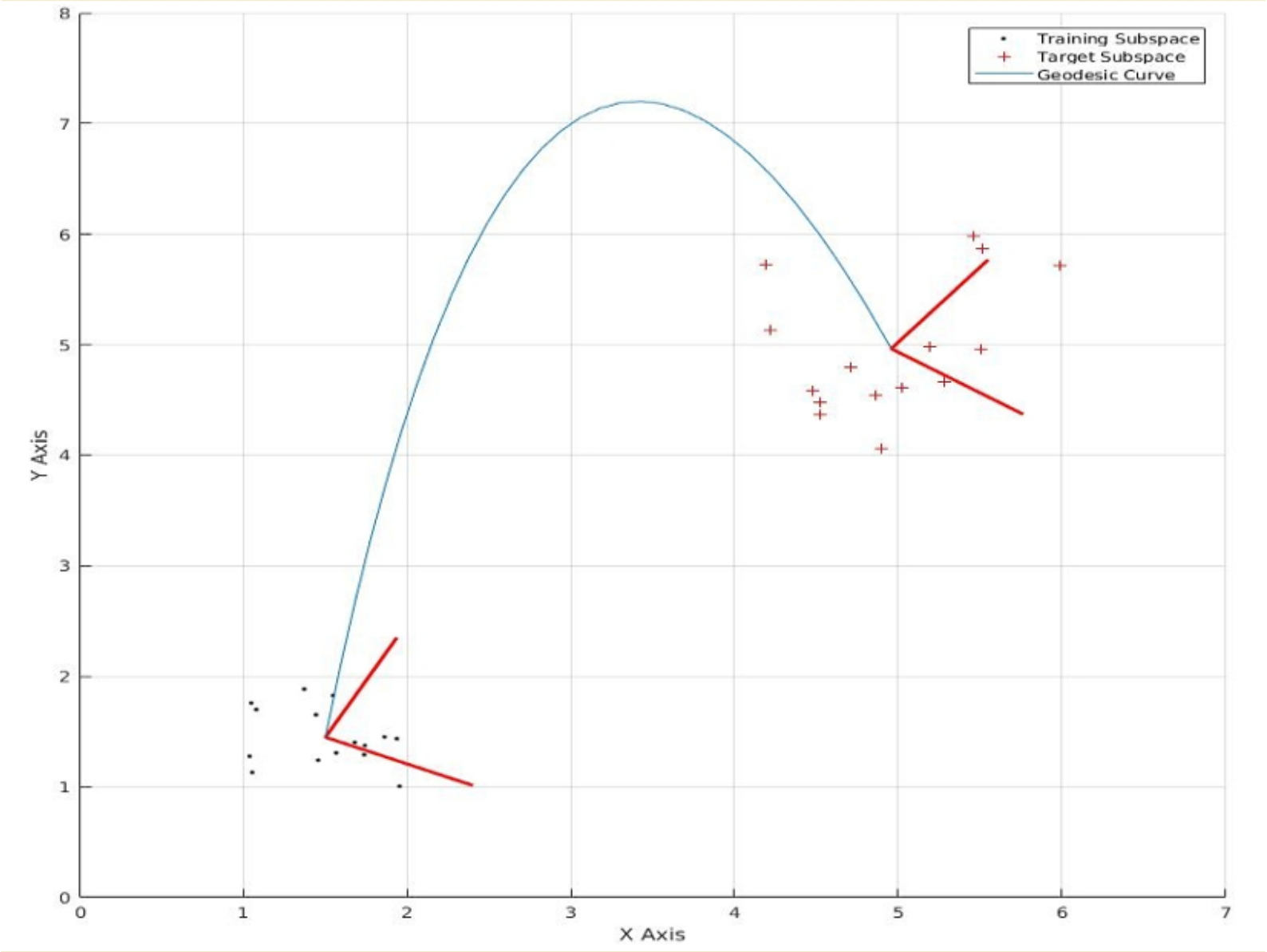


José Miguel Mateus Maurício de Carvalho

Traffic Surveillance using Visual Domains Adaptation

September 2017

· U C ·

Universidade de Coimbra

Department of Electrical and Computer Engineering,
Faculty of Sciences and Technology, University of Coimbra,

A Dissertation for Graduate Study in MSc Program
Master of Science in Electrical and Computer Engineering

Traffic Surveillance using Visual Domains Adaptation

José Miguel Mateus Maurício de Carvalho

Research Developed Under Advisory of:
Prof. Jorge Manuel Moreira de Campos Pereira Batista

Jury:
Prof. Doutor Fernando Manuel dos Santos Perdigão
Prof. Doutor Paulo Jorge Carvalho Menezes
Prof. Doutor Jorge Manuel Moreira de Campos Pereira Batista

September 2017

Work developed in the Institute of Systems and Robotics of the University of Coimbra.

Thanks

I would like to start by thanking my mom and dad. Without their guidance and full support I would not be the man I am today and wouldn't have made it this far. This work is most of all dedicated to them.

Secondly I would like to thank my sister for showing me that no matter the adversities and unfavorable odds, we can push forward and reach our objectives.

I would like to thank my supervisor Professor Doutor Jorge Batista for accepting me as his master student as well as for the guidance and suggestions he gave me through the development of this work.

A special thanks to my colleagues in the Brisa Project Laboratory. A big thanks to the laboratory master, Pedro Martins, for the good welcoming and the knowledge passed on when it was needed. Thanks to André Ferreirinha for the insights and years of friendship, inside and outside the department walls. Thanks to Rafael Duarte for the help getting acquaintance to the operating system and for being a real firefighter when was needed. It was a good place to develop this work and achieve my master degree.

Lastly, I would like to thank all my other friends and family for believing in me even when I didn't. You all define the person I am today and this work is, in one way or another, a contribution of you all.

Form the bottom of my heart, thank you all.

Abstract

Over the past decade, traffic surveillance systems development have attracted the interest of many in the computer vision community. Mainly due to possible improvement of drivers security such as implementations in systems capable of predict real-time accidents, detection of infractions on roads or even time and fuel reduction by selecting the right way of traveling. The use of computer vision techniques to monitoring traffic as proven to be a non-invasive, cost effective, automated option when it comes to traffic surveillance. The challenge today is to efficiently develop an Intelligent Transportation System capable of real-time detecting roads with high affluence of traffic and for example, sending that information so that drivers can choose another way in advance, make an efficient and autonomous management of traffic lights, or in extreme scenarios, like a car accident, a system that automatically notifies authorities to provide quicker medical assistance.

The purpose of this work is to implement some visual domain adaptation based approaches when it comes to identify the existence or not of a vehicle in an intersection. To accomplish the purpose of adapting dynamic events on traffic surveillance, or similar tasks, we conducted along this thesis several approaches with holistic classification exploring domain adaptation of evolutionary events to some GIST features extracted from the dataset images and also apply the same approaches on AlexNet neural networks features of the same dataset images. This approaches are being implemented in order to be used on situations where a dynamic evolution of domains is needed and where we have an unlabeled target data.

Keywords: Traffic Surveillance, Domain Adaptation, Smooth Interpolation, Subspace Update, Holistic Classification.

Resumo

Durante a última decada, o desenvolvimento de sistemas direcionados para controlo e manutenção de trafego tem desplotado imenso interesse na comunidade de visão por computadores. Isto deve-se muito ao facto do grande número de oportunidades no melhoramento de técnicas para segurança dos condutores, como por exemplo, predição em tempo real de acidentes, deteção de comportamentos ilegais nas estradas ou até aplicar estas técnicas a aplicações que permitam poupar tempo e combustível ao escolher o melhor caminho. A utilização destas técnicas para monitorização de trafego tem provado ser uma opção não invasiva, barata e autonoma. Hoje em dia é bastante desejado um sistema inteligente capaz de monitorizar em tempo real densidade de tráfego para que com antecedência se possam calcular novas rotas para que condutores evitem tráfego indesejado, outra aplicação será a gestão automática de semáforos, ou até em casos mais extremos, fazer a predição de acidentes e em caso de acidente notifique as autoridades para que possam as pessoas envolvidas possam receber cuidados médicos o mais rápido possível.
Este trabalho tem como propósito a implementação de abordagens de adaptação de domínios visuais para a detecção de veículos em imagens na aproximação de um cruzamento. Para atingir os nossos objectivo de adaptar dinamicamente eventos relacionados com monitorização de trafego, implementámos algumas abordagens baseadas em classificação holística para explicar a adaptação evolutiva de domínios, inicialmente aplicadas a caracteristicas GIST extraídas das imagens incluídas no dataset utilizado. Posteriormente aplicamos as mesmas abordagens a características extraídas com a ajuda da rede neuronal AlexNet. Estas abordagens que estamos a implementar pretendem ser aplicadas em situações onde se seja necessária uma evolução dinâmica de domínios e onde temos dados sem labels no treino.

Palavras-Chave: Monitorização de Tragego, Adaptação de Domínios, Interpolação Suave, Classificação Holística.

Acronyms and symbols

Abbreviation	Meaning
2D	Two-dimensional space
3D	Three-dimensional space
CMA	Continuous Manifold Adaptation
DA	Domain Adaptation
GFK	Geodesic Flow Kernel
KNN	K-Nearest Neighbor
MDA	Marginalized Denoised Autoencoder
ML	Machine Learning
PCA	Principal Components Analysis
SA	Subspace Alignment
SIC	Smooth Interpolation Curve
SS	Semi-Supervised DA
SU	Subspace Update
SVD	Single Vector Decomposition
SVM	Support Vector Machine
TDA	Transductive Domain Adaptation
TL	Transfer Learning
US	Unsupervised DA

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The wheel was one of the great inventions of humankind, some even say it was THE invention. With it the world became an open road to the nomad tribes that could finally walk for hundreds of miles a lot faster than before and were able to take with them more food and cloth to survive in harsh weathers.

Since then, the world has vastly evolve and now we have a lot of vehicles circulating the roads of our cities. The invention of the automobile has transformed modern life by being a beacon of freedom, mobility and comfort. However due to the population exodus to big cities there's been an exponential increasing on traffic affluence within those cities.

Due to this ever-increasing traffic demand, modern societies with well-planned road management systems, and sufficient infrastructure for transportation still face the problem of traffic congestion. This results in loss of travel time, and societal and economic costs. One of the solutions for this problem might be the construction of new roads but due to political and environmental concerns, that is often the less feasible possibility. A more elegant way would be to make more efficient use of the existing infrastructure. This latter approach is getting a lot of interest in the computer vision community, due to its high number of possibilities.

Traffic management and control approaches are used to control the traffic flows and to prevent or reduce traffic jams, or more generally to improve the performance of traffic systems. Possible performance measures in this context are travel time, safety, fuel consumption, emissions, etc.

In recent years there was also a rapidly increasing of digital cameras, computation power, and

new techniques in video compression standards has led to a recent growth of video digital content. Additionally, in the last few years there has been a fast increase of Internet-connect cameras, not only for personal but also for commercial use. A significant amount of them are being used around the world for traffic surveillance cameras. The data received from these cameras can be analyzed in order to prevent traffic jams, road kills and overall improve road safety. Several developed countries have already adopted many real-time technology approaches to address this issue that is road safety. Today's challenge is to have more and more developed countries doing the same and replicate it in underdeveloped ones.

The objective of this work is to identify the existence or not of a vehicle within a certain image, in this specific case, an oncoming intersection. Doing this classification will allow future implementation such as management of traffic or traffic lights in intersections.

Fig. 1.1: Intersection used for the experiment

This can be achieved with the traditional vehicle detection methods using descriptors such as HOG (Histogram of Oriented Gradients), SIFT (Scale-Invariant Feature Transform), etc. Normally this methods are embedded within certain approaches usually divided into two distinguished stages, which consists in first detecting the vehicle and then analyze, if needed, the classification of the vehicle or an abnormal behavior. This is how the traditional computer vision techniques work when it comes to detection of an object, in this case, a vehicle. Every one of

this approaches can be implemented in a real-time continuous environment but there's a main drawback caused by the problem this approaches have from the acquisitions variation such as illumination, light saturation, fog, snow, etc. This non interrupting time domain as well as the constant variation of the environment, causes the need for domain adaptation methods that can counter those variations and differences. Unless the detection method is robust and can extract features capable of mitigate those domain variation,i.e., those features are invariant to variations of the domain, we have to use approaches that, as time goes by, based on new observations can change and adapt their detection features (e.g. dynamic background subtraction).

In recent years, there has been an increase on holistic[1] approaches. In this case, instead of trying to detect the car and extract the features, we extract the features of the whole image and all the environment within the image. This approaches basically receives an image, which we say that's positive (with car) or negative (without car), extract the features of the all scene and then understand within those features what differs and what really distinguish one image from the other.In other words, instead of detection and tracking these approaches try to make a pattern recognition with all the image.

Fig. 1.2: Examples of images with and without cars with only 12 minutes apart. This small timespan allows us to limit the environmental differences on the environment.

[1]Holistic means to be concern with wholes or with complete system rather than with the analysis of only parts of the system.

If we apply any of those approaches to these two images we get a vector of features for each one of them. In our thesis we will be using a descriptor called GIST.
Given an input image, a GIST descriptor is computed by

1. Convolve the images with 32 Gabor filters at 4 scales, 8 orientation, producing 32 features maps of the same size of the input image.

2. Divide each feature map into 16 regions (4×4 grid), and then average the feature values within each region.

3. Concatenate the 16 averaged values of all 32 feature maps, resulting in a $16 \times 32 = 512$ GIST descriptor.

The discriminative ability of GIST for one of those 512 feature vectors can be expressed with the help of a bar plot. If we see the following image, the differences are pretty clear.

