Surrogate Approximations for Similarity Measures

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

Doctor of Philosophy
in
Computer Science and Engineering

by

Nagendar G
201099002
nagendar.g@research.iiit.ac.in

International Institute of Information Technology, Hyderabad
(Deemed to be University)
Hyderabad - 500032, INDIA
March 2023

International Institute of Information Technology

Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "**Surrogate Approximations for Similarity Measures**" by Nagendar G, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: C. V. Jawahar

Acknowledgments

I would like to express my sincere gratitude to everyone who has provided guidance, assistance, and support throughout this part of my life's journey. Special thanks to my advisor C.V Jawahar, without whom I couldn't have completed this journey.

Abstract

This thesis targets the problem of surrogate approximations for similarity measures to improve their performance in various applications. We have presented surrogate approximations for popular dynamic time warping (DTW) distance, canonical correlation analysis (CCA), Intersection-over-Union (IoU), PCP, and PCKh measures. For DTW and CCA, our surrogate approximations are based on their corresponding definitions. We presented a surrogate approximation using neural networks for IoU, PCP, and PCKh measures.

First, we propose a linear approximation for the naïve DTW distance. We try to speed up the DTW distance computation by learning the optimal alignment from the training data. We propose a surrogate kernel approximation over CCA in our next contribution. It enables us to use CCA in the kernel framework, further improving its performance. In our final contribution, we propose a surrogate approximation technique using neural networks to learn a surrogate loss function over IoU, PCP, and PCKh measures. For IoU loss, we validated our method over semantic segmentation models. For PCP, and PCKh loss, we validated over human pose estimation models.

Contents

List of Figures

List of Tables

Table Page

Chapter 1

Introduction

1.1 Motivation

Humans primarily differentiate two objects by measuring their similarities. The concept of similarity is fundamentally essential in almost every scientific field. For instance, the relationship between two topological spaces is studied using similarity concepts like congruence and isomorphism. Measuring the similarity between a given query sample and indexed samples is an important problem in search engines. In machine learning, similarity concepts play a fundamental role in evaluating the performance of learned models. A similarity measure is a function that computes the degree of similarity between two objects. Several forms of these similarity measures routinely come up in many real-world problems.

In a typical learning task, we are given access to labeled examples (x, y) drawn from some distribution $\mathcal{P}$ over $X \times Y$, where X is the input space, and Y is the target/output space. Here, we have not made any assumptions on the input space X. The objective of a learning algorithm is to learn a discriminative function $h : X \to Y$, such that, for given an example $(x, y) \in \mathcal{P}$, $(x, h(x))$ is in some sense similar to the training data $\mathcal{D}$. In other words, we want similar inputs to lead to similar outputs. In machine learning, to study the problem of learning, we need additional structure in input and output space. On this front, we need a similarity measure over the input space X and the output space Y. The discriminative function h is computed using these similarity measures. For the input space X, we need a function,

$$f : X \times X \to \mathbb{R} \tag{1.1}$$

satisfying, $\forall x, x^{'} \in X,\ f(x, x^{'}) = f(x^{'}, x)$. It defines a similarity measure over X. If $f(x, x_i) \leq f(x, x_j), \forall\ j \neq i$ then $h(x)(= y)$ should be similar to $h(x_i)(= y_i)$.

In the output space Y, the similarity is usually measured in terms of a loss function. The similarity in the output space evaluates a learned model's performance. The loss function over the output space Y is defined as,

$$L : Y \times Y \to \mathbb{R} \tag{1.2}$$

satisfying, $L(y_i, y_j) \geq 0, \forall y_i, y_j \in Y$. The loss function is used to evaluate the performance of discriminative function h (learned model) for the given sample $x \in X$.

1.1.1 Similarity Measures in Machine Learning

Comparing two images/objects is a fundamental operation in many computer vision applications like classification, clustering, recognition, and retrieval. Comparing two images or, more generally, two samples mainly relies on the concept of the similarity measure. Many similarity measures are proposed in the literature for comparing different types of objects. Different similarity measures give different types of similarities between the samples. No single similarity measure is best suited for all types of data. For example, the standard Euclidean distance works well for comparing the samples in $\mathbb{R}^n$ (samples in the n-dimensional space). However, if we consider two-time series sequences of the same dimension, then Euclidean distance may not be able to capture all their similarities. For example, consider the two-time series X and Y, given in figure 1.1(a). It shows that the Euclidean distance cannot capture their similarities for the given two-time series. In Euclidean distance, i^{th} sample point of X (X_i) is mapped to i^{th} sample point of Y (Y_i). However, X_i is not similar to Y_i for the given time series. On the other hand, Dynamic time warping (DTW) distance (Figure 1.1(b)) can capture the similarities between X and Y. In DTW distance, i^{th} sample point of X (X_i) is mapped to its similar point in Y. In the figure, X_i is similar to Y_{i+2}, so X_i is mapped to Y_{i+2}. For time-series data, DTW distance works well compared to Euclidean distance.

The similarity measures are domain-specific. The applicability of similarity measures may differ from one domain to another domain of data. Such as the similarity measure used to compare two speech signals may not work well for comparing two DNA sequences. There are similarity measures that best suit specific problems. For example, each of the similarity measures canonical correlation analysis (CCA) [18, 51], Edit distance [43], χ^2 Histogram distance [14] and Earth movers distance (EMD) [134, 135] works well for a specific type of problems. The canonical correlation analysis

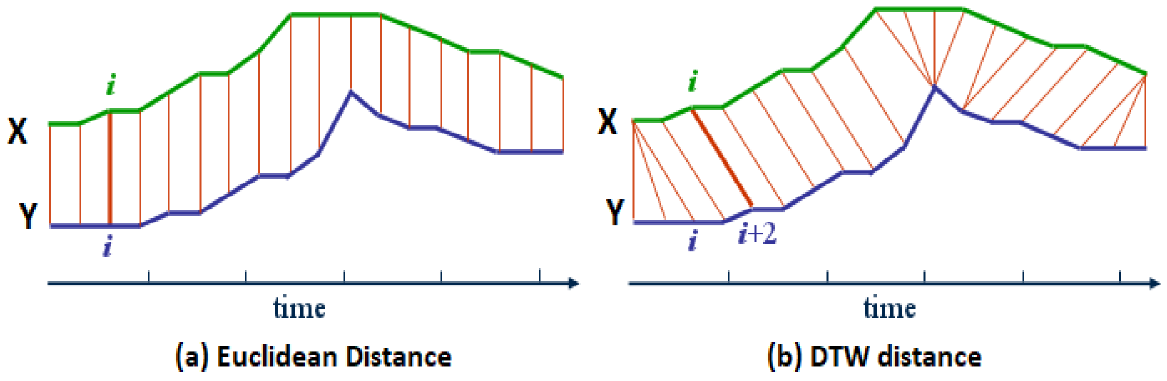


Figure 1.1: Euclidean distance and Dynamic time warping (DTW) for comparing time series data. Here, i corresponds to i^{th} sample point of X (Y), i.e. X_i (Y_i)

(CCA) [18, 51] is best suited to compute the similarity between two random variables. CCA is popularly used for comparing videos, where videos are represented as a tensor variable. CCA has applications in action recognition [108]. On the other hand, the similarity measure Edit distance [43] works well for the problem of string matching. The Edit distance between two strings is the cost of the sequence of operations like insertion, deletion, and substitution of a symbol required to transform one string into another. Edit distances find applications in natural language processing (NLP) and bioinformatics. χ^2 Histogram distance [14] is useful in comparing the histograms. For many computer vision tasks, each object of interest can be presented as a histogram by using visual descriptors. Earth movers distance (EMD) [134, 135] is popularly used for comparing two probability distributions. EMD is widely used in content-based image retrieval. In general, these similarity measures work well with the nearest neighbor (NN) classifier. NN classifier classifies the data based on the given similarity function. One advantage of using the NN classifier is that we can use it with any similarity measure and its performance depends on the given similarity measure.

1.1.1.1 Kernels

The objective of a similarity function is to output small values for similar samples and large values for dissimilar samples. Consider the example given in Figure 1.2. In Figure 1.2(a), the samples marked in red belong to class 1, and those marked in blue belong to class 2. Here, the data is non-linear, and we cannot draw a straight line that separates the given two classes. Consider the Euclidean distance

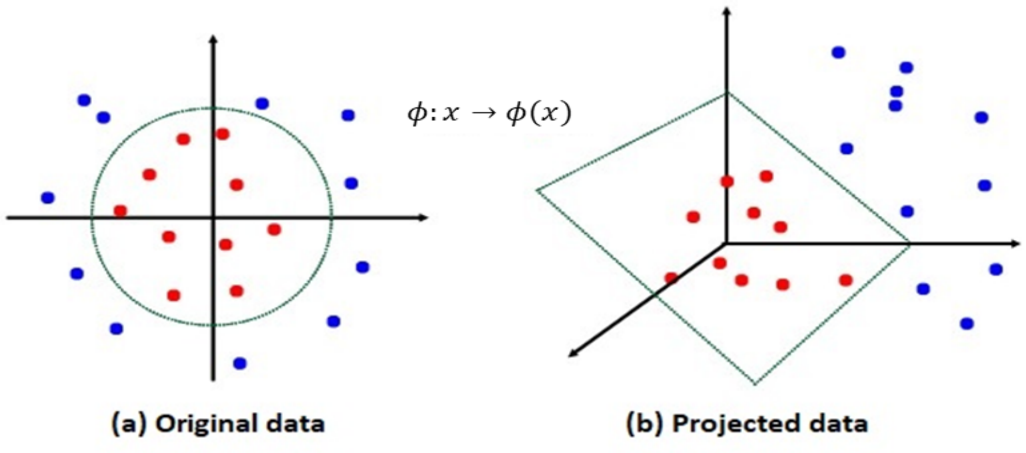


Figure 1.2: Projecting the given data using feature maps. Here, 2-D data is projected into 3-D.

over this data for measuring the similarity. Ideally, the Euclidean distance between class 1 and class 2 samples should be high. However, the Euclidean distance between class 1 and 2 samples around the circle drawn is minimal. Many similarity measures will not work over these types of data to measure the similarity. The primary reason for this is that the given data is non-linear. One possible solution for this problem is to project the non-linear data into a linear space and use the above similarity measures over the projected data. Kernel functions are suitable for this task. Kernels are a class of similarity measures that handles the non-linearities present in the given data. For a given sample $x, x^{'} \in X$, a kernel $\kappa : X \times X \rightarrow \mathbb{R}$ is defined as

$$\kappa(x, x^{'}) = \langle \phi(x), \phi(x^{'}) \rangle \tag{1.3}$$

where ϕ is the feature map, which projects the given data into some dot product space H (Hilbert space). The inner product between the projected samples $\phi(x)$ and $\phi(x^{'})$ can be viewed as a similarity between the given samples in the inner product space H. The inner product between the projected samples is computed using the kernel function (Eq 1.3). Kernels project the given data into an inner product space H using a feature map ϕ, and the similarities are computed in this projected space. Consider the example given in Figure 1.2(b). Here, the original data in $\mathbb{R}^2$ (Figure 1.2(a)) is projected on to $\mathbb{R}^3$ using the featuremap $\phi(x = (x_1, x_2)) = x_1^2 + x_2^2 + 2x_1x_2$. Now, the projected data is linear, and we can find a hyperplane that separates the given two classes. The dot product in this projected space can compute the similarities between the original samples. In general, the dimensionality of the projected space is very high.

Many kernels proposed in the literature describe different notions of similarity between the samples in various settings. For example, the RBF kernel is popularly applied for the samples in $\mathbb{R}^n$. In the RBF kernel, the original data is projected into infinite-dimensional space and the similarities between the given samples are computed in this projected space. For the samples x and $x^{'}$, the RBF kernel is computed as $\kappa_{RBF}(x, x^{'}) = exp(-\gamma||x - x^{'}||^2)$. Histogram intersection kernel [14] serves as a good measure for the histograms. It measures the degree of similarity between two histograms. Pyramid match kernel [53] works well for data where the samples are sets of features with different cardinalities. Kernels are widely used as similarity measures in support vector machines (SVM) classifier. SVMs are the most well-known learning systems based on kernel methods. For the labeled training data, SVMs output an optimal hyperplane, which categorizes the new samples. If there is an appropriate choice of kernel, the SVMs perform well compared to the NN classifier [32,43]. The generalization ability of SVMs contributes to their success.

1.1.1.2 Loss Functions

The objective of a learning algorithm is to learn a discriminative function $h : X \rightarrow Y$ according to a similarity function $f : X \times X \rightarrow \mathbb{R}$, such that $h(x_i) = y_i, i = 1 \; to \; n$, where, $\mathcal{D} = \{(x_i, y_i) : i = 1 \; to \; n\}$ is the given training data. However, during the training, $h(x_i)$ may not give the exact output y_i. We need a similarity measure over the output space to evaluate how well the discriminative function h fits the given training data. Loss function $L : Y \times Y \rightarrow \mathbb{R}$, defines a similarity measure over the output space Y. These are also used to evaluate the performance of given discriminative functions. For a given sample $(x, y) \in X \times Y$, if $h(x)$ deviates from y then the loss function outputs a higher score/error ($L(h(x), y)$ will be high) otherwise it gives a small error or no error ($L(h(x), y)$ will be a small value). There are many loss functions proposed in the literature for measuring the similarity in the output space. Few loss functions commonly used in machine learning are L_2 loss (Mean squared error), L_1 loss, Hinge loss, cross entropy, and intersection over union (IoU).

The L_2 loss or Mean squared error (MSE) is measured as the average squared difference between predictions and actual observations. For a given training sample $(x, y) \in X \times Y$, the L_2 loss corresponding to the discriminative function h is computed as

$$L_{MSE} = \frac{1}{p}\|y - h(x)\|_2^2 \tag{1.4}$$

where, y is the target value for x and $p = dim(y)$. It is the Euclidean distance between y and $h(x)$. Consider the case, where the target space Y contains probability scores ($[0, 1]$), i.e the discriminative function $h(x)$ outputs a probability value between 0 and 1 ($y \in [0, 1]$). If there are c number of classes present in the given data, then $y \in [0, 1]^c$. In this setting, cross-entropy loss is used to measure the performance of a given learning algorithm. For a given sample $x \in X$, the cross-entropy loss corresponding to the discriminative function h is computed as

$$L_{CE} = -\sum_{i=1}^{c} y^i log(h^i(x)) \tag{1.5}$$

where, $y = (y^1, \ldots, y^c) \in \mathbb{R}^c$ is the target value for x and $h(x) = (h^1(x), \ldots, h^c(x))$. Similar to cross-entropy, IoU is also popularly used in measuring the performance of the given discriminative function, where the target space Y contains probability scores.

More recently, in machine learning, neural network-based classifiers are performing well compared to the SVM based methods. In particular, convolutional neural networks (CNN) achieve state-of-the-art performance on a wide variety of problems. In CNNs, the features are learned along with the classifier in an end-to-end fashion. Loss functions play an essential role in the performance of CNNs. For the given problem, the network is trained to minimize the training error, measured using the given loss function. The performance of the CNN network is evaluated using this training error (accuracy).

1.2 Surrogate Approximations in Machine Learning

In machine learning, surrogate approximation techniques are widely used for approximating functions in various applications. It includes approximating a discriminative function using the training data, approximating an objective function in a simpler form [49], and approximating similarity functions to improve their performance [35, 79, 173]. In mathematical programming, sometimes the objective functions are approximated by one of the simpler forms to reduce the computational overhead [49]. Few approximation techniques estimate the decision function using the given functional data, like approximating the optimal discriminative function using SVM [31]. For the given training data, SVM approximates its decision function by minimizing its training loss. Linear regression and SVM regressions are used to approximate the functions whose output is a real or continuous variable. Neural networks are popularly used to find surrogate approximations for various real-valued functions. These are widely used in approximating the non-differentiable functions, which do not have a closed-form expression [58, 66]. Differentiable similarity functions have better properties than non-differentiable similarity functions like the computation of gradients for finding the optimal solution. For example, the similarity measure, intersection-over-union (IoU), is popularly used as the performance measure in segmentation problems. However, due to its non-differentiable property, it cannot be used as a loss function in CNN. The differentiable similarity functions are widely used as loss functions in CNN, like L_2 loss and cross-entropy. For finding the differentiable approximation of non-differentiable similarity measures like edit distance and intersection-over-union (IoU), a few surrogate approximation techniques [90, 127] approximate the given similarity function using some modifications in its definition. Similar to linear techniques, non-linear approximation techniques are also used in functional approximation [55]. These techniques use Gaussian kernels in the functional approximation. Surrogate approximations are also widely used in kernel approximations. In general positive definite kernels guarantees a unique global optimal solution in SVM. However, many of the kernels in the literature are not positive definite, like the sigmoid kernel [61], hyperbolic tangent kernel [149], and edit distance kernel [90]. These kernels are converted into positive definite kernels using surrogate approximation techniques [38, 90].

In our work, we would like to focus on the problem of surrogate approximations for similarity measures. In this space, we propose approximation techniques for different similarity measures to improve their performance in various applications. We introduce different approximation techniques for (i) Speedup and (ii) Improved accuracy.

1.2.1 Problems of Interest

(a) **Linear approximation of DTW distance:** In general, DTW distance has quadratic complexity, which limits its use in various applications with the nearest neighbor classifier. For a given two sequences of length n and m, respectively, the computational complexity for DTW distance is $O(nm)$. It is computationally expensive for large datasets. To compute the DTW distance for a pair of given two sequences, we need to find the optimal alignment with the least cost from all the possible alignments. This is the computationally expensive operation in DTW distance. This work aims to find a linear approximation to the DTW distance. The surrogate approximation will speed up the DTW distance computation, which enables to use of DTW distance for various applications like the word image retrieval over a large corpus.

(b) **Linear approximation kernel over DTW distance:** The DTW distance is a popular similarity measure for comparing the time series data. In the past, there have been attempts to define kernels over DTW distance. These kernels enable the use of DTW distance in kernel methods like SVM, which improves its performance compared to NN classifier. However, these kernels have quadratic complexity and are computationally expensive for large datasets. Also, these kernels are not definite, limiting their applicability in kernel methods. An explicit feature map is a popular method for speedup the non-linear kernels. It approximates the original large dimensional (infinite-dimensional) feature map of non-linear kernels by a small finite-dimensional feature map. This small-dimensional feature map gives a linear approximation of the original non-linear kernel. The explicit feature maps are quite popular for different non-linear kernels like intersection, Hellinger's, χ^2, and RBF kernels. However, this technique does not appear widespread in the time series community. Our second problem of interest aims to find a linear surrogate approximation kernel over DTW distance using explicit feature maps. The linear approximation kernel is computationally efficient compared to all other kernels over DTW distance and can also be used with linear SVM.

(c) **Surrogate approximation kernel over canonical correlation analysis (CCA):** Videos are generally represented using 3D tensors. To compare the videos in the problems like action recognition in videos, the 3D tensors are factorized into lower-dimensional factors, and similarity measures are applied to these factors. Canonical correlation analysis (CCA) is widely used as the similarity measure for comparing these lower-dimensional tensors. It measures the similarity using specially selected discriminative correlation coefficients. Kernelized CCA is proposed to improve its performance in a kernel setting. However, it cannot be applied in all the kernel methods. In this work, rather than kernelizing the CCA, we are interested in defining a surrogate kernel over CCA, which can be applied in many situations where the canonical correlation is used.

(d) **Learning a Surrogate Loss for Semantic Segmentation and Human Pose Estimation** Semantic segmentation is a key topic in computer vision today, and deep neural network models have emerged as the state-of-the-art solution to this problem in recent times. Cross-entropy loss is the typical loss function used to train deep neural networks for this task. However, the success of the learned models is measured using Intersection-Over-Union (IoU), which is inherently non-differentiable. Similarly, CNNs for human pose estimation typically use the regression loss for training the network. However, the accuracy of human pose estimation methods (networks) is measured using metrics such as the percentage of correct parts (PCP) and the percentage of correct key points (PCK). This gap between performance measure and loss function results in a fall in performance, which few recent efforts have also studied. This work aims to learn surrogate loss functions (networks) approximating the IoU, PCP, and PCK loss.

1.3 Major contributions

The thesis has the following major contributions.

1. **Linear approximation of DTW distance:** In this work, we propose Fast Surrogate DTW distance, a linear approximation of naive DTW distance. We need to find the optimal warping path from the possible alignments for computing the DTW distance. This is a computationally expensive operation and requires quadratic complexity. We try to speed up the DTW distance by learning the optimal alignment from the training data. We learn a small set of global principal alignments from the training data, and the optimal alignment for the new test sequences is approximated using these alignments. As far as we are aware, none of the previous methods have exploited the hidden structure of the alignments. We approximate the DTW distance as a sum of multiple weighted Euclidean distances that are known to be amenable to indexing and efficient retrieval. The performance of the proposed Fast Surrogate DTW distance is as good as DTW distance and computationally performs equally as simple Euclidean-based matching. The details of the Fast Surrogate DTW distance are present in Chapter 3.

2. **Linear approximation kernel over DTW distance:** In this work, we have introduced the Fast Surrogate DTW kernel, which is a linear kernel over DTW distance. We have also proposed an explicit feature map for our Fast Surrogate DTW kernel, that enables the kernel to be applied with linear SVM. The explicit feature map for the Fast Surrogate DTW kernel is computed using the global principal alignments learned from the training data. In chapter 3, we present our Fast Surrogate DTW kernel along with its evaluation studies.

3. **Efficient query specific DTW distance for document retrieval:** We present an application using Fast Surrogate DTW distance in chapter 4. These applications widen the scope of our work and validate its robustness to different modalities.

    - This chapter addresses the problem of faster indexing in classifier-based retrieval methods using fast surrogate DTW distance. We introduce the query specific Fast Surrogate DTW distance for faster indexing, which has linear time complexity.

4. **Surrogate approximation kernel over canonical correlation analysis (CCA):** In this work, we have introduced a surrogate kernel over CCA. The proposed canonical correlation kernel (CCK) enables us to compare videos in a kernel framework. The kernel function works well for action recognition as it embeds the temporal context in the videos. It seamlessly integrates the advantages of lower-dimensional representation of videos with a discriminative classifier like SVM. We have also shown that multiple features can be seamlessly integrated into CCK to enhance recognition performance further. We present the details of the proposed CCK kernel in chapter 5.

5. **Learning a Surrogate Loss for Semantic Segmentation and Human Pose Estimation** This work proposes a novel method to automatically learn a surrogate loss function that approximates the IoU, PCP, and PCKh loss, and is hence better suited for providing stronger performance. To the best of our knowledge, this is the first such work that attempts to learn a loss function for improved performance. The proposed loss can be directly learned over any base network. For IoU loss, we validated our method over semantic segmentation models. For PCP and PCKh loss, we validated over human pose estimation models. The proposed approach can also be used for other tasks evaluated using non-differentiable measures. We present the details of the proposed method in chapter 6.

1.4 Publications

Part of the work described in this thesis has previously been presented in the following publications.
Journals:

1. **G Nagendar**, Viresh Ranjan, Gaurav Harit, C. V. Jawahar: Efficient Query Specific DTW Distance for Document Retrieval with Unlimited Vocabulary. J. Imaging, 2018.

2. **G Nagendar**, Vineeth N. Balasubramanian, C. V. Jawahar. Surrogate Loss Networks for Discrete and Non-Decomposable Metrics. PR, 2022 (Under Review)

Conferences:

1. **G Nagendar**, Sai Ganesh Bandiatmakuri, Mahesh Goud Tandarpally, C. V. Jawahar, Action Recognition Using Canonical Correlation Kernels, ACCV 2012

2. **G Nagendar**, C. V. Jawahar, Efficient word image retrieval using fast DTW distance, ICDAR 2015

3. **G Nagendar**, C. V. Jawahar, Fast approximate dynamic warping kernels, CODS 2015

4. **G Nagendar**, Digvijay Singh, Vineeth N. Balasubramanian, C. V. Jawahar, Neuro-IoU: Learning a Surrogate Loss for Semantic Segmentation, BMVC 2018

Other publications during PhD which are not part of this thesis are as follows:

1. Tejaswi Kasarla, **G Nagendar**, Guruprasad Hegde, Vineeth N. Balasubramanian, C.V. Jawahar, Region-Based Active Learning for Efficient Labelling in Semantic Segmentation, WACV 2019

1.5 Organization

- In **Chapter 2**, we provide the necessary background from the domain of computer vision and machine learning for understanding the concepts and methods presented in this thesis.

- **Chapter 3** presents our first major contribution by proposing a linear approximation of the popular DTW distance. We present a detailed analysis of the proposed approximation technique and evaluate it over standard datasets. We also proposed a linear kernel over DTW distance. The proposed kernel is computationally faster compared to all the kernels over DTW distance.

- In **Chapter 4**, we present an interesting application using the proposed linear approximation of DTW distance. We address the problem of faster indexing in classifier-based retrieval methods using the proposed linear approximation of naive DTW distance.

- In **Chapter 5**, we introduce a surrogate kernel approximation over canonical correlation analysis (CCA). The proposed surrogate kernel reports state-of-the-art results for the task of action recognition in videos. It also enables multiple feature fusion for enhancing recognition performance.

- In **Chapter 6**, we present a novel method to automatically learn a surrogate loss function that approximates discrete and non-decomposable metrics - in particular, the IoU, PCP and PCKh loss, and is hence better suited for providing stronger performance.
- Finally, in **Chapter 7** we present the summary of the thesis proposal.

Chapter 2

Background

2.1 Introduction

This chapter provides a brief background for some of the concepts we use to design our approaches for various tasks.

2.2 Similarity Measures

A similarity function is a real-valued function that quantifies the similarity between two objects. This section provides a brief overview of a few popular similarity measures.

2.2.1 Canonical Correlation Analysis (CCA)

Canonical correlation analysis (CCA) [18, 51, 60, 75] identifies and measures the linear relationships between two sets of random variables. It consists of finding a linear combination of variables in each set such that the resultant linear combinations best express the correlations between the given two sets of variables. Let $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_m\}$ be two vectors of random variables and there exists correlations among the variables. The CCA finds a linear combination of variables in X and Y, i.e., $U_k = \sum_{i=1}^{n} a_{ki}x_i = a_{k1}x_1 + a_{k2}x_2 + \ldots + a_{kl}x_n$ and $V_k = \sum_{i=1}^{m} b_{ki}y_i = b_{k1}y_1 + b_{k2}y_2 + \ldots + b_{km}y_m$ such that U_k and V_k have maximum correlation with each other and best express the correlations between X and Y. The random variables $U_1 = a_1^T X$ and $V_1 = b_1^T Y$, where $a_1 \in \mathbb{R}^n$ and $b_1 \in \mathbb{R}^m$, are the first pair of canonical variables. The cross covariance $\sum_{XY} = cov(X, Y)$ for X and Y is an $n \times m$ matrix, whose (i, j) entry is the covariance between x_i and y_j. The first pair of canonical variables (U_1, V_1) is defined using the coefficient vectors a_1 and b_1, which are defined as

$$(a_1, b_1) = argmax_{a_1,b_1} Corr(U_1, V_1) \tag{2.1}$$

where, $Corr(U_1, V_1)$ is the correlation between U_1 and V_1 and defined as

$$Corr(U_1, V_1) == \frac{Cov(U_1, V_1)}{\sqrt{Var(U_1)}\sqrt{var(V_1)}} = \frac{a_1^T \Sigma_{XY} b_1}{\sqrt{a_1^T \Sigma_X a_1}\sqrt{b_1^T \Sigma_X b_1}} \tag{2.2}$$

where, $\sum_X = cov(X, X)$ and $\sum_Y = cov(Y, Y)$. The correlation $Corr(U_1, V_1)$ between U_1 and V_1 is the first canonical correlation ρ_1 between X and Y. Similarly, the second pair of canonical variables (U_2, V_2) are computed using Eq 2.1, subject to the constraint that they should to be uncorrelated with the first pair of canonical variables (U_1, V_1). The 2^{nd} canonical correlation is given by $\rho_2 = Corr(U_2, V_2)$. The k^{th} canonical correlation is the square root of the k^{th} eigenvalue of $\Sigma_X^{\frac{-1}{2}} \Sigma_{XY} \Sigma_Y^{-1} \Sigma_{YX} \Sigma_X^{\frac{-1}{2}}$. For a given random vectors X and Y, we are interested in p canonical correlations, where $p = min\{n, m\}$. The coefficient vectors a_k and b_k for the k^{th} pair of canonical variables (U_k, V_k) are computed as

$$a_k = d_k^T \Sigma_X^{\frac{-1}{2}} \quad and \quad b_k = e_k^T \Sigma_Y^{\frac{-1}{2}} \tag{2.3}$$

where, d_k is the k^{th} eigenvector of $\Sigma_X^{\frac{-1}{2}} \Sigma_{XY} \sum_Y^{-1} \Sigma_{YX} \Sigma_X^{\frac{-1}{2}}$ and e_k is the k^{th} eigenvector of $\Sigma_Y^{\frac{-1}{2}} \Sigma_{YX} \sum_Y^{-1} \Sigma_{XY} \sum_Y^{\frac{-1}{2}}$.

For given random vectors X and Y, its CCA is given as

$$CCA(X, Y) = \sum_{i=1}^{p} \rho_i \tag{2.4}$$

where, ρ_i is the i^{th} canonical correlation between X and Y and $p = min\{n, m\}$.

CCA is often defined using the theory of subspaces. In terms of subspace concepts, canonical correlations are defined as the principal angles between the linear subspaces generated from the given set of random variables. Canonical correlations are invariant to affine transformations with respect to the given inputs [20], i.e. for the given inputs $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$

$$CCA(Px + q, Ry + s) = CCA(x, y)$$

where, $P \in \mathbb{R}^{n\times n}$, $q \in \mathbb{R}^n$, $R \in \mathbb{R}^{m\times m}$ and $s \in \mathbb{R}^m$.

Applications of CCA CCA can be successfully applied for various visual classification tasks [138]. It has been successful in comparing two sets of images [45, 174, 178], where a set of linearly independent vectors represents each image set. It has also been used as a set similarity measure for object recognition problem [40]. Here, each image is represented as a collection of image descriptors and these image descriptors are compared using CCA. CCA has also been applied to the problems of robot localization [148], underwater target classification [120], estimation of face depth maps from color textures [132], and detection of neural activity in functional MRI [44].

2.2.2 Dynamic Time Warping (DTW)

For a given two time series, a naive approach for computing their similarity could be first to sample the equal number of points from both the time series; here, the time series may have variable length. Then, compute the Euclidean distance between the sampled points. This method may not always produce the desired results, mainly due to the variable time series length, as its results match the points that might not correspond well. Also, it cannot capture the local dependencies between neighboring states of the time series. Consider the example given in Figure 1.1(a). From the figure, we can observe that Euclidean distance cannot capture their similarities for the given two-time series X and Y. In Euclidean distance, i^{th} sample point of X (X_i) is mapped to i^{th} sample point of Y (Y_i). However, in the figure, X_i does not correspond well with Y_i.

A successful method to compare time series data (sequences) is dynamic time warping (DTW) [12, 139, 146]. The DTW is used to compute the similarity between two time series [144].DTW has been widely used for matching the time series (1-D signals) in many areas [79], including bio-informatics [1], speech recognition [107], and word recognition [21,128]. For a given two time series, the DTW yields an optimal alignment from all possible alignments. The optimal alignment is used to find the corresponding points between the given time series. These correspondences are used to find the similarity between the given time series. The example given in Figure 1.1(b) shows the DTW distance better capture the similarities between X and Y compared to Euclidean distance In DTW distance, i^{th} sample point of X (X_i) is mapped to its similar point in Y. Since X_i is similar to Y_{i+2}, X_i is mapped to Y_{i+2}. In general, for time series data, DTW distance works well compared to Euclidean distance.

Given two time series, $X = (x_1, \dots, x_n)$ and $Y = (y_1, \dots, y_m)$ of lengths n and m respectively, an alignment π of length $|\pi| = p$ is a pair of increasing p-tuples (π_1, π_2) such that

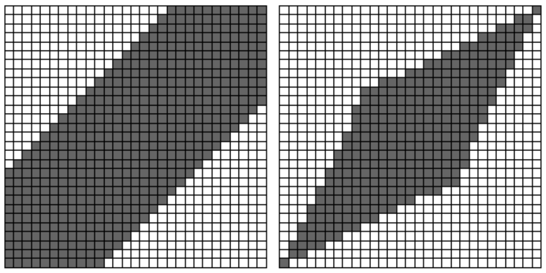


Figure 2.1: Two constraints for speed up the DTW: Sakoe-Chuba Band (left) and an Itakura Parallelogram (right).

$$1 = \pi_1(1) \leq \ldots \leq \pi_1(p) = n,$$
$$1 = \pi_2(1) \leq \ldots \leq \pi_2(p) = m$$

with unitary increments and no simultaneous repetitions. That is, for $\forall 1 \leq i \leq p-1$,

$$\pi_1(i+1) \leq \pi_1(i) + 1, \;\; \pi_2(i+1) \leq \pi_2(i) + 1$$
$$(\pi_1(i+1) - \pi_1(i)) + (\pi_2(i+1) - \pi_2(i)) \geq 1.$$

Intuitively, an alignment π between X and Y describes a way to associate each element of X to one or possibly more elements in Y, and vice-versa. Let $A(X, Y)$ be the set of all possible alignments between X and Y. The DTW between X and Y is given as the minimum distance over all possible alignments,

$$DTW(X, Y) = min_{\pi \in A(X,Y)} \sum_{i=1}^{|\pi|} \varphi(X_{\pi_1(i)}, Y_{\pi_2(i)}) \tag{2.5}$$

where, $\varphi(X_{\pi_1(i)}, Y_{\pi_2(i)})$ is the ground distance between the given sequences indexed by the alignment π. In general, Euclidean distance is used as the ground distance. The resulting alignment is the optimal alignment between the given two sequences. The DTW between the time series X and Y is the distance between X and Y after mapped together with the optimal alignment.

For comparing two time-series X and Y of length n and m respectively, the time and space complexity of DTW is $O(nm)$. The quadratic complexity mainly involves finding the optimal alignment from a large set of possible alignments. The quadratic complexity is particularly prohibitive for large-length time series. The time series containing a few thousands of observations requires much memory and is computationally unattractive. In practice, heuristic constraints are used to speed up DTW. Among these, the most commonly used constraints are Sakoe-Chiba [139] and Itakura [65] constraints. These constraints are shown in Figure 2.1. The shaded region gives the constraint window. Instead of searching in the entire space for optimal path, it restricts the search space to the constraint window. When constraints are used, the DTW algorithm finds the optimal warp path from the constraint window. Using constraints speed up the DTW distance by a constant factor, but the quadratic complexity remains the same. In addition, using these methods, the global optimal path cannot be found if it does not pass through the constraint window. The constraint windows work well if the optimal path is expected closer to the diagonal.

2.3 Classifiers

In machine learning, a large number of problems can be posed as classification tasks. In classification, we will be given a set of observations called "input variables" and the objective is to assign/predict its category ("output variable") from a set of categories. For a given input space X and its corresponding output space Y, the problem can be formulated as finding a classifier $f : X \to Y$ that can correctly classify the given input samples. Formally, the classification problem has the following inputs

- A set of input-output pairs, called the training data $D = \{(x_1, y_1), \ldots, (x_n, y_n)\} \in X \times Y$
- A query sample x_t

For the given query sample x_t, its corresponding predicted category y_t is formulated as

$$y_t = f(x_t, D, P) \tag{2.6}$$

where, P is the parameters for the classification algorithm. Classification has been widely used in many vision tasks, which include semantic segmentation [92], speaker identification [152], document categorization [70], and face recognition [116]. Examples of few popular classification algorithms in machine learning include k-nearest neighbor (k-NN) classifier, support vector machines (SVM), and convolutional neural networks (CNN).

2.3.1 Nearest Neighbor Classifier

The k-nearest neighbor (k-NN) [32, 43] is a supervised classification technique. k-NN has been studied over the past few decades and is widely used as the baseline classifier in many pattern classification problems [17]. It is a non-parametric technique [76]. In non-parametric methods, there are no parameters or a fixed number of parameters irrespective of the data size. If there is no prior knowledge available about the data distribution, then k-NN could be the best choice for the classification. It requires labeled training data for classifying the new samples. In k-NN, for a given new query sample, it first computes its distance from all the training samples using the base similarity measure. The sample is classified by a majority vote of its k-nearest neighbors. It is simply assigned to the most common class among its k nearest neighbors. Here, k is a positive integer. If k=1, then the sample is assigned to the class of its nearest training sample. The choice of k depends on the data. In general, a larger value for k helps to reduce the effects of noisy samples in the training data. For a given query sample x_t, k-NN classifies it as follows,

$$y_t = \underset{c \in \{c_1, \ldots, c_l\}}{argmax} \sum_{x \in NN(x_t, k)} D(x, c) \tag{2.7}$$

where y_t is the predicted category for the given query sample x_t, $\{c_1, \ldots, c_l\}$ are the set of categories and $NN(x_t, k)$ is the k nearest neighbors for x_t according to the given base similarity measure.

$$D(x, c) = \begin{cases} 1 & \text{if } y = c \\ 0 & \text{else} \end{cases}$$

where y is the given category/output for the sample x. The performance of k-NN algorithm mainly depends on the given base similarity measure used for measuring the similarity between the query sample and the training samples [4]. This raises the importance of the similarity measure (distance function) used in k-NN algorithm. Several studies have been conducted to analyze the effect of similarity/distance measures on k-NN classifier performance [4, 29, 62, 123]. In [62], Hu *et.al.* analyzed the effect of four distance measures, including Euclidean, Cosine, Chi square, and Minkowski distances on k-NN performance over four medical domain datasets (Blood, Breast cancer, Ecoli, and Pima) which are chosen from the UCI machine learning repository. The results of this comparative study are given in figure 2.2. From the results, we can observe that the performance of the k-NN classifier depends on the distance function and the choice of distance function depends on the given dataset. Minkowski distance is performing well on Breast cancer and Ecoli datasets, whereas, Chi-square is performing well on Blood and Pima datasets.

k-NN is one of the simplest classification algorithms, however, it has some limitations. It needs to store all the training data to classify a given sample, which results in memory overhead. For problems

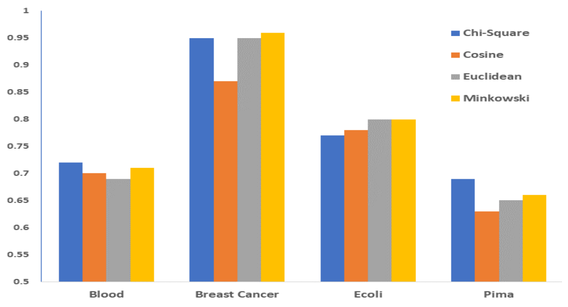


Figure 2.2: Effect of similarity measures on k-NN performance. Here, the similarity measures Euclidean, Cosine, Chi square, and Minkowski are compared over Blood, Breast cancer, Ecoli, and Pima datasets.

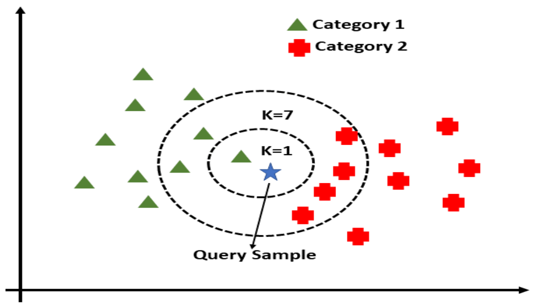


Figure 2.3: Role of k value on k-NN performance. Here, the results are given for k=1 and k=7.

where the training set is large, the k-NN classifier is computationally expensive, and testing is impractical. For classification, the k-NN classifier gives equal weightage to all the features. It may result in undesirable performance for problems where only a subset of features are important. The value for k also plays an important role in k-NN performance. Consider the example in figure 2.3. In the figure, the samples marked in blue belong to category 1, and samples marked in green belong to category 2. For a given query sample (marked in red), the regions for k=1 and k=7 are marked in dotted circles. From the figure, we can observe that if k=1, the query sample belongs to category 1, and if k=7, it belongs to category 2. Also, while computing the nearest neighbors during the testing phase, it does not consider their class distribution. It may not always produce desirable results on imbalanced data.

2.3.2 Support Vector Machines (SVM)

Support vector machines (SVM) are introduced by Vapnik [31] *et al.* It is a popular learning method for binary and multiclass classification [22, 165]. The basic idea is to find the hyperplanes that separate the given data classes. It finds a hyperplane that separates the data into two classes for binary classification. This subsection presents an overview of support vector machines, starting with hard margin SVM and followed by soft margin SVM.

2.3.2.1 Hard margin SVM (Separable Case)

In a binary classification setting, let $D = \{(x_i, y_i); i = 1 \text{ to } n\}$ be the given training data, where $x_i(\in \mathbb{R}^d)$s are the data samples and $y_i(= \{+1, -1\})$s are their corresponding labels. Linear classification is the problem of finding a linear hyper plane, which separates the given two classes. All the hyper planes in $\mathbb{R}^d$ can be characterized by the two parameters $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$, and expressed as

$$w^T x + b = 0 \tag{2.8}$$

This gives a decision function

$$f(x) = sign(w^T x + b) \tag{2.9}$$

However, a given hyperplane characterized by the parameters (w, b) is equivalently expressed as $(\lambda w, \lambda b)$, $\forall \lambda \in \mathbb{R}^+$. Thus we define the hyperplane (w, b) as the one which separates the two classes by a distance of at least one. This can be expressed as

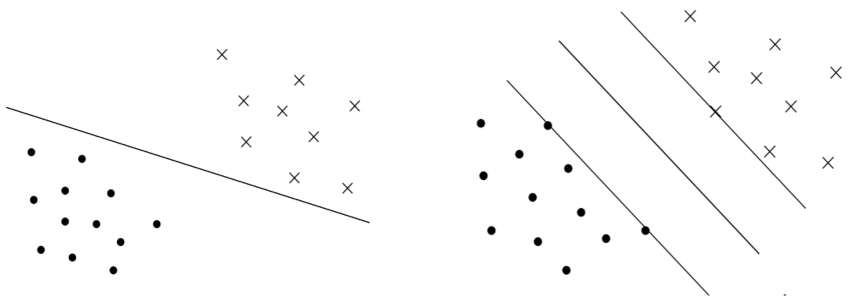


Figure 2.4: Two separating hyperplanes for a linearly separable data. The hyperplane in the right hand side has maximum margin compared to the other hyperplane.

$$\begin{gathered} w^T x_i + b \geq 1 \text{ if } y_i = +1 \\ w^T x_i + b \leq -1 \text{ if } y_i = -1 \end{gathered} \tag{2.10}$$

or in a compact way

$$y_i(w^T x_i + b) \geq 1 \tag{2.11}$$

Consider a binary classification problem given in Figure 2.4. In the figure, both the hyperplanes correctly separate the given two classes of data. The margin of a hyperplane is defined as its distance from the closest data sample. In figure 2.4, the hyperplane on the right-hand side has the maximum margin compared to the other hyperplane. In general, the samples around the decision hyperplane have high uncertainty. They represent very uncertain classification decisions. A hyperplane with a maximum margin has very few uncertain samples, which results in minimal uncertain classification decisions. It also guarantees minimal test error compared to other hyperplanes. The maximum margin hyperplane is less sensitive to the noise in the data. For a given classification problem, support vector machines find a hyperplane that separates the given classes with the maximum margin. The maximum margin hyperplane is computed by minimizing the norm of w, i.e., $\|w\|$. The SVM problem for classification is formulated as

$$\min \ \frac{\|w\|}{2} \tag{2.12}$$

subject to

$$y_i(w^T x_i + b) \geqslant 1 \ \ \forall i = 1 \ to \ n \ \ ; y_i \in \{+1, -1\}$$

The above optimization problem is solved using the concept of Lagrangian dual. The Lagrangian for the above problem is given as

$$L(w, b, \alpha) = \frac{1}{2} w^T w - \sum_{i=1}^{n} \alpha_i \Big[y_i(w^T x_i + b) - 1 \Big] \tag{2.13}$$

where α_is are the Lagrangian multiplier.

To get the Lagrangian dual, the Lagrangian has to be minimized with respect to w and b. Thus, differentiating $L(w, b, \alpha)$ w.r.to w and b, and setting it to zero gives the following expressions

$$\frac{\partial L(w, b, \alpha)}{\partial w} = 0 \implies w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{2.14}$$

$$\frac{\partial L(w, b, \alpha)}{\partial b} = 0 \implies \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.15}$$

Substituting (2.14) and (2.15) in (2.13) gives the following lagrangian dual

$$Q(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{2.16}$$

The lagrangian dual $Q(\alpha)$ has to be maximized w.r.to the lagrangian multiplier α, this gives the following lagrangian dual formulation for the SVM primal problem (2.12).

$$\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{2.17}$$

subject to

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\alpha_i \geq 0 \text{ for } i = 1, 2, \ldots, n$$

This is a quadratic programming (QP) problem, and many techniques are available for solving this formulation. After the computation of Lagrangian multipliers α_i, the expression (2.14) is used to find the optimal hyperplane.

2.3.2.2 Soft margin SVM (Non-Separable case)

In the previous section, we discussed the case where the data is linearly separable. Consider the data given in Figure 2.5. For this data, we cannot find a separating hyperplane using hard margin SVM as

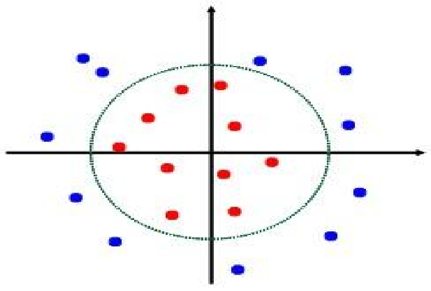


Figure 2.5: Example of a non-linear data. Here, we cannot find a hyperplane, which separates the two classes.

discussed in Section 2.3.2.1. Even for the linearly separable data, an outlier may cause a hard margin SVM to overfit the training data in its search for a separating hyperplane. One solution to this problem is to allow some training error during the training. It prompted the development of soft margin SVM [31], which can handle non-separable data. It introduces a positive slack variable ξ in the constraints 2.10 and allows some misclassification error during the training. The resulting problem is solved by minimizing the misclassification error. This approach gives the following soft margin SVM formulation

$$\min_{w,\xi} \qquad \frac{1}{2}w^T w + C\sum_{i=1}^{n}\xi_i \tag{2.18}$$

subject to

$$y_i(w^T\phi(x_i)+b) \geq 1-\xi_i$$

$$\xi_i \geq 0 \ \ \forall i = 1,\ldots,n$$

Here C is a regularization parameter. It controls the tradeoff between the complexity of the machine and the number of non-separable points. If the value of C is large, it leads to overfitting, and if it is very small, it allows many misclassifications, leading to underfitting. The value for C is chosen using cross-validation techniques.

2.3.3 Convolutional Neural Networks (CNN)

Over the past few years, there has been increasing interest in feature learning for various tasks like object detection, recognition, segmentation, etc, using machine learning methods. Deep learning models offer a solution to this problem. These are based on multi-layer neural networks. In these networks,

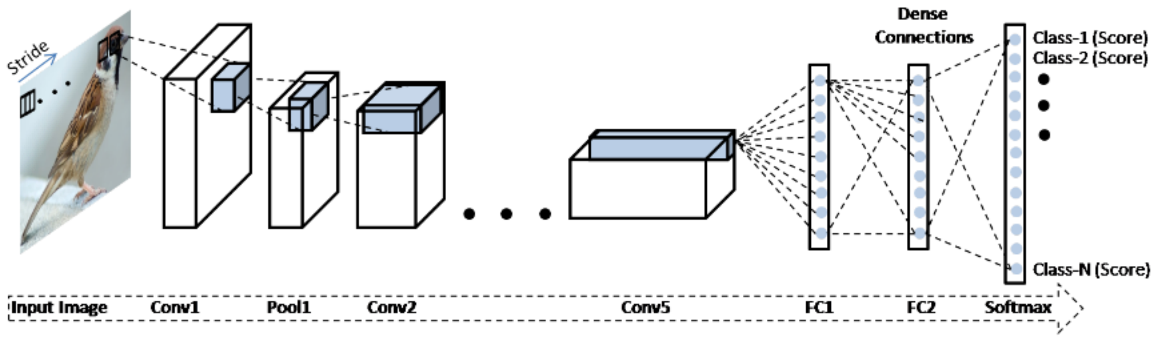


Figure 2.6: Typical architecture of a convolutional neural network for image classification.

each layer learns a set of features for the given problem. A feedforward neural network involves an input layer, multiple hidden layers, and an output layer. Each neuron in the hidden layer contains an activation function. Input to a neuron in the k^{th} layer is the linear mapping of the output of $k-1$- layer neurons. Given an input $x_0 \in \mathbb{R}^d$, it maps to an output $x_m \in \mathbb{R}^p$ through a sequence of functions f_i called layers. The output at j^{th} layer is given as $f_j(x_{j-1}, w_j)$, where w_j is the j^{th} layer parameters and x_{j-1} is input to the j^{th} layer. In general, feed-forward neural networks are trained using the back propagation technique. These are susceptible to the vanishing gradient problem. The gradient of the loss with respect to the network parameters becomes increasingly smaller in the initial layers. Since these gradients become increasingly smaller in each iteration, the weights and biases in the initial layers will not be updated effectively. It is the main limitation of neural networks in terms of the number of layers. In fully connected networks, neurons are connected to all the input dimensions. If we consider a typical image classification problem using fully connected networks with images of size 128×128. It results in an input dimension of 16384. Training the network for this higher dimensional data is practically very difficult.

A Convolutional Neural Network (CNN) is a multi-layer feed-forward neural network that can learn multiple layers of non-linear features. CNNs are particularly useful for image data. In CNN, instead of connecting all the input image pixels to the neurons in the next layers, each neuron in the immediate layer only connects to the pixels in a small patch of the image using receptive fields. This makes training simpler compared to fully connected models. Given an image $I \in \mathbb{R}^{w \times h \times c}$, where w, h, c are the width, height and number of channels (for RGB image $c = 3$), the output at j^{th} layer of CNN is given as a

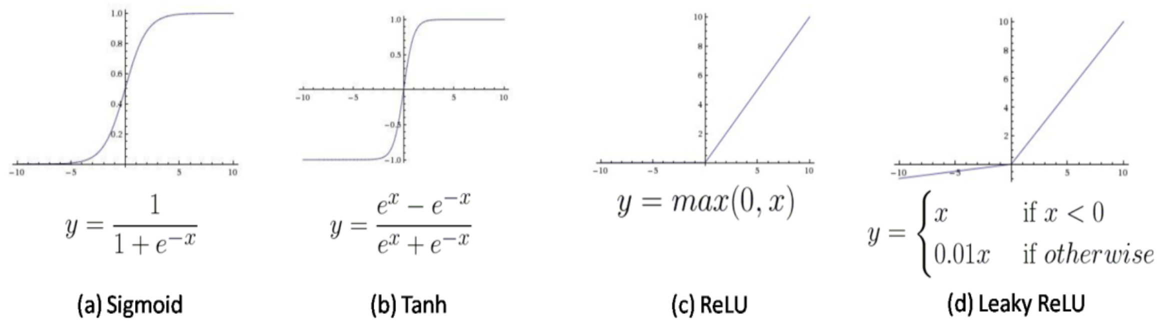


Figure 2.7: Popular activation functions used in convolutional neural networks

spatial map $x_j \in \mathbb{R}^{w_j \times h_j \times c_j}$, where w_j, h_j are the spatial resolution of x_j and c_j is the number of feature channels in j^{th} layer.

2.3.3.1 Layers of CNNs

The CNN contains different functional layers, and each of these layers performs different types of tasks. Some important functional layers in CNN are the convolutional layer, pooling layer, and fully connected layer. A typical CNN network is given in figure 2.6. Here we show five convolutional layers and one pooling layer.

Convolutional Layer: In convolutional neural networks, convolutional layer is the most important layer. A convolutional layer computes the convolution between an input $x \in \mathbb{R}^{w \times h \times c}$ and a set of k filters $f \in \mathbb{R}^{w' \times h' \times c}$ to produce an output $y \in \mathbb{R}^{w'' \times h'' \times k}$. The convolutional operation is performed as follows

$$y_j^n = f(\sum_{i=1}^{F} x_i^{n-1} * w_{ij}^n), j = 1, \ldots, k \tag{2.19}$$

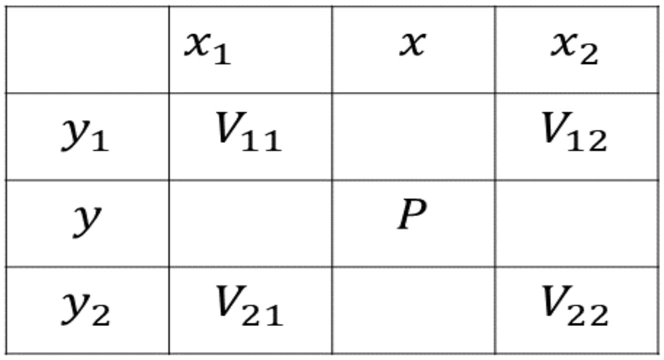
where, n indicates the layer index, i is the input channel index, w_{ij}^n are the filter weights, $*$ denotes convolution operation, $f()$ is the activation function and y_j^n is the j^{th} output featuremap. Each of the filters is applied as a sliding window over the input. It acts as the local feature detector for images. As the layers go deeper, it computes more complex features. The parameters (weights) are shared among all the neurons in a convolutional layer. These features do not change according to spatial positions. Convolution is a translation invariant operator. After each convolution operation, non-linear activation functions are applied to learn complex transformations in the input space. Few popular activation functions are sigmoid, tan hyperbolic, rectified linear units (ReLU)s [111], and leakyReLU. These activation functions are shown figure 2.7. Here both the sigmoid and tanh functions suffer from vanishing gradient problem. This is mainly due to its smoothness towards the lower and higher domain values. This leads to slower training. The activation function ReLU solves this problem by having non-saturating gradients. The hyper-parameters of a convolutional layer are weights in the filters (w, h), the number of filters (k), the step size (s), and the padding width (p). Padding is used to preserve spatial dimensionality.

Pooling Layer: In CNN architecture, a pooling layer is followed by a convolution layer. The pooling layers are used to progressively reduce the spatial size of feature maps obtained from the convolutional layer. The pooling layers play an important role in CNN architecture. In general, the convolutional layers preserve the spatial dimension of the feature maps. If the input image is of size $w \times h \times 3$, then convolutional operation produces an output $w^{'} \times h^{'} \times$ (number of filters). For large images, it results in a huge number of features. Training on such a huge number of features is very difficult and may also result in overfitting. Many pooling techniques are available for reducing the feature dimensions, such as max pooling and average pooling. The most popular technique is max pooling, as it extracts the most dominant features from its corresponding region. As we go further down from the input layer, pooling layers reduce the featuremap resolution and thereby capture the rich semantic features.

Fully Connected Layer: In a typical CNN architecture, after a set of convolutional and pooling layers, a set of fully connected (FC) layers are applied. These are treated as a classifier in the form of a multi-layer perceptron. It maps the convolution features to an output space. For a classification problem, its output space dimension is equal to the number of classes. Neurons in a fully connected layer have full connections to all activations in the previous layer. In figure 2.6, we show 3 fully connected layers. After the fully connected layers, a softmax layer is applied to convert the FC layer scores in the range (0,1). The nodes in the softmax layer generally contain different activation functions. In general, the softmax function is used as the activation function, which is given as,

$$y_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.20}$$

The output from this softmax function can be treated as the probability values for each label.



	x_1	x	x_2
y_1	V_{11}		V_{12}
y		P	
y_2	V_{21}		V_{22}

Figure 2.8: The 4 nearest neighbors for an interpolated sample in bilinear interpolation.

Upsampling Layer: In the classification task, the spatial dimension of the featuremaps is reduced using the above layers. In the final layer, the number of neurons equals the number of classes. In general, downsampling is needed for the classification task. However, downsampling is not always needed in CNN. For the problems like semantic segmentation, where the objective is to classify each pixel in the image, we also need an upsampling layer. Here we need to construct an output with the exact spatial dimensions of the input. Upsampling layers are needed for increasing the spatial dimension of the deep features in the later layers. Many upsampling techniques are available for increasing the spatial dimensions in CNN. A few popular upsampling techniques are bilinear interpolation and deconvolution.

Bilinear interpolation is the simplest upsampling technique and is an extension of linear interpolation for functions of two variables. In bilinear interpolation, interpolation is performed using the nearby values. To interpolate at a location, it needs four nearest neighbors. The weighted average of these four nearest neighbor values is assigned to the new location. These weights are calculated using the distance from the nearest neighbors. Suppose if we want to interpolate at a location (x, y) using the bilinear interpolation and we know its 4 nearest neighbors (x_1, y_1), (x_1, y_2), (x_2, y_1) and (x_2, y_2). The location of interpolation is given in figure 2.8. The bilinear interpolation is given as

$$P = \frac{(x_2 - x)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)}V_{11} + \frac{(x - x_1)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)}V_{21} + \frac{(x_2 - x)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)}V_{12} + \frac{(x - x_1)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)}V_{22} \tag{2.21}$$

Deconvolution is the inverse function of the convolution operation. The deconvolutional layer is used for upsampling the down-sampled featuremaps obtained from the CNN. In CNN, the deconvolution term is generally referred to as convolution operation for upsampling or transposed convolution. The

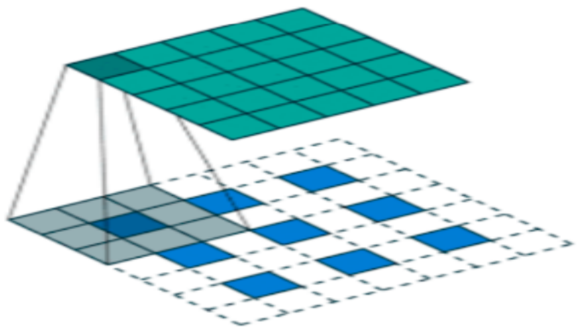


Figure 2.9: Deconvolution operation over a 3×3 image (blue squares). Here the 3×3 image is upsampled to a 5×5 image (green).

deconvolution operation performed over a 3×3 image is explained in figure 2.9. Here the 3×3 image is upsampled to a 5×5 image. The white boxes denote the zero-padding around each pixel. In general, the deconvolution for an input image is the convolution operation over the padded image. In [113], the authors proposed a DeconvNet, where zeros padding around each pixel can be avoided with an unpooling layer. The convolution operation is directly applied over the unpooled images. Deconvolution gives better results compared to bilinear interpolation. This is mainly due to the learning process involved in the deconvolutional layer.

2.3.3.2 Training

Training a CNN network requires finding the optimal weights for each learnable parameter in the network. These are filter weights (W) and biases (b), which correspond to the convolutional and fully connected layers. The most popular optimization algorithm used for finding the optimal parameters is the mini-batch stochastic gradient descent (SGD) with momentum. The backpropagation algorithm [136] is used for updating the parameters of each layer in CNN network.

For finding the optimal weights, we formulate the learning process in terms of an optimization problem with the following objective function

$$\theta^* = argmin_\theta \sum_{i=1}^{N} L(x^i, y^i) + \lambda \|W\|_2^2 \tag{2.22}$$

where $L(.)$ denotes the loss function which is parameterized by θ and $\theta = (w, b)$. Here x^i, y^i denotes the input sample and the ground truth label vector for the i^{th} sample, respectively. The second part of the equation minimizes the L_2 norm of the weight coefficients, also referred to as the weight decay, which prevents over-fitting by reducing the complexity of the network. This is also known as the regularization process.

In general, the loss function for a deep convolutional network with non-linear activation functions is highly complex and non-convex. The most popular optimization algorithm used is the mini-batch stochastic gradient descent (SGD) with momentum. SGD is a first-order iterative optimization method that computes the gradient of the loss and iteratively update the weights in the direction where the loss decreases as given below:

$$\begin{aligned} \theta_{n+1} &= \theta_n - \eta\Delta \\ \Delta &= \mu\Delta + \frac{\partial L}{\partial \theta_n} \end{aligned} \tag{2.23}$$

where η is the learning rate that decides the step size of the update. μ is the momentum factor that determines how much gradient direction from the previous step should be considered in the current update. Note that there exist many weight initialization schemes for better learning the optimal weights.

2.3.3.3 Regularization

The goal of any machine learning model is to generalize well on unseen examples and prevent over-fitting of the training data. The same is applicable in the context of deep learning. In this section, we discuss a few regularization techniques from the literature,

Dropouts: Dropouts [153] is a regularization technique proposed in [153]. In dropouts, at each iteration of training, a few neurons are selected randomly with a probability p and dropped out during training. All the connections to the selected neurons are made zero in the forward pass, and in the backward pass, the weights are not updated. It makes each neuron learn individually during the training, which avoids overfitting. It is similar to the concept of learning an ensemble of networks. Now, the learned network is less sensitive to specific neurons, which results in a generalized network. All neurons are preserved but scaled during testing with a factor of p.

Data Augmentation: Convolutional neural networks have been performing well on many computer vision problems. However, their success depends on the amount of training data. In practical scenarios, sometimes collecting the labeled data is very expensive. If the network is trained on small datasets, it may not generalize well on test data, leading to poor generalization. Data augmentation provides a solution to these problems. It extracts more information from the training data using augmentation techniques. Few popularly used augmentation techniques are translation, rotation, cropping, and scaling. All of these techniques are affine transformation over given training images, which can be expressed as,

$$y = Wx + b$$

Here x is the original image. However, during these transformations, one must be careful in preserving the label information.

2.3.3.4 Loss functions in CNN

In convolutional neural networks, the goal is to minimize or maximize a function with respect to the network parameters. Minimization is the most common case in CNN. The function that is minimized/maximized in CNN is typically called the objective function. It can also be called a loss, error, or cost function. In CNN, the loss function measures how far the network output is from the target/desired output. It is essential to choose the correct loss function in CNN. Since the gradient of the loss function is used for updating the network parameters during the training, the loss function plays an essential role in CNN. In CNN, if an appropriate loss function is not used, it directly affects the convergence of the network. Loss functions can be broadly divided into two categories depending on the learning task, classification loss and regression loss. In this section, we discuss a few loss functions in each category.

2.3.3.4.1 Classification Loss: For classification, we generally use cross-entropy and hinge loss.

Cross Entropy Loss: Cross entropy is widely used as the loss function in classification models, where the model output is a probability value between 0 and 1. In CNN, a softmax layer is applied before computing the cross entropy loss for converting the network scores in to probability values. For multiclass classification, cross entropy loss for a sample x_i is calculated as,

$$L = -\sum_{i}^{c} y_i log(y_i^{'}) \tag{2.24}$$

where, y_i is the ground truth, $y_i^{'}$ is the network output for the sample x_i and c is the number of classes. For binary classification problem, the cross-entropy is calculated as

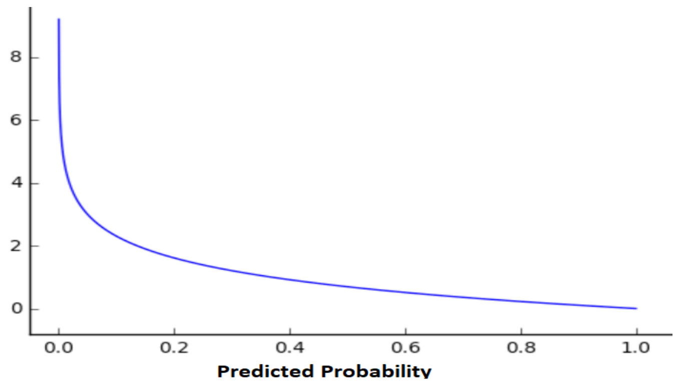


Figure 2.10: The behaviour of cross entropy loss over the predicted probability scores. Here, X- axis contains the predicted probability scores and their corresponding cross entropy loss values are plotted on Y-axis.

$$L = -(y_i log(y_i^{'}) + (1 - y_i)(1 - log(y_i^{'})) \tag{2.25}$$

The behavior of cross-entropy loss is shown in figure 2.10. As the predicted probability of the true class gets closer to one, the loss becomes small. If it reaches closer to zero, the loss increases rapidly.

Hinge loss Hinge loss is used for maximum margin classification. It is popularly used in support vector machines. Hinge loss is not a differentiable function, however, due to its convex nature, it is widely used in the machine learning community. For binary classification, where the class labels are either +1 or -1, the hinge loss is given as,

$$L = \sum_i max(0, 1 - y_i h(x_i)) \tag{2.26}$$

where, y_i is the ground truth and $h(x_i)$ is the network output.

2.3.3.4.2 Regression Loss For regression, we generally use mean square or L_2 loss.

Mean Square Loss/L_2 loss For a given problem, the mean square loss/error (MSE) is measured as the average of squared difference between predictions and actual observations. The mean square loss is defined as,

$$L = \frac{1}{n}\sum_{i=1}^{n}\|y_i - y_i^{'}\|_2^2 \tag{2.27}$$

where y_i is the ground truth and $y_i^{'}$ is the predicted value for the i^{th} training example. n is the number of training samples. The mean square loss is only concerned with its magnitude and does not look into its direction. Due to the squaring operation involved in the definition, the predictions far from the ground truth are more penalized compared to the near ones.

In addition to these loss functions, other loss functions exist, such as KL divergence, margin-based raking loss, negative log-likelihood, cosine embedding, and others. These loss functions are problem-specific and suit well for a certain class of problems. Also, note that, except hinge loss, all these loss functions are continuous and differentiable. In CNN, the loss function must be differentiable for training the network.

2.3.3.4.3 Loss functions in Generative Adversarial Networks (GAN) Generative adversarial networks (GAN) is a deep learning architecture consisting of two neural networks competing with each other to improve their prediction accuracy. The two neural networks are generally referred to as the generator and the discriminator. Min-max loss is the standard loss function used in GAN, which is given as

$$L = E_x[log(D(x))] + E_z[log(1 - D(G(z)))] \tag{2.28}$$

where, $D(x)$ represents the discriminator estimate of the probability when x come from the real training data and $G(z)$ is the generator output when given noise z. E_x and E_z are the expected values respectively over the real data and generated fake instances $G(z)$. The generator tries to minimize the above loss function while the discriminator tries to maximize it. The min-max loss function is derived from the cross entropy loss over actual and generated fake (noise) data distributions.

2.4 Kernel Methods in Machine Learning

Linear models can be computed efficiently and are well understood compared to non-linear models. However, many real-world problems are inherently non-linear, and these problems are difficult to interpret using linear models. One solution to visualize the non-linear data using linear models is to project the data into a linear space. Now, all the linear models can be used to interpret the projected data.

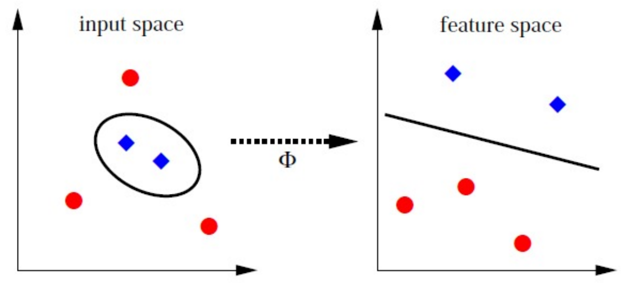


Figure 2.11: Projecting the data using a feature map. Here the projected data is linearly separable.

The projection has to preserve all the similarities in the original space to get a better interpretation. In machine learning, feature maps offer a solution to this problem. Feature maps are based on the idea of cover's theorem, which states that a "complex pattern classification problem cast in a high dimensional space non linearly is more likely to be linearly separable than in a low dimensional space".

2.4.1 Kernels and Feature maps

Feature map It is a mapping function $\phi : X \to V$, which maps the input data X into some higher dimensional vector space V so that the data is linearly separable in the projected space and the projected space is equivalent to the original input space. Here V is called the feature space. A feature map over non-linear data is shown in figure 2.11. Here the projected data is linearly separable.

Kernel of a Function The kernel of a function F is an equivalence relation on the function's domain that roughly expresses the idea of "equivalent as far as the function F can tell". Let X and Y be two sets and F be a function from X to Y. The elements x_1 and x_2 of X are said to be equivalent if $F(x_1)$ and $F(x_2)$ are equal, i.e. they mapped to the same element in Y. Formally, if $F : X \to Y$ is a function then kernel of F is defined as

$$ker(F) = \{(x_1, x_2) \in X \times X : F(x_1) = F(x_2)\}$$

Kernels are used to find a similarity between the elements of a set with respect to a function defined over that set.

Kernel Function A kernel is a function from $X \times X$ to $\mathbb{R}$. A kernel function $\kappa : X \times X \to \mathbb{R}$ which is either continuous or has a finite domain is said to be a **valid kernel (positive definite)** if it is symmetric and can be decomposed as

$$\kappa(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle \tag{2.29}$$

where $\phi : X \to F$ is a feature map from X to some higher dimensional feature space F and $\langle . \rangle$ is the inner product in F.

or

It is a (i) Symmetric function (ii) Kernel matrix corresponds to any finite subset of X is a positive definite matrix, i.e, $\forall S = \{x_1, \ldots, x_{|S|}\} \subseteq X$, $|S| < \infty$; the kernel matrix $\kappa_{S,S}$ is a positive definite matrix, where $\kappa_{S,S} \in M_{|S| \times |S|}(\mathbb{R})$ and $\kappa_{S,S}(i,j) = \kappa(x_i, x_j)$.

The kernel generally corresponds to a feature map over its input space. In feature maps, the similarities in the projected space are computed using the associated kernel function. Since the dimension of the feature space is usually high, the operations in the feature space require a huge computational cost. The kernel trick allows us to do the computations in the feature space using kernel functions, which depend on the dimension of the original space. Consider the problem of distance computation (Euclidean) in the feature space F. It can be computed as

$$\begin{aligned} d(\phi(x_1), \phi(x_2))^2 &= (\phi(x_1) - \phi(x_2))^T (\phi(x_1) - \phi(x_2)) \\ &= \phi(x_1)\phi(x_1)^T - 2\phi(x_1)\phi(x_2)^T + \phi(x_2)\phi(x_2)^T \\ &= \kappa(x_1, x_1) - 2\kappa(x_1, x_2) + \kappa(x_2, x_2) \end{aligned}$$

Here, $x_1, x_2 \in X$ and $\phi(x_1), \phi(x_2) \in F$. The computational cost of Euclidean distance in F depends on the dimension of the projected samples ($dim(F)$). Using the kernel trick, now it only depends on the dimension of original space ($dim(X)$).

Examples of Kernels

- Polynomial Kernel : $\kappa(x_1, x_2) = \langle x_1, x_2 \rangle^d + c$
- Gaussian Kernel : $\kappa(x_1, x_2) = exp(\frac{-||x_1 - x_2||^2}{2\sigma^2})$
- Tan hyperbolic Kernel : $\kappa(x_1, x_2) = tanh(\alpha x_1 x_2 + c)$

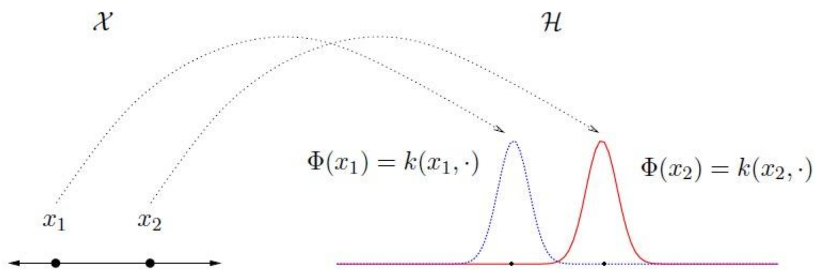


Figure 2.12: Projection of the data into its corresponding Reproducing kernel Hilbert space.

2.4.2 Reproducing Kernel Hilbert Space (RKHS)

Let X be an arbitrary set and H be a Hilbert space of real or complex valued functions over X then H is said to be a reproducing kernel Hilbert space (RKHS) if the linear map $F_x : H \rightarrow \mathbb{R}$ defined as, $F_x : f \rightarrow f(x)$ is continuous $\forall x \in X$.

Let H^* be a dual space of H, consisting set of all continuous linear functions from H to $\mathbb{R}/\mathbb{C}$. For every element $g \in H$, we can define a function $\psi_g : H \rightarrow \mathbb{R}$ as

$$\psi_g(f) = \langle g, f \rangle \forall f \in H \tag{2.30}$$

The functions ψ_g, $\forall g \in H$ are elements of H^*. From Riesz representation theorem, every element in H^* can be uniquely expressed in the above form (2.30). Since F_x is a continuous function over H, it belongs to H^*. Thus it can be expressed as

$$f(x) = F_x(f) = \langle f, K_x \rangle, \forall f \in H \text{ where } K_x \in H \tag{2.31}$$

This implies that $\forall x \in X$, there exists a function $K_x \in H$, such that $f(x)$ can be evaluated by taking its inner product with K_x. Since K_x is a function over X, it can be written as $K_x(y)$. The space of all these functions can be encoded into a single function $\kappa : X \times X \rightarrow \mathbb{R}$, which is defined as

$$\kappa(x, y) = K_x(y) \tag{2.32}$$

This function is called the reproducing kernel for the Hilbert space H. It is completely determined by the Hilbert space H. Since $K_x(y) = \kappa(x, y)$, $K_x(y)$ is a valid kernel. Every reproducing kernel Hilbert space is associated with a unique valid kernel. Also, given a valid kernel κ, we can construct a feature space H (RKHS), such that k computes the dot product in H. In figure 2.12, we show a sample's projection into its corresponding reproducing kernel Hilbert space.

2.4.3 Kernels in Machine Learning

In Section 2.4.1, we have presented some examples of kernels over $\mathbb{R}^n$, but kernels can also be applied over other domains such as sets, text, strings, and graphs. This section presents examples of a few popular kernels over different domains.

Kernels over Histograms The simplest way to represent the images is by using histograms. The histogram intersection kernel is one of the popular kernels applied over histograms. Histogram intersection is a technique proposed in [157] for color indexing and successfully applied for object recognition. It measures the degree of similarity between two histograms. For a given two histograms $H_1 = (h_{11}, \ldots, h_{1n})$ and $H_2 = (h_{21}, \ldots, h_{2n})$, the histogram kernel is defined as

$$\kappa(H_1, H_2) = \sum_{i=1}^{n} \min\{h_{1i}, h_{2i}\} \tag{2.33}$$

Kernels over Sets Let D be any set and $\mathcal{P}(D)$ is its corresponding power set. For $A_1, A_2 \in \mathcal{P}(D)$, a kernel over D can be defined as follows

$$\kappa(A_1, A_2) = 2^{|A_1 \cap A_2|} \tag{2.34}$$

where $|A_1 \cap A_2|$ is the cardinality of $A_1 \cap A_2$.
Its corresponding feature map ϕ is defined as

$$\phi(A)_U = \begin{cases} 1 & \text{if } U \subseteq A \\ 0 & \text{otherwise} \end{cases}$$

Here, the elements of $\phi(A)_U$ are enumerated using the elements of $\mathcal{P}(D)$ and the dimension of $\phi(A)_U$ is $|\mathcal{P}(D)|$.

Kernels over Text Let $\mathcal{D}$ be the set of documents, consider a dictionary which is the set of all words in the documents. For a given document $d \in \mathcal{D}$ define a feature map as follows

$$\phi(d) = (f(t_1, d), f(t_2, d), ..., f(t_N, d)) \tag{2.35}$$

where N is the size of the dictionary and $f(t_i, d)$ is the frequency of i^{th} word in the document d. The corresponding kernel is defined as

$$\begin{aligned}\kappa(d_1, d_2) &= \langle \phi(d_1), \phi(d_2) \rangle \\ &= \sum_{i=1}^{N} f(t_i, d_1) f(t_i, d_2)\end{aligned}$$

Kernels over Strings Consider an alphabet Σ consisting of m symbols. A string $s = s_1.....s_{|s|}$ over Σ is a finite sequence of symbols from Σ and Σ^n is the set of all strings of length n.

p-Spectrum Kernel For a given string s, the featuremap for the p-Spectrum kernel is defined as

$$\phi^p(s) = (\phi^p_u)_{u \in \Sigma^p}; \phi^p_u = |\{(v_1, v_2) : s = v_1 u v_2\}| \tag{2.36}$$

Here Σ^p is the feature space. The associated p-Spectrum kernel is defined as

$$\begin{aligned}\kappa_p(s, t) &= \langle \phi^p(s), \phi^p(t) \rangle \rangle \\ &= \sum_{u \in \Sigma^p} \phi^p_u(s) \phi^p_u(t)\end{aligned}$$

Fisher Kernel Let X be the given data, for a data sample x, $P(x/\theta)$ be a probability model where θ is a vector of model parameter. $\delta\theta$ is the gradient operator with respect to θ, $log_e P(x/\theta)$ is the log-likelihood of x with respect to the model over the given set of parameters θ. The Fisher score for the given data sample x is its gradient of the log-likelihood with respect to the set of model parameters θ.

$$F_x = \delta_\theta log_e P(x/\theta) \tag{2.37}$$

The Fisher score gives an embedding into the feature space. The Fisher kernel [68, 118] over this feature space is defined as

$$\kappa(x_1, x_2) = F_{x_1} I^{-1} F_{x_2} \tag{2.38}$$

where I is the Fisher information matrix. All the above kernels are standard kernels over different domains. These kernels are problem dependent, a single kernel may not work well for all the problems.

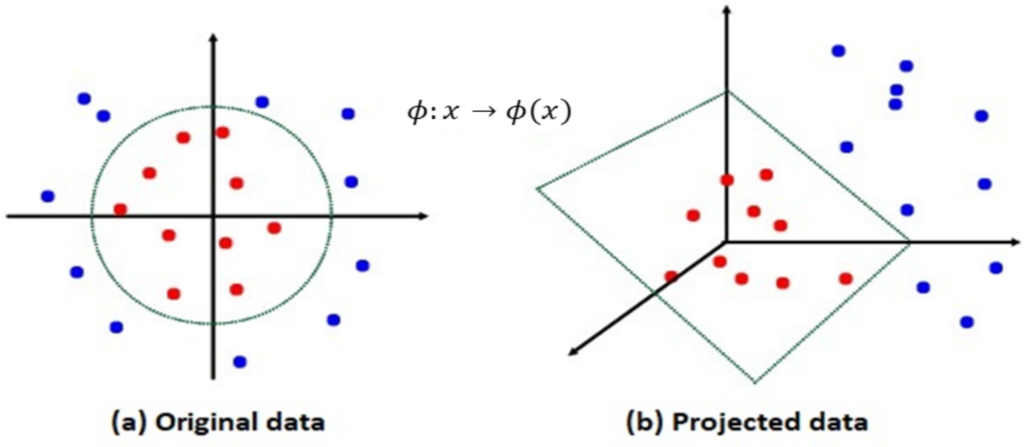


Figure 2.13: Projecting the data using non linear feature map. Here the projected data is linearly separable.

2.4.4 Kernels in SVM

Kernels play an essential role in support vector machines (SVM). In SVM, the non-linearities present in the data are handled using kernels and their associated feature maps. Since, for the non-linear separable data (figure 2.13(a)), it is not possible to find a separable hyperplane, SVM uses the concept of a kernel induced feature space. In this, the data is projected into a higher dimensional feature space where it is linearly separable. The projection is performed using the feature map associated with the given kernel. In figure 2.13(b), we can observe that the projected data is linearly separable. The SVM finds a separating hyperplane in the kernel induced projected space. One such separating hyperplane is shown in figure 2.13(b). The computational problems arising from the high dimensional feature space are handled using the kernel trick. The SVM formulation using kernels is given as

$$\min_w \ \frac{\|w\|}{2} \tag{2.39}$$

subject to

$$y_i(w^T\phi(x_i) + b) \geqslant 1 - \xi_i \ \ \forall i = 1 \text{ to } n \ \ ; y_i \in \{+1, -1\}$$

The main difference between the above formulation and the formulation given in Eq 2.12 is the usage of the feature map. In the above formulation, the data is first projected using the feature map ϕ, then, the SVM formulation given in Eq 2.12 is applied over the projected data. The Lagrangian dual for this formulation is given as

$$\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i x_j) \tag{2.40}$$

subject to the constraints
$(1) \sum_{i=1}^{n} \alpha_i y_i = 0$
$(2) 0 \leq \alpha_i \leq C \ \ for \ \ i = 1, 2, \ldots, n$

In this formulation, the inner product between the feature maps $\phi(x_i)$ and $\phi(x_j)$ is replaced using its corresponding kernel function $\kappa(x_i, x_j)$. This avoids the computations in the higher dimensional feature space. It is called kernel trick.

importance of positive definite kernels in SVM If the kernel is not a positive definite, then the Lagrangian formulation given in 2.40 will not be convex, which does not guarantee a unique global optimal solution [28]. The uniqueness of the solution gives the superiority of SVM over other classifiers like nearest neighbor classifier and neural networks.

2.4.5 Indefinite kernels

In machine learning, there is a large class of kernels that are not positive semidefinite (indefinite), such as sigmoid kernel [61] and hyperbolic tangent kernels [149]. Also, the kernels induced from similarity measures like edit distance [90], canonical correlation analysis (CCA) [18], and dynamic time warping (DTW) [146] are not positive definite. Even though these kernels are indefinite, their corresponding similarity measures found applications in many problems. For example, the kernel computed using the edit distance, $\kappa(s, s^{'}) = edit(s, s^{'})$, where $edit(s, s^{'})$ is the edit distance between the strings s and$s^{'}$ is not positive definite. However, edit distance is a widely used similarity measure for classifying biological sequences. It is successfully used with the nearest neighbor classifier. Although these similarity measures have applications in many problems, they can not be used as kernels in SVM (as discussed in Section 2.4.4), which limits their applicability. This is mainly due to their indefinite property. The positive definite kernels over similarity measures allow us to explore the non-linearities in the given data. Since these kernels can be used with SVM, it guarantees a unique global optimal solution for the given problem compared to the nearest neighbor classifier. For a given problem, these kernels with SVM perform well compared to its corresponding similarity measure with the nearest neighbor classifier.

Indefinite kernels are previously explored in SVMs [38, 61, 90, 169]. Lin and Lin [61] converted the indefinite sigmoid kernel into a conditionally positive definite kernel using some restrictions on kernel parameters. They solved the resultant non-convex dual problem using SMO type decomposition methods. Cheng *et al.* [115] represented the indefinite kernels in reproducing kernel Krein spaces. Due to its non-positivity, they stabilize the loss function instead of minimizing it. Some previous methods [38, 169], directly used the indefinite kernels with SVM. Although these methods perform well, the convergence of SVM with these kernels to the unique global optimal solution is not guaranteed. This is mainly due to the non-convexity of SVM with indefinite kernels.

The kernel over edit distance, $\kappa(s, s^{'}) = edit(s, s^{'})$ is not positive definite. However, Haifeng *et al.* [90] converted this indefinite kernel in to a positive definite kernel by defining it as,

$$\kappa(s, s^{'}) = e^{-\beta \; edit(s, s^{'})} \tag{2.41}$$

The positive definiteness depends on the value of β, which is data dependent. In general, the DTW distance does not yield a positive definite kernel. This is mainly due to its non-metric property. However, few kernels are defined over DTW distance using different techniques [12, 54, 146, 183]. In [54], the authors corrected the negative definite kernel matrix by its square. There are also some attempts where minor changes are incorporated in the definition of DTW distance for defining a positive definite kernel. Hayashi *et al.* [56] projected the time series into a Euclidean space such that the resulting representation approximates the DTW distance.

Chapter 3

Fast Surrogate DTW: A Surrogate Approximation of DTW Distance

This chapter presents a surrogate approximation for dynamic time warping (DTW). The proposed surrogate approximation is a linear approximation of the DTW distance. The main goal of this chapter is to speed up the computation of DTW using its surrogate approximation. This is achieved by introducing the global principal alignments, which avoids the computation of optimal alignments for new query images. These pre-computed global principal alignments capture all the correlations in the given dataset. The proposed linear approximation technique makes the DTW based document image retrieval computationally feasible. The performance of the proposed method is as good as naive DTW distance and computationally performs equally as simple Euclidean-based matching. Using the proposed surrogate approximation technique, we also present a linear approximation kernel over DTW, which can be used with linear SVM.

3.1 Introduction

Distance (Similarity) functions play an important role in a wide variety of problems, including regression, classification, and clustering. They are used to find the similarity/dissimilarity between the samples. There are several distance functions available in machine learning literature like Euclidean, Geodesic, Earth Movers Distance (EMD) [135], dynamic time warping (DTW) [12, 16, 146], and so on. Each of these distance functions computes different types of similarity and is useful for different problems. For example, Euclidean distance is widely used to compare the sequences in the same dimensional space. The main limitation of using Euclidean distance for time series data is that it does not capture the local dependencies between neighboring states of the time series and is sensitive to distortion in the time axis. For a given pair of sequences, it finds the optimal alignment by ignoring both global and local shifts in the time dimension. DTW distance is popularly used for comparing word image representations [130]. This is mainly due to its ability to capture local dependencies and handle variable-length representations. DTW is often used in speech recognition [107] to determine the similarity between two speech signals representing similar spoken words. In speech recognition, the length of the signals is permitted to vary for a given same spoken word, but the overall speech signals are similar. In addition to speech recognition, DTW has also been found useful in many other disciplines [79], including word recognition [21, 128], bioinformatics [1], data mining and gesture recognition. DTW is commonly used in data mining as a distance measure between time series.

For a pair of given two sequences, to compute their DTW distance, we need to find the optimal alignment which has the least cost of all the possible alignments. This is the computationally expensive operation in DTW distance. For a given two sequences of length n and m respectively, the computational complexity for DTW is $O(nm)$. The Euclidean distance can be computed in linear time, however its main limitation is that it does not capture the local dependencies. In the first part of this work, we try to speed up the DTW distance by learning the optimal alignment from the given training data. As far as we know, none of the previous methods have exploited the hidden structure of the alignments. