 0.008	0.546 ± 0.020	0.406 ± 0.038	0.723 ± 0.026	0.765 ± 0.060	0.519 ± 0.016	0.261 ± 0.048	0.517 ± 0.042	0.188 ± 0.011
LeNet5-nonlinear	**0.357 ± 0.078**	0.603 ± 0.049	**0.374 ± 0.008**	0.504 ± 0.015	0.476 ± 0.101	0.731 ± 0.018	0.792 ± 0.049	0.490 ± 0.012	0.282 ± 0.050	0.641 ± 0.131	0.188 ± 0.014
LeNet5-linear	0.403 ± 0.070	0.613 ± 0.029	0.376 ± 0.006	0.524 ± 0.015	**0.397 ± 0.038**	**0.715 ± 0.021**	**0.717 ± 0.050**	0.489 ± 0.015	0.275 ± 0.051	0.491 ± 0.055	**0.183 ± 0.009**
VGG16-nonlinear	0.548 ± 0.154	0.652 ± 0.020	0.386 ± 0.007	0.537 ± 0.034	0.527 ± 0.080	1.399 ± 0.520	1.858 ± 0.406	0.514 ± 0.021	1.384 ± 0.373	0.614 ± 0.038	1.154 ± 2.816
VGG16-linear	0.883 ± 0.045	0.670 ± 0.014	0.402 ± 0.020	0.588 ± 0.025	0.514 ± 0.041	1.773 ± 0.492	2.280 ± 0.031	0.607 ± 0.043	1.841 ± 0.213	0.666 ± 0.007	0.413 ± 0.204
Number of DTs in a RF is equal to 100 and its depth is equal to 5											
RF_1	0.493 ± 0.044	0.592 ± 0.016	0.401 ± 0.002	0.516 ± 0.010	**0.394 ± 0.028**	0.741 ± 0.026	1.019 ± 0.017	**0.483 ± 0.009**	0.350 ± 0.024	0.466 ± 0.016	0.259 ± 0.004
RF_{min}	0.406 ± 0.038	–	0.335 ± 0.006	0.511 ± 0.017	–	–	–	–	0.241 ± 0.054	0.433 ± 0.033	0.168 ± 0.008
$RF - nonlinear_{min}$	0.415 ± 0.071	0.659 ± 0.108	**0.334 ± 0.006**	0.527 ± 0.026	0.453 ± 0.103	0.940 ± 0.052	1.064 ± 0.030	0.536 ± 0.028	0.222 ± 0.045	**0.424 ± 0.030**	**0.166 ± 0.005**
$RF - linear_{min}$	0.424 ± 0.055	0.604 ± 0.041	0.337 ± 0.006	0.516 ± 0.017	0.411 ± 0.044	0.928 ± 0.049	1.099 ± 0.039	0.514 ± 0.020	0.239 ± 0.078	0.432 ± 0.030	0.169 ± 0.009
$CNN1 - R - nonlinear$	0.369 ± 0.049	0.643 ± 0.029	0.349 ± 0.008	0.530 ± 0.023	0.567 ± 0.089	0.723 ± 0.0263	0.759 ± 0.043	0.501 ± 0.022	0.131 ± 0.024	0.476 ± 0.033	0.183 ± 0.002
$CNN1 - R - linear$	0.387 ± 0.063	0.606 ± 0.018	0.352 ± 0.006	0.531 ± 0.012	2.16 ± 1.398	0.684 ± 0.027	0.619 ± 0.011	0.497 ± 0.010	**0.112 ± 0.016**	0.446 ± 0.034	0.176 ± 0.007
CNN2-R-nonlinear	**0.364 ± 0.060**	0.620 ± 0.085	0.378 ± 0.007	0.527 ± 0.024	0.472 ± 0.062	0.726 ± 0.032	0.735 ± 0.052	0.515 ± 0.032	0.131 ± 0.025	0.507 ± 0.108	0.184 ± 0.010
CNN2-R-linear	0.378 ± 0.065	**0.591 ± 0.034**	0.379 ± 0.008	0.519 ± 0.022	2.5101.434	0.683 ± 0.027	0.609 ± 0.026	0.503 ± 0.017	0.122 ± 0.024	1.928 ± 2.910	0.177 ± 0.010
LeNet5-nonlinear	0.415 ± 0.072	0.884 ± 0.180	0.374 ± 0.008	0.568 ± 0.062	1.701 ± 0.763	0.712 ± 0.031	0.679 ± 0.043	0.567 ± 0.026	0.127 ± 0.023	2.143 ± 0.824	0.182 ± 0.012
LeNet5-linear	0.368 ± 0.051	0.594 ± 0.032	0.376 ± 0.006	**0.510 ± 0.018**	0.425 ± 0.051	**0.673 ± 0.022**	**0.589 ± 0.027**	0.499 ± 0.022	0.122 ± 0.024	0.456 ± 0.049	0.166 ± 0.010
VGG16-nonlinear	0.479 ± 0.230	0.611 ± 0.033	0.386 ± 0.007	0.515 ± 0.013	0.500 ± 0.066	1.297 ± 0.538	1.088 ± 0.147	0.513 ± 0.045	0.396 ± 0.271	0.606 ± 0.055	1.747 ± 3.206
VGG16-linear	2.533 ± 3.268	0.652 ± 0.036	0.402 ± 0.020	0.585 ± 0.032	1.772 ± 1.573	2.098 ± 0.393	2.287 ± 0.020	0.560 ± 0.053	1.391 ± 0.671	0.667 ± 0.017	0.440 ± 0.229
A number of DTs in a ETs is equal to 100 and its depth is equal to 2											
RF_1	0.725 ± 0.032	0.669 ± 0.007	0.561 ± 0.005	0.565 ± 0.009	0.452 ± 0.028	1.453 ± 0.031	1.782 ± 0.013	0.584 ± 0.004	1.161 ± 0.0265	0.593 ± 0.009	0.590 ± 0.008
RF_{min}	0.472 ± 0.043	0.648 ± 0.015	0.434 ± 0.009	0.527 ± 0.016	0.405 ± 0.031	1.415 ± 0.142	1.692 ± 0.027	0.508 ± 0.017	1.084 ± 0.071	0.482 ± 0.051	0.300 ± 0.014
$RF - nonlinear_{min}$	0.414 ± 0.038	0.638 ± 0.029	0.420 ± 0.011	0.519 ± 0.018	0.404 ± 0.039	1.607 ± 0.138	1.718 ± 0.034	0.501 ± 0.021	1.139 ± 0.040	**0.439 ± 0.042**	0.285 ± 0.013
$RF - linear_{min}$	0.447 ± 0.037	0.632 ± 0.030	0.434 ± 0.009	0.522 ± 0.020	0.406 ± 0.035	1.577 ± 0.143	1.693 ± 0.020	0.498 ± 0.018	1.131 ± 0.048	0.483 ± 0.051	0.278 ± 0.016
$CNN1 - R - nonlinear$	**0.349 ± 0.060**	0.629 ± 0.017	0.371 ± 0.007	**0.495 ± 0.006**	0.388 ± 0.027	0.693 ± 0.030	0.746 ± 0.052	**0.477 ± 0.018**	0.179 ± 0.031	0.458 ± 0.060	0.220 ± 0.015
$CNN1 - R - linear$	0.508 ± 0.081	0.668 ± 0.011	0.373 ± 0.006	0.569 ± 0.014	0.400 ± 0.035	0.673 ± 0.019	0.649 ± 0.023	0.543 ± 0.027	0.153 ± 0.032	0.574 ± 0.032	0.240 ± 0.017
CNN2-R-nonlinear	0.369 ± 0.073	0.613 ± 0.038	0.369 ± 0.011	0.508 ± 0.017	**0.380 ± 0.039**	0.693 ± 0.026	0.779 ± 0.097	0.488 ± 0.012	0.143 ± 0.026	0.476 ± 0.048	0.222 ± 0.013
CNN2-R-linear	0.417 ± 0.071	0.664 ± 0.012	0.371 ± 0.010	0.539 ± 0.020	0.396 ± 0.049	0.670 ± 0.022	0.648 ± 0.029	0.551 ± 0.020	**0.136 ± 0.021**	0.564 ± 0.060	0.235 ± 0.019
LeNet5-nonlinear	0.355 ± 0.064	**0.588 ± 0.030**	**0.366 ± 0.010**	0.512 ± 0.017	0.779 ± 0.915	0.688 ± 0.025	0.680 ± 0.044	0.496 ± 0.045	0.172 ± 0.019	0.489 ± 0.062	**0.218 ± 0.013**
LeNet5-linear	0.397 ± 0.062	0.651 ± 0.018	0.369 ± 0.009	0.519 ± 0.017	0.382 ± 0.041	**0.665 ± 0.019**	**0.594 ± 0.023**	0.495 ± 0.015	0.145 ± 0.019	0.531 ± 0.071	0.228 ± 0.014
VGG16-nonlinear	0.559 ± 0.146	0.673 ± 0.008	0.392 ± 0.013	0.524 ± 0.022	0.569 ± 0.047	2.059 ± 0.098	2.055 ± 0.413	0.552 ± 0.028	2.613 ± 3.776	0.590 ± 0.057	0.364 ± 0.201
VGG16-linear	0.868 ± 0.083	0.675 ± 0.008	0.501 ± 0.118	0.591 ± 0.027	0.532 ± 0.060	1.933 ± 0.497	2.294 ± 0.007	0.617 ± 0.025	1.699 ± 0.433	0.666 ± 0.017	0.618 ± 0.126
Number of DTs in a ETs is equal to 100 and its depth is equal to 5											
RF_1	0.539 ± 0.041	0.642 ± 0.007	0.479 ± 0.007	0.527 ± 0.010	0.405 ± 0.026	0.874 ± 0.019	1.238 ± 0.014	0.530 ± 0.007	0.600 ± 0.015	0.488 ± 0.011	0.491 ± 0.010
RF_{max}	0.398 ± 0.035	**0.604 ± 0.021**	0.349 ± 0.008	**0.504 ± 0.013**	0.401 ± 0.044	–	1.101 ± 0.031	**0.485 ± 0.015**	0.307 ± 0.031	0.399 ± 0.017	0.197 ± 0.017
$RF - nonlinear_{max}$	0.392 ± 0.039	0.613 ± 0.041	0.345 ± 0.007	0.517 ± 0.014	0.425 ± 0.097	0.921 ± 0.040	1.119 ± 0.024	0.500 ± 0.018	0.305 ± 0.020	0.399 ± 0.012	**0.187 ± 0.014**
$RF - linear_{max}$	0.412 ± 0.028	0.611 ± 0.028	0.354 ± 0.008	0.516 ± 0.013	0.394 ± 0.041	0.907 ± 0.037	1.084 ± 0.018	0.499 ± 0.012	0.313 ± 0.027	0.417 ± 0.010	0.193 ± 0.011
$CNN1 - R - nonlinear$	**0.370 ± 0.017**	0.625 ± 0.018	0.347 ± 0.010	0.519 ± 0.018	0.452 ± 0.056	0.708 ± 0.019	0.643 ± 0.042	0.501 ± 0.020	0.138 ± 0.033	0.434 ± 0.058	0.203 ± 0.016
$CNN1 - R - linear$	0.404 ± 0.048	0.654 ± 0.016	0.385 ± 0.008	0.555 ± 0.014	**0.392 ± 0.025**	0.664 ± 0.015	0.563 ± 0.021	0.518 ± 0.019	0.108 ± 0.017	0.413 ± 0.058	0.221 ± 0.014
CNN2-R-nonlinear	0.375 ± 0.056	0.626 ± 0.034	0.348 ± 0.004	0.527 ± 0.025	0.440 ± 0.061	0.695 ± 0.024	0.657 ± 0.046	0.526 ± 0.033	0.130 ± 0.023	0.429 ± 0.036	0.190 ± 0.015
CNN2-R-linear	0.386 ± 0.044	0.643 ± 0.023	0.378 ± 0.010	0.522 ± 0.015	0.395 ± 0.047	0.665 ± 0.022	0.542 ± 0.023	0.499 ± 0.017	**0.098 ± 0.015**	1.121 ± 2.107	0.216 ± 0.018
LeNet5-nonlinear	0.451 ± 0.072	0.633 ± 0.052	**0.345 ± 0.004**	0.563 ± 0.018	0.626 ± 0.477	0.672 ± 0.027	0.593 ± 0.035	0.547 ± 0.037	0.122 ± 0.022	1.177 ± 0.556	0.191 ± 0.015
LeNet5-linear	0.379 ± 0.049	0.628 ± 0.028	0.347 ± 0.002	0.514 ± 0.010	0.393 ± 0.040	**0.650 ± 0.024**	**0.539 ± 0.030**	0.493 ± 0.021	0.101 ± 0.020	**0.386 ± 0.024**	0.189 ± 0.015
VGG16-nonlinear	0.490 ± 0.222	0.650 ± 0.035	2.386 ± 4.010	0.517 ± 0.011	0.524 ± 0.073	1.135 ± 0.288	1.253 ± 0.397	0.526 ± 0.046	0.923 ± 0.487	0.552 ± 0.056	0.881 ± 1.852
VGG16-linear	1.712 ± 2.426	0.676 ± 0.007	0.607 ± 0.074	0.590 ± 0.026	0.778 ± 0.833	1.766 ± 0.554	2.283 ± 0.041	0.590 ± 0.044	1.703 ± 0.535	0.655 ± 0.040	0.397 ± 0.192

Table 13: Learning classifier interactions (advanced experiments): means and standard deviations of prediction accuracy (shown in %) for each method on the benchmark datasets. A number of trees in ET is equal to 100 and its depth is equal to 2.

Method/Dataset	Balance	Bupa	Gamma	German	Heart	Mfeat-mor	Mfeat-zer	Pima	Segment	Sonar	Spambase
Best Zhang [84]	85.66 ± 4.14	––	**87.51 ± 0.71**	76.52 ± 3.37	82.59 ± 7.03	69.89 ± 2.53	76.01 ± 2.16	76.01 ± 3.87	97.24 ± 0.99	81.91 ± 9.33	––
Best Kuncheva ([47])	––	73.00	––	77.70	80.50	––	––	77.30	**98.20**	81.80	––
Best Ko et al. [43]	––	65.33 ± 0.04	––	––	––	––	––	**80.51 ± 0.04**	91.43 ± 0.12	––	––
Best Zhang [87]	**91.83**	––	85.81	**78.20**	**88.43**	––	––	78.13	94.67	82.69	91.39
Best ours	89.17 ± 1.26	**74.19 ± 3.00**	86.77 ± 0.31	76.40 ± 0.85	83.38 ± 2.09	**73.61 ± 0.65**	**79.29 ± 0.86**	78.20 ± 1.61	97.11 ± 0.53	**82.79 ± 1.94**	**94.38 ± 0.32**

- c–the number of cascades of classifier ensembles
- d–the number of features
- k–the number of nearest neighbours in case of k-NN classifier
- l_i–the number of neurons in the l_i layer
- L–the number of classes
- m–the number of neurons
- n–the number of samples

- p–the number of support vectors in SVM classifier
- t–the number of epochs to train NN (MLP, CNN, LSTM, etc)
- T–the number of classifier in a classifier ensemble

In Table 14 the classification complexities for some of the conventional algorithms have been demonstrated. As we can see from the Table the SVM and RF have exponential or linear-logarithmic complexities for training correspondingly while have a very low-cost computational complexities for the inference. The ET and k-NN have linear complexity for both training and inference. If we do not need to retrain our models too often we are able to use SVM even for larger datasets, especially for real-time predictions or classifications. A good compromise can be RFs. We provided computational complexities for conventional classifiers because they will be involved to build stacking-based classification algorithms.

Table 14: Computational complexity of some conventional classification algorithms

Algorithm	RF	ET	k-NN	SVM	MLP
Training complexity	$O(n\log(n)dT)$	$O(ndT)$	$O(knd)$	$O(n^2d+n^3)$	$O(tdnm)$
Testing complexity	$O(dT)$	$O(ndT)$	$O(nd)$	$O(pd)$	$O(dl_1+l_1l_2+...)$

Next, we are going to analyse stacking-based classifiers where aforementioned conventional classifiers will be used as meta-classifiers and as base classifiers to produce classification predictions, we apply decision trees. There are linear and nonlinear versions of classifier stacking obtained by using R-manifold. To compute the total complexity of all stacking-based classifiers when meta-classifiers are aforementioned conventional classifiers, we need to take a complexity of RF plus complexity of a conventional meta-classifier when new feature number is equal to TL. For simplicity we use L instead of $L-1$ independent SPD matrices (see Table 15). In case of linear and nonlinear versions of classifier stacking, the number of features will be proportional to T^2L because we have to vectorize L SPD matrices. The number of operations to compute SPD matrix is proportional to T^2 and computing the logarithm of this matrix while linearizing it requires T^3 operations. That is how we compute the complexities in Tables 16 and 17. We then simplified those complexities by putting the number of predictors T and the number of instances n high. We then simplified them more by assuming that in most of the cases the next conditions are satisfied: $p \gg 1; n \gg 1; Ll_1 \gg 1; pL \gg 1; kL \gg 1; nL \gg 1, T^2Ll_1 \gg 1; n \gg L; T \gg p; T \gg L, T \gg k; n \gg k$. We are not going to provide a computational complexity for DNF due to the novelty of the algorithms and the non-trivial evaluation of the computational complexity.

Table 15: Computational complexity of some conventional stacking-based classification algoritms

Algorithm	k-NN-stacking	SVM-stacking	MLP-stacking	RF cascades	ET cascades
Training complexity	$O(n\log(n)dT+knTL)$	$O(n^2TL+n\log(n)Td)+n^3)$	$O(n\log(n)Td+TLtnm)$	$O(cn\log(n)T^2L)$	$O(cnT^2L)$
Testing complexity	$O(TnL+Td)$	$O(TpL+Td)$	$O(Td+TLl_1+l_1l_2+...)$	$O(cT^2L)$	$O(cnT^2L)$

Lastly, we provide the complexities of stacking-based architectures based on R-and-G-manifolds. To learn on R-manifold of SPD matrices we first need to compute these matrices. Computing L SPD matrices is of $O(T^2L)$ complexity and linearization of them is about $O(T^3L)$. Approximate simplified complexity of CNNs is calculated taking into account the

following assumptions. First, we are going to determine the computational complexity of CNN as a function of the size of SPD matrices. We then assume that the number of convolutional layers is equal to the number of convolutional filters and the number of training epochs. Knowing that every convolution is of $O(T^3)$ complexity we determine that backpropagation complexity will be about $O(T^5)$. The feed forward path has approximate complexity about $O(T^4)$. Taking into account that the number of SPD matrices is equal to L, we have the final complexity presented in the Table 18. We remind that those complexities are not very accurate due to assumptions we have made. This only shows the order of the complexity based on structure of the CNN architecture and the dimension of the input data. Some of the DL architectures may be way simpler than assumed here, so this upper bound can be too overrated. One of the examples is pyramidal-based architecture. It has a property that the number of convolutional filters in the largest layer is larger than the sum of convolutional filters in all previous layers. This means that the upper bound of its training complexity can be of order $O(T^4)$ while inference one doean not exceed $O(T^3)$.

Table 16: Computational complexity of some conventional nonlinear stacking-based classification algorithms

Algorithm	k-NN-stacking-nonlinear	SVM-stacking-nonlinear	MLP-stacking-nonlinear
Training complexity	$O(n\log(n)dT + knT^2L + nLT^2)$	$O(n\log(n)dT + n^2T^2L + n^3 + nLT^2)$	$O(n\log(n)dT + T^2Ltnm + nLT^2)$
Testing complexity	$O(T^2nL + Td + LT^2)$	$O(Td + T^2pL + LT^2)$	$O(Td + T^2Ll_1 + l_1l_2 + ... + LT^2)$

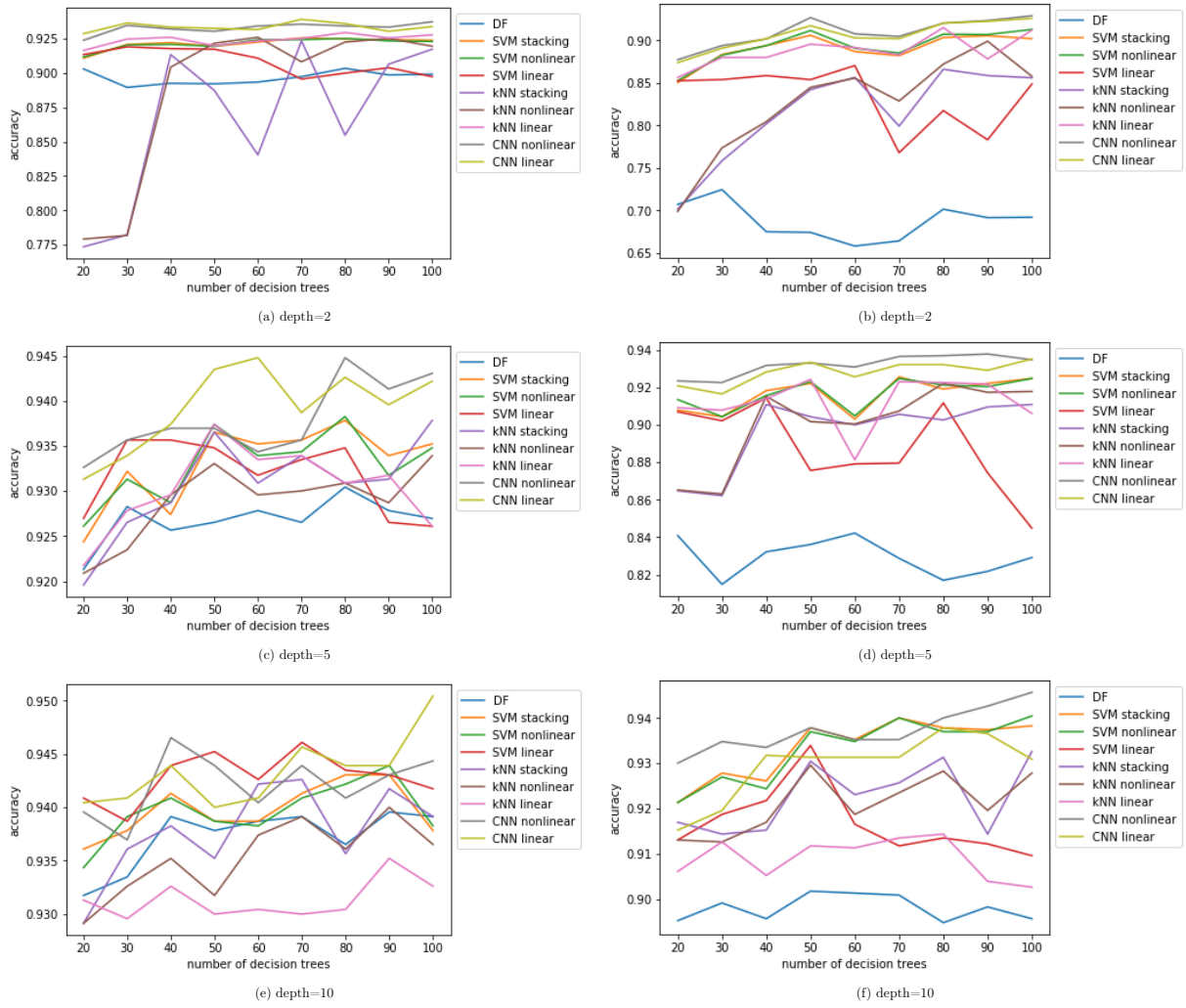
Table 17: Computational complexity of some conventional linear stacking-based classification algoritms

Algorithm	k-NN-stacking-linear	SVM-stacking-linear	MLP-stacking-linear
Training complexity	$O(n\log(n)dT + knT^2L + nLT^2 + nLT^3)$	$O(n\log(n)dT + n^2T^2L + n^3 + nLT^2 + nLT^3)$	$O(n\log(n)dT + T^2Ltnm + nLT^2 + nLT^3)$
Testing complexity	$O(T^2nL + Td + LT^2 + LT^3)$	$O(Td + T^2pL + LT^2 + LT^3)$	$O(Td + T^2Ll_1 + l_1l_2 + ... + LT^2 + LT^3)$

Next, we are going to consider four architectures based on G-manifold. Since, computational complexity for the aforementioned architectures involves many terms, we focus only on the largest terms. Computational complexity for GC based on DP consists of three terms: the complexity to build predictions by RF (ET), the complexity for two SPD and the complexity for classification using k-NN classifier. We leave only two first terms because othersare much less than any of two first terms. In the case of inference, we keep all three terms. In the case of GC we need applying RF c times to perform a data transformation. Other terms are the same as in the previous case and we keep only the largest one. For the inference, we keep two terms as in the previous case. For linear and nonlinear GC we first need to compute CPPM matrix which has $O(T^2)$ complexity. Linearization has additional complexity $O(T^3)$. The first cascade takes a vector of features obtained by vectorizing CPPM matrix with linearization or not. Then all other cascades take vector of features equal to TL. A DP has been built by using predictions from the last cascade. After that the number of operations to compute two SPD matrices and using k-NN classifier is the same as in the previous cases. We then leave only the largest term. The inference complexity consists of two terms as before. In case of ET the computational complexity is proportional to n instead of $n\log(n)$ everywhere during the training phase.

Table 18: Computational complexity of some manifold-based architectures

Algorithm	CNN-linear	CNN-nonlinear	G-DP	GC	GC-linear	GC-nonlinear
Training complexity	$O(nL(T^5+T^3+T^2))$	$O(nL(T^5+T^2))$	$T^2Ln+n\log(n)dT$	$n\log(n)T^2Lc$	$n\log(n)T^3L$	$n\log(n)T^3L$
Testing complexity	$O(L(T^4+T^3+T^2))$	$O(L(T^4+T^2))$	$T^2L+dT+nL$	T^2Lc+nL	T^3L+nL	T^3L+Ln

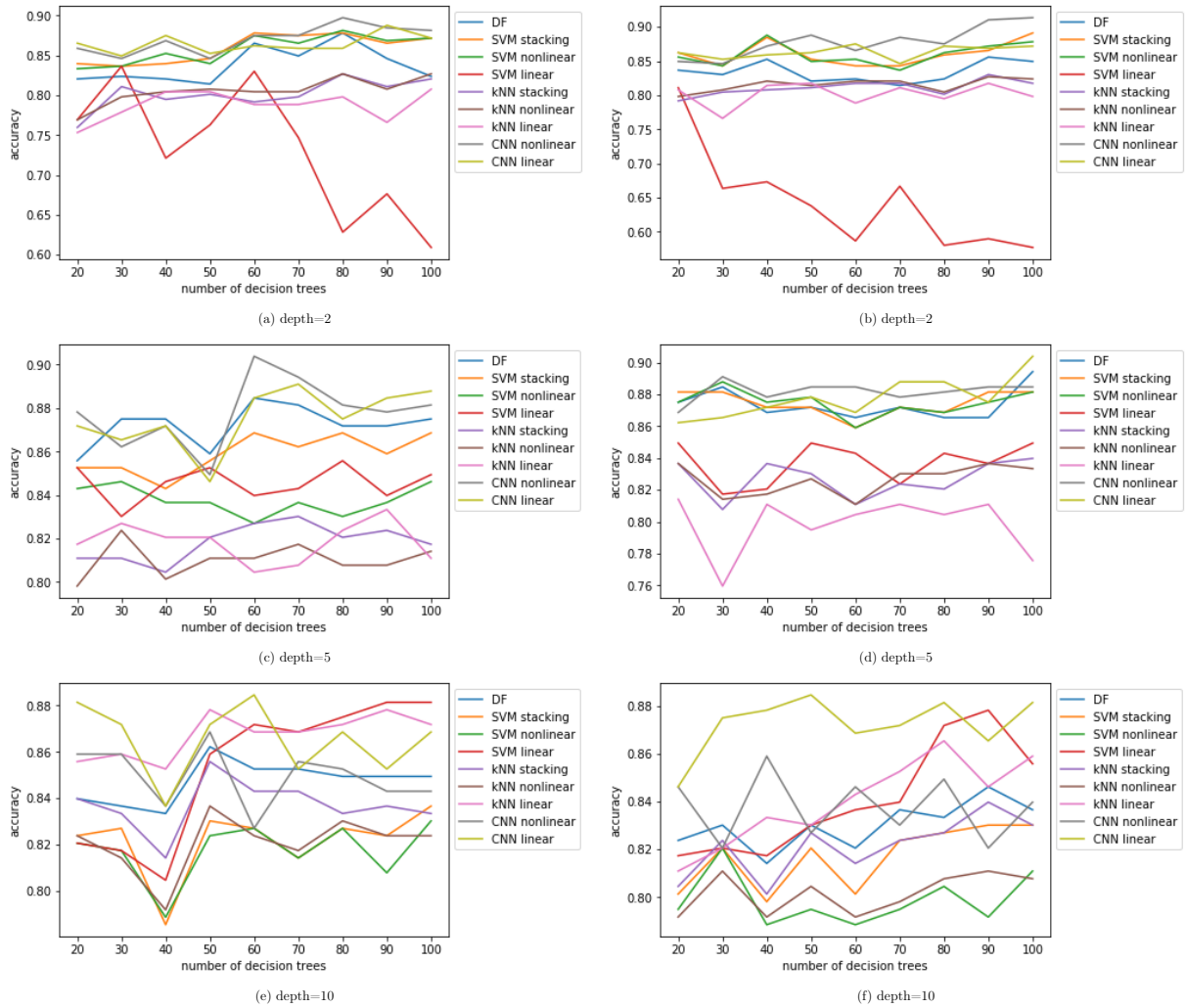


(a) depth=2 (b) depth=2

(c) depth=5 (d) depth=5

(e) depth=10 (f) depth=10

Figure 29: Classification performance as a function of the number of trees and their depth for "spambase" dataset: present the results for RFs (left column) and for ETs (right column) as new data generators correspondingly.



(a) depth=2

(b) depth=2

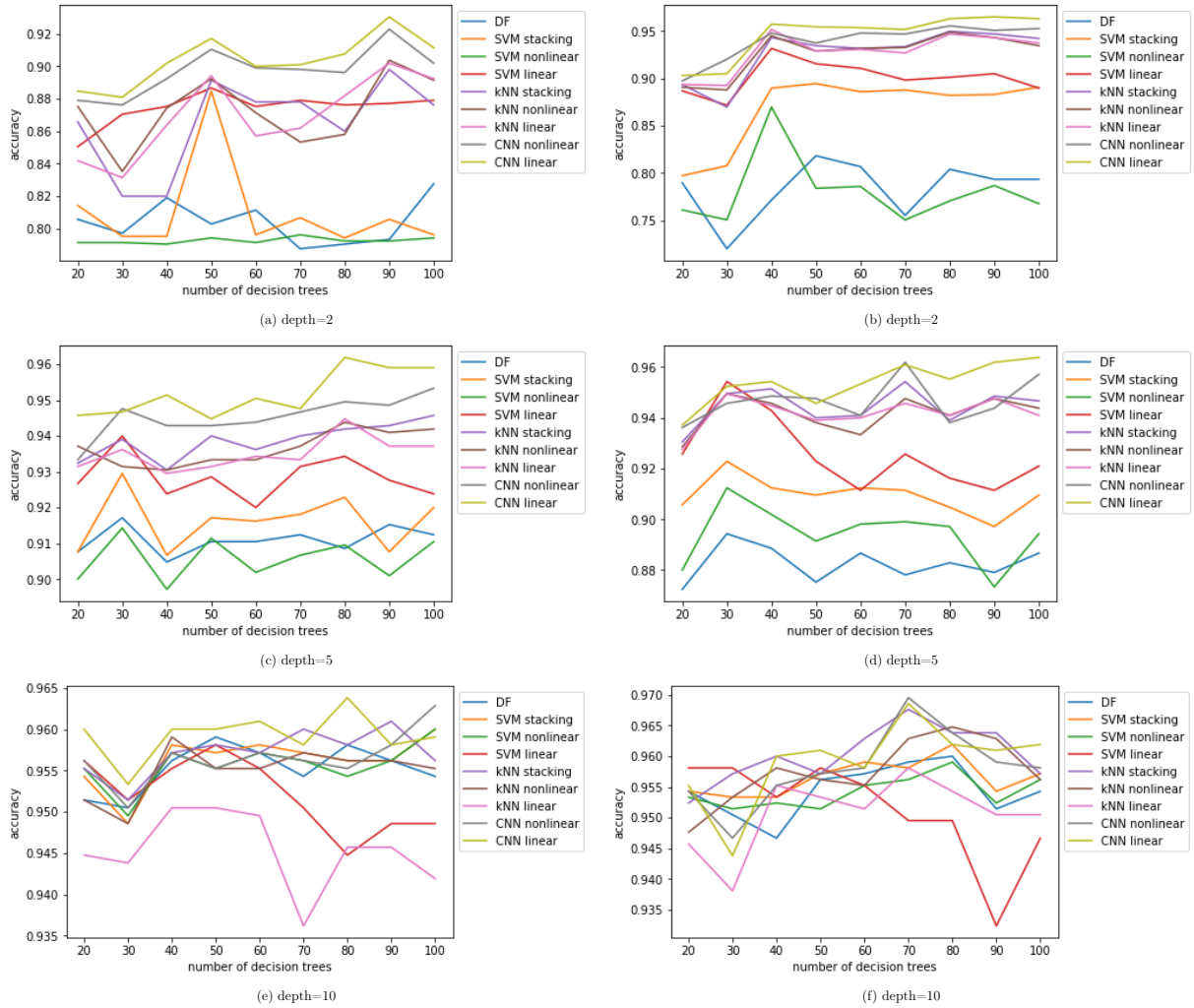
(c) depth=5

(d) depth=5

(e) depth=10

(f) depth=10

Figure 30: Classification performance as a function of the number of trees and their depth for "balance" dataset: present the results for RFs (left column) and for ETs (right column) as new data generators correspondingly.



(a) depth=2

(b) depth=2

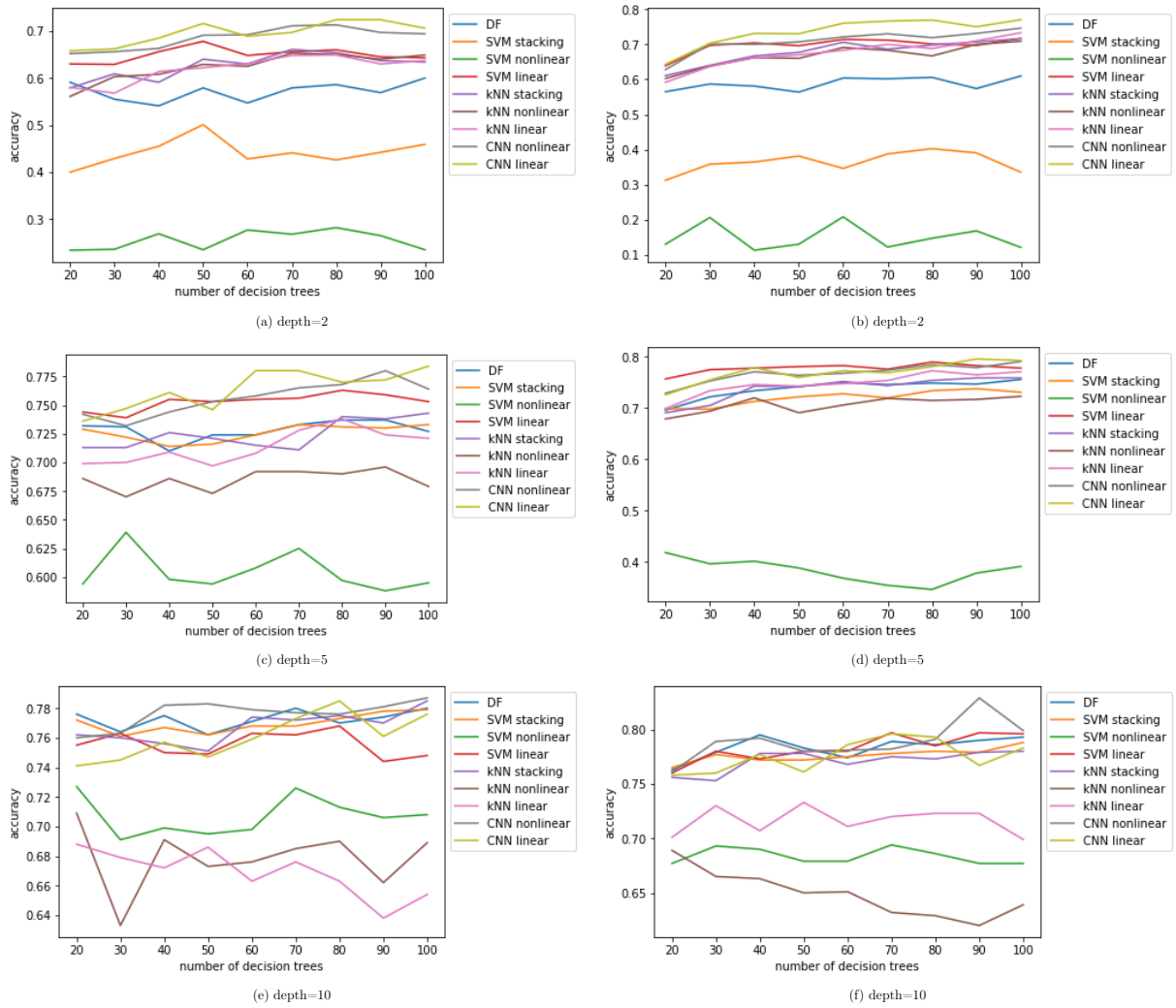
(c) depth=5

(d) depth=5

(e) depth=10

(f) depth=10

Figure 31: Classification performance as a function of the number of trees and their depth for "segmentaion" dataset: present the results for RFs (left column) and for ETs (right column) as new data generators correspondingly.



(a) depth=2

(b) depth=2

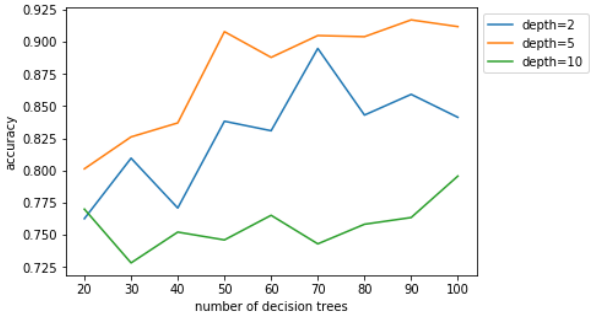
(c) depth=5

(d) depth=5

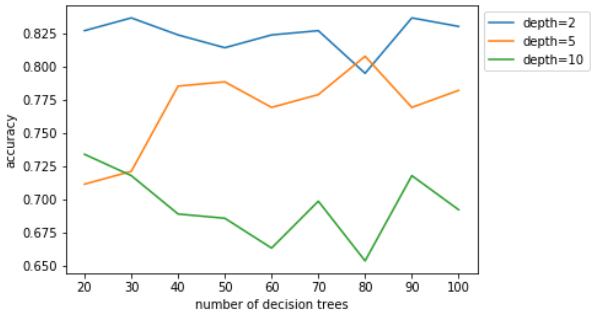
(e) depth=10

(f) depth=10

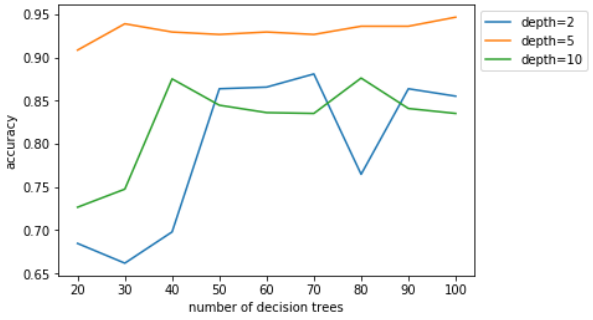
Figure 32: Classification performance as a function of the number of trees and their depth for "mfeat-zer" dataset: present the results for RFs (left column) and for ETs (right column) as new data generators correspondingly.



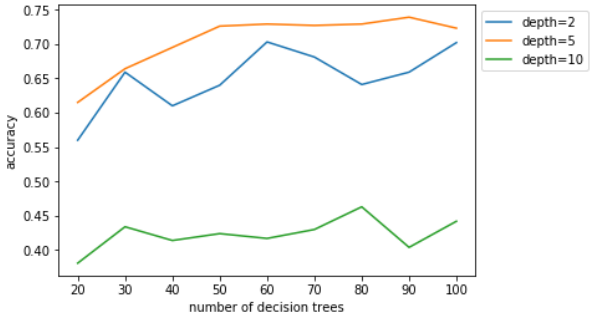
(a) "spambase" dataset on RF



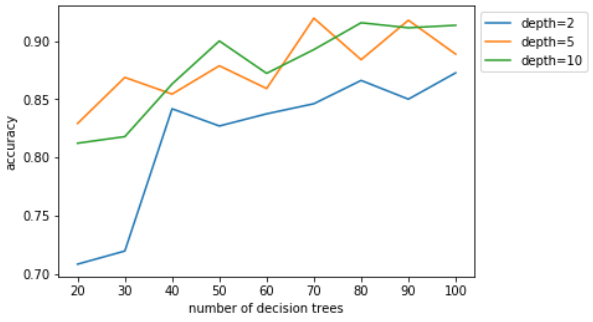
(b) "balance" dataset on RF



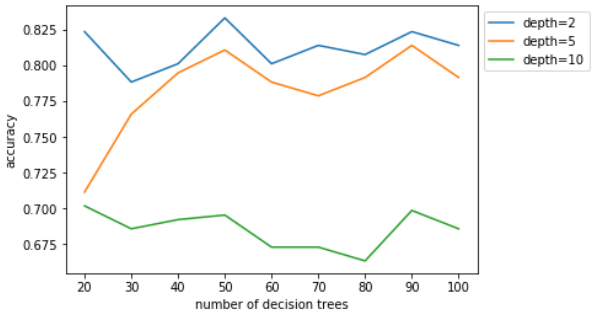
(c) "segmentation" dataset on RF



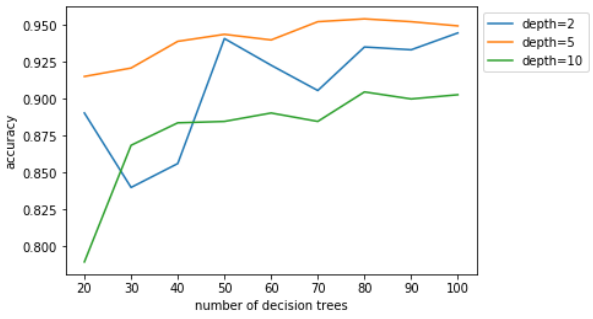
(d) "mfeat-zer" dataset" on RF



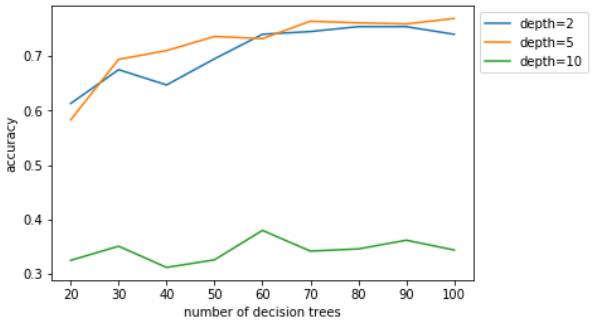
(e) "spambase" dataset on ETs



(f) "balance" dataset on ETs



(g) "segmentation" dataset on ETs



(h) "mfeat-zer" dataset" on ETs

Figure 33: Performance of G-manifolds on four datasets as function of number of trees and their depth

5 Application of Riemannian and Grassmann manifolds to video classification and detection problems

5.1 Introduction

This section is devoted to application of nonlinear manifolds for the problems of video analysis. There are two problems that will be considered: gesture phase segmentation and face forgery detection. For the first dataset features have been already generated while for the second one we have to detect the region of interest (ROI) and then generate features for it using NN-based DL architecture. As a performance measure we have used validation accuracy and validation log-loss for the first dataset and training and public validation log-loss–for the second one. Similarly to the previous series of experiments we have used hold-out with stratification to split our dataset into training and validation subsets. For the previous experiments, among all we compared our architectures with the state of the art algorithms, while here we compare our methods against the leader board ones during the competition organized by Kaggle for the face forgery detection.

5.2 Gesture Phase Segmentation dataset

Gesture segmentation dataset, which consists of 7 raw subsets corresponding to 7 videos of gestures (see Figure 19). Thus, we used every subset as a separate dataset to carry out our experiments. These are 5 class classification problems with 18 features. This dataset also contains processed subsets, but we are not going to use them due to a very low classification accuracy by all the algorithms, which may signify poor information contained in initial feature vectors. We kept the same testing protocol as in the previous set of experiments.

Table 19: Summary of characteristics of Gesture Phase Segmentation dataset: raw data

Dataset	**Size**	**Features**	**Classes**	**Tr,** %	**Ts,** %
gesture-raw1	1747	18	5	50	50
gesture-raw2	1264	18	5	50	50
gesture-raw3	1834	18	5	50	50
gesture-raw4	1073	18	5	50	50
gesture-raw5	1424	18	5	50	50
gesture-raw6	1111	18	5	50	50
gesture-raw7	1448	18	5	50	50

5.3 Experiments

In Table 20 we provide the classification accuracy for different schemes of data transformation and different classifiers. As seen, the nonlinear version of classifier stacking using classifier interactions outperforms other classifier stacking techniques, including a nonlinear version of DPs based on G-manifolds and (GCs). For an GC as a distance measure, a geodesic's length has been applied. GCs are built according to the homotopy diagram presented in Figure 26. Simple GCs are obtained from DPs built on data produced by CRFs where the number of

RFs or ETs changes from one to eight. Nonlinear and linearized versions of GCs are built on the corresponding versions of CRF using R-manifolds of SPD matrices which are CPPMs. As one can see, in this case, we have a combination of two manifolds applied together at the same time. As seen from the aforementioned Table, GCs and CNN-based classifiers that learn on R-manifold of SPD matrices outperform by a large margin forest-based classifiers including their cascades and stacking-based SVM in both geometries. This happens when the depth of DTs as base classifiers in RFs and ETs is equal to 2. When the depth of DTs is equal to 5 this margin is less significant. Lower depths of DTs leads to more diverse base classifiers in RFs and ETs which leads to more diversity between elements of CPPMs or S_+^d. The diversity of elements in aforementioned matrices also depends on the initial features X because $Z_1^X = f_1(X)$ (see Figure 25). To create more diverse classifiers, we need lower depths of DTs in RFs or ETs, but those classifiers are characterized by lower accuracy. As we mentioned before, according to research carried out in [47], for base classifiers in a classifier ensemble it is more important to be accurate than diverse when MV is applied as a decision-making rule. As we can see from the Table 20, the difference between the accuracy of classifier stacking using classifier interactions and CNN as a meta-classifier as well as in the case of GC for aforementioned depths is less significant in comparison with a classical RFs or ETs. Knowing that motion is a nonlinear process, we can conclude that using nonlinear manifolds which present classifier interactions, allows taking into account the geometry of data which depends on the geometry of initial features and mapping function $f_1 : X \to Z_1^X$ depending, for instance, on the depth of DTs in our case. We can also see that SVMs do not perform well on the initial feature set X but improve significantly using predictions from the first CRFs as transformed features Z_1^X. This means that generally, it is not a very good idea to use SVM on the initial features when dealing with motion analysis problems, as opposed to what authors did in [62] using SVM as a classifier for fused temporal and spatial features.

In Tables 21-22 we summarized the accuracy and logarithmic losses from different CNN architectures used as meta-learners for classifier stacking build on R-manifold of SPD matrices. As we can see from the tables pyramidal-based architecture and LeNet5 outperform base-line architecture for most of the problems. Since it outperforms LeNet5 for most of the datasets, it has been ranked as a top architecture according to average ranks ranking. Similar conclusion can be made by using other ranking methods, however it requires some computations. Pyramidal-based architecture is the simplest one in terms of computational complexity (see section 4). Indeed, among all CNN architectures its has the least amount of convolutional filters. In Table 22 we provided logarithmic losses for RFs and CRFs in both geometries. Based on all three tables we can conclude that the best classification accuracy is achieved for CNN-based architectures that are used to learn on R-manifold of SPD matrices which are CPPMs of classifier interactions. As one more deep architecture we compared DNFs vs CNNs. We wanted to see the performance of the new deep architecture aiming to learn optimal splits in DTs of RFs. As we can see from the results presented in Table 21, DNFs are comparable to CRFs but inferior to the CNN-based architectures, especially LeNet5 and pyramidal CNN architecture. As in the previous set of experiments on the generalized datasets from the UCI repository we can affirm that DNFs are characterized by significant instability of classification accuracy for some datasets when using different subsets of the initial dataset. This can be seen by the sufficiently large variance of the classification accuracy.

Table 20: Learning classifier predictions using different classifier stacking techniques: means and standard deviations of prediction accuracy (shown in %) for each method on the raw Gesture Phase Segmentation dataset.

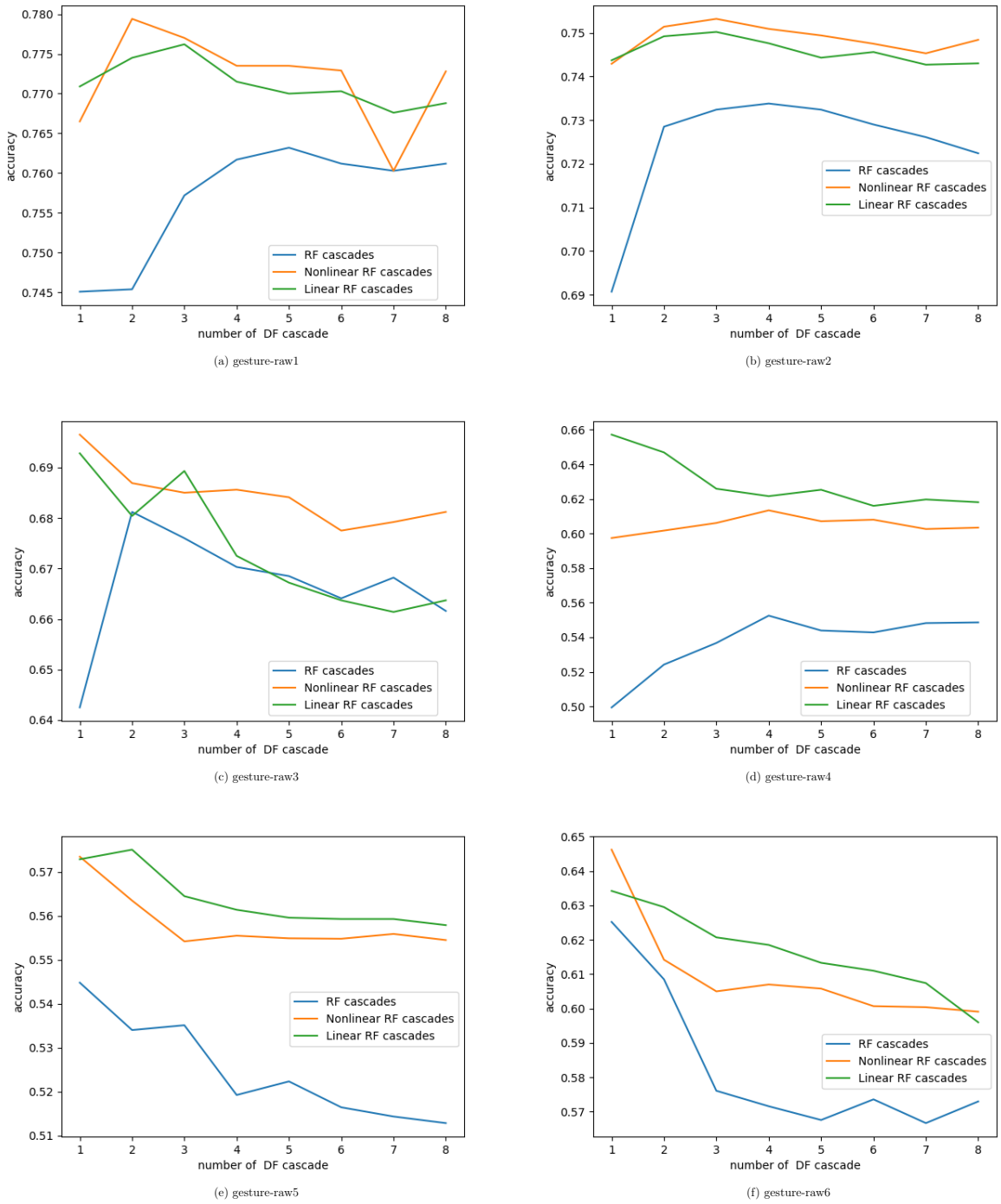
Method/Dataset	gesture-raw1	gesture-raw2	gesture-raw3	gesture-raw4	gesture-raw5	gesture-raw6	gesture-raw7
A number of DTs in a RFs is equal to 100 and its depth is equal to 2							
RFs_1	74.45 ± 1.36	69.07 ± 1.44	64.25 ± 2.01	49.94 ± 3.75	54.48 ± 1.48	62.52 ± 3.48	58.49 ± 2.37
RFs_{max}	76.31 ± 1.50	73.38 ± 1.44	68.12 ± 1.39	55.25 ± 5.96	–	–	–
$RFs - nonlinear_{max}$	77.94 ± 1.42	75.31 ± 1.41	69.65 ± 1.98	61.34 ± 3.55	57.35 ± 2.25	64.62 ± 2.93	60.33 ± 1.94
$RFs - linear_{max}$	77.62 ± 1.75	75.01 ± 2.14	69.28 ± 1.96	65.72 ± 2.73	57.51 ± 2.02	63.42 ± 3.85	60.55 ± 2.57
$CNN - R - nonlinear$	85.46 ± 1.21	84.29 ± 1.73	86.93 ± 0.91	80.88 ± 2.94	**80.62 ± 1.78**	79.59 ± 2.16	81.81 ± 1.55
$CNN - R - linear$	**85.69 ± 1.23**	**84.94 ± 1.53**	**87.19 ± 1.02**	**80.99 ± 3.25**	80.46 ± 2.00	**80.54 ± 2.15**	**82.17 ± 1.66**
GC_{max}	83.41 ± 1.47	82.06 ± 1.45	84.96 ± 1.35	75.70 ± 3.52	79.35 ± 2.19	77.73 ± 1.66	78.81 ± 1.48
$GC - nonlinear_{max}$	81.28 ± 1.83	78.88 ± 1.33	80.37 ± 2.26	71.02 ± 3.16	70.41 ± 2.56	71.13 ± 3.80	71.49 ± 2.88
$GC - linear_{max}$	81.75 ± 1.58	78.86 ± 2.41	81.71 ± 1.27	74.12 ± 3.42	71.19 ± 2.30	71.87 ± 2.03	73.31 ± 3.08
$SVM(RBF)$	74.19 ± 1.24	68.05 ± 1.46	70.12 ± 1.53	46.16 ± 1.79	41.19 ± 2.15	48.15 ± 1.77	49.28 ± 1.27
$SVM(RBF) - stacking$	80.41 ± 1.49	77.91 ± 2.23	81.25 ± 1.69	69.78 ± 3.07	69.44 ± 3.12	73.33 ± 2.01	74.59 ± 1.76
Number of DTs in an RFs is equal to 100 and its depth is equal to 5							
RFs_1	83.46 ± 0.93	85.35 ± 1.12	85.81 ± 0.71	77.17 ± 2.66	77.44 ± 2.11	79.10 ± 1.98	79.16 ± 1.29
RFs_{max}	89.13 ± 1.88	89.68 ± 1.04	92.04 ± 1.12	89.70 ± 0.94	89.05 ± 1.75	85.27 ± 1.06	86.20 ± 1.41
$RFs - nonlinear_{max}$	90.29 ± 1.52	89.98 ± 0.86	92.16 ± 1.14	89.81 ± 1.10	90.08 ± 1.45	86.40 ± 1.54	87.00 ± 1.08
$RFs - linear_{max}$	89.41 ± 1.22	89.29 ± 0.92	91.48 ± 1.18	89.35 ± 1.25	88.82 ± 1.69	85.27 ± 1.20	85.88 ± 1.23
$CNN - R - nonlinear$	89.91 ± 1.39	89.46 ± 0.83	92.34 ± 1.00	89.29 ± 1.02	89.31 ± 1.27	86.29 ± 1.31	86.34 ± 0.84
$CNN - R - linear$	**90.67 ± 1.14**	**90.33 ± 1.16**	**93.04 ± 1.06**	**90.73 ± 1.31**	**90.45 ± 1.22**	**87.27 ± 1.53**	**87.87 ± 1.14**
GC_{max}	86.01 ± 1.01	85.05 ± 1.61	89.17 ± 1.20	88.57 ± 1.76	87.32 ± 1.58	81.46 ± 1.76	84.13 ± 1.36
$GC - nonlinear_{max}$	78.68 ± 1.10	74.67 ± 1.55	80.56 ± 1.62	79.74 ± 2.36	73.41 ± 2.02	68.27 ± 2.90	73.74 ± 1.34
$GC - linear_{max}$	77.05 ± 1.17	72.97 ± 1.38	80.40 ± 1.16	77.30 ± 2.22	70.15 ± 2.17	65.13 ± 2.99	71.46 ± 1.83
$SVM(RBF) - stacking$	89.15 ± 1.54	89.95 ± 1.15	91.55 ± 0.83	87.09 ± 1.50	89.41 ± 1.73	86.98 ± 1.60	86.71 ± 1.33
Number of DTs in an ETs is equal to 100 and its depth is equal to 2							
ETs_1	74.98 ± 0.75	68.97 ± 1.00	62.99 ± 0.92	45.90 ± 2.21	49.90 ± 2.95	58.62 ± 3.01	55.40 ± 3.11
ETs_{max}	76.99 ± 1.53	71.90 ± 1.51	68.21 ± 2.36	50.00 ± 3.03	54.23 ± 2.51	60.81 ± 2.37	59.92 ± 2.62
$ETs - nonlinear_{max}$	77.87 ± 1.21	73.96 ± 1.64	70.92 ± 1.58	55.51 ± 4.56	56.76 ± 1.59	63.21 ± 2.55	61.15 ± 2.46
$ETs - linear_{max}$	77.84 ± 0.78	73.58 ± 1.07	68.78 ± 1.48	61.15 ± 3.75	55.74 ± 2.59	63.41 ± 1.86	60.69 ± 2.38
$CNN - R - nonlinear$	87.14 ± 1.66	85.92 ± 1.43	87.21 ± 0.68	81.73 ± 2.34	83.89 ± 2.03	80.40 ± 2.43	**84.03 ± 1.51**
$CNN - R - linear$	**88.07 ± 1.67**	**87.15 ± 1.65**	**87.75 ± 0.54**	**83.48 ± 2.95**	**83.89 ± 1.69**	**81.55 ± 1.94**	83.49 ± 1.39
GC_{max}	86.99 ± 0.70	84.48 ± 1.82	85.27 ± 1.84	82.74 ± 2.40	81.29 ± 1.72	81.44 ± 1.76	81.98 ± 1.60
$GC - nonlinear_{max}$	81.50 ± 1.59	80.28 ± 1.79	81.26 ± 1.34	70.11 ± 6.98	68.78 ± 5.00	71.56 ± 2.25	75.47 ± 1.84
$GC - linear_{max}$	83.02 ± 0.98	80.87 ± 1.27	81.81 ± 2.57	76.57 ± 3.59	70.10 ± 3.27	74.01 ± 1.30	75.44 ± 1.50
$SVM(RBF) - stacking$	81.69 ± 1.00	78.51 ± 2.20	82.88 ± 1.00	71.34 ± 2.61	73.19 ± 2.74	77.28 ± 2.97	79.41 ± 1.69
Number of DTs in an ETs is equal to 100 and its depth is equal to 5							
ETs_1	78.24 ± 1.40	77.88 ± 1.79	77.20 ± 2.01	72.33 ± 3.65	70.62 ± 1.87	74.30 ± 2.06	72.75 ± 2.38
ETs_{max}	87.93 ± 0.95	87.93 ± 0.86	88.94 ± 1.39	88.70 ± 1.36	85.52 ± 1.53	82.52 ± 1.73	84.46 ± 1.83
$ETs - nonlinear_{max}$	88.58 ± 1.42	88.39 ± 0.80	90.05 ± 0.81	89.01 ± 1.32	86.33 ± 1.33	84.37 ± 2.10	84.85 ± 1.73
$ETs - linear_{max}$	87.79 ± 0.87	87.59 ± 1.26	88.33 ± 1.62	87.56 ± 1.18	84.61 ± 1.47	82.07 ± 1.72	83.20 ± 2.22
$CNN - R - nonlinear$	89.28 ± 0.98	89.41 ± 1.13	90.88 ± 0.66	89.39 ± 2.14	89.14 ± 1.04	86.22 ± 1.79	86.93 ± 1.52
$CNN - R - linear$	**90.90 ± 0.86**	**89.98 ± 0.93**	**92.06 ± 1.05**	**90.15 ± 0.92**	**90.00 ± 0.98**	**87.43 ± 1.47**	**87.89 ± 1.35**
GC_{max}	88.38 ± 1.23	87.01 ± 1.54	90.61 ± 0.78	87.21 ± 1.55	87.53 ± 1.56	83.83 ± 1.75	85.10 ± 1.07
$GC - nonlinear_{max}$	83.22 ± 1.71	80.35 ± 1.73	85.80 ± 1.06	83.20 ± 1.42	81.71 ± 2.82	75.31 ± 2.71	79.48 ± 1.60
$GC - linear_{max}$	80.69 ± 2.10	77.86 ± 1.93	82.76 ± 1.36	81.58 ± 2.16	77.89 ± 2.60	69.19 ± 1.77	77.49 ± 1.97
$SVM(RBF) - stacking$	87.49 ± 1.41	87.63 ± 0.93	88.17 ± 1.22	85.49 ± 1.85	86.18 ± 1.19	85.00 ± 1.65	85.46 ± 1.60

For the second half of the first series of experiments, we also want to see whether increasing the number of cascades leads to improvement of classification accuracy for the Gesture Phase Segmentation dataset. Accuracy has been evaluated for every number of cascades in CRFs (see Figures 34-37). For the experiments, we put the depth of trees in RFs and ETs equal to 5 as the accuracy is much higher in comparison when depth is equal to 2. As seen from Figures 34-37, more cascades are required to get the maximum accuracy using cascades of ETs. As we mentioned before, ETs produce more nonlinear outputs (predictions) than RFs. Thus, we may assume that more nonlinear data require more cascades. Also, it can be noticed that after every next cascade, the improvement in accuracy is less significant than after the previous one. This can be seen in slope decrease on all curves using more cascades. Thus, we can conclude that it is true that adding the next cascade is less efficient than the previous

Table 21: Learning classifier predictions using different classifier stacking techniques: means and standard deviations of prediction accuracy (shown in %) for each method on the raw Gesture Phase Segmentation dataset.

Method/Dataset	gesture-raw1	gesture-raw2	gesture-raw3	gesture-raw4	gesture-raw5	gesture-raw6	gesture-raw7
A number of DTs in a RFs is equal to 100 and its depth is equal to 2							
$DNF-nonlinear$	82.67 ± 1.39	78.13 ± 2.34	82.62 ± 1.03	69.55 ± 4.87	70.23 ± 3.21	71.23 ± 2.88	73.52 ± 2.66
$DNF-linear$	82.58 ± 0.75	80.82 ± 1.94	83.64 ± 0.73	73.09 ± 4.44	73.66 ± 3.83	74.43 ± 2.32	75.06 ± 2.41
$CNN1-R-nonlinear$	85.46 ± 1.21	84.29 ± 1.73	86.93 ± 0.91	80.88 ± 2.94	80.62 ± 1.78	79.59 ± 2.16	81.81 ± 1.55
$CNN1-R-linear$	85.69 ± 1.23	84.94 ± 1.53	87.19 ± 1.02	80.99 ± 3.25	80.46 ± 2.00	80.54 ± 2.15	82.17 ± 1.66
$CNN2-R-nonlinear$	87.72 ± 1.01	85.52 ± 1.90	88.30 ± 1.25	$\mathbf{82.77 \pm 3.12}$	$\mathbf{85.22 \pm 1.22}$	81.83 ± 1.55	82.68 ± 1.52
$CNN2-R-linear$	$\mathbf{88.19 \pm 1.37}$	$\mathbf{86.46 \pm 1.93}$	$\mathbf{88.80 \pm 1.21}$	82.18 ± 2.81	84.85 ± 1.92	$\mathbf{81.83 \pm 1.33}$	$\mathbf{83.41 \pm 1.27}$
$LeNet5-nonlinear$	86.20 ± 0.95	84.41 ± 2.18	85.52 ± 1.63	76.44 ± 3.16	77.25 ± 2.22	77.77 ± 1.79	77.79 ± 1.52
$LeNet5-linear$	85.37 ± 1.52	84.00 ± 1.68	85.87 ± 1.68	76.98 ± 3.22	78.57 ± 1.55	79.39 ± 1.46	79.86 ± 1.12
$VGG16-nonlinear$	70.93 ± 15.40	65.30 ± 14.32	74.21 ± 4.08	44.95 ± 4.44	57.39 ± 2.43	55.09 ± 11.83	55.91 ± 10.42
$VGG16-linear$	52.31 ± 16.12	46.27 ± 13.37	50.87 ± 16.48	40.28 ± 2.34	51.87 ± 8.66	35.09 ± 9.15	51.59 ± 4.17
Number of DTs in a RFs is equal to 100 and its depth is equal to 5							
$DNF-nonlinear$	88.19 ± 1.31	87.52 ± 1.54	90.88 ± 1.60	87.38 ± 3.28	86.19 ± 2.07	85.00 ± 1.91	85.47 ± 1.50
$DNF-linear$	75.04 ± 19.19	88.09 ± 1.56	92.15 ± 0.85	84.43 ± 11.42	88.47 ± 2.08	85.41 ± 1.87	86.48 ± 1.07
$CNN1-R-nonlinear$	89.91 ± 1.39	89.46 ± 0.83	92.34 ± 1.00	89.29 ± 1.02	89.31 ± 1.27	86.29 ± 1.31	86.34 ± 0.84
$CNN1-R-linear$	90.67 ± 1.14	90.33 ± 1.16	93.04 ± 1.06	90.73 ± 1.31	90.45 ± 1.22	87.27 ± 1.53	87.87 ± 1.14
$CNN2-R-nonlinear$	91.17 ± 0.71	90.05 ± 1.45	92.52 ± 0.72	90.50 ± 1.24	90.41 ± 1.03	87.61 ± 1.21	88.26 ± 0.82
$CNN2-R-linear$	$\mathbf{92.06 \pm 0.88}$	$\mathbf{91.14 \pm 1.30}$	$\mathbf{93.22 \pm 0.78}$	$\mathbf{91.73 \pm 1.10}$	$\mathbf{91.90 \pm 1.23}$	$\mathbf{88.58 \pm 1.54}$	$\mathbf{89.23 \pm 0.74}$
$LeNet5-nonlinear$	91.10 ± 1.01	90.08 ± 1.40	92.29 ± 0.71	89.76 ± 1.19	90.21 ± 1.22	87.09 ± 1.29	88.08 ± 1.02
$LeNet5-linear$	91.67 ± 0.92	90.81 ± 1.30	92.81 ± 0.73	91.08 ± 1.06	91.24 ± 0.96	88.20 ± 1.95	88.85 ± 0.77
$VGG16-nonlinear$	73.01 ± 21.64	60.95 ± 24.95	77.20 ± 22.44	71.30 ± 17.35	79.07 ± 15.24	70.88 ± 16.72	82.89 ± 2.45
$VGG16-linear$	64.03 ± 20.74	61.61 ± 18.73	57.99 ± 13.91	39.76 ± 2.30	49.28 ± 10.91	42.81 ± 9.06	56.09 ± 12.09
Number of DTs in an ETs is equal to 100 and its depth is equal to 2							
$DNF-nonlinear$	81.50 ± 2.06	77.11 ± 2.79	83.56 ± 1.34	68.69 ± 9.17	68.69 ± 3.43	71.50 ± 3.15	76.70 ± 3.20
$DNF-linear$	81.91 ± 1.83	80.37 ± 2.56	84.29 ± 2.59	73.55 ± 4.55	69.70 ± 8.20	77.56 ± 1.94	74.08 ± 5.12
$CNN1-R-nonlinear$	87.14 ± 1.66	85.92 ± 1.43	87.21 ± 0.68	81.73 ± 2.34	83.89 ± 2.03	80.40 ± 2.43	84.03 ± 1.51
$CNN1-R-linear$	88.07 ± 1.67	87.15 ± 1.65	87.75 ± 0.54	83.48 ± 2.95	83.89 ± 1.69	81.55 ± 1.94	83.49 ± 1.39
$CNN2-R-nonlinear$	89.07 ± 1.41	87.59 ± 1.18	88.92 ± 1.02	85.83 ± 2.26	87.15 ± 1.51	84.60 ± 1.43	$\mathbf{85.94 \pm 1.04}$
$CNN2-R-linear$	$\mathbf{89.42 \pm 1.05}$	$\mathbf{88.15 \pm 1.26}$	$\mathbf{89.56 \pm 1.12}$	$\mathbf{86.59 \pm 1.16}$	$\mathbf{87.16 \pm 1.61}$	$\mathbf{84.96 \pm 1.88}$	85.90 ± 1.12
$LeNet5-nonlinear$	86.18 ± 1.58	84.87 ± 1.08	85.94 ± 1.42	76.78 ± 1.79	79.03 ± 1.05	79.32 ± 1.97	79.85 ± 2.07
$LeNet5-linear$	86.51 ± 1.06	85.47 ± 0.81	86.22 ± 0.97	81.02 ± 1.99	82.71 ± 1.90	81.76 ± 2.18	82.06 ± 1.20
$VGG16-nonlinear$	73.95 ± 1.16	71.16 ± 3.35	70.04 ± 12.25	44.54 ± 4.19	56.60 ± 2.64	56.10 ± 11.09	53.09 ± 1.32
$VGG16-linear$	61.51 ± 1.83	53.29 ± 15.24	56.66 ± 15.07	41.45 ± 4.14	37.51 ± 6.54	34.64 ± 8.40	45.12 ± 1.33
Number of DTs in an ETs is equal to 100 and its depth is equal to 5							
$DNF-nonlinear$	88.07 ± 0.84	86.82 ± 1.33	88.96 ± 1.34	85.06 ± 1.09	84.00 ± 1.24	82.05 ± 1.51	84.17 ± 1.58
$DNF-linear$	88.84 ± 1.23	86.54 ± 2.50	89.51 ± 2.82	87.77 ± 0.84	86.64 ± 1.32	84.86 ± 1.70	84.41 ± 2.06
$CNN1-R-nonlinear$	89.28 ± 0.98	89.41 ± 1.13	90.88 ± 0.66	89.39 ± 2.14	89.14 ± 1.04	86.22 ± 1.79	86.93 ± 1.52
$CNN1-R-linear$	90.90 ± 0.86	89.98 ± 0.93	92.06 ± 1.05	90.15 ± 0.92	90.00 ± 0.98	87.43 ± 1.47	87.89 ± 1.35
$CNN2-R-nonlinear$	90.82 ± 1.20	89.76 ± 1.59	90.89 ± 1.72	89.83 ± 1.02	90.17 ± 1.12	86.91 ± 1.62	88.55 ± 1.04
$CNN2-R-linear$	86.72 ± 16.53	$\mathbf{91.30 \pm 1.22}$	$\mathbf{92.28 \pm 1.27}$	85.31 ± 1.71	$\mathbf{91.54 \pm 0.96}$	$\mathbf{88.90 \pm 1.20}$	$\mathbf{89.50 \pm 0.80}$
$LeNet5-nonlinear$	90.31 ± 1.21	89.22 ± 1.72	90.34 ± 1.12	88.27 ± 1.35	88.93 ± 1.34	86.47 ± 1.61	87.60 ± 0.65
$LeNet5-linear$	$\mathbf{91.57 \pm 1.06}$	90.46 ± 1.42	91.53 ± 1.23	$\mathbf{90.45 \pm 1.68}$	90.74 ± 1.07	88.13 ± 1.33	88.74 ± 0.62
$VGG16-nonlinear$	61.45 ± 22.37	72.94 ± 1.74	79.47 ± 14.99	71.15 ± 18.07	72.04 ± 20.75	69.21 ± 14.58	74.85 ± 10.04
$VGG16-linear$	54.62 ± 19.63	59.64 ± 17.94	57.73 ± 18.57	46.95 ± 11.46	43.72 ± 8.71	36.73 ± 14.32	49.38 ± 9.80

one for all cascades in both geometries. More cascades are needed in the E-geometry (up to three for RFs and up to five for ETs) whereas for the R-manifold only two and three cascades are required correspondingly. The improvements in accuracy are higher in the E-geometry, while the maximum is still lower than in the case of the R-manifold. This is because the first cascade already took into account the nonlinearity of data using vectorized data from S_+^d and its linearized version. Using nonlinear R-manifold is computationally expensive. That is why it is important to know how close are accuracy maximums for CCE in both geometries. As we can see from the aforementioned figures for several datasets, the difference is not significant. This means that classification of nonlinear data can be tackled using more CCE (CRFs for instance) whereas the nonlinear manifolds are not necessary for some classification problems.



(a) gesture-raw1

(b) gesture-raw2

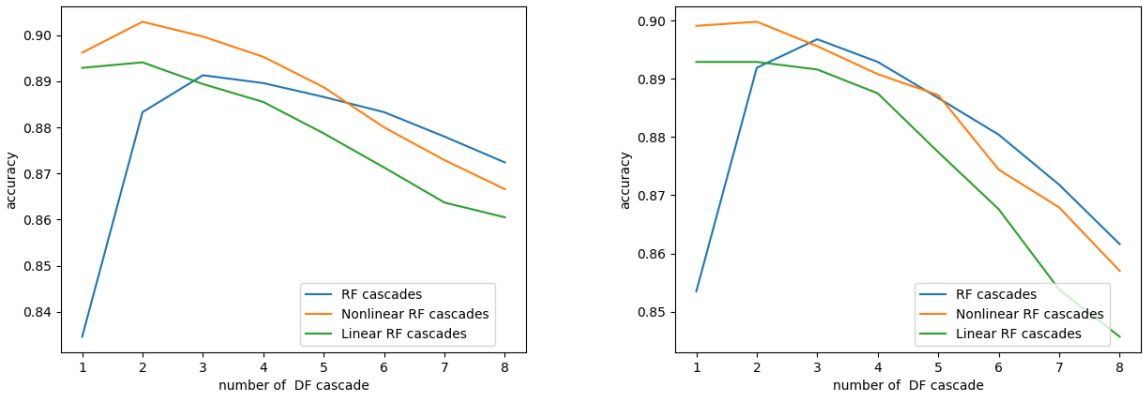
(c) gesture-raw3

(d) gesture-raw4

(e) gesture-raw5

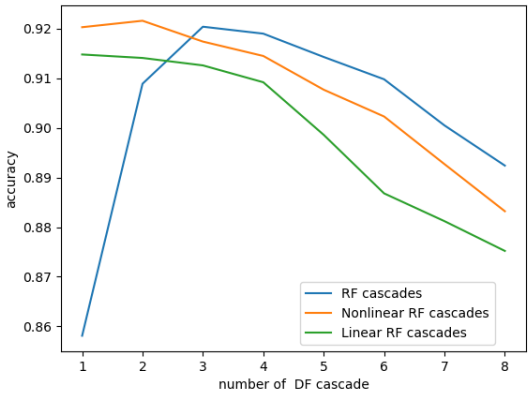
(f) gesture-raw6

Figure 34: Classification accuracy as a function of the number of CRFs plotted for six experiments from Gesture Phase Segmentation dataset. The depth of DTs in RFs is equal to 2 in all six experiments.

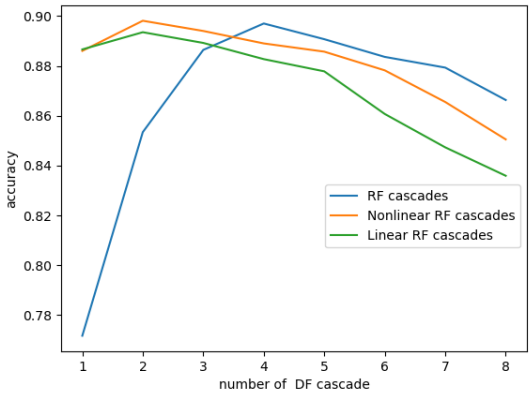


(a) gesture-raw1

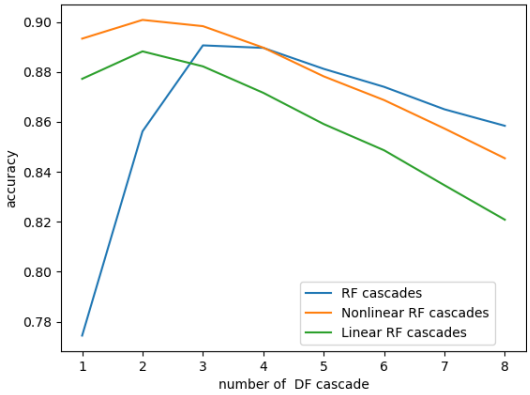
(b) gesture-raw2



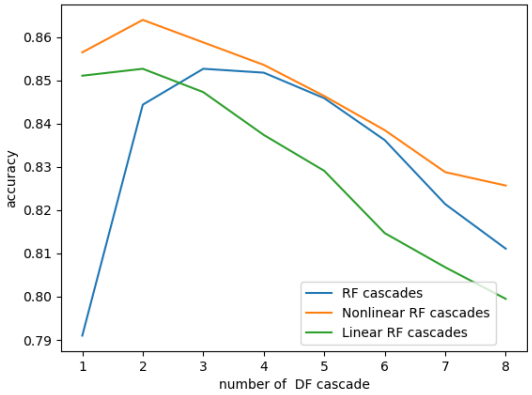
(c) gesture-raw3



(d) gesture-raw4

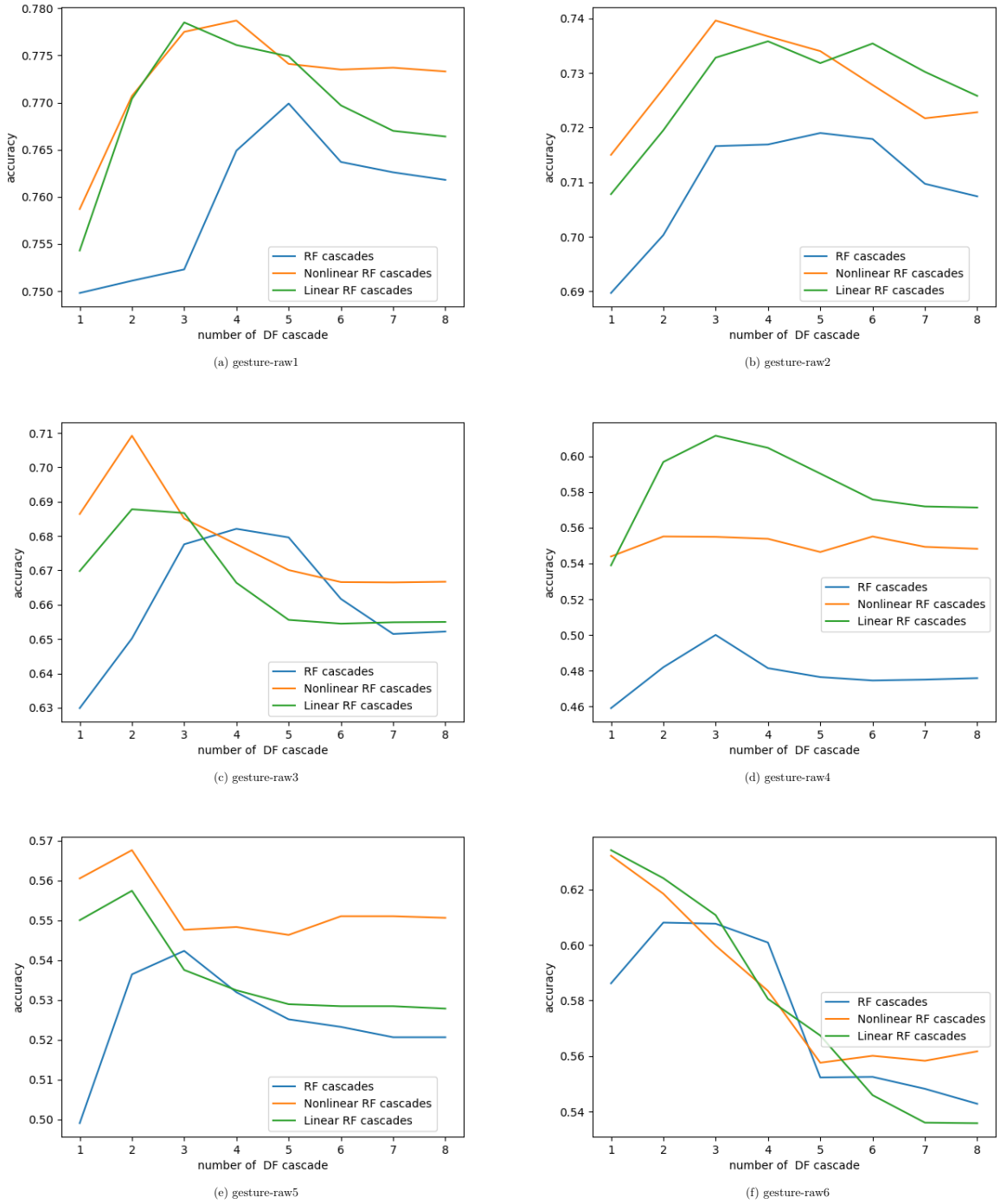


(e) gesture-raw5



(f) gesture-raw6

Figure 35: Classification accuracy as a function of the number of CRFs plotted for six experiments from Gesture Phase Segmentation dataset. The depth of DTs in RFs is equal to 5 in all six experiments.



(a) gesture-raw1

(b) gesture-raw2

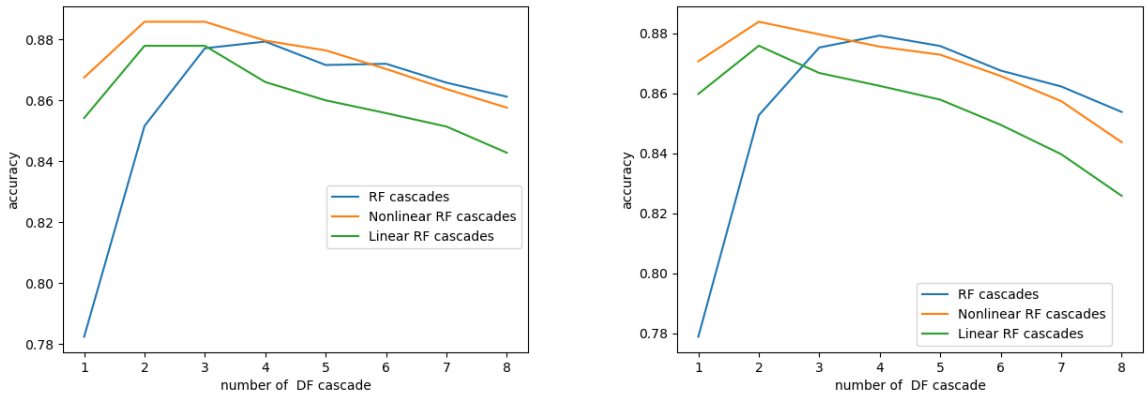
(c) gesture-raw3

(d) gesture-raw4

(e) gesture-raw5

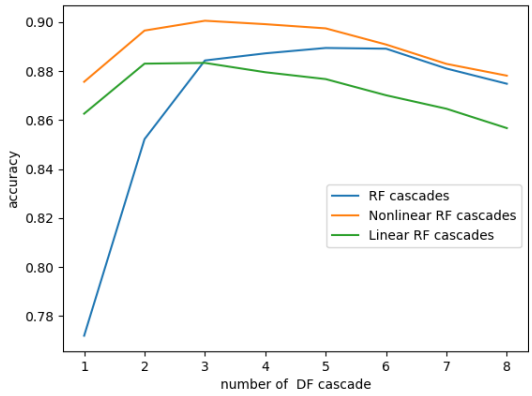
(f) gesture-raw6

Figure 36: Classification accuracy as a function of the number of cascades in CRF plotted for six experiments from Gesture Phase Segmentation dataset. The depth of DTs in ETs is equal to 2 in all six experiments.

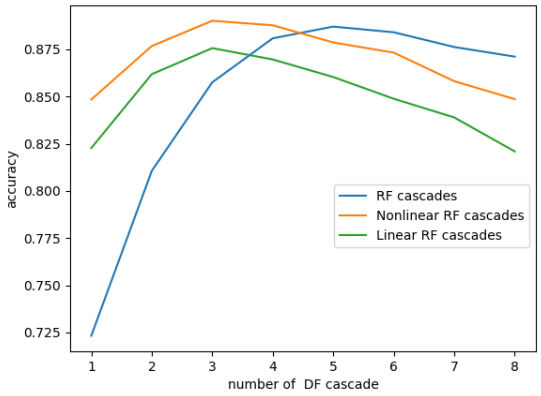


(a) gesture-raw1

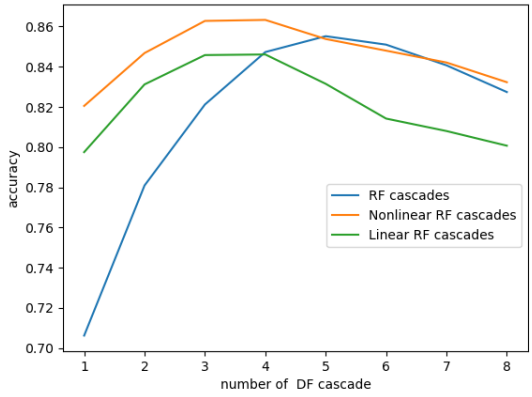
(b) gesture-raw2



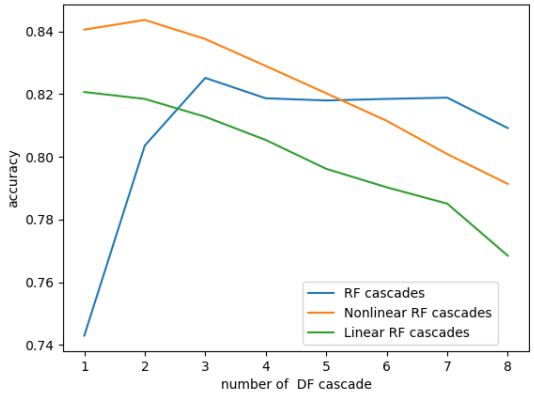
(c) gesture-raw3



(d) gesture-raw4



(e) gesture-raw5



(f) gesture-raw6

Figure 37: Classification accuracy as a function of the number of cascades in CRF plotted for six experiments from Gesture Phase Segmentation dataset. The depth of DTs in ETs is equal to 5 in all six experiments.

Table 22: Learning classifier predictions using different architectures of NNs: means and standard deviations of validation loss for each method on the raw Gesture Phase Segmentation dataset.

Method/Dataset	gesture-raw1	gesture-raw2	gesture-raw3	gesture-raw4	gesture-raw5	gesture-raw6	gesture-raw7
A number of DTs in a RFs is equal to 100 and its depth is equal to 2							
RFs_1	0.823 ± 0.015	0.861 ± 0.023	0.949 ± 0.016	1.143 ± 0.026	1.162 ± 0.012	1.111 ± 0.017	1.073 ± 0.014
RFs_{min}	0.683 ± 0.025	0.744 ± 0.039	0.821 ± 0.015	1.083 ± 0.064	1.113 ± 0.035	1.066 ± 0.028	1.033 ± 0.026
$RFs - nonlinear_{min}$	0.643 ± 0.025	0.701 ± 0.036	0.772 ± 0.014	1.009 ± 0.042	1.053 ± 0.016	0.968 ± 0.032	0.991 ± 0.023
$RFs - linear_{min}$	0.644 ± 0.024	0.700 ± 0.027	0.770 ± 0.009	0.929 ± 0.034	1.047 ± 0.025	0.955 ± 0.031	0.995 ± 0.018
$CNN1 - R - nonlinear$	0.470 ± 0.049	0.534 ± 0.027	0.481 ± 0.060	0.652 ± 0.121	0.589 ± 0.082	0.651 ± 0.041	0.644 ± 0.043
$CNN1 - R - linear$	$\mathbf{0.438 \pm 0.061}$	0.489 ± 0.034	0.449 ± 0.036	0.634 ± 0.108	0.602 ± 0.053	0.636 ± 0.023	0.624 ± 0.056
$CNN2 - R - nonlinear$	0.504 ± 0.063	0.507 ± 0.049	0.453 ± 0.055	$\mathbf{0.594 \pm 0.090}$	$\mathbf{0.517 \pm 0.053}$	0.606 ± 0.085	0.584 ± 0.048
$CNN2 - R - linear$	0.469 ± 0.045	0.477 ± 0.052	0.413 ± 0.047	0.604 ± 0.075	0.559 ± 0.043	0.581 ± 0.075	$\mathbf{0.566 \pm 0.037}$
$LeNet5 - nonlinear$	0.475 ± 0.050	0.484 ± 0.051	0.451 ± 0.041	0.633 ± 0.043	0.586 ± 0.035	0.589 ± 0.061	0.610 ± 0.053
$LeNet5 - linear$	0.466 ± 0.043	$\mathbf{0.475 \pm 0.048}$	$\mathbf{0.413 \pm 0.044}$	0.613 ± 0.042	0.570 ± 0.041	$\mathbf{0.568 \pm 0.060}$	0.570 ± 0.038
$VGG16 - nonlinear$	1.623 ± 2.828	1.730 ± 2.712	0.768 ± 0.235	2.083 ± 2.659	1.073 ± 0.155	1.135 ± 0.191	1.062 ± 0.134
$VGG16 - linear$	6.416 ± 4.365	3.983 ± 4.339	2.992 ± 3.712	2.196 ± 2.561	1.112 ± 0.169	3.614 ± 4.168	2.399 ± 3.192
Number of DTs in a RFs is equal to 100 and its depth is equal to 5							
RFs_1	0.491 ± 0.020	0.479 ± 0.023	0.490 ± 0.011	0.769 ± 0.036	0.720 ± 0.023	0.686 ± 0.015	0.660 ± 0.018
RFs_{min}	0.321 ± 0.027	0.355 ± 0.046	0.289 ± 0.024	0.344 ± 0.034	0.397 ± 0.054	0.509 ± 0.126	0.495 ± 0.035
$RFs - nonlinear_{min}$	$\mathbf{0.308 \pm 0.035}$	0.324 ± 0.031	0.266 ± 0.014	0.346 ± 0.077	0.362 ± 0.042	0.452 ± 0.053	0.447 ± 0.036
$RFs - linear_{min}$	0.329 ± 0.023	0.350 ± 0.038	0.283 ± 0.017	0.349 ± 0.043	0.402 ± 0.038	0.495 ± 0.084	0.495 ± 0.048
$CNN1 - R - nonlinear$	0.413 ± 0.030	0.471 ± 0.060	0.396 ± 0.040	0.571 ± 0.105	0.559 ± 0.129	0.617 ± 0.054	0.573 ± 0.068
$CNN1 - R - linear$	0.352 ± 0.064	0.350 ± 0.062	0.299 ± 0.039	0.384 ± 0.047	0.374 ± 0.081	$\mathbf{0.380 \pm 0.052}$	0.427 ± 0.033
$CNN2 - R - nonlinear$	0.401 ± 0.051	0.420 ± 0.060	0.350 ± 0.041	0.502 ± 0.092	0.492 ± 0.073	0.567 ± 0.101	0.537 ± 0.087
$CNN2 - R - linear$	0.323 ± 0.046	$\mathbf{0.318 \pm 0.038}$	0.271 ± 0.049	0.336 ± 0.064	0.329 ± 0.045	0.428 ± 0.080	0.399 ± 0.047
$LeNet5 - nonlinear$	0.336 ± 0.049	0.368 ± 0.052	0.282 ± 0.033	0.414 ± 0.067	0.396 ± 0.064	0.494 ± 0.088	0.454 ± 0.066
$LeNet5 - linear$	0.318 ± 0.047	0.319 ± 0.045	$\mathbf{0.249 \pm 0.039}$	$\mathbf{0.331 \pm 0.068}$	$\mathbf{0.320 \pm 0.044}$	0.388 ± 0.075	$\mathbf{0.393 \pm 0.067}$
$VGG16 - nonlinear$	3.429 ± 4.345	2.430 ± 3.746	4.768 ± 5.081	2.636 ± 3.659	0.925 ± 0.399	0.851 ± 0.255	0.959 ± 0.370
$VGG16 - linear$	1.809 ± 2.571	3.899 ± 4.144	3.872 ± 4.254	2.239 ± 2.666	1.132 ± 0.257	1.454 ± 0.189	1.218 ± 0.268
Number of DTs in an ETs is equal to 100 and its depth is equal to 2							
ETs_1	1.013 ± 0.016	1.068 ± 0.023	1.149 ± 0.012	1.206 ± 0.014	1.277 ± 0.014	1.255 ± 0.012	1.274 ± 0.016
ETs_{min}	0.690 ± 0.027	0.753 ± 0.034	0.799 ± 0.020	1.156 ± 0.075	1.135 ± 0.027	1.061 ± 0.042	1.012 ± 0.018
$ETs - nonlinear_{min}$	0.648 ± 0.029	0.716 ± 0.038	0.773 ± 0.018	1.042 ± 0.045	1.099 ± 0.033	1.022 ± 0.036	0.992 ± 0.016
$ETs - linear_{min}$	0.652 ± 0.027	0.716 ± 0.044	0.784 ± 0.016	0.984 ± 0.033	1.080 ± 0.029	1.027 ± 0.036	0.995 ± 0.016
$CNN1 - R - nonlinear$	$\mathbf{0.340 \pm 0.037}$	0.443 ± 0.030	0.428 ± 0.047	0.566 ± 0.124	0.515 ± 0.077	0.624 ± 0.048	0.577 ± 0.055
$CNN1 - R - linear$	0.388 ± 0.027	$\mathbf{0.377 \pm 0.041}$	0.354 ± 0.030	0.489 ± 0.131	0.481 ± 0.058	0.576 ± 0.035	0.562 ± 0.040
$CNN2 - R - nonlinear$	0.414 ± 0.041	0.437 ± 0.039	0.362 ± 0.042	0.520 ± 0.071	0.498 ± 0.088	0.599 ± 0.075	0.505 ± 0.073
$CNN2 - R - linear$	0.398 ± 0.026	0.428 ± 0.047	$\mathbf{0.333 \pm 0.052}$	$\mathbf{0.488 \pm 0.054}$	$\mathbf{0.481 \pm 0.055}$	0.544 ± 0.050	$\mathbf{0.478 \pm 0.048}$
$LeNet5 - nonlinear$	0.416 ± 0.039	0.449 ± 0.041	0.399 ± 0.029	0.603 ± 0.038	0.584 ± 0.043	0.614 ± 0.025	0.556 ± 0.040
$LeNet5 - linear$	0.409 ± 0.023	0.421 ± 0.023	0.37 ± 10.021	0.494 ± 0.024	0.482 ± 0.042	$\mathbf{0.532 \pm 0.039}$	0.493 ± 0.037
$VGG16 - nonlinear$	2.577 ± 3.687	0.813 ± 0.120	0.822 ± 0.323	1.216 ± 0.080	1.075 ± 0.064	1.238 ± 0.239	1.092 ± 0.193
$VGG16 - linear$	5.543 ± 4.721	3.004 ± 3.943	1.359 ± 0.193	1.224 ± 0.136	1.399 ± 0.121	1.522 ± 0.065	1.250 ± 0.216
Number of DTs in an ETs is equal to 100 and its depth is equal to 5							
ETs_1	0.701 ± 0.013	0.707 ± 0.015	0.742 ± 0.021	0.888 ± 0.019	0.917 ± 0.009	0.889 ± 0.013	0.885 ± 0.010
ETs_{min}	0.336 ± 0.023	0.366 ± 0.017	0.314 ± 0.032	0.387 ± 0.045	0.415 ± 0.033	0.519 ± 0.072	0.471 ± 0.021
$ETs - nonlinear_{min}$	0.323 ± 0.019	0.335 ± 0.021	0.298 ± 0.043	0.358 ± 0.028	0.389 ± 0.028	0.487 ± 0.070	0.435 ± 0.029
$ETs - linear_{min}$	0.363 ± 0.021	0.368 ± 0.020	0.326 ± 0.038	0.398 ± 0.066	0.441 ± 0.035	0.516 ± 0.037	0.505 ± 0.018
$CNN1 - R - nonlinear$	0.437 ± 0.027	0.488 ± 0.045	0.398 ± 0.034	0.621 ± 0.071	0.521 ± 0.084	0.642 ± 0.055	0.518 ± 0.060
$CNN1 - R - linear$	0.306 ± 0.041	0.314 ± 0.032	0.289 ± 0.016	0.383 ± 0.076	$\mathbf{0.286 \pm 0.0370}$	0.413 ± 0.040	$\mathbf{0.338 \pm 0.038}$
$CNN2 - R - nonlinear$	0.367 ± 0.046	0.443 ± 0.052	0.380 ± 0.050	0.446 ± 0.070	0.501 ± 0.052	0.575 ± 0.120	0.531 ± 0.077
$CNN2 - R - linear$	0.283 ± 0.037	0.315 ± 0.034	0.275 ± 0.049	0.279 ± 0.063	0.320 ± 0.031	0.401 ± 0.083	0.385 ± 0.059
$LeNet5 - nonlinear$	0.307 ± 0.024	0.357 ± 0.048	0.307 ± 0.039	0.371 ± 0.066	0.403 ± 0.035	0.474 ± 0.116	0.449 ± 0.058
$LeNet5 - linear$	$\mathbf{0.264 \pm 0.033}$	$\mathbf{0.313 \pm 0.043}$	$\mathbf{0.255 \pm 0.037}$	$\mathbf{0.265 \pm 0.055}$	0.291 ± 0.030	$\mathbf{0.368 \pm 0.078}$	0.360 ± 0.057
$VGG16 - nonlinear$	4.354 ± 4.416	2.489 ± 3.712	1.599 ± 2.891	3.494 ± 4.198	0.666 ± 0.053	0.919 ± 0.258	0.687 ± 0.088
$VGG16 - linear$	2.540 ± 3.542	3.929 ± 4.380	1.990 ± 2.770	1.300 ± 0.124	1.280 ± 0.213	2.424 ± 3.363	1.323 ± 0.251

5.4 Face forgery detection

5.4.1 Datasets

There are 4 datasets created to test fake videos [16, 15]:

- DFDC Training Set: 124k 10 sec videos, featuring eight facial modification algorithms.
- Preview dataset: 5k videos, featuring two facial modification algorithms.
- Public Test Set: The dataset is completely withheld and is what Kaggle's platform computes the public leaderboard against. It consists of $4,000$ 10 sec videos and contains 50% of fake videos. In the aforementioned dataset, 214 new subjects were engaged previously unseen in the training dataset. This dataset contains a new generation method StyleGAN to produce tempered faces on videos. Data augmentation has been applied to approximately 79% of videos.
- Private Test Set: This dataset is privately held outside of Kaggle's platform, and is used to compute the private leaderboard. It contains $10,000$ videos with a similar format and nature as the Training and Public Validation/Test Sets, but are real, organic videos with and without deepfakes. To build this dataset 216 more subjects have been involved unseen in two previous datasets. In this dataset, there are no forged faces generated by StyleGAN. Two new data augmentation techniques such as a dog mask and a crown flower filter have been applied for the current dataset. Same as in the Public Test Set for around 79% of videos, data augmentation techniques were applied.

Approximate gender distribution of videos: 74% female and 26% male. Ethnicity distribution: 68% Caucasian, 20% African-American, 9% East-Asian and 3% South-Asian [16].

5.4.2 Preliminary video processing

Image processing such as face detection is required because forgery is applied to a human face, and the face region should be cropped from the whole video frame. For the face detection we used the MTCNN face detector [86] which is known to be sufficiently fast. Examples of applying such a face detector to a video with a human are shown in Figure 38. The middle line of images correspond to the original face, and images up and down from it show different types of forgeries applied to the same video. The number of frames for every video is equal to 300 with $fps = 30$, so the length of the video is 10s. However, during video processing, the face detector does not detect all the faces due to poor conditions and the low quality of video frames. As seen from the aforementioned figure, the upper line of forged images are of better quality and are more challenging to detect. In opposite, the bottom line of images are of poor quality and forgery is applied to some parts of the face. Some other types of forgeries could be applied as well. However, usually, in real-world videos, the forgery applied is of rather good quality. We already know that for the DFDC challenge 8 types of forgeries have been applied and the best quality videos with forged faces are obtained using $DeepFakes$.

We also applied face alignment. We think it is important to analyze forgeries in time, and face alignment can help to reduce the effect of spatial displacements and focus on changes

Figure 38: Face forgery examples

related to the face in time. To accomplish the face alignment, we used the *dlib* library in Python. Face alignment is performed using the position of eyes, which needs eye detection. We also added some frames around the face to keep the natural background around the head after the alignment. After face detection and alignment, the size of faces is 224×224 pixels.

5.4.3 Feature extraction and training a NN to detect the face forgery

We applied VGG16 trained on the FaceNet dataset to extract features from every frame of the video. Then we write all features for all frames to the csv file. The number of features for every frame is equal to 512. In the csv file, features are written as rows and the number of rows depends on how many faces were detected in the video. To speed up the training and to avoid the temporal noise, we can use down sampling for every video in time using 20 points evenly distributed. For example, for a video where 300 faces were detected, we can use every 15th frame.

The learning process is accomplished using LSTMs with attention. It has 2 LSTM layers, one input layer, one attention layer and a classification layer. The shape of a tensor fed into pre-trained VGG16 is $224 \times 224 \times 3$. We do not use DLAs for feature extraction and classification. The size of the dataset allows using deeper and more complex architectures, but since the final validation dataset used to rank participants in the Kaggle challenge is completely unavailable, we think that using deeper architectures will lead to overfitting.

We applied the feature transformation technique using CRFs. So, for every feature vector obtained using VGG16 and that corresponds to a frame, we applied CRFs. Because face forgery detection is a binary classification problem, we can take into account only predictions

input_1 (InputLayer)	(None, 256, 256, 3)	0
conv1_1 (Conv2D)	(None, 256, 256, 64)	1792
conv1_2 (Conv2D)	(None, 256, 256, 64)	36928
pool1 (MaxPooling2D)	(None, 128, 128, 64)	0
conv2_1 (Conv2D)	(None, 128, 128, 128)	73856
conv2_2 (Conv2D)	(None, 128, 128, 128)	147584
pool2 (MaxPooling2D)	(None, 64, 64, 128)	0
conv3_1 (Conv2D)	(None, 64, 64, 256)	295168
conv3_2 (Conv2D)	(None, 64, 64, 256)	590080
conv3_3 (Conv2D)	(None, 64, 64, 256)	590080
pool3 (MaxPooling2D)	(None, 32, 32, 256)	0
conv4_1 (Conv2D)	(None, 32, 32, 512)	1180160
conv4_2 (Conv2D)	(None, 32, 32, 512)	2359808
conv4_3 (Conv2D)	(None, 32, 32, 512)	2359808
pool4 (MaxPooling2D)	(None, 16, 16, 512)	0
conv5_1 (Conv2D)	(None, 16, 16, 512)	2359808
conv5_2 (Conv2D)	(None, 16, 16, 512)	2359808
conv5_3 (Conv2D)	(None, 16, 16, 512)	2359808
pool5 (MaxPooling2D)	(None, 8, 8, 512)	0
global_average_pooling2d_1 (	(None, 512)	0

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

Figure 39: Architecture of VGG16

for one of the classes (all prediction probabilities sum up to one). To have the same size of feature vector as in the previous case we created RFs and ETs containing 512 DTs. Then those new feature vectors were fed to our LSTM model. We already know from the previous experiments that CRFs can provide a good approximation (in terms of classification accuracy or logarithmic loss) of data transformation performed by applying nonlinear R-manifold and have much lower computational complexity. This is specifically true for motion-related problems, to which face forgery detection problem belongs.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 20, 512)	0
lstm_1 (LSTM)	(None, 20, 256)	787456
lstm_3 (LSTM)	(None, 20, 32)	36992
attention_weighted_average_1	[(None, 32), (None, 20)]	32
last_dense (Dense)	(None, 1)	33

Total params: 824,513
Trainable params: 824,513
Non-trainable params: 0

Figure 40: Architecture of LSTM

5.4.4 Experiments

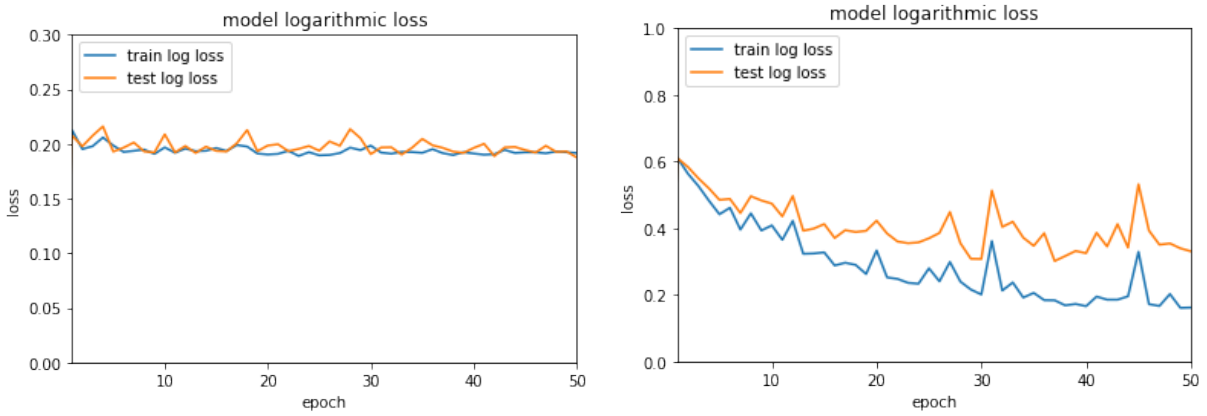
For the set of experiments, we created our own, balanced dataset of $10,000$ videos: $5,000$ fake and $5,000$ real faces. This split corresponds to splits in Public and Private Test Sets. We used a smaller dataset to avoid overfitting and to receive the preliminary results quickly. The size of our dataset is comparable to the size of Public and Private Test Sets. This way, we tried to carry out the experiments as close as possible when testing on two aforementioned datasets. At the first glance, we optimize the batch size using to train our LSTM. As optimization criteria, we use the log-loss function.

Table 23: Batch size optimization on DFDC dataset

Batch Size	Best Epoch	Training Log-Loss	Validation Log-Loss
4	21	0.277	0.301
8	25	0.210	0.322
16	33	0.169	0.310
32	41	0.138	0.315
64	43	0.163	0.350
128	33	0.032	0.335

As seen from Table 23, the lowest log-loss and the lowest overfitting has been achieved for batch size equal to 4. This means better generalization. Taking into consideration that the final validation set is completely unseen, we are going to use a batch size equal to 4 for further experiments.

From Table 24 we can see that the lowest logarithmic loss is obtained for VGG16+CRF+LSTM scheme. It is much lower than the VGG16+LSTM scheme. For the VGG16+CRF+LSTM scheme, we have applied CRF (ETs) where the number of cascades is equal to two and the depth of DTs is equal to 20. In this case, we obtained the lowest logarithmic loss. Our logarithmic loss is also lower than the loss of the first-place winner on the public validation dataset. But the most important and valuable is that our algorithm does not overfit (see



(a) VGG16+LSTM+CRF (b) VGG16+LSTM

Figure 41: Training and validation logarithmic losses for two algorithms: VGG16+CRF+LSTM and VGG16+LSTM

Table 24 and Figure 41). Among all five winners, the algorithm of the first-place winner shows the most difference in logarithmic loss when validating on public and hidden test sets, which, of course, should be taken into account when ranking all the participants. However, organizers did not take this into account. As we can see, the relatively higher logarithmic loss on a public test set does not necessarily mean that it will be proportionally higher on the hidden test set (see the results for the second-place winner who has used XceptionNet). The difference between the first and the second place is negligible and taking into account all aforementioned, we can conclude that the solution proposed by the second-place participant is more promising. Comparing all the results, we can conclude that complexity reduction via data transformation leads to less overfitting and loss decrease.

Table 24: Experimental results on DFDC dataset

Method	Training Log-Loss	Public Validation Log-Loss	Hidden Validation Log-Loss
EfficientNet B-7	– –	0.2034	0.4280
XceptionNet	– –	0.2868	0.4284
Ensemble of EfficientNets	– –	0.2297	0.4345
Ensemble of frame/video models	– –	0.2376	0.4348
Ensemble of 2D-and-3D CNNs	– –	0.2532	0.4371
VGG16+CRFs+LSTM	0.1917	**0.1876**	
VGG16+LSTM	0.2770	0.3010	

6 Conclusions and Directions for Future Research

During current research, we developed two approaches, allowing us to partially solve the overfitting problem and boost the classification accuracy. The overfitting problem is known to be one of the principal issues in ML. It is known that the main reason for overfitting is data complexity [2, 74]. Thus, reducing data complexity should lead to overfitting reduction. Overfitting can also be reduced by building classifier ensembles appropriately [74]. But even in [74], it is mentioned that the main reason for overfitting is data complexity and not a way of combining classifiers. The overfitting of every single base classifier in a classifier ensemble is less than the overfitting of the whole ensemble of base classifiers. According to VC-theory, overfitting is a linear function of the number of classifiers in a classifier ensemble. However, in [73, 74] it was shown that VC bounds can be drastically improved for every single classification problem. Thus, generally, for every single problem overfitting of a classifier ensemble is much less than what we can expect by computing VC bounds for the classifier ensemble.

The idea behind data complexity reduction by the first method consists in splitting data into functional groups depending on the classification complexity. Eliminating misclassified and easy classification patterns from training the classifiers, leaving only ambiguous ones which are of average complexity, allows dealing with the overfitting phenomenon. The general concept of the proposed approach is to use some part or category of instances that are the most general if considering different subsets of a generalized dataset. Our intuition and belief concerning this aspect are about the hypothesis that the easy for classification patterns as well as too difficult ones (misclassified) are not representatives of the class on which the trained classifiers should be focused. Easy instances can be reliably classified with relatively weak classifiers. In opposite to easy patterns, misclassified ones can not be reliably classified by most or all of the existing classifiers. To split ambiguous instances from all others, we use classifier agreement. The minimal number of classifiers needed for this is equal to two. Ambiguous patterns are those on which consensus is not reached. After that, a separate classifier is trained on those instances. The experiments show that, trained on a subset of ambiguous patterns classifiers, overfit much less than those trained on the dataset containing all groups of instances. Based on classifier agreement, we have found bounds for minimal and maximal classification error probabilities in the case of a binary classification problem. The minimal classification probability error is the probability of misclassified patterns, and the maximal one is the sum of probabilities of misclassified and half of the ambiguous patterns. To find a bound for the maximal error, we consider the case of perfectly balanced classes in the group of ambiguous patterns. This is the worst case because in this case, we have the highest entropy of the split of two classes. Comparing classifiers tested in [47] on some binary classification problems against the bound of the maximal error, we can see that some well-known classifiers have error probabilities higher than the bound of the maximal error probability. For every particular classification problem, we can split all classifiers into two groups, depending on whether they have probabilities of classification error higher or lower than the bound of the maximal error. Then we can question those classifiers that have probabilities of classification error higher than the bound of the maximal one.

The second method assumes data transformation by embedding our initial data on nonlinear manifolds (R-and-G manifolds) and then back projecting it to a linear E-space. Thus, we developed a generalized classifier stacking using interactions between classifiers or stacking of

classifier subensembles. Those interactions can not be presented using means of E-geometry and are objects that lie on a nonlinear manifold (R-manifold). This approach has linearized and nonlinearized versions. It was compared to other classifier stacking algorithms such as classical classifier stacking and recursive stacking based on CCE such as RFs and ETs. To have our experiments more fair and reliable, we use two measures: accuracy and logarithmic loss. The logarithmic loss was used as a principal measure in DFDC organized by Kaggle [15]. The approach proves to be superior to the aforementioned classifier stacking approaches in both geometries. Using CCE is especially beneficial for nonlinear problems such as gesture classification in opposite to a regular SVM. Cascades are also advantageous for the DFDC dataset. We did not apply R-manifolds for face forgery detection due to the extremely high computational load. The CCE significantly reduce data complexity, which can be seen from the experimental results. LSTM applied to transformed data using CRFs does not overfit. Notwithstanding LSTM applied to features obtained by VGG16 only, for every single frame, overfits rather significantly. Applying logarithmic loss to compare two pipelines where data transformation using CCE have been applied and another one where this data transformation has not been applied, we see that the first pipeline significantly outperforms the latter one. It also outperforms all other algorithms from DFDC, including first place. All first five winners from this competition did not propose original solutions, they have used well-known architectures such as XceptionNet, EfficientNet, Resnet and others in 2D or 3D variants. They did not perform complexity reduction, which led to enormous overfitting when their algorithms have been tested on the hidden private dataset.

CNNs proved to be superior over other classifiers when learning on R-manifolds for most of the problems from the UCI repository and Phase Gesture Segmentation dataset. Because of the structure of learning patterns (SPD matrices) DCNNs do not perform well. Even VGG16 is too deep for such kinds of objects. To this end, we have used much smaller architectures which are much less computationally expensive as well. Transfer learning and fine-tuning can not be applied when learning from patterns such as SPD matrices because predictions are unique for every single dataset and have meaning only for this dataset. The CCE perform similar to CNNs for some classification problems. The main objective of their application proposed initially in [89] and [52] was to create DLAs based on RFs and ETs that have fewer parameters and are much less computationally expensive than CNNs. At the same time, they had to have a similar level of accuracy on the most popular image datasets. DNFs did not prove their applicability for such kinds of learning instances. Indeed, they are not stable in terms of classification, although they perform relatively well for some classification problems. We believe that they can perform reliably after some modification.

The nonlinearity of the R-manifold can be regulated by choosing base classifiers with certain properties. It depends on how dissimilar classifiers in an ensemble are. The dissimilarity depends for instance on the depth of the DT as a base classifier if forest-based classifier ensembles are used. Also, it is known that ETs are characterized by more independent classifiers than RFs which additionally allows augmenting the nonlinearity of the appropriate R-and-G-manifolds. The accuracy of the individual classifiers may have some influence on the accuracy of nonlinear classifier stacking but this phenomenon should be studied separately. Indeed, the accuracy is related to the depth of DTs which determines the nonlinearity of the R-and-G-manifolds. All aforementioned before allows to have a geometrical interpretation of RFs and CRFs using nonlinear manifolds.

The number of experiments showed that the optimal number of classifiers (in terms of classification accuracy and computational load) to be involved in a classifier interaction or create small classifier subensembles from the generated pool of classifiers is equal to two. This is because of the structure of $(0, 2)$ sub-tensors of the tensors of higher orders. Using some properties of SPD matrices such as eigenvalues does not lead to any improvements in classification accuracy or logarithmic loss. We could not use nonzero elements of CS as new feature vectors because their computing requires the computation of the determinant of SPD matrices, which approaches zero with the growth of the matrix size. This leads to the impossibility to compute geometric curvature and Ricci flow (Ricci tensor) as principal characteristics of a manifold. Compression of vectorized SPD matrices using PCA or autoencoders is also not advantageous.

G-manifolds are built using information from DPs. In this case, we do not exploit the information about classifier interactions. To build G-manifolds from DPs, we need to create an orthonormal basis from these DPs. This is done by SVD. Then, every such orthonormal basis is considered as a point on a G-manifold. To measure a distance between those points, appropriate nonlinear measures have been applied. Finally, the classification is accomplished using k-NN-based classifiers.

We tested them on the general classification datasets of different scales from the UCI repository and Phase Gesture Segmentation dataset. We did not apply them to DFDC dataset as well as R-manifolds due to an extremely high computational complexity. They perform above average on the general classification datasets when compared to other most popular classifiers and their stacked versions. On the Phase Gesture Classification dataset, they are principally inferior to only CNN-based architectures learned on R-manifolds. However, it should be noticed that implementing G-manifolds is a very computationally expensive procedure. It is even more critical if we want to integrate R-and-G-manifolds (see Figure 26). According to this homotopy diagram, we first have to project points on an R-manifold to E-space and then use corresponding projection vectors as input feature vectors for CRFs. A G-manifold is built after every such cascade. In most cases, using CRFs to build a DP and then G-manifold does not lead to any boost of classification accuracy. However, we are going to optimize the algorithm and carry out some more experiments.

For future experiments, we are going to extend the idea of DNFs to the CCE and back propagate through all the cascades to learn splits on the decision nodes. We are also going to adapt DNFs for SPD matrices as learning patterns. There are some recent works where R-and-G-manifolds have been applied to build geometric-based NNs, including CNNs and generative models [26, 7, 61, 33]. We are going to adapt these approaches for CRFs to build a geometric version of CCE. We also want to carry out some research about the possibility of the usage of CCE as generative models. For some datasets such as MNIST, CIFAR-10 or CIFAR-100 and others the resolution of images is not very high: 28×28 for MNIST and 32×32 for CIFAR-10 or CIFAR-100. In the case of our SPD matrices, to reduce the size of the $(0, 2)$ tensor, we need to perform a classifier selection procedure. So, from a generated pool of classifiers (in our case DTs) we need to select a subset according to some criteria (dissimilarity, accuracy). Thus, we are going to dedicate some efforts to this subject as well.

Also, we are going to learn a mapping between different manifolds using deep generative models such as Variational Autoencoder (VAE) or Deep Convolutional Generative Adversarial Network (DCGAN). This allows us to analyze the structure of different manifolds and

determine their similarity. Moreover, this generates a manifold $\mathcal{M}_z$ from a manifold $\mathcal{M}_x$ avoiding computing predictions Z and mapping $Z \to \mathcal{M}_z$. Generating a $(0,3)$ tensor as a stack of $(0,2)$ tensors is very useful taking into account that computing a logarithm on $(0,2)$ tensors is very computationally demanding in the case of linearization and using nonlinear properties of a manifold. To see how effective and accurate the generator is, one needs to carry out a series of experiments on datasets of different scales. Generative models can also be applied for G-manifolds to avoid computationally costly SVD decomposition.

Bibliography

[1] Darius Afchar, Vincent Nozick, Junichi Yamagishi, and Isao Echizen. Mesonet: a compact facial video forgery detection network. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–7. IEEE, 2018.

[2] Mitra Basu and Tin Kam Ho. *Data Complexity in Pattern Recognition*. Springer Science & Business Media, 2006.

[3] Edwin J Beggs and Shahn Majid. *Quantum Riemannian Geometry*. Springer, 2020.

[4] Aleksandr A Borisenko and Yuri A Nikolaevskii. Grassmann manifolds and the Grassmann image of submanifolds. *Russian Mathematical Surveys*, 46(2):45–94, 1991.

[5] Pavel B Brazdil and Carlos Soares. A comparison of ranking methods for classification algorithm selection. In *European conference on machine learning*, pages 63–75. Springer, 2000.

[6] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[7] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[8] Gavin Brown. *Diversity in Neural Network Ensembles*. PhD thesis, Citeseer, 2004.

[9] Gavin Brown and Ludmila I Kuncheva. "good" and "bad" diversity in majority vote ensembles. In *International Workshop on Multiple Classifier Systems*, pages 124–133. Springer, 2010.

[10] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[11] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1251–1258, 2017.

[12] Hao Dang, Feng Liu, Joel Stehouwer, Xiaoming Liu, and Anil K Jain. On the detection of digital face manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5781–5790, 2020.

[13] Arthur P Dempster. A generalization of bayesian inference. *Journal of the Royal Statistical Society: Series B (Methodological)*, 30(2):205–232, 1968.

[14] Jiankang Deng, Jia Guo, Evangelos Ververas, Irene Kotsia, and Stefanos Zafeiriou. Retinaface: Single-shot multi-level face localisation in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5203–5212, 2020.

[15] Brian Dolhansky, Joanna Bitton, Ben Pflaum, Jikuo Lu, Russ Howes, Menglin Wang, and Cristian Canton Ferrer. The deepfake detection challenge (dfdc) dataset. *arXiv preprint arXiv:2006.07397*, 2020.

[16] Brian Dolhansky, Russ Howes, Ben Pflaum, Nicole Baram, and Cristian Canton Ferrer. The deepfake detection challenge (dfdc) preview dataset. *arXiv preprint arXiv:1910.08854*, 2019.

[17] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625–2634, 2015.

[18] Ganggang Dong and Gangyao Kuang. Target recognition in SAR images via classification on Riemannian manifolds. *IEEE Geoscience and Remote Sensing Letters*, 12(1):199–203, 2014.

[19] Nick Dufour and Andrew Gully. Contributing data to deepfake detection research. *Google AI Blog*, 1(2):3, 2019.

[20] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6202–6211, 2019.

[21] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1933–1941, 2016.

[22] Andrew Frank. Uci machine learning repository. *http://archive. ics. uci. edu/ml*, 2010.

[23] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The Annals of Statistics*, 28(2):337–407, 2000.

[24] Richard S Hamilton. Three-manifolds with positive Ricci curvature. *Journal of Differential Geometry*, 17(2):255–306, 1982.

[25] Jihun Hamm and Daniel D Lee. Grassmann discriminant analysis: a unifying view on subspace-based learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 376–383. ACM, 2008.

[26] Michael Hauser and Asok Ray. Principles of Riemannian geometry in neural networks. In *Advances in Neural Information Processing Systems*, pages 2807–2816, 2017.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference On Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[28] Tao Hu, Honggang Qi, Qingming Huang, and Yan Lu. See better before looking closer: Weakly supervised data augmentation network for fine-grained visual classification. *arXiv preprint arXiv:1901.09891*, 2019.

[29] Dong Huang and Fernando De La Torre. Facial action transfer with personalized bilinear regression. In *European Conference on Computer Vision*, pages 144–158. Springer, 2012.

[30] Yea S. Huang and Ching Y. Suen. A method of combining multiple experts for the recognition of unconstrained handwritten numerals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1):90–94, 1995.

[31] Zhiwu Huang, Ruiping Wang, Shiguang Shan, and Xilin Chen. Projection metric learning on Grassmann manifold with application to video based face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 140–149, 2015.

[32] Zhiwu Huang, Ruiping Wang, Shiguang Shan, Luc Van Gool, and Xilin Chen. Cross Euclidean-to-Riemannian metric learning with application to face recognition from video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2827–2840, 2017.

[33] Zhiwu Huang, Jiqing Wu, and Luc Van Gool. Building deep networks on Grassmann manifolds. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[34] Sadeep Jayasumana, Richard Hartley, Mathieu Salzmann, Hongdong Li, and Mehrtash Harandi. Kernel methods on Riemannian manifolds with Gaussian RBF kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(12):2464–2477, 2015.

[35] Liming Jiang, Zhengkui Guo, Wayne Wu, Zhaoyang Liu, Ziwei Liu, Chen Change Loy, Shuo Yang, Yuanjun Xiong, Wei Xia, Baoying Chen, et al. Deeperforensics challenge 2020 on real-world face forgery detection: Methods and results. *arXiv preprint arXiv:2102.09471*, 2021.

[36] Liming Jiang, Ren Li, Wayne Wu, Chen Qian, and Chen Change Loy. Deeperforensics-1.0: A large-scale dataset for real-world face forgery detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2889–2898, 2020.

[37] Boris E Kapustiy, Bogdan P Rusyn, and Vitaliy A Tayanov. Comparative analysis of different estimates of recognition probability. *Journal of Automation and Information Sciences*, 38(8):8–16, 2006.

[38] Hermann Karcher. Riemannian center of mass and mollifier smoothing. *Communications on Pure and Applied Mathematics*, 30(5):509–541, 1977.

[39] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[40] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[41] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

[42] Davis E King. Dlib-ml: A machine learning toolkit. *The Journal of Machine Learning Research*, 10:1755–1758, 2009.

[43] Albert HR Ko, Robert Sabourin, Alceu de Souza Britto, and Luiz Oliveira. Pairwise fusion matrix for combining classifiers. *Pattern Recognition*, 40(8):2198–2210, 2007.

[44] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1467–1475, 2015.

[45] 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.

[46] Ludmila I Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. John Wiley & Sons, 2004.

[47] Ludmila I Kuncheva. A bound on kappa-error diagrams for analysis of classifier ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 25(3):494–501, 2013.

[48] Ludmila I Kuncheva, James C Bezdek, and Robert PW Duin. Decision templates for multiple classifier fusion: an experimental comparison. *Pattern Recognition*, 34(2):299–314, 2001.

[49] Ludmila I Kuncheva and Christopher J Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207, 2003.

[50] John M Lee. *Introduction to Riemannian Manifolds*. Springer, 2018.

[51] Y Li, X Yang, P Sun, H Qi, and S Celeb-DF Lyu. A large-scale challenging dataset for deepfake forensics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA*, pages 14–19, 2020.

[52] Kevin Miller, Chris Hettinger, Jeffrey Humpherys, Tyler Jarvis, and David Kartchner. Forward thinking: Building deep random forests. *arXiv preprint arXiv:1705.07366*, 2017.

[53] Maher Moakher and Mourad Zéraï. The Riemannian geometry of the space of positive-definite matrices and its application to the regularization of positive-definite matrix-valued data. *Journal of Mathematical Imaging and Vision*, 40(2):171–187, 2011.

[54] Yuval Nirkin, Yosi Keller, and Tal Hassner. FSGAN: Subject agnostic face swapping and reenactment. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7184–7193, 2019.

[55] Adam Polyak, Lior Wolf, and Yaniv Taigman. TTS skins: Speaker conversion via ASR. *arXiv preprint arXiv:1904.08983*, 2019.

[56] Zhaofan Qiu, Ting Yao, and Tao Mei. Deep quantization: Encoding convolutional activations with deep generative model. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 6759–6768, 2017.

[57] Lior Rokach. Decision forest: Twenty years of research. *Information Fusion*, 27:111–125, 2016.

[58] Andreas Rossler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics++: Learning to detect manipulated facial images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1–11, 2019.

[59] BP Rusyn, VA Tayanov, and OA Lutsyka. Upper-bound estimates for classifiers based on a dissimilarity function. *Cybernetics and Systems Analysis*, 48(4):592–600, 2012.

[60] Jorge Sánchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek. Image classification with the fisher vector: Theory and practice. *International Journal of Computer Vision*, 105(3):222–245, 2013.

[61] Hang Shao, Abhishek Kumar, and P Thomas Fletcher. The Riemannian geometry of deep generative models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 315–323, 2018.

[62] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, pages 568–576, 2014.

[63] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[64] Vitaliy Tayanov, Adam Krzyżak, and Ching Suen. Some properties of consensus-based classification. In *International Conference on Computer Recognition Systems*, pages 276–285. Springer, 2017.

[65] Vitaliy Tayanov, Adam Krzyżak, and Ching Y Suen. Prediction-based classification using learning on Riemannian manifolds. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 591–596. IEEE, 2018.

[66] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

[67] Justus Thies, Michael Zollhofer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of RGB videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2387–2395, 2016.

[68] 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.

[69] Stephen B Vardeman and Max D Morris. Majority voting by independent classifiers can increase error rates. *The American Statistician*, 67(2):94–96, 2013.

[70] Vivek Veeriah, Naifan Zhuang, and Guo-Jun Qi. Differential recurrent neural networks for action recognition. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4041–4049. IEEE, 2015.

[71] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *Advances in Neural Information Processing Systems*, 29:550–558, 2016.

[72] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–511–I–518. IEEE, 2001.

[73] KV Vorontsov. Combinatorial probability and the tightness of generalization bounds. *Pattern Recognition and Image Analysis*, 18(2):243–259, 2008.

[74] KV Vorontsov. Splitting and similarity phenomena in the sets of classifiers and their effect on the probability of overfitting. *Pattern Recognition and Image Analysis*, 19(3):412–420, 2009.

[75] David H Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

[76] Zuxuan Wu, Yu-Gang Jiang, Xi Wang, Hao Ye, and Xiangyang Xue. Multi-stream multi-class fusion of deep networks for video classification. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 791–800. ACM, 2016.

[77] Zuxuan Wu, Xi Wang, Yu-Gang Jiang, Hao Ye, and Xiangyang Xue. Modeling spatial-temporal clues in a hybrid deep learning framework for video classification. In *Proceedings of the 23rd ACM International Conference on Multimedia*, pages 461–470. ACM, 2015.

[78] Xiaofeng Xie, Zhu Liang Yu, Haiping Lu, Zhenghui Gu, and Yuanqing Li. Motor imagery classification based on bilinear sub-manifold learning of symmetric positive-definite matrices. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25(6):504–516, 2016.

[79] Xin Yang, Yuezun Li, and Siwei Lyu. Exposing deep fakes using inconsistent head poses. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8261–8265. IEEE, 2019.

[80] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4694–4702, 2015.

[81] Egor Zakharov, Aliaksandra Shysheya, Egor Burkov, and Victor Lempitsky. Few-shot adversarial learning of realistic neural talking head models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9459–9468, 2019.

[82] Wei Zeng, Dimitris Samaras, and David Gu. Ricci flow for 3d shape analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):662–677, 2010.

[83] Shengxin Zha, Florian Luisier, Walter Andrews, Nitish Srivastava, and Ruslan Salakhutdinov. Exploiting image-trained CNN architectures for unconstrained video classification. *arXiv preprint arXiv:1503.04144*, 2015.

[84] Chun-Xia Zhang, Jiang-She Zhang, Nan-Nan Ji, and Gao Guo. Learning ensemble classifiers via restricted boltzmann machines. *Pattern Recognition Letters*, 36:161–170, 2014.

[85] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[86] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016.

[87] Le Zhang. *Ensemble Classification and Its Application to Visual Tracking*. PhD thesis, Nanyang Technological University, School of Electrical Electronic Engineering, 2016.

[88] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman and Hall/CRC, 2012.

[89] Zhi-Hua Zhou and Ji Feng. Deep forest: Towards an alternative to deep neural networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, pages 3553–3559, 2017.

[90] Yuri I Zhuravlev. On an algebraic approach to solving recognition or classification problems. *Problems of Cybernetics*, 33:5–68, 1978 (in Russian).

Appendices

A Riemannian and Grassman manifolds

A.1 Connection as a principal topological and geometrical property of Riemannian manifolds. Ricci flow.

For a Riemannian manifold $\mathcal{M}$ equipped with a metric g_{ij} the torsion free connection or Levi-Civita connection can be expressed using CS [50, 3]:

$$\Gamma_{ij}^{k} = \frac{1}{2} g^{k\ell} \Big(\frac{\partial g_{j\ell}}{\partial x^{i}} + \frac{\partial g_{i\ell}}{\partial x^{j}} - \frac{\partial g_{ij}}{\partial x^{\ell}} \Big), \tag{37}$$

where $g^{ij} = (g_{ij})^{-1}$. The Riemannian curvature tensor is given by

$$R_{ij\ell}^{k} = \frac{\partial \Gamma_{j\ell}^{k}}{\partial x^{i}} - \frac{\partial \Gamma_{i\ell}^{k}}{\partial x^{j}} + \Gamma_{ip}^{k} \Gamma_{j\ell}^{p} - \Gamma_{jp}^{k} \Gamma_{i\ell}^{p}. \tag{38}$$

By lowering the third index we get

$$R_{ijk\ell} = g_{kp} R_{ij\ell}^{p}. \tag{39}$$

The curvature tensor $R_{ijk\ell}$ is anti-symmetric in the pairs i, j and k, ℓ and symmetric in their interchange:

$$R_{ijk\ell} = -R_{jik\ell} = -R_{ij\ell k} = R_{k\ell ij}. \tag{40}$$

According to the first Bianchi identity we have

$$R_{ijk\ell} + R_{jki\ell} + R_{kij\ell} = 0. \tag{41}$$

The Ricci tensor is the contraction

$$R_{ik} = g^{j\ell} R_{ijk\ell}. \tag{42}$$

The scalar curvature is

$$R = g^{ij} R_{ij}. \tag{43}$$

In case of SPD matrices the scalar curvature depends only on the size of a diagonal n [53]:

$$R(n) = \frac{1}{8} n(n-1)(n+2). \tag{44}$$

Ricci flow by Hamilton [24] is the evaluation equation

$$\frac{\partial g_{ij}}{\partial t} = -2R_{ij}. \tag{45}$$

A Riemannian metric g_{ij} is called **Einstein** if for some constant λ

$$R_{ij} = \lambda g_{ij}. \tag{46}$$

A smooth manifold with an Einstein metric is called an **Einstein manifold**.

A.2 Riemannian manifolds of SPD matrices

A pair of Gauss maps for the affine invariant metric can be written as follows [18, 78, 34]

$$\begin{aligned} \exp_{\mathbf{P}}(\mathbf{\Delta}) &= \mathbf{P}^{\frac{1}{2}} \exp\left(\mathbf{P}^{-\frac{1}{2}} \mathbf{\Delta} \mathbf{P}^{-\frac{1}{2}}\right) \mathbf{P}^{\frac{1}{2}} = \mathbf{Q} \\ \log_{\mathbf{P}}(\mathbf{Q}) &= \mathbf{P}^{\frac{1}{2}} \log\left(\mathbf{P}^{-\frac{1}{2}} \mathbf{Q} \mathbf{P}^{-\frac{1}{2}}\right) \mathbf{P}^{\frac{1}{2}} \end{aligned} \tag{47}$$

and for the log Euclidean metric [32]

$$\begin{aligned} \exp_{\mathbf{P}}(\mathbf{\Delta}) &= \exp(\log(\mathbf{P}) + \Delta) = \mathbf{Q} \\ \log_{\mathbf{P}}(\mathbf{Q}) &= \log(\mathbf{Q}) - \log(\mathbf{P}) = \Delta \end{aligned} \tag{48}$$

The length of a geodesic on Sym_+^d in case of affine invariant metric can be computed as

$$d(\mathbf{P}, \mathbf{Q}) = \sqrt{tr[\log^2\left(\mathbf{P}^{-\frac{1}{2}} \mathbf{Q} \mathbf{P}^{-\frac{1}{2}}\right)]} \tag{49}$$

and for the log Euclidean metric as

$$d(\mathbf{P}, \mathbf{Q}) = ||\log(\mathbf{Q}) - \log(\mathbf{P})|| \tag{50}$$

Alternative computation of geodesic's length using eigenvalues can be done as follows,

$$d(\mathbf{P}, \mathbf{Q}) = \sqrt{\sum_{i=1}^{n} \ln^2 \lambda_i(\mathbf{P}, \mathbf{Q})}, \tag{51}$$

where $\{\lambda_i(\mathbf{P}, \mathbf{Q})\}_{i=1}^{n}$ are generalized eigenvalues of $\mathbf{P}$ and $\mathbf{Q}$ computed from the equation $\lambda_i \mathbf{P}\mathbf{x}_i - \mathbf{Q}\mathbf{x}_i = 0, i = 1, ..., d, \mathbf{x}_i \neq 0$ are generalized eigenvectors.

A.3 Grassmann manifolds

Projection distance between two subspaces $[Y_1], [Y_2]$ presented as matrices with D orthogonal columns ($m \times D$ matrices) is given as

$$d_P([Y_1], [Y_2]) = 2^{-1/2} ||\mathbf{Y}_1\mathbf{Y}_1^T - \mathbf{Y}_2\mathbf{Y}_2^T||_F. \tag{52}$$

As seen from 52 computation of geodesic's length is rather computationally expensive. To deal with this problem, one uses properties of the matrix trace and of the fact that $Y_1^T Y 1 = Y_2^T Y_2 = I_D$ [34]:

$$d_P^2([Y_1], [Y_2]) = 2^{-1} ||\mathbf{Y}_1\mathbf{Y}_1^T - \mathbf{Y}_2\mathbf{Y}_2^T||_F^2 = D - ||\mathbf{Y}_1^T\mathbf{Y}_2||_F^2. \tag{53}$$

It is possible to define more metrics using angles θ_i between two subspaces $\mathbf{Y}_1$ and $\mathbf{Y}_2$. Then a number of distance measures can be applied. The angle can be computed using SVD. Using SVD we can write $\mathbf{Y}\mathbf{Y}^T = \mathbf{U}(\Lambda)\mathbf{U}^T$, where Λ is the diagonal matrix of principal angles

$$\Lambda = diag(\cos(\theta_1), \cos(\theta_2), ..., \cos(\theta_n)),$$

$\mathbf{U}$ and $\mathbf{U}^T$ are left and right eigenvectors. Elements of the diagonal matrix Λ are known as canonical correlations. For illustration, see Figure 42.

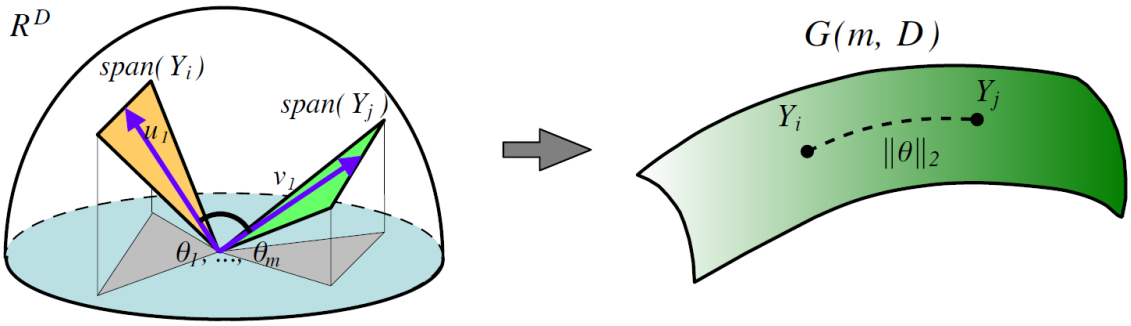


Figure 42: Principal angles between two subspaces and how to compute distances on a Grassmann manifold

Using notation of the angle between two subspaces Y_1 and Y_2 the arc length (geodesic) is given by

$$d_A(\mathbf{Y}_1, \mathbf{Y}_2) = \Big(\sum_i \theta_i^2\Big)^{1/2} \tag{54}$$

Projection metric is defined as follows:

$$d_P(\mathbf{Y}_1, \mathbf{Y}_2) = \Big(\sum_{i=1}^{m} \sin^2\theta_i\Big)^{1/2}. \tag{55}$$

Then Binet-Cauchy metric

$$d_{BC}(\mathbf{Y}_1, \mathbf{Y}_2) = \Big(1 - \prod_i \cos^2 \theta_i\Big)^{1/2}. \tag{56}$$

Procrustes metric

$$d_{CF}(\mathbf{Y}_1, \mathbf{Y}_2) = 2\Big(\sum_{i=1}^{m} \sin^2(\theta_i/2)\Big)^{1/2}. \tag{57}$$

Finally max and min correlations can be found as

$$d_{max}(\mathbf{Y}_1, \mathbf{Y}_2) = \Big(1 - \cos^2 \theta_1\Big)^{1/2} = \sin \theta_1. \tag{58}$$

$$d_{min}(\mathbf{Y}_1, \mathbf{Y}_2) = \Big(1 - \cos^2 \theta_m\Big)^{1/2} = \sin \theta_m. \tag{59}$$

B Algorithms

Algorithm 1 Part 1: data transformation procedures

procedure ComputeTensorNonlinear(X)
  for $k \leftarrow 1, |X|$ **do**
    for $\ell \leftarrow 1, L$ **do**
      for $i \leftarrow 1, T$ **do**
        for $j \leftarrow 1, T$ **do**
          $T(k)(l)(i)(j) = h_i^\ell(x_k)h_j^\ell(x_k)$
        end for
      end for
    end for
    return T
end procedure
procedure ComputeTensorLinear(X)
  for all $k \in |X|$ **do**
    for all $\ell \in L$ **do**
      $T(k)(l) \leftarrow \log(\mathbf{A}^\ell(x_k))$
    end for
  end for
  return T
end procedure
procedure TensorVectorizing($\mathbf{T}$)
  for $k \leftarrow 1, |X|$ **do**
    $n \leftarrow 1$
    for $\ell \leftarrow 1, L$ **do**
      for $i \leftarrow 1, T$ **do**
        for $j \leftarrow i, T$ **do**
          if $j = i$ **then**
            $Z(k)(n) = T(k)(l)(i)(j)$
            $n \leftarrow n + 1$
          end if
          if $j > i$ **then**
            $Z(k)(n) = T(k)(l)(i)(j)\sqrt{2}$
            $n \leftarrow n + 1$
          end if
        end for
      end for
    end for
  end for
  return Z
end procedure

Algorithm 2 Part2: learning from classifier interactions on R-manifolds

Require: $\mathbf{T}, \tilde{\mathbf{T}}, \mathbf{Z}, \tilde{\mathbf{Z}}$ ▷ Linear and nonlinear versions
$CNN \leftarrow \mathbf{T}$
$CNN \leftarrow \tilde{\mathbf{T}}$
$SVM(RBF) \leftarrow \mathbf{Z}$
$SVM(RBF) \leftarrow \tilde{\mathbf{Z}}$
$kNN(k = 3) \leftarrow \mathbf{Z}$
$kNN(k = 3) \leftarrow \tilde{\mathbf{Z}}$
$MLP \leftarrow \mathbf{Z}$
$MLP \leftarrow \tilde{\mathbf{Z}}$

Algorithm 3 Learning from decision profiles on G-manifolds

Require: DP_t, DP_s ▷ Decision profiles
procedure GC(DP_t, DP_s)
  for $j \leftarrow 1, |X_t|$ **do**
    $\{S, U, V\} \leftarrow SVD(DP_t(i))$
    $T_t(i) \leftarrow V$
  end for
  for $i \leftarrow 1, |X_s|$ **do**
    $\{S, U, V\} \leftarrow SVD(DP_s(i))$
    $T_s(i) \leftarrow V$
  end for
  for $i \leftarrow 1, |X_s|$ **do**
    for $j \leftarrow 1, |X_t|$ **do**
      $\{S, U, V\} \leftarrow SVD(T_t(i)T_s(j)^T)$
      $d(i) \leftarrow Geodesic(U)$
    end for
    $kNN \leftarrow d(i)$
  end for
end procedure
procedure GEODESIC(U)
  $d \leftarrow 0$
  for $i \leftarrow 1, L$ **do**
    $d \leftarrow d + \arccos^2(U_i)$
  end for
  $d \leftarrow \sqrt{d}$
end procedure
procedure PROJECTION(U)
  $d \leftarrow 0$
  for $i \leftarrow 1, L$ **do**
    $d \leftarrow d + U_i^2$
  end for
  $d \leftarrow \sqrt{L - d}$
end procedure
procedure BINETCAUCHY(U)
  $d \leftarrow 1$
  for $i \leftarrow 1, L$ **do**
    $d \leftarrow d * U_i^2$
  end for
  $d \leftarrow \sqrt{1 - d}$
end procedure