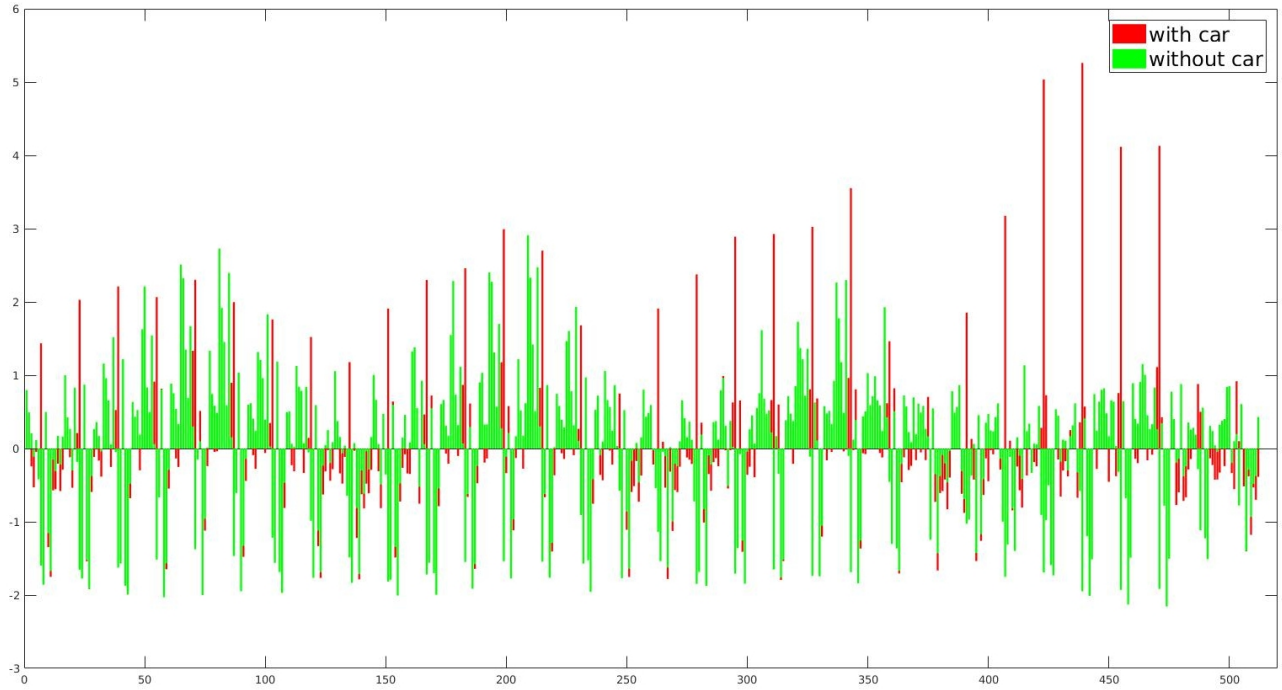


Fig. 1.3: GIST features extracted from images above in form of a graphic

As we can see in the graphic 1.3, the vector for an image with car (red bars) is not so far apart from the vector without a car (green bars), this was expected since the images are not very different. However, if we look closely to the region between 350 and 512 we can see a clear difference between the red and green bars, as ones go higher to the positive side and others go

lower to the negative side.

Neural networks trained with a great number of images (thousands or millions of pictures) can usually make a robust descriptor since it can arrange a generalization from all the training data. Due to the lack of training data and the temporal requirements for the method to function, this approaches require new methods for domain adaptation in order to dynamically adapt the classifiers to the intrinsic changes within the descriptors.

This line of thought gives us our objective for this thesis. The main objective is to explore a bunch of approaches capable of dynamic domain adaptation in a temporal interval.

1.2 Summary of the work developed

In the pursuing of this goals, having an approach capable of classifying in real-time and with domain variation, we are going to use as foundation the work proposed by Judy Hoffman et al. at [14]. This work was possibly one of the groundbreaking works in evolutionary domain adaptation. It explores the representation of feature points into a subspace that in turn is a single point on a Grassman manifold. The theory behind subspaces and the Grassam manifold is explained further on the thesis.

Basically, their approach is to represent a dynamic adaptation between two subspaces, one for training data and another for testing data, and there's a continuously adaptation of the testing subspace (see figure 1.4).

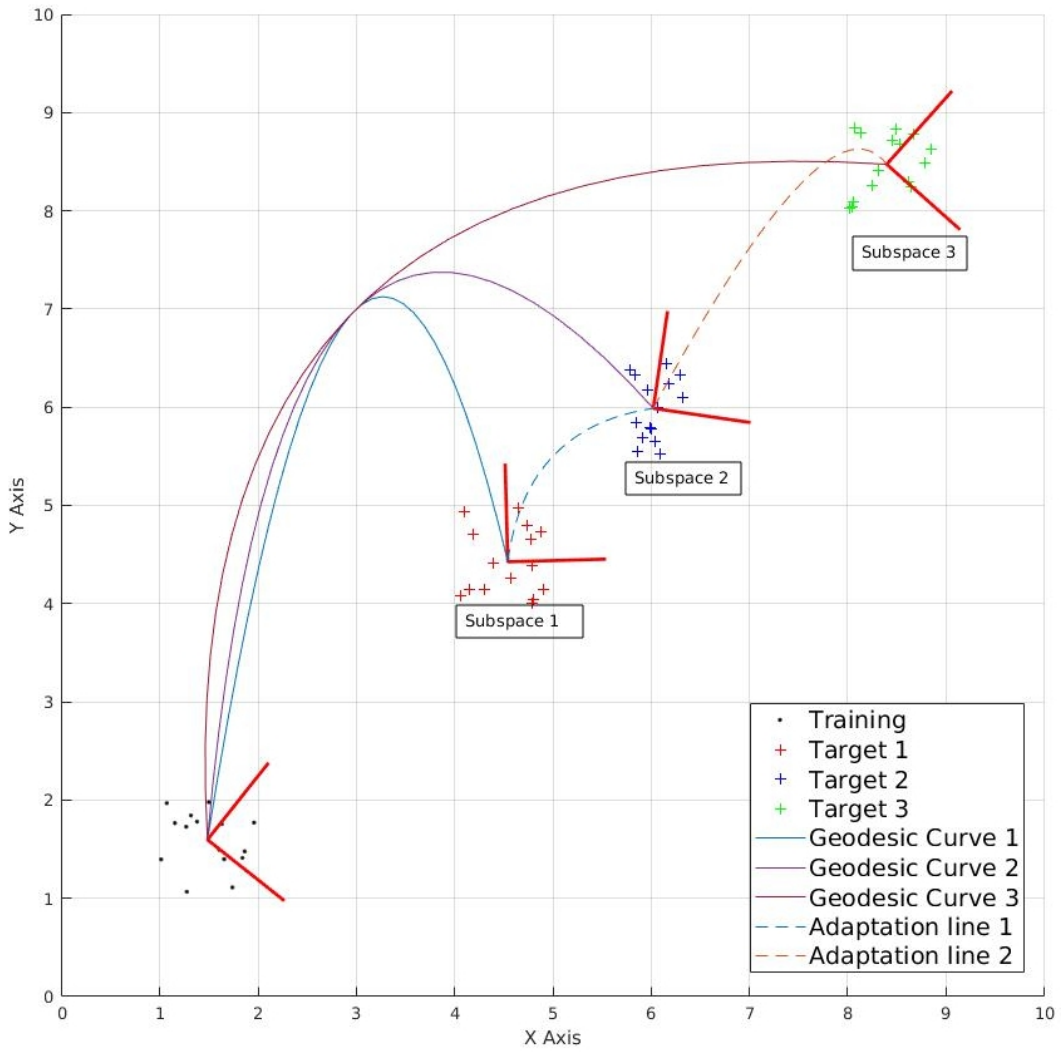


Fig. 1.4: Training and Adapting Target Subspaces on a Grassman Manifold.

This approach is much more generalist and the objective is to, with this adaptation of the testing subspace, find a geodesic[2] that better represents the testing subspace on the target subspace.
Hoffman's work is based on the notion that only the target subspace is being adapted. Our work is a little divergent from theirs.

Taking into consideration that the application we want to achieve has a cyclical meta-descriptor[3] and the objective is to classify a set of images during an whole day, what we are going to do is, instead of having only one training subspace, with the help of this meta-descriptor we can group samples into distinct subspaces. What we hope to achieve with this is that each subspace can be more representative accordingly to the samples within it, something we think can never happen if we have only one subspace representing all the distinct samples from an whole day.
With this multiple training subspaces, we'll use an approach developed at the Institute of System and Robotics consisting of a smooth interpolation between all those source subspaces [1]. Basically, we propose a different approach from Hoffman (adaptation of target subspace only), we propose a continuously adaption of the four training subspaces computed with the help of Time, our meta-parameter (as we can see in figure 1.5). This will require each time to compute new interpolations in order to project the new samples and proceed to the classification of those same samples.

[2]A geodesic is a generalization of the notion of a "straight line" to "curved spaces". In this case, a geodesic is the shortest line between two points on a Grassman manifold

[3]In our case, Time. (the hours of a day)

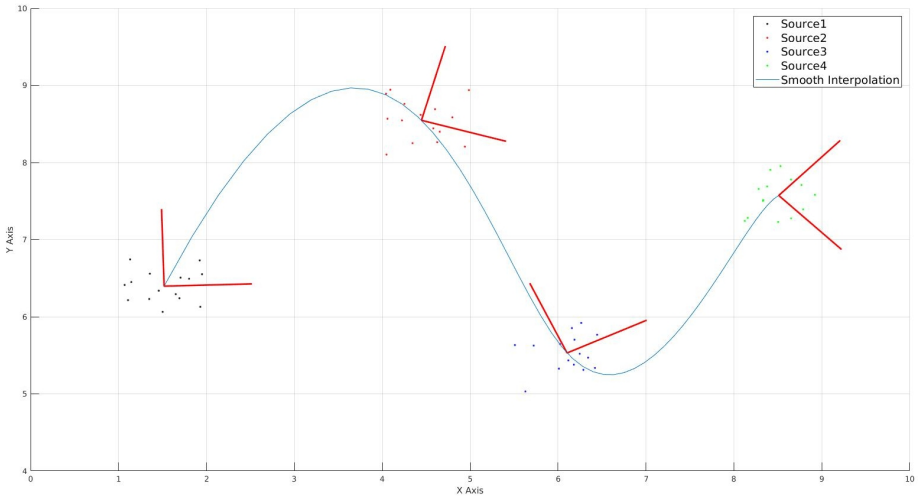


Fig. 1.5: Figure representing the four training subspaces and the smooth interpolation line

Finally we take a look at a method called Transductive Doman Adaptation [5], explained further in the thesis, that basically allows us from a unsupervised testing environment extract labels that we can later incorporate into the source subspaces, thus increasing the variety of those same sources which we hope will improve the performance of the classifier for new samples.

1.3 Outline of the thesis

- Chapter 2: State of the Art. Goes over relevant research related to Visual Domain Adaptation and Deep Learning oriented for domain adaptation.

- Chapter 3: Math Used on the Different Approaches. It presents the math theory on which we support our approaches. It gives us an insight on how the methods theory works, mathematical speaking.

- Chapter 4: Description of the Used Approaches. Is similar to chapter 3 but here we give an insight on how the methods and approaches really work and how they are implemented. Additionally we also have some code and pseudo-code to make it easier to understand the functionality of the code itself.

- Chapter 5: Development and Results. How it was developed, the dataset used with figures examples as well as the results obtained from the experiments and an evaluation of those same results.

- Chapter 6: Conclusions and Future Work. Presents the final conclusions from the work and suggested some possible future work.

Chapter 2

State of the Art

In recent years, video monitoring and surveillance systems have been widely used in traffic management, extracting useful information such as traffic density and vehicle types. Development of intelligent systems that extract traffic density and vehicle classification information from traffic surveillance systems is crucial in traffic management.
It is important to know the traffic density of the roads real time especially in mega cities for signal control and effective traffic management. Time estimation of reaching from one location to another and recommendation of different alternative routes using real time traffic density information is extremely valuable for residents of mega cities. Furthermore, classification of vehicles is also very important for traffic control centers.

Domain adaptation has received a lot of attention in computer vision[2, 3, 6, 7, 8, 9, 10, 12, 13, 16, 18, 21, 22] where domain variation is a consequence of changing conditions, such as background, location and pose, etc. and, more generally, a strong bias between datasets.This kind of domain shift is very common in real-life applications, in particular in outdoor traffic surveillance tasks.
Having a classifier robust and reliable requires a lot of labeled samples that will train that said classifier. Even nowadays, the acquisition of these data labels has a high cost since there's gigantic volumes of unlabeled data generated in many distinct domains. Trying to overcome that issue of exploiting unlabeled data, alternative solutions have been proposed in literature. Domain Adaptation (DA) is a particular case of transfer learning (TL) (see figure 2.1) that uses labeled data from one or many source domains, to learn a classifier for unlabeled data. The source domains, although it doesn't have to be the same as the target domains, must be somehow

similar. Thus, this "problem" becomes a standard machine learning (ML) where the test data is drawn from a similar distribution as the training data.

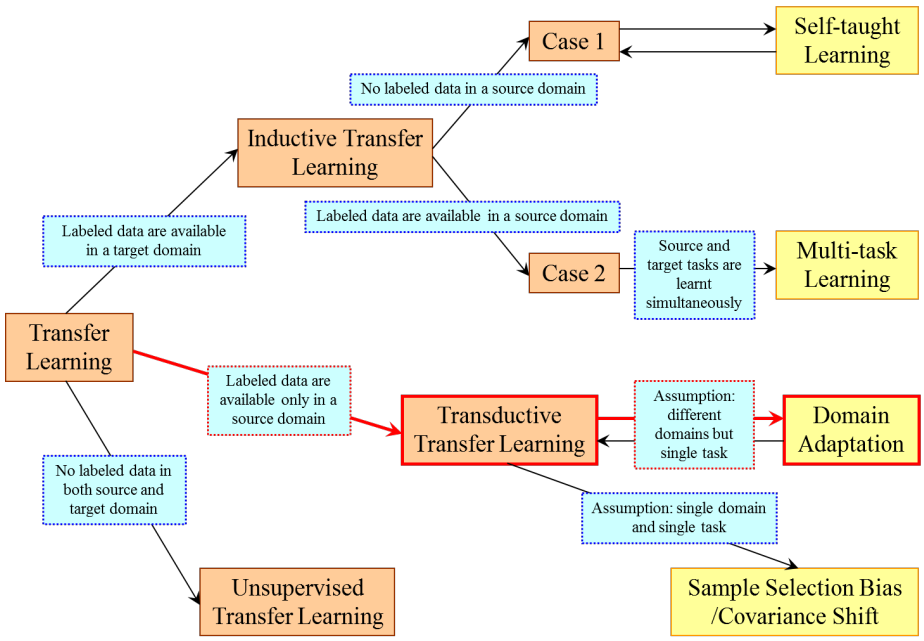


Fig. 2.1: An overview of different transfer learning approaches.[19]

2.1 Visual Domain Adaptation

In visual applications, there can be a lot of domain shift, such as background, location, pose changes, but the domain mismatch might be more severe when the source and target domains contain images of different types, for example comparing photos with sketches. In surveillance and urban traffic understanding, pretrained models on previous locations might need adjustment to the new environment. This adjustment needs either the acquisition of labeled data in the the location or the adaptation of those pretrained models to achieve the performance needed in the new situation. However, the first solution (data labeling), is expensive and time consuming due to the human effort involved. Therefore, the latter solution is preferred when possible. This can be achieved by adapting the pretrained models taking advantage of the unlabeled (and if available labeled) target data.

DA methods are transductive TL solution (see figure 2.1), where is assumed that the tasks are the same. In general they refer to a categorization task, where both the set of labels and the conditional distributions are assumed to be shared between the two domains. However, the assumption that the conditional distributions are the same does not always hold in real-life

applications. Therefore, only the assumption that the labels are the same is required.
The DA can be distinguish between *semi-supervised*(SS) case where a small set of target samples are labeled and *unsupervised*(US) case where the labels are available only for the source domain. In this thesis we'll be focus on the unsupervised approach.

2.1.1 Feature Augmentation

Gopalan et al. at [13] proposed one of the simplest method for DA, where the original representation $\mathtt{x}$ is augmented with itself and a vector of the same size filled with zeros as follows: the source become $\begin{bmatrix} \mathtt{x}^s \\ \mathtt{x}^s \\ 0 \end{bmatrix}$ and the target features $\begin{bmatrix} \mathtt{x}^t \\ 0 \\ \mathtt{x}^t \end{bmatrix}$, for the detailed work see [13]. Then a Support Vector Machine (SVM) is trained on these augmented features to figure out which parts of the representation is shared between the domains and which are the domains specific ones.
The idea of feature augmented is also behind the Geodesic Flow Kernel[1] (GFK)[12], where domains are embedded in d-dimensional linear subspaces that can be seen as points on the Grassman manifold[2] corresponding to the collection of all d-dimensional subspaces. GFK proposes a kernel that makes the solution equivalent to integrating over all common subspaces lying on the geodesic path.
These methods use cross-domain representations and can be used either to train a classifier or to label the target samples, in both scenarios unsupervised and semi-supervised.

2.1.2 Feature Space Alignment

Instead of augmenting the features, these methods tries to align the source features with the target ones. As such, the Subspace Alignment (SA)[8] learns an alignment between the source subspace obtained by PCA[3] and the target PCA subspace, where the PCA dimensions are selected by minimizing the Bregman divergence between the subspaces. It advantage is its simplicity, as shown in the algorithm 1.

[1] GFK is explained furthermore in the thesis
[2] Grassman manifolds are also explain in the following chapters
[3] Principal Component Analysis. Explain in detailed in the next chapters.

Algorithm 1: Subspace Alignment (SA) [8]

Input: Source data $\mathtt{X}^s$, target data $\mathtt{X}^t$, subspace dimension d

Output: Aligned source, $\mathtt{X}_a^s$ and target $\mathtt{X}_a^t$ data

1 $P_s \leftarrow PCA(\mathtt{X}^s, d), \quad P_t \leftarrow PCA(\mathtt{X}^t, d);$

2 $\mathtt{X}_a^s = \mathtt{X}^s\mathtt{P}_s\mathtt{P}_s^T\mathtt{P}_t, \quad \mathtt{X}_a^t = \mathtt{X}^t\mathtt{P}_t;$

2.1.3 Unsupervised Feature Transformation

Chen et al. at [4] exploits the correlation between the source and target set to learn a robust representation by reconstructing the original features from their noised counterparts. The method, called Marginalized Denoising Autoencoder (MDA), is based on a quadratic loss and a drop-out noise level that factorizes over all feature dimensions. This allows the method to avoid explicit data corruption by marginalizing out the noise and to have a closed-form solution for the feature transformation. These method can be used with only one layer or with a stack of several layers with the option of non-linearity between layers to obtain a multi-layer network with the parameters for each layer obtained in a single forward pass (see algorithm 2).

Algorithm 2: Stacked Marginalized Denoising Autoencoder (sMDA) [4]

Input: Source data $\mathtt{X}^s$, target data $\mathtt{X}^t$

Input: $p \leftarrow$ noise level, $w \leftarrow$ regularizer and $k \leftarrow$ number of layers

Output: Denoised features $\mathtt{X}_k$

1 $\mathtt{X} = [\mathtt{X}^s, \mathtt{X}^t], \quad \mathtt{S} = \mathtt{X}^\mathtt{T}\mathtt{X}, \quad \mathtt{X}_0 = \mathtt{X};$

2 $\mathtt{P} = (1-p)\mathtt{S}$ and $\mathtt{Q} = (1-p)^2\mathtt{S} + p(1-p)diag(\mathtt{S})$

3 $\mathtt{W} = (\mathtt{Q} + w\mathtt{I}_\mathtt{D})^{-1}\mathtt{P}$

4 (Optionally), stack K layers with $\mathtt{X}_{(\mathtt{k})} = \mathtt{tanh}(\mathtt{X}_{(\mathtt{k}-1)}\mathtt{W}^{(\mathtt{k})})$

Our thesis has in its foundation, the work proposed by Hoffman[4] et al. at [14], where they take advantage of both Subspace Alignment and GFK approaches adding an approach of their own called continuous manifold adaptation (CMA). This latter approach tries to classify streaming data drawn from a continuously evolving visual domain. This is a space alignment unsupervised method, ie, utilizes PCA to reduce the dimension of its features and assumes labeled source data but unlabeled target data. CMA assumes each target sample has been extracted from a distribution corresponding to a PCA subspace on the Grassman manifold of subspaces. Since the data arrives in a continuous sequentially stream, the target subspace is constantly changing and is found by a variant of the Karhunen-Loeve method [15].

[4]Hoffman method explained later on the thesis

Chapter 3

Theory Behind The Approaches

In this chapter we take a look at the theory behind the algorithms used for the different approaches we tested. The theories used were essentially three: *Interactive Subspace Estimation*, *Smooth Subspace Interpolation* and *Transductive Domain Adaptation*. In the next sections we try to take a deep look at each one of them. Additionally we take a look at the classifiers used to make the predictions that are used to ascertain the accuracy of each method.

3.1 Principal Component Analysis

The Principal Component Analysis (PCA), as the name suggests, is the analyses of a set of observations into their principal components. PCA is one method to reduce the number of features used to represent data. The benefit of this dimension reduction include providing a simpler representation of the data, reduction in memory, and faster classification. We accomplish this with sophisticated underlying mathematical principles to transform a number of possibly correlated variables into a smaller number of variables called principal components (see figure 3.1). In general terms, PCA uses a vector space transform to reduce the dimensionality of large data sets. Using mathematical projection, the original dataset, which may have involved many variables, can often be interpreted in just a few variable (principal components). Therefore it is often that the use of this reduction in the dimension of data allows to find trends and patterns far more easily than would have been possible with the whole data set.

Given data points $x_1, x_2, \ldots, x_n \in \mathbb{R}^p$.

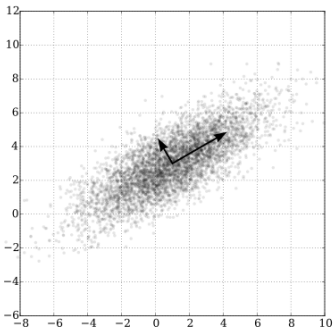


Fig. 3.1: PCA of a multivariate Gaussian distribution. Image from Wikipedia

We define the reconstruction of data in $\mathbb{R}^q \rightarrow \mathbb{R}^p$ as

$$f(\lambda) = \mu + v_q\lambda \tag{3.1}$$

In this rank q model, the mean is $\mu \in \mathbb{R}^p$ and v_q is a $p \times q$ matrix with q orthogonal unit vectors. Finally, $\lambda \in \mathbb{R}^q$ is the low-dimensional data points we are projecting.

Creating a good low-dimensional representation of the data requires that we carefully choose $\mu, v_q,$ and λ. One is by minimizing the reconstruction error given by

$$min \sum_{n=1}^{N} ||x_n - \mu - v_q\lambda_n|| \tag{3.2}$$

In Equation 3.2, μ is the intercept of the lower space in the higher space. Next, $\lambda_{1...N}$ is the $\mathbb{R}^q$ coordinate of x. Last, the quantity inside the sum is the distance between the original data and the low-dimensional representation reconstruction in the original space.

3.1.1 Single Value Decomposition (SVD)

Consider

$$X = UDV^T \tag{3.3}$$

where

- X is an $n \times p$ matrix
- U is an $n \times p$ orthogonal matrix and the columns of U are linearly independent
- D is a positive $p \times p$ diagonal matrix with $d_{11} \geqslant d_{22} \geqslant \cdots \geqslant d_{pp}$
- V is a $p \times p$ orthogonal matrix

We represent each data point as linear combinations.

$$x_1 = u_{11}d_1\bar{v_1} + u_{12}d_2\bar{v_2} + \cdots + u_{1p}d_p\bar{v_p}$$

$$x_2 = u_{21}d_1\bar{v_1} + u_{22}d_2\bar{v_2} + \cdots + u_{1p}d_p\bar{v_p}$$

$$\cdots$$

We can embed x into an orthogonal space via rotation. D scales, V rotates, and U is a perfect circle.
PCA cuts off SVD at q dimensions. U is a low-dimensional representation, D reflects the variance and V are the principal components.

3.2 Geodesic Flow Kernel (GFK)

The GFK approach was first proposed by Gong et al. at [12].
Let $P_{\mathcal{S}}, P_{\mathcal{T}} \in \mathbb{R}^{\mathrm{D}\times\mathrm{d}}$ denote the two sets of basis of the subspaces for the source and target domains. Let $R_{\mathcal{S}} \in \mathbb{R}^{\mathrm{D}\times(\mathrm{D}-\mathrm{d})}$ denote the orthogonal complement to $P_{\mathcal{S}}$ namely $R_{\mathcal{S}}^T P_{\mathcal{S}} = 0$. Using the canonical Euclidean metric for the Riemannian manifold, the geodesic flow is parameterized as $\Phi : t \in [0,1] \rightarrow \Phi(t) \in c|G(\mathrm{d},\mathrm{D})$ under the constraints $\Phi(0) = P_{\mathcal{S}}$ and $\Phi(1) = P_{\mathcal{T}}$. For other t,

$$\Phi(t) = P_{\mathcal{S}}U_1\Gamma(t) - R_{\mathcal{S}}U_2\Sigma(t), \tag{3.4}$$

where $U_1 \in \mathbb{R}^{\mathrm{d}\times\mathrm{d}}$ and $U_2 \in \mathbb{R}^{(\mathrm{D}-\mathrm{d})\times\mathrm{d}}$ are orthonormal matrices. They are given by the following pair of SVDs,

$$P_{\mathcal{S}}^T P_{\mathcal{T}} = U_1 \Gamma V^T, \qquad R_{\mathcal{S}}^T P_{\mathcal{T}} = -U_2 \Sigma V^T. \tag{3.5}$$

Γ and Σ are $\mathrm{d}\times\mathrm{d}$ diagonal matrices. The diagonal elements are $\cos\theta_i$ and $\sin\theta_i$ for $i = 1, 2, \ldots, \mathrm{d}$. Particularly, θ_i are called the pricipal angles between $P_{\mathcal{S}}$ and $P_{\mathcal{T}}$:

$$0 \leqslant \theta_1 \leqslant \theta_2 \leqslant \cdots \leqslant \theta_d \leqslant \frac{\pi}{2} \tag{3.6}$$

They measure the degree that subspaces "overlap". More over, $\Gamma(t)$ and $\Sigma(t)$ are diagonal matrices whose elements are $\cos(t\theta_i)$ and $\sin(t\theta_i)$ respectively.

3.2.1 Computation of GFK

Consider the subspace $\Phi(t)$ for a $t \in (0, 1)$ and compute $\Phi(t)^T x$, ie, the projection of a feature vector x into this subspace. If x is from the source domain and t is close to 1, then the projection will appear more likely coming from the target domain and conversely for t close to 0.
For two original $\mathcal{D}$-dimensional feature vectors x_i and x_j, we compute their projections into $\Phi(t)$ for a continuous t from $0 \to 1$ and concatenate all the projections into infinite-dimensional feature vectors z_i^∞ and z_j^∞. The inner product between them defines their geodesic-flow kernel,

$$\langle z_i^\infty, z_j^\infty \rangle = \int_0^1 (\Phi(t)^T x_i)^T (\Phi(t)^T x_j) dt = x_i^T G x_j \tag{3.7}$$

where $G \in \mathbb{R}^{\mathcal{D}\times\mathcal{D}}$ is a positive semidefinite matrix. This is precisely the "kernel trick", where a kernel function induces inner products between infinite-dimensional features.
The matrix G can be computed in a closed-form from previously defined matrices:

$$G = [P_{\mathcal{S}} U_1 \quad R_{\mathcal{S}} U_2] \begin{bmatrix} \Lambda_1 & \Lambda_2 \\ \Lambda_2 & \Lambda_3 \end{bmatrix} \begin{bmatrix} U_1^T & P_{\mathcal{S}}^T \\ U_2^T & R_{\mathcal{S}}^T \end{bmatrix} \tag{3.8}$$

where Λ_1 to Λ_3 are diagonal matrices, whose diagonal elements are

$$\lambda_{1i} = 1 + \frac{sin(2\theta_i)}{2\theta_i}, \lambda_{2i} = \frac{cos(2\theta_i) - 1}{2\theta_i}, \lambda_{3i} = 1 - \frac{sin(2\theta_i)}{2\theta_i}$$

3.3 Interactive Subspace Estimation

For this estimation we used the model proposed by [20] where they approach the problem with some resemblance to the classic filter algorithms, but with a more informative representation

through the use of an eigenbasis. Their approach is constantly update the eingenbasis using a computationally efficient algorithm.

3.3.1 Incremental Update of Eigenbasis

The Ross et al [20] method is a variant of the efficient sequential Karhunen-Loeve algorithm to update the eignbasis ([15]), which in turns is based on the classic $R-SVD$ method[1] [11].
Let $X = U\Sigma V^T$ be the SVD of a data $M \times P$ matrix X where each column vector is an observation. The $R-SVD$ algorithm provides an efficient way to carry out th SVD of a larger matrix $X^* = (X|E)$, where E is a $M \times K$ matrix consisting of K additional observations (incoming samples) as follows.

- Use and orthonormalization process on $(U|E)$ to obtain an orthonormal matrix $U' = (U|\tilde{E})$.
- Form the matrix $V' = \begin{pmatrix} V & 0 \\ 0 & I_K \end{pmatrix}$, where I_K is a K dimensional identity matrix.
- Let $\Sigma' = U'^T X^* V' = (\frac{U^T}{\tilde{E}^T})(X|E)\begin{pmatrix} V & 0 \\ 0 & I_K \end{pmatrix} = \begin{pmatrix} U^TXV & U^TE \\ \tilde{E}^TXV & \tilde{E}^TE \end{pmatrix} = \begin{pmatrix} \Sigma & U^TE \\ 0 & \tilde{E}^TE \end{pmatrix}$ since $\Sigma = U^TXV$ and $\tilde{E}^TXV = 0$. Notice that the K rightmost columns of Σ' are the new samples, represented in the update orthonormal basis spanned by the columns of U'.
- Compute the SVD of $\Sigma' = \tilde{U}\tilde{\Sigma}\tilde{V}^T$ and the SVD of X^* is

$$X^* = U'(\tilde{U}\tilde{\Sigma}\tilde{V}^T)V'^T = (U'\tilde{U})\tilde{\Sigma}(\tilde{V}^TV'^T) \tag{3.9}$$

By exploiting the orthonormal properties and block structure, the SVD computation of X^* can be efficiently carried by using the smallet matrices, U',V',Σ' and the SVD of smaller matrix Σ'. The computational complexity analysis and details of $R-SVD$ algorithm are described in [11]. Based on the $R-SVD$ method, the sequential Karhunen-Loeve algorithm further exploits the low dimensional subspace approximation and only retains a small number of eigenvectors as new data arrive. See [20] for details of an update strategy and the computational complexity analysis.

[1] Recursive Singular Value Decomposition

3.4 Smooth Subspace Interpolation

The Smooth Subspace Interpolation approach was developed by Batista et al. at [1] and aims to generate a smooth interpolation curve intrinsically on the Grassman manifold, by means of the Casteljau algorithm.

3.4.1 Grassman Manifold

Let $s(n)$ and $so(n)$ denote the set of all $n \times n$ real symmetric matrices and the set of all $n \times n$ real skew-symmetric matrices respectively.
The (real) Grassman manifold $\zeta_{n,k}$ is the set of all k-dimensional linear subspace in the $\mathbb{R}^n$, where $n \geqslant k \geqslant 1$. This manifold has a matrix representation

$$\zeta_{n,k} := \{P \in s(n) : P^2 = P \text{ and } \texttt{rank}(P) = k\} \tag{3.10}$$

so that it is considered a sub-manifold of $\mathbb{R}^{n\times n}$ with dimension $k(n-k)$. For $P \in \zeta_{n,k}$, define

$$so_P(n) := \{X : X \in so(n) \text{ an } XP + PX = X\} \tag{3.11}$$

The tangent space to a point $P \in \zeta_{n,k}$ is given by

$$T_P\zeta_{n,k} = \{[X, P] : X \in so_P(n)\} \tag{3.12}$$

The Grassman manifold will be equipped with the metric inherited from the Euclidean space $\mathbb{R}^{n\times n}$

$$\langle [X_1, P], [X_2, P] \rangle = tr(X_1^T X_2) \tag{3.13}$$

A geodesic γ in $\zeta_{n,k}$ starting from P with initial velocity $\dot{\gamma}(0) = [X, P]$ is given by

$$\gamma(t) = e^{tX} P e^{-tX} \tag{3.14}$$

It has been proven that the geodesic joining a point P to a point Q is of the form 3.14, with $X = \frac{1}{2} log((I - 2Q) \cdot (I - 2P))$ where 'log' stands for the *principal logarithm* of a matrix, so that if the orthogonal matrix $(I - 2Q) \cdot (I - 2P)$ has no negative real eigenvalues then this *geodesic is unique*.

3.4.2 The Casteljau Algorithm

3.4.2.1 Generating a 2nd-order geometric polynomial

Given a set of three points $x_{i\,i=0}^{2}$ in $\zeta_{n,k}$ let t $\rightarrow \sigma_1(t, x_i, x_{i+1})$ be geodesic curves joining x_i to x_{i+1}, for $i = 0, 1$. Define a family of curves $\gamma : [0,1] \times [0,1] \rightarrow \zeta_{n,k}$ as follows. For a fixed $t_0 \in [0,1]$, the map $t \rightarrow \gamma(t, t_0)$ is a smooth curve joining $\sigma_1(t_0, x_0, x_1)$ to $\sigma_1(t_0, x_0, x_2)$. Then $\sigma_2 : [0,1] \rightarrow \zeta_{n,k}$ given by $\sigma_2(t) = \gamma(t,t)$ is a smooth curve joining $x_0 \rightarrow x_2$, as illustrated in Figure 3.2. This is a second order geometric polynomial produced by the Casteljau algorithm.

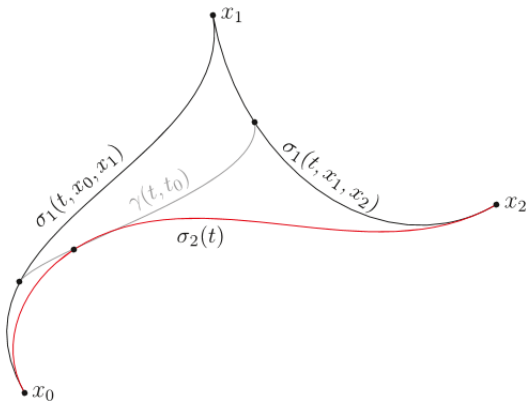


Fig. 3.2: Illustration of the two-step Casteljau algorithm [1]

It turns out that the initial and final velocities of the curve σ_2 are related to the initial and final velocities of the curves σ_1. More precisely:

$$\dot{\sigma_2}(0) = 2\dot{\sigma_1}(0, x_0, x_1) \text{ and } \dot{\sigma_2}(1) = 2\dot{\sigma_1}(1, x_0, x_1) \tag{3.15}$$

3.4.2.2 Piecing together polynomial

Using the Casteljau algorithm in each time interval $[i, i+1]$, starting with an arbitrary control point in the first interval, and computing the control points for the remaining intervals so that we can compute spline curves that are differentiable and obtained by piecing together quadratic polynomials. With this kind of process we can fine the whole C^1-smooth curve between all intervals.

The curve σ may be generated by piecing together quadratic polynomials defined on each subinterval $[i, i+1]$ and joining P_i to P_{i+1} with control point C_i, that is

$$\sigma(t) \mid_{[i,i+1]} = \sigma_2(t - i, P_i, C_i, P_{i+1}) \tag{3.16}$$

Each control point is calculated as

$$C_i = e^{\Omega_i} P_i e^{-\Omega_i} \tag{3.17}$$

and represents the end point of the geodesic σ that starts at the point P_i with the initial velocity equal to $[\Omega_i, P_i]$. In order to ensure that σ is C^1-smooth, the initial velocity Ω_{i+1} of each subsequent spline segment must equal the final velocity of the previous segment, i.e.

$$\Omega_{i+1} = \frac{log((I - 2P_{i+1}) \cdot (I - 2C_i))}{2} \tag{3.18}$$

To find the points $\sigma(t)$ on the spline, first compute 3.17 to find the control point C_i, for the segment $\sigma([i, i+1])$, where $t \in [i, i+1]$. Then with the triplet C_i,P_i and P_{i+1} we can find a set of K intermediate subspaces $\sigma(t)$ and the final velocity Ω.

3.5 Transductive Domain Adaptation

The TDA approach is used in the case of source classifiers available as a black box. These classifiers can only be used for predicting labels, including for target domain instances. Such an unsupervised domain adaptation is of high interest for us because we can get new labels to use as new training labels and features for posterior classification with some reliability.

In this method we consider class predictions $f_k(x_n)$ as relevant but corrupted. Hence, we can exploit the correlation between the target data x_n and the source predictions $f(x_n^t)$ to reconstruct both the target data and the source classifiers predictions of these target data. We then apply the sMDA(see algorithm 2) to the augmented dataset $U_n^t = [x_n^t; f(x_n^t)]$ and compute W.

W is the linear mapping matrix which tries to minimize the corruption. The basic form of this method is a one-layer denoising autoencoder where a set of inputs x_n are corrupted M times by random feature dropout probability p and reconstructed with W by minimizing the square reconstruction loss:

$$\mathcal{L}(W) = \sum_{n=1}^{N} \sum_{m=1}^{M} ||x_n - W\tilde{x}_{nm}||^2 \tag{3.19}$$

The mapping W can be expressed in closed form as $W = \mathbb{E}[P]\mathbb{E}[Q]^{-1}$, where

$$\mathbb{E}[Q]_{ij} = \begin{cases} S_{ij} q_i q_j & \text{if } i \neq j \\ S_{ij} q_i & \text{if } i = j \end{cases} \tag{3.20}$$

and

$$\mathbb{E}[P]_{ij} = S_{ij} q_i \tag{3.21}$$

where $\mathbb{E}[x]$ is the mathematical expectation, $q = [1-p, ..., 1-p, 1] \in R^{D+1}$, p is the dropout probability, D is the feature dimension and $S = XX^T$ is the covariance matrix of the uncorrupted data X.

As said before, the basic form of this method is one-layered. However we can stack several MDA layers and create a deep architecture with $(l-1)^{th}$ denoising layer as input to the l^{th} layer and learn the transformation W^l to reconstruct the previous output from its corrupted equivalent. We can also extend the mapping beyond just a linear transformation between layers, for each output we can apply a hyperbolic tangent function $h_l = \tanh(W^l h_{l-1})$ or a rectified linear units $h_l = max(W^t h_{l-1}, 0)$.

After this computation, we can get denoised class predictions for x^t as $y^t = \sum W_{:,D+1:D+C} f(x)$ and use these to reconstruct class predictions $\hat{f}(x_n^t)$ to make the classification decisions. Our set $\mathcal{F}$ includes a single source classifier f_1 with one prediction per class, the class with the maximum reconstructed value, $c^* = \text{argmax}_c\{y_c^t | y^t\}$.

The main advantage os this method is that it doesn't require class labels; hence we can take advantage of the unlabeled target data and apply it for unsupervised domain adaptation.

To marginalized the corrupted features, a corrupting distribution is first defined to transform observations x into corrupted versions $\tilde{x}$. The corrupting distribution is assumed to factorize over all feature dimension and, each individual distribution is assumed to be a member of a natural exponential family, $p(\tilde{\mathrm{x}}|\mathrm{x}) = \prod_{d=1}^{D} P(\tilde{x}_d | x_d; \theta_d)$, where $\mathtt{x} = (x_1, ..., x_d)$ and $\theta_d, d = 1, ..., D$ is a parameter of the corrupting distribution on dimension d.

The corrupting distribution can be unbiased or biased. Known examples of distribution P are the blankout [23], Gaussian, Laplace and Poisson noise [17].

3.6 Classifiers

3.6.1 K-Nearest Neighbors

KNN is a *non parametric lazy learning* algorithm which means that it does not make any assumptions on the underlying data distribution. This is pretty useful, as most of the practical data does not obey the typical theoretical assumptions made (eg gaussian mixtures, linearly separable, etc).

It is also a lazy algorithm. This means it does not use the training data points to do and *generalization*, i.e., there is *no explicit training phase* or it is very minimal. This means the training phase is pretty fast. Lack of generalization means that KNN keeps all the training data. More exactly, almost all the training data is needed during the test phase.
The dichotomy is pretty obvious here − There is a non existent or minimal training phase but a costly testing phase. The cost is in terms of both time and memory. More time might be needed as in the worst case, all data points might take point in decision. More memory is needed as we need to store all training data.

KNN assumes that the data is in a metric space. The data can be scalars or possibly multidimensional vectors. Since the points are in feature space, they have a notion of distance − This need not necessarily be Euclidean distance although is the one commonly used.
Each of the training data consists of a set of vectors and class labels associated with each vector. In the simplest case (our case), it will be either positive or negative classes.
We are also given a single number "k". This number decides how many neighbors influence classification. This is usually an odd number if the number of classes is 2. This neighbors are defined based on the metric distance.

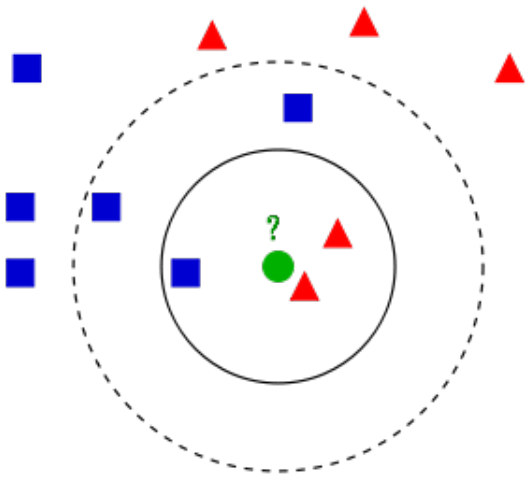


Fig. 3.3: Example of a 2D KNN classification [24]

As we can see in figure 3.3 the test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If $k = 3$ (solid line circle) it is assign to the second class because there are two triangles and only one square inside the area bounded by that solid line circle. If $k = 5$ (dashed line circle) it is assign to the first class, like the first case, here we have three squares and two triangles inside the circle bounded by the dashed line.

The choice of k is very critical − A small value of k means that noise will have a higher influence on the results. A large value make it computational expensive and defeats the basic philosophy behind KNN. A simple approach to select k is set $k = \sqrt{n}$, where n is the number of features.

3.6.2 Support Vector Machine

The idea behind SVM is that it creates a hyperplane to separate a set of data. This hyperplane is chosen as to provide the largest distance possible to the nearest data points of the different classes. With this separation it is possible to classify unknown points depending on which side of the hyperplane they fall on. This simple and intuitive yet powerful concept has made SVM one of the standards methods for data classification.

3.6.2.1 Linear SVM

Let us assume we have some training dataset of n points $(\overrightarrow{x}_1, y_1), ..., (\overrightarrow{x}_n, y_n)$,where y_i are either 1 or -1, each indicating the class to which the point $\overrightarrow{x}_i$ belongs. Each $\overrightarrow{x}_i$ is a p-dimensional real vector. Note that SVM deals only with binary classification, i.e., classification between two classes (positive and negative). We want to find the "maximum-margin hyperplane" that divides the group of points $\overrightarrow{x}_i$ for which $y_i = 1$ from the group of points for $y_i = -1$, which is defined so that the distance between the hyperplane and the nearest point $\overrightarrow{x}_i$ from either group is maximized.
The goal of SVM (finding those margins) can be better seen in the following figure 3.4.

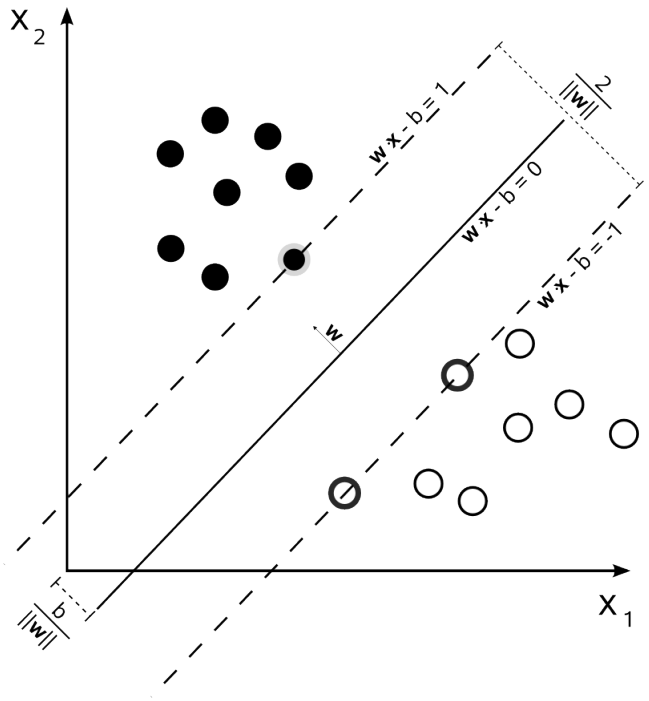


Fig. 3.4: Visualization of the margins for 2D data [25]

An hyperplane can be described as:

$$\overrightarrow{w}.\overrightarrow{x} - b = 0 \tag{3.22}$$

where $\overrightarrow{w}$ is the normal vector to the hyperplane and $\frac{b}{||\overrightarrow{w}||}$ represents the offset of the hyperplane from the origin.

Supposing the training data is linearly separable then we can define:

$$\overrightarrow{w}.\overrightarrow{x} - b = 1 \tag{3.23}$$

$$\overrightarrow{w}.\overrightarrow{x} - b = -1 \tag{3.24}$$

that represents the boundaries of the hyperplane. This means that any point in our training data satisfying

$$\overrightarrow{w}.\overrightarrow{x} - b \geqslant 0 \tag{3.25}$$

belongs to class 1, and points satisfying

$$\overrightarrow{w}.\overrightarrow{x} - b \leqslant 0 \tag{3.26}$$

belongs to class -1.

We then want to maximize the distance between these two boundaries. This distance is given by

$$\text{width} = (\overrightarrow{x}_+ - \overrightarrow{x}_-)\frac{\overrightarrow{w}}{||\overrightarrow{w}||} \tag{3.27}$$

where $\overrightarrow{x}_+$ is a positive sample in the boundary, $\overrightarrow{x}_-$ is a negative sample in the boundary and $\frac{\overrightarrow{w}}{||\overrightarrow{w}||}$ is the normalized normal vector of the hyperplane. From

$$\overrightarrow{w}.\overrightarrow{x} = 1 + b\overrightarrow{w}.\overrightarrow{x} = -1 + b \tag{3.28}$$

we then get

$$\text{width} = (\overrightarrow{w}.\overrightarrow{x}_+ - \overrightarrow{w}.\overrightarrow{x}_-)\frac{1}{||\overrightarrow{w}||} = \frac{2}{||\overrightarrow{w}||} \tag{3.29}$$

We conclude that, in order to maximize the width we need to minimize $||\overrightarrow{w}||$. Since this minimization problem involves the norm it is necessary to determine a square root. As a mathematical convenience we can minimize for $\frac{1}{2}||\overrightarrow{w}||$ since we'll arrive at the same answer. This minimization can be achieved using Lagrange multipliers:

$$L\frac{1}{2}||\overrightarrow{w}||^2 - \sum \alpha_i[y_i(\overrightarrow{w}.\overrightarrow{x} - b) - 1] \tag{3.30}$$

where $\alpha_i = (\alpha_1, \alpha_2, ..., \alpha_n)$ are the Lagrange multipliers. Since we want to find an extreme of the function we need to find the zeros of its derivatives:

$$\frac{\partial L}{\partial \overrightarrow{w}} = \overrightarrow{w} - \sum \alpha_i y_i \overrightarrow{x}_i \implies \overrightarrow{w} = \sum \alpha_i y_i \overrightarrow{x}_i \tag{3.31}$$

$$\frac{\partial L}{\partial b} = -\sum \alpha_i y_i = 0 \implies \sum \alpha_i y_i = 0 \tag{3.32}$$

From this we were able to come up with a value of $\overrightarrow{w}$. By using this value on equation 3.30 we get

$$L = \sum \alpha_i - \frac{1}{2}\sum\sum \alpha_i \alpha_j y_i y_j \overrightarrow{x}_i.\overrightarrow{x}_j \tag{3.33}$$

At this stage quadratic programming techniques have to be employed. Once the vector $\alpha^* = (\alpha_1^*, ..., \alpha_N^*)$ solution of the maximization problem has been found, the optimal separating hyperplane is given by,

$$w^* = \sum_{i=1}^{N} \alpha_i^* y_i x_i$$

$$b^* = -\frac{1}{2}(w^*, x_r + x_s)$$

where x_r and x_s are any support vector from each class satisfying, $\alpha_r, \alpha_s > 0$ and $y_r = -1, y_s = 1$.

Chapter 4

Description of the Used Approaches

We reserved this chapter to explain how our approaches really work a part from the theory. Here we show how the math explained before (chapter 3) is implemented into our approaches' algorithms.

4.1 Initial Parameters

First we start with a function that is transversal to all approaches. This function is called *config_caltran.m* and is responsible for the configuration of the train and test parameters. These parameters are presented in the following table 4.1.

Parameter	Value	Description
norm_type	l1_zscore	normalization method utilized
ns	480	number of labeled training points
alpha	1.5	forgetting factor for online subspace learning
Tmax	480 or 2400	number of test images to use
block_size	5	number of test images seen at each time step
classes	[1 -1]	label classes, in this case either positive or negative
start	start_index	index of the first training image
dim	10	dimension of the principal components

Table 4.1: Table summarizing the initial parameters

All these parameters are then stored like a structure on a variable (*expt*) to facilitate the use of them throughout the code

.

The normalization method is composed by two different techniques, L_1 *Normalization* and *Z-Score Normalization.*

4.1.1 L1 Normalization

L_1 norm of x is defined as

$$||x||_1 = \sum |x_i|$$

L_1 norm is the sum of absolute differences between the target value and the estimated values. This norm is quite commong and is known as the *Manhattan norm.* If the L_1 norm is computed for a difference between two vectors or matrices, that is

$$SAD(x_1, x_2) = ||x_1 - x_2||_1 = \sum |x_{1_i} - x_{2_i}|$$

it is called *Sum of Absolute Difference (SAD).*
In more general cases, it may be scaled to a unit vector by:

$$MAE(x_1, x_2) = \frac{1}{n}||x_1 - x_2||_1 = \frac{1}{n}\sum |x_{1_i} - x_{2_i}|, \text{ where } n \text{ is a size of } x.$$

This unit vector is also known as *Mean-Absolute Error (MAE).*

4.1.2 Z-Score Normalization

Z-Score is a measure of how many standard deviation above or below the dataset mean a data point is. It's also known as a standard score and it can be placed on a normal distribution curve. Z-Scores range from -3 standard deviations (which would fall to the far left of the normal distribution curve) up to $+3$ standard deviations (which would fall to the far right of the normal distribution curve), but it doesn't always have to be those values for standard deviation.

$$\bar{x_i} = \frac{x_i - \mu}{\sigma}$$

where

$$\sigma = \sqrt{\frac{1}{(n-1)}\sum_{i=1}^{n}(x_i - \mu)^2}$$

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

4.1.3 Initial PCAs

After getting those initial parameters and compute both training and testing pair of features and labels ($[X_s, Y_s]$ for training; $[X_t, Y_t]$ for testing), the next step we take in order to achieve our objective is compute those first PCAs with the training components. Basically, we separate those 480 images[1] from training into 4 groups of 120 images, leaving us with 4 separate groups of images that will be the computed into our initial 4 PCAs.

The way we do this is by simply apply a MatLab function called *myPCA* which receives our initial parameters (see table 4.1) and the features of the training data we wish to use in order to compute each PCA. In this function we start by using the MatLab's *svd* function to compute the singular value decomposition of those input features given. This allows us to extract three matrices but we'll only use two: S, which is a matrix containing the diagonal non-negative values in decreasing order and V, are the right singular vectors fo corresponding singular values. With these two matrices we then compute four variables that will make the structure of our PCA to be used later.

For better understanding, *myPCA algorithm* is presented next (see algorithm 3).

Algorithm 3: *myPCA algorithm.* Adapted form Hoffman code.

Input: Initial Parameters *expt*, features X_s

Output: PCA structure *pca*

1 $dim \leftarrow expt.di$

2 $[\sim, Ss, V] \leftarrow svd(Xs)$

3 $P \leftarrow V(:, 1 : dim)$

4 $Ut \leftarrow P$

5 $m \leftarrow min(size(S, 1), dim)$

6 $S \leftarrow diag(Ss(1 : m, 1 : m))$

7 $S((m + 1) : dim) \leftarrow 0$

8 $mu \leftarrow mean(X_s)$

9 $nprev \leftarrow size(X_s, 1)$

10 $pca \leftarrow Ut, S, mu, nprev$

11 **return** *pca*

[1] Images extracted from Hoffman's Dataset, explained further

4.1.4 Initial Interpolation

Now that our Training PCAs are computed we can progress to the computation of two segments that make the interpolation curve between all those four PCAs. To compute each segment we needed three points (as explain in chapter 3). The first segment is computed with the first three PCAs as points, we achieved this first segment with the following algorithm (see algorithm 4):

Algorithm 4: *Initial Curve algorithm.* [1]

Input: K, P, C, R, where P is the initial point, R is the final point and C is the control point

Output: Z, Ω_1

1 Calculate velocity components Ω and Ω_1 for the geodesics $\sigma(t, P, C)$ and $\sigma_1(t, C, R)$, respectively:

$$\Omega = \frac{log((I-2C)\cdot(I-2P))}{2}$$

$$\Omega_1 = \frac{log((I-2R)\cdot(I-2C))}{2}$$

2 **for** $k \leftarrow 0$ ***to*** K **do**

3 $t = k/K$

4 Calculate first step end points M_1 and M_2 at t:

$$M_1 = e^{t\Omega} \cdot P \cdot e^{-t\Omega}$$

$$M_2 = e^{t\Omega_1} \cdot C \cdot e^{-t\Omega_1}$$

5 Compute the point on the geodesic from M_1 to M_2 at t:

$$\Theta_1 = \frac{log((I-2M_2)\cdot(I-2M_1))}{2}$$

$$Z[k] = e^{t\Theta_1} \cdot M_1 \cdot e^{-t\Theta_1}$$

6 **end**

7 **return** Z,Ω_1

To compute the next segment of the interpolation curve we have to previously compute one control point between the last two PCAs in order to apply a similar algorithm like the previous one. This control point is calculated with the following algorithm:

Having this linking point we can now proceed to compute the second segment. This new algorithm is very much alike with algorithm 4 just with some minor changes. See the following

Algorithm 5: *Control Point Calculation.* [1]

and open t **Input:** Q,Ω

Output: C

Calculate the control point C:

$$C = e^{\Omega} \cdot Q \cdot e^{-\Omega}$$

return C.

algorithm:

Algorithm 6: *Compute the second segment.* [1]

Input: Multiple domain data $X_d, d = 1, ..., D$, subspace dimension k, number of intermediate samples per curve segment K

Output: Sampled points $S = P(j), j = 1, ..., J$. Each point represents an intermediate subspace.

Perform PCA on each X_d to obtain the orthogonal k-frame in $\mathbb{R}^n$, and compute P_d.

Perform ordering (indexation) of P_d

$$sort(P_d) \leftarrow min \sum -tr(\Omega^2)|_{[i,i+1]}$$

and open t Consider P_1 as the control point for 1st curve segment.

Get K sample points and velocity components (for first segment)

$$[S, \Omega] = Alg_4(K, P_0, P_1, P_2)$$

for $i \leftarrow 2 \textit{to} D - 1$ **do**

  Get control point for next segment

$$C = Alg_5(P_i, \Omega)$$

  Get K sample points and velocity components (next segments)

$$[S', \Omega] = Alg_4(K, P_i, C, P_{i+1})$$

  Concatenate $[S] \leftarrow [S, S']$

end

return S

4.2 Continuous Manifold Adaptation with Subspace Update

Continuous Manifold Adaptation is the approach used by Hoffman et al. at [14]. This approach was the foundation of our thesis and is the baseline to our results. Their objective is more generic than ours. Having a source subspace fixed and limited in time that characterizes all

the source samples, with the computation of a geodesic curve they can project the features of this source subspace into intermediate subspaces between the source and target subspaces. This target subspace is time-variant and aims to simulate the variations on the testing domain.

Suppose that at test time, we receive a stream of observations $z_1, \ldots, z_{nT} \in \mathbb{R}^D$, which arrive one at a time[2]. Assuming the distribution of possible points arriving at t can be represented by a lower dimensional subspace P_t.

To align the training and test data, we seek to learn a time-varying transformation, W_t, between source and target points, where t indexes the order in which the examples are received. This transformation can be obtained by learning two time-varying embeddings that map between points of the two lower dimensional subspaces, $\tilde{A}_t$ and $\tilde{B}_t$, with the mapping in the original space being defined as $W_t = \tilde{A}^T U^T P_t \tilde{B}_t$. This transformation is the correspondent G in equation 3.7 and it is computed with the GFK algorithm explained in the previous chapter. This Since we have a continuously changing target subspace, we must simultaneously learn the lower dimensional subspace, P_t, representing the distribution from which the data was drawn at each time t. We will search for a subspace that minimizes the projection error of the data:

$$R_{err}(z_t, P_t) = ||z_t - P_t(P_t^T z_t)||^2 \tag{4.1}$$

This allows us to apply a smoothness to the subspace learning, with the assumption that the target subspace does not change quickly.
At each step, the goal is to optimize the following equation:

$$\min_{P_t^T P_t = I, \tilde{A}_t, \tilde{B}_t} r(P_{t-1}, P_t) + R_{err}(z_t, P_t) + \psi(U\tilde{A}_t, P_t\tilde{B}_t) \tag{4.2}$$

where $r(\cdot)$ is a regularizer that encourages the new subspace learned at time t to be close to the previous subspace of time $t-1$.

Equation 4.2 is non-convex and because of that they chose to solve it by alternating between three steps:

1. Receive data z_t
2. Given $\tilde{A}_{t-1}$ and $\tilde{B}_{t-1}$ compute P_t

[2]On our case, we have a batch of samples

3. Given P_t compute $\tilde{A}_t$ and $\tilde{B}_t$

To optimize step 2, they fixed $\tilde{A}_{t-1}$ and $\tilde{B}_{t-1}$ and then examined the third term of the optimization function. Grouping the first and third term into a single regularizer of P_t, we can solve this problem. Doing this enforces a smoothness between the subsequent learned subspaces. Fianlly, we can express this subproblem as:

$$\min_{P_t} \quad r(P_{t-1}, P_t) + R_{err}(z_t, P_t), \qquad P_t^T P_t = I \tag{4.3}$$

Solving Equation 4.3 for $r(\cdot, \cdot) =$ constant would result in P_t which is equal to the d largest singular vectors of the data z_t, which can be obtained via SVD. Due to the lack of enough data at time t to compute a robust SVD, this optimization problem was solved with a variant of sequential Karhunen-Loeve [15], which adapts a subspace incrementally (see algorithm 3).

4.3 Cyclic Temporal Clusters

This section has three subsection because is one approach but with three variants.
What these variants have in common is that all utilize two base concepts: smooth interpolation and clustering the train and test data.
Starting from the latter, both, training and test data, are divided into groups of 6 hours (120 images), the train side has 4 divisions and the test side has also 4 divisions if we are testing 480 images (1 day) and has 20 divisions if testing for 2400 images (5 days).
This data clustering was thought and conceived with the idea that if we group together images from those 6 hours the probability of the scene conditions being nearly the same was higher and when testing images from those same time periods the accuracy of the test was greater, trying to mitigate the problems mention early (image resolution, different light and weather conditions).
The second concept, Smooth Interpolation, as explain deeper on chapter 3 utilizes those clusters (PCAs) and with the computation of a control point creates a smooth polynomial curve that will serve as the base of the testing. That curve, in our case two curves for the training part and one for the testing part, is divided into K intervals (10 in our work) and each point is then projected into all the remaining points given us a training vector, which later is gonna be used for testing.

4.3.1 Smooth Interpolation with Target Subspace Update

This is the first method we applied the smooth interpolation.

Even though this approach is inside this section, we only made clusters from the training data and the only subspace that's being updated is the target subspace. This training data clusters allows us to compute the first two segments of the interpolation curve and get the interpolation samples needed to make the classification (see algorithms [4,5,6]).

Here we assume the target data as a whole and use a continuous adaptation of the target PCA, each step advancing a fixed number (*block_size* see 4.1) and computing a new PCA with the help of *skml.m* function which uses a variant of the efficient sequential Karhunen-Loeve algorithm to update the eignbasis ([15]). This function receives as input the new batch of $block_size$ images as well as the old PCA structure (see algorithm 3) and returns new elements for a new PCA structure. After this subspace update we compute an interpolation curve from the last training PCA to the one created with those new elements (using algorithm 6) and use that curve to classify those new features. The process is then iterated until all the training images are classified and the accuracy of that classification is stored to be analyzed after.

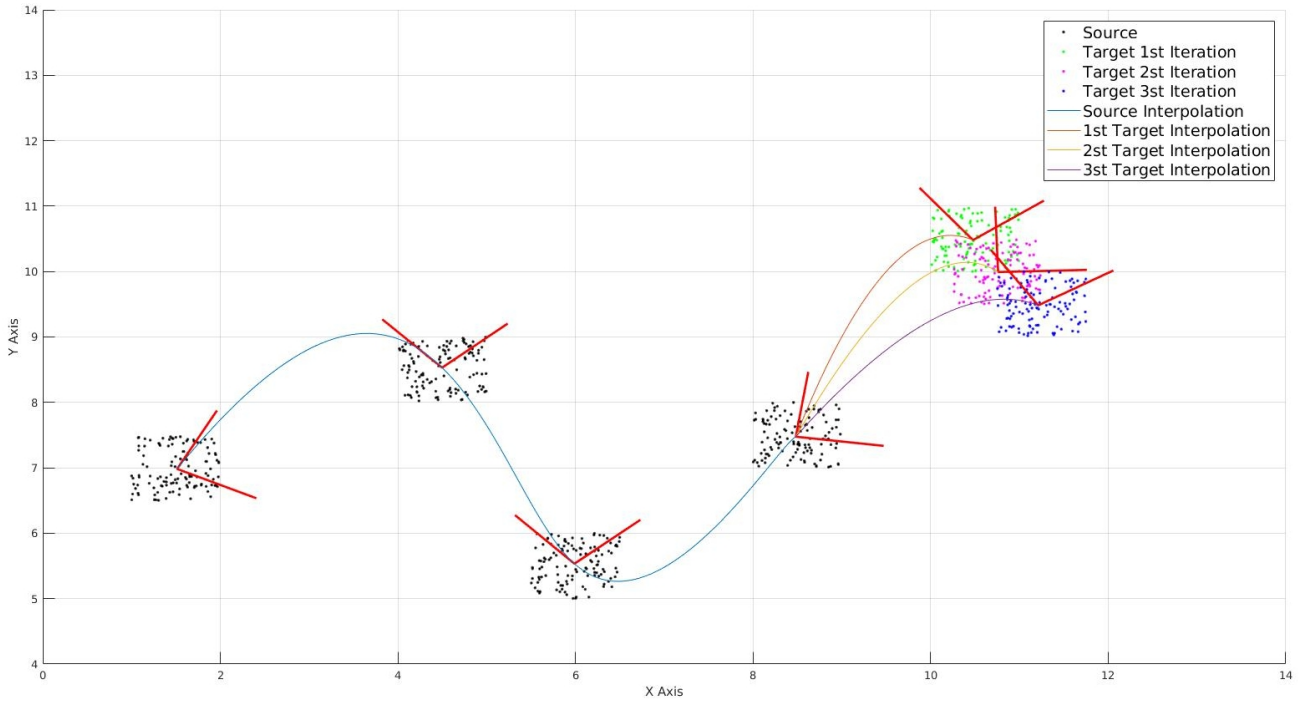


Fig. 4.1: Smooth interpolation with Target Subspace Update

As we can see in figure 4.1, after computing the initial source PCA (black dots), the new samples arrive (green dots) form a target PCA and the first target segment of the curve is

drawn. As the source data stays fixed and unchanged, the interpolation segments stay the same and with the arriving of the new batch of samples (purple dots), it is required another target interpolation segment and so on. We repeat the same process until all the images are classified.

4.3.2 Smooth Interpolation with Training Subspace Update

This method is very much alike with the previous one. However in this one at each batch of new images arrive, those points are being projected onto both interpolation segments and store that information for classification purposes. After this projection, those same samples are used to update the training PCA to which those samples correspond to.
As the previous one, we start by computing four training PCAs and with the help of algorithm 3 and then with those same three algorithms ([4,5,6]) we compute the first two segments of the interpolation curve. After this is done, we then proceed to project those samples on those segments. With the help of *sklm.m* function to extract the new elements for that PCA, we update the training PCA with the new samples. The difference between this approach and the previous one is that we have the target data with clusters of 6 hours (120 images) each. This allows us to correspond each of those clusters to one of the four training clusters (either $0h - 6h$, $6h - 12h$, $12h - 18h$ or $18h - 0h$). When we have that correspondence we can then update that corresponding training PCA by adding the new elements of the target PCA. Thus, adding more information to the training part and, hopefully,get better results as time goes by. As we can see, the following figure tries to explain what is happening with this approach.

Figure 4.2 shows us how this approach works. Initially we have 4 sources (black dots) corresponding to the 4 initial PCA with the first interpolation curve, transversal to all approaches. The new samples arrive (green dots) after they are projected onto this first curve, those green dots are going to change the initial PCA and a new curve is compute between this new PCA and the other previous ones. The next batch arrives (purple dots) do exactly the same to the second training PCA and a new curve is drawn with the two new subspaces and the last two initial ones. Similarly, the same occurs with the 3rd and 4th initial subspace and it continues until all the samples are analyzed.

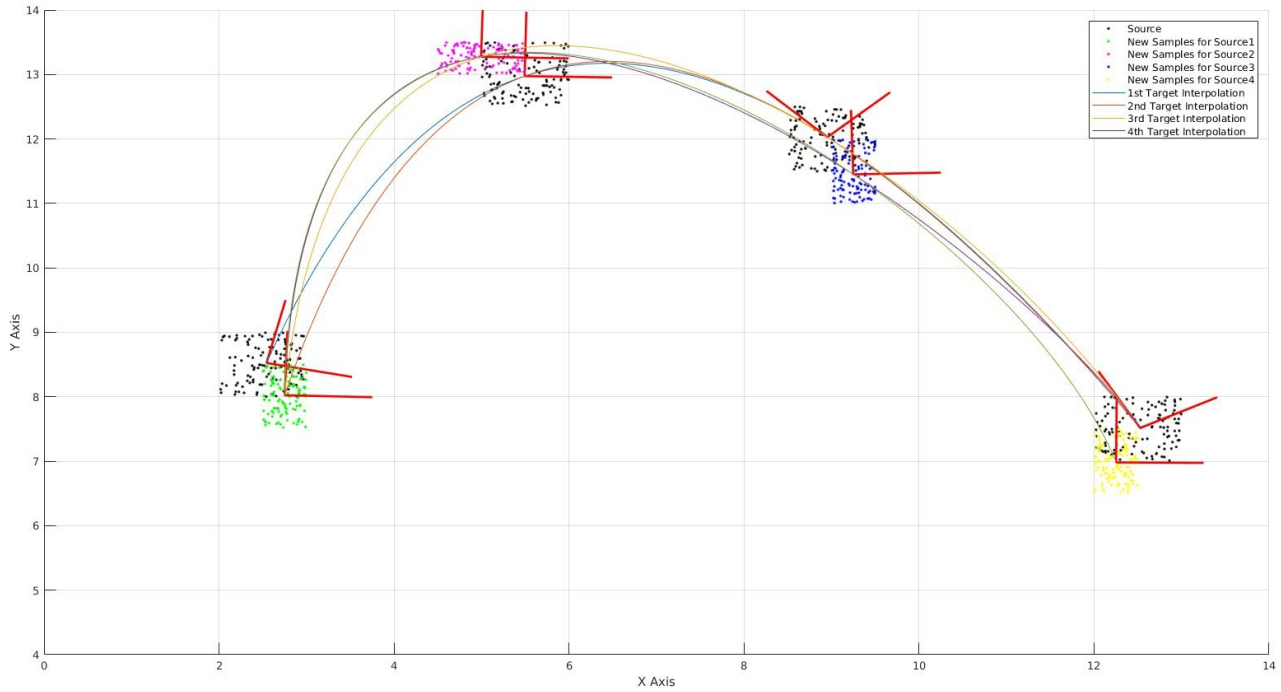


Fig. 4.2: Smooth interpolation with Source Subspace Update

4.3.3 Smooth Interpolation with Training Subspace Update and TDA

The Smooth Interpolation with Training Subspace Update and TDA is the same as the last one but with a little difference. This assumes that the dataset samples have some kind of noise[3] and tries to mitigate that noise. This allows us to predict labels from unlabeled data with a relatively high probability of being correct.

The advantage of this method, compared to the previous one, is that we can choose which of the samples to incorporate onto the existing training PCA. We can only add, for example, samples that scored an accuracy above 90%. This allows us to only have *good* samples in our training subspace.

The following figure tries to demonstrate this process.

[3]In this case, blankout noise

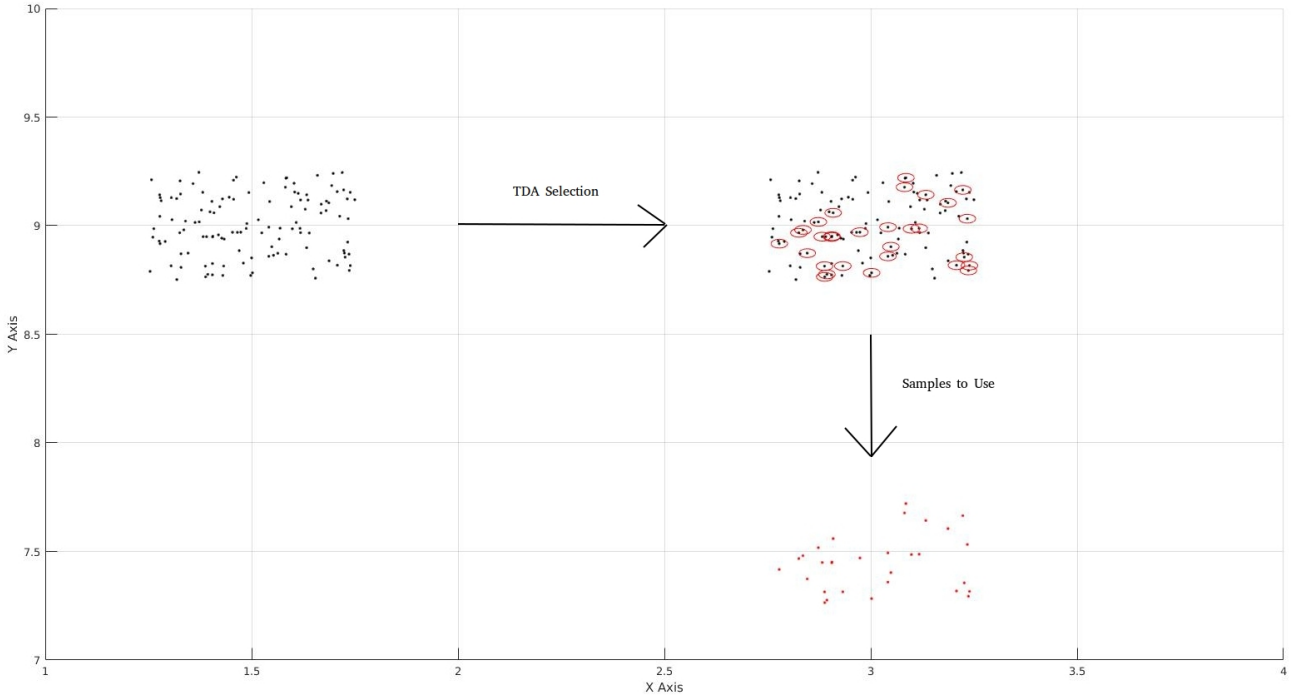


Fig. 4.3: TDA process for selecting new samples

As we can see in figure 5.6, the black dots represent the original samples, the ones this method assumes to have noise and to be corrupted. After applying the method, there were same of those samples (now with mitigated noise) that meet the criteria of accuracy and are flagged with the red circles around them. Those flagged samples (red dots) were originally unlabeled samples but, with the help of algorithm 7, we can predict with a reliable probability the label of each one of them. After this *labeling* we include those samples onto corresponding training subspace.

Algorithm 7: *Transductive Domain Adaptation with mSDA.*[5]

Input: Target dataset $X^t \in \mathbb{R}^{N \text{x} D}$ without labels, Source classifiers $f_k \in \mathbb{R}^D \rightarrow [0..1]^C$

Output: Labels target Y^t

1 Generate class predictions $\mathbf{f}(\mathbf{x_n^t})$ for all $x_n^t \in X^t$

2 Compose an augmented dataset U with $u_n^t = [x_n^t; f(x_n^t)]$

3 Estimate $[W, h] = Alg_2(X^t, p)$

4 Get denoised class predictions for x^t as $\mathbf{y^t} = \sum W_{:,D+1:D+C} f(x)$

5 Get Y^t with $c^* = argmax_c\{y_c^t | \mathbf{y^t}\}$

6 **return** Y^t

Chapter 5

Development and Results

5.1 Development of the work

Succinctly, our work was based on the development of different algorithms trying to discover which one was better. All these algorithms have the same initial data and the same separation of train and test data, only differentiating when it comes on how to manipulate that previously computed separation. As explain, in detail, in chapter 4, one of the developed algorithm utilizes all the train data and all the testing data. Another one of those algorithms, uses PCA to reduce the training and testing data to their principal components and then make an continuously update of the training PCA. The next one, does exactly the same thing but with the slightly difference that the test data is arranged in clusters of 6 hours each. And finally, the last one which is divided into three sub-algorithms, aims to update no only the testing PCA but also the training PCA, with the help of the labels received from TDA. In all of these, we compute a TDA-free approach and a TDA approach trying to visualize if it improves the final accuracy results.

5.1.1 Dataset

Since this work was meant to compare and try to find an improvement over the work done by Hoffman et al. in [14], we had to used the same dataset they used, in order to maintain the integrity and objective conclusions of the test results.

The dataset images were captured from a fixed traffic camera observing an intersection. Frames

were updated at intervals of 3 minutes each with a resolution of 320x240. This resolution posted a problem because conventional detection methods were unable to identify most of the cars. They collected images for a period of two weeks which gives us the possibility of testing in different conditions, which is also a challenging problem for domain adaptation because the changes include illumination, shadows, fog, snow, light saturation, night time infra-red mode, etc. Because of the resolution problem as well as this environment changes, insted of using conventional methods (eg scanning-window car detection or deformable parts model) they (and we) opted for a scene labeling, i.e., we compute features from over the whole image and create a label of the scene instead of focusing only on the vehicles.

After getting all these images from the intersection, labels were human-handed input into a vector as two classes: positive (label = 1) and negative (label = −1). The following figure 5.1 shows five positive and negative images from the used dataset, in different environment scenarios.

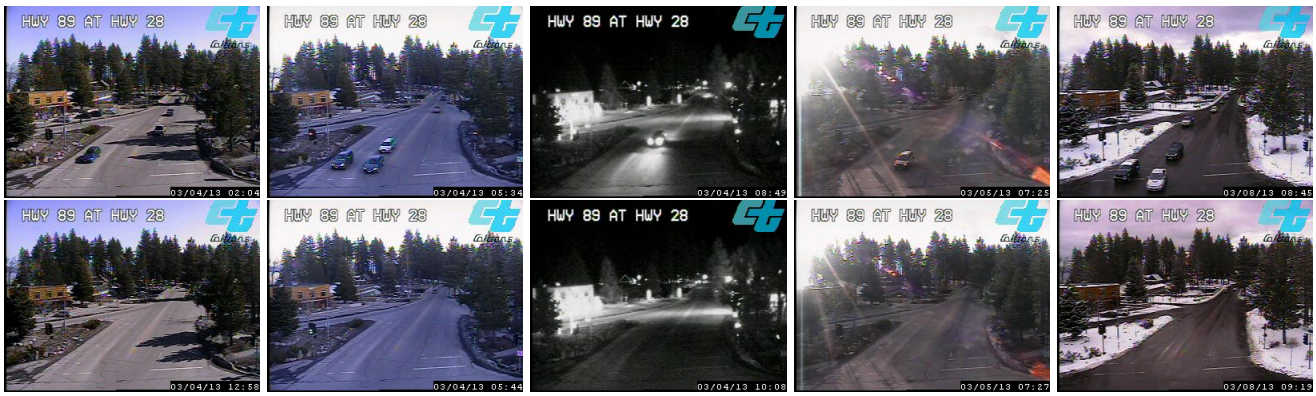


Fig. 5.1: Dataset images used for intersection traffic classification. Positive images in the top row, negative images in the bottom row.

5.1.2 Experiment Setup

When it came to the setup of the experiment, here we didn't use the same training time as them. They only used 50 labeled images (the equivalent of 2.5 hours) for training and then evaluate each of their algorithms on the immediately following 24 hours (480 images) ad 5 days (2400 images).

We, however, used one day of training (480 images) because we predicted that with the information of a whole day (passing by the different stages of light) we'd have a better training dataset for the following evaluation of the next 480 images (one day) and 2400 images (5 days).

As said before, this task is extremely challenging and cannot be solved with conventional ap-

proaches because the images (and especially the cars within those images) are of very low resolution.

They use GIST and SIFT-SPM features to test their CMA algorithm, based on two approaches: Geodesical Flow Kernel (GFK) and Subspace Alignment (SA). We, as the feature dimension is so extensive, first tested our approaches with GIST (512 features) and after tested with features extracted from AlexNet neural network (4096 features).

5.2 Results and Evaluation

We reserved this section to present the results of our work. The idea was to, for each approach, test it with the two set of features, GIST[1] and AlexNet[2]. Unfortunately, that was not possible for the Smooth Interpolation approaches due to the highly computation requirements and the exorbitant time consuming it took to compute each segment. Thus, we test the differences between these two features without adaptation and with the adaptation proposed by Hoffman, CMA algorithm. The rest of the approaches were only tested with GIST features.

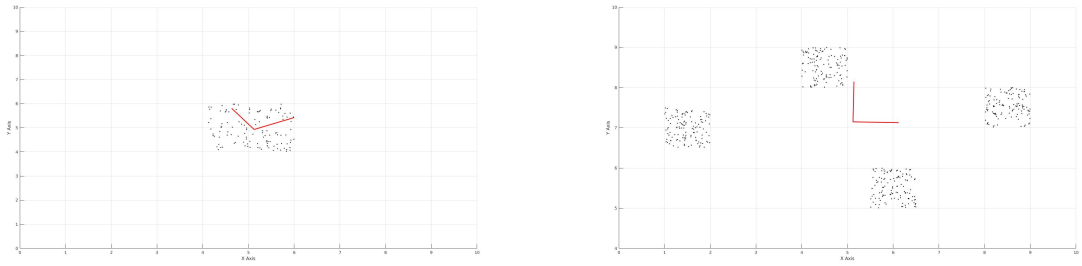
5.2.1 First Test - Influence of feature type and number of training samples

The first test we did had two practical objectives: (1) see the influence of the feature type either on approaches with no adaptation as well as on CMA approach; (2) see if the number of training samples influences the accuracy of those methods due to the application of PCA techniques with a lot of distinct information.

Our hypothesis for this latter objective is that a classification method that uses PCA to lower its features dimension has a better performance when that PCA is composed with samples relatively similar (eg. samples from only 3 hours of a day) instead of a PCA composed with distinct samples (eg. samples from a whole day). We can see the difference in figure 5.2. The results of this test are presented in table 5.1.

[1] GIST has 512 features

[2] AlexNet has 4096 features



(a) **(b)**

Fig. 5.2: Two images showing the difference between 5.2a PCA with similar samples and 5.2b PCA with distinct samples. As can be seen, in the left figure the center of the axis falls inside the point cloud, this implies that it's easier to associate new samples if it fall near that area. Contrariwise, in the figure on the right, the center of the axis is placed outside any point cloud, being tougher to associate new samples to any of the point clouds.

				50 train samples		480 train samples	
				480 test samples	2400 test samples	480 test samples	2400 test samples
NO CMA	GIST	SVM		82.12	58.52	**73.60**	**81.51**
		KNN		47.19	51.10	**76.30**	74.84
	AlexNet	SVM		**88.98**	**76.88**	67.50	78.63
		KNN		**75.05**	**64.51**	71.10	**81.13**
CMA Approach	GIST	SVM	GFK	**83.37**	60.27	81.29	81.67
			SA	**83.99**	59.68	81.29	81.63
		KNN	GFK	46.57	51.44	77.55	79.38
			SA	46.57	51.35	77.55	79.51
	AlexNet	SVM	GFK	81.50	**84.09**	**88.98**	**84.59**
			SA	81.50	**84.05**	**89.19**	**84.80**
		KNN	GFK	**74.22**	**77.93**	**89.39**	**80.51**
			SA	**74.43**	**77.97**	**89.61**	**80.59**

Table 5.1: First test: Accuracy Percentages of Hoffman's Approach with different number of training and testing samples, as well as with different feature type.

From table 5.1 we can withdraw some pertinent conclusions concerning the objectives we wanted to achieve with this test. In response to our first hypotheses, see if the feature type is influential, we can see that when there's no adaptation, the neural network extracted features dominated when less training samples on both tests (either 480 and 2400 samples) but was somewhat precarious when the training samples are vast. In the presence of adaptation, the

neural network features are the overwhelmingly winners. For our second hypothesis, understand the role of training samples, we can see that the accuracies are way higher when we have a bigger number of training samples.
In short

- Choosing the type of feature is essential
- If there's enough training samples, the more we use to train the classifier the better

5.2.2 Second Test - Smooth Interpolation Approaches

This second test is reserved to the approaches we tried to implement. As explained earlier, we proposed three approaches based on cyclical clusters and a smooth interpolation curve to make the classification.

We are going to ascertain four statistical measures to better analyze the performance of our methods. Those measures are: Accuracy, Precision, Sensitivity and Specificity.

- **Accuracy:** the degree to which the result of a measurement conforms to the correct value or the standard. Our algorithms already return the accuracy, so we get the value for this measurement directly from MatLab.

$$\textbf{Accuracy} = \frac{\text{True positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{True Negatives} + \text{False Negatives}} \tag{5.1}$$

- **Precision:** the degree to which repeated measurements under unchanged conditions show the same results. Precision has to be calculated apart from accuracy and with the help of confusion matrices. We use equation 5.2 to calculate the precision for each case.

$$\textbf{Precision} = \frac{\text{True positives}}{\text{True Positives} + \text{False Positives}} \tag{5.2}$$

- **Sensitivity:** measures the proportion of positives that are correctly identified as such. Sensitivity also needs the help of confusion matrices to be calculated. We use equation 5.3 to calculate the precision for each case.

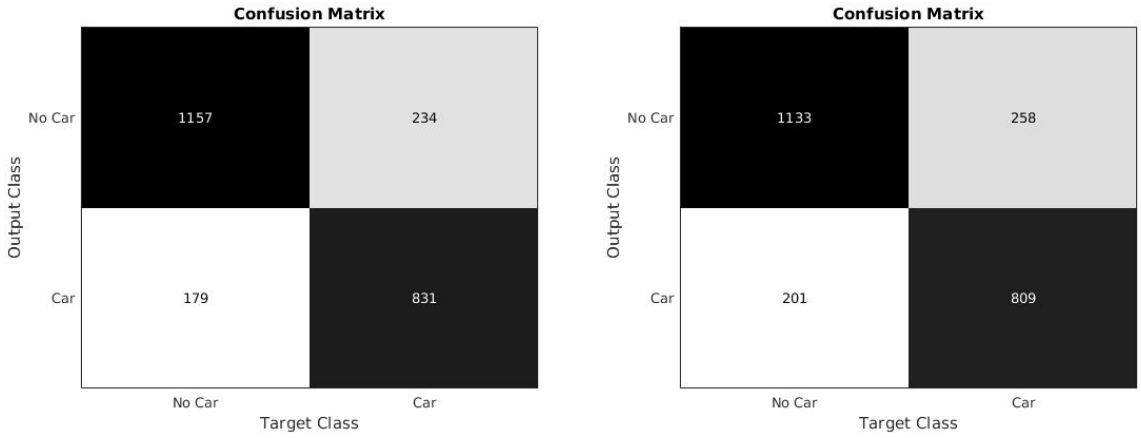
$$\textbf{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{5.3}$$

- **Specificity:** measures the proportion of negatives that are correctly identified as such.Specificity also needs the help of confusion matrices to be calculated. We use equation 5.4 to calculate the precision for each case.

$$\textbf{Specificity} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}} \tag{5.4}$$

We define the variables used on these equations as:

- **True Positives:** The number of images with vehicles that were correctly classified as having at least one vehicle.
- **True Negatives:** The number of images without vehicles that were correctly classified as not having any vehicle.
- **False Positives:** The number of images without vehicles classified as having vehicles.
- **False Negatives:** The number of images with vehicles classified as not having any vehicle.

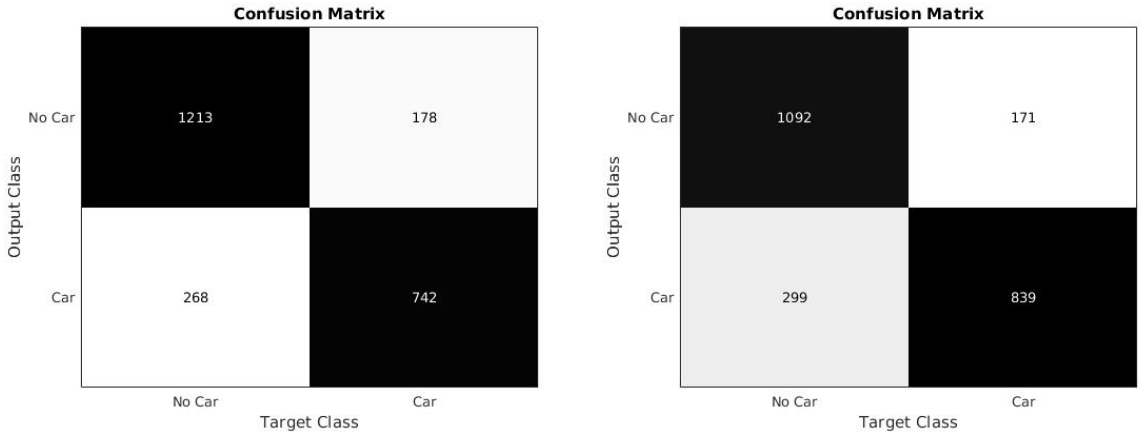


(a) SVM Classifier **(b)** KNN Classifier

Fig. 5.3: Confusion matrix for no adaptation method

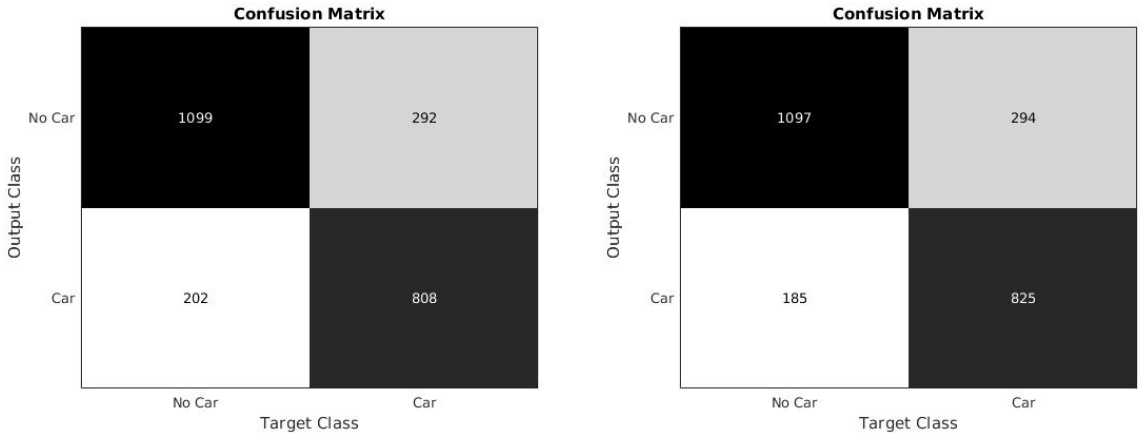
Adaptation Method	Classifier	Accuracy	Precision	Sensitivity	Specificity
No Adaptation	SVM	82.80	78.03	82.28	83.18
	KNN	80.88	75.82	80.10	81.45
SI+TgSU	SVM	81.42	80.65	73.47	87.20
	KNN	80.42	83.07	73.73	86.46
SI+TrSU	SVM	79.43	73.45	80.00	79.01
	KNN	80.05	73.73	81.68	78.86

Table 5.2: Second Test: Smooth Interpolation approaches and their results



(a) SVM Classifier **(b)** KNN Classifier

Fig. 5.4: Confusion matrix for the Target Subspace Update approach



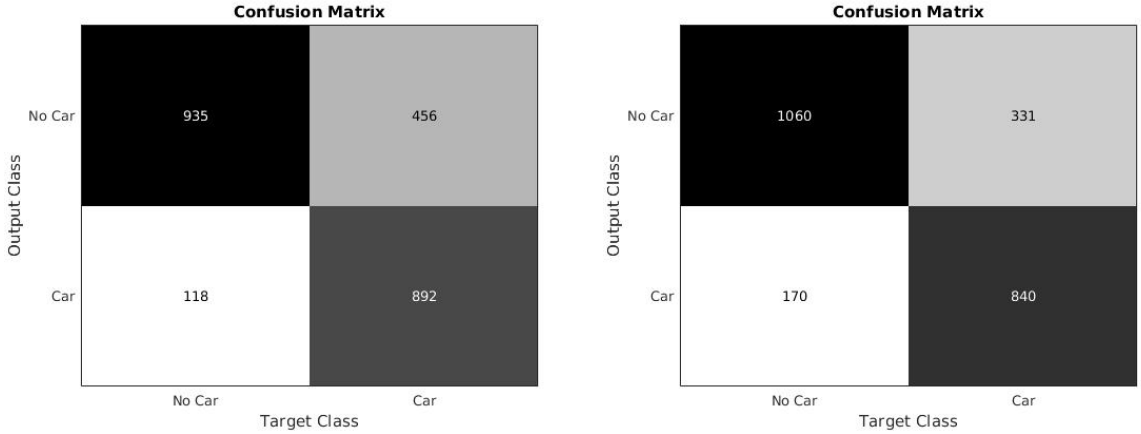
(a) SVM Classifier **(b)** KNN Classifier

Fig. 5.5: Confusion matrix for the Train Subspace Update approach

The results on table 5.2 have for comparison the no adaptation computed with the Hoffman's algorithm. As we can see, for our first approach (Smooth Interpolation plus Target Subspace Update) in terms of accuracy the results were just a little bit worse but the precision was considerably higher ($\pm 8\%$) for the KNN classifier and just a little higher ($\pm 2.5\%$) when the SVM classifier was used. These results weren't accordingly to our expectations since we expected them to be much higher than the ones we got. This may be due to a lack of classifier optimization.

Unfortunately, we couldn't test the TDA approach with the other ones because of how the classifier models are trained on Hoffman's method and we couldn't implement those models on the TDA algorithm. Instead, we used the KNN and SVM from MatLab to train new models and

compared the TDA approach with those new models.



(a) SVM Classifier **(b)** KNN Classifier

Fig. 5.6: Confusion matrix for TDA approach

Adaptation Method	Classifier	Accuracy	Precision	Sensitivity	Specificity
No Adaptation	SVM	75.47	65.47	88.22	66.21
	KNN	78.88	75.99	85.94	71.48
SI+TDA	SVM	76.09	66.17	88.32	67.22
	KNN	79.13	71.73	83.17	76.20

Table 5.3: Results from Smooth Interpolation plus TDA

As expected and shown by table 5.3, the TDA method improved the classification for both classifiers. However, it wasn't a significantly improve, comparing to the time consuming computation it took to compute the results. We ran some more tests and the TDA almost always increased the performance of the classification. It's safe to say that TDA approach should be used as complement when classifying on an unsupervised domain.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

With the development of this work it is possible to conclude that choosing the right type of feature is of the utmost importance since it has been revealed that the performance of the classifiers greatly improves when comparing both features (GIST and AlexNet. See table 5.1). Apart from this conclusion, we could also see that the number of training samples to train the classifiers models is also an important part for the classification process.

Regarding our approaches, the results shows that there is no great improvement relatively to the no adaptation approach. The TDA approach was tested independently from the other two approaches and we conclude that it is helpful since it improves a little the classification performance. However, with our dataset, the time it took to generate all the results, the approaches tested are not beneficial. This is due to the great amount of time needed for each test.

Ultimately, we could conclude that the GIST features though weaker than neural networks, are an excellent descriptors for holistic methods since they have considerably less features (512 `vs` 4096) and the computation time associated with it is much lower. Our approaches, although not achieving the results we were expecting, performed well and with a considerably reliable accuracy for vehicle existence or not. Down the line, hopefully, this work could be done with other circumstances and achieve even greater results.

6.2 Future Work

As future work, there are several new tasks that can be implemented in an intelligent system as well as other ones that can be improved. Some of them are listed bellow.

- Optimize this method and implement an efficient algorithm that uses real-time video acquisition. This will allow the method to be used on traffic surveillance cameras and return accurate information.
- Add to the method a vehicle classification and a counter algorithms in order to give a more discriminative idea of the traffic being analyzed instead of only giving the existence or not of car.
- Optimize classifiers models, in order to design a more robust and accurate model for image classification.
- It would be very interesting, although time-consuming, to see how the neural networks would perform when applying the smooth interpolation approaches. In our case was not possible due to the great amount of time needed.

References

[1] Jorge Batista, Krzysztof A Krakowski, Luís Machado, Pedro Martins, and Fatima Silva Leite. Multi-source domain adaptation using Cλ1-smooth subspaces interpolation. *2016 IEEE International Conference on Image Processing (ICIP)*, pages 2846–2850, 2016.

[2] John Blitzer, Dean Foster, and Sham Kakade. Domain Adaptation with Coupled Subspaces. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 15:173–181, 2011.

[3] Minmin Chen, Kilian Q Weinberger, and John C Blitzer. Co-training for Domain Adaptation. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2456–2464, USA, 2011. Curran Associates Inc.

[4] Minmin Chen, Kilian Q Weinberger, Fei Sha, and Los Angeles. Marginalized Denoising Autoencoders for Domain Adaptation. *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 767—-774, 2012.

[5] Boris Chidlovskii, Stephane Clinchant, and Gabriela Csurka. Domain Adaptation in the Absence of Source Domain Data. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 451–460, New York, NY, USA, 2016. ACM.

[6] Lixin Duan, Ivor W Tsang, Dong Xu, and Tat-Seng Chua. Domain Adaptation from Multiple Domain adaptation from multiple sources via auxiliary classifiers. *Proceedings of the Annual International Conference on Machine Learning*, pages 289–296, 2009.

[7] Lixin Duan, Dong Xu, and Shih Fu Chang. Exploiting web images for event recognition in consumer videos: A multiple source domain adaptation approach. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1338–1345, 2012.

[8] Basura Fernando, Amaury Habrard, Marc Sebban, and Tinne Tuytelaars. Unsupervised visual domain adaptation using subspace alignment. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2960–2967, 2013.

[9] Yaroslav Ganin and Victor Lempitsky. Unsupervised Domain Adaptation by Backpropagation. *Proceedings of the International Conference on Machine learning (ICML)*, (i):1180–1189, 2015.

[10] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach. *Proceedings of the 28th International Conference on Machine Learning*, (1):513–520, 2011.

[11] Gene H Golub and Charles F Van Loan. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)(3rd Edition)*, volume 208-209. 1996.

[12] Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2066–2073, 2012.

[13] Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. Domain adaptation for object recognition: An unsupervised approach. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 999–1006, 2011.

[14] Judy Hoffman, Trevor Darrell, and Kate Saenko. Continuous manifold based adaptation for evolving visual domains. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 867–874, 2014.

[15] a Levey and M Lindenbaum. Sequential Karhunen-Loeve basis extraction and its application to images. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, 9(8):1371–4, 2000.

[16] Mingsheng Long, Jianmin Wang, Guiguang Ding, Jiaguang Sun, and Philip S. Yu. Transfer joint matching for unsupervised domain adaptation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1410–1417, 2014.

[17] Laurens Maaten, Minmin Chen, Stephen Tyree, and Kilian Q Weinberger. Learning with Marginalized Corrupted Features. *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 28:410–418, 2013.

[18] Sinno Jialin Pan, Xiaochuan Ni, Jian-Tao Sun, Qiang Yang, and Zheng Chen. Cross-domain sentiment classification via spectral feature alignment. *Proceedings of the 19th international conference on World wide web - WWW '10*, page 751, 2010.

[19] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning, 2010.

[20] David Ross, Jongwoo Lim, and Ming-Hsuan Yang. Adaptive Probabilistic Visual Tracking with Incremental Subspace Update. *Computer Vision - ECCV 2004*, 3022:470–482, 2004.

[21] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6314 LNCS, pages 213–226, 2010.

[22] Shiliang Sun, Honglei Shi, and Yuanbin Wu. A survey of multi-source domain adaptation. *Information Fusion*, 24:84–92, 2015.

[23] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 1096–1103, 2008.

[24] Wikipedia. k-nearest neighbors algorithm.

[25] Wikipedia. Support vector machine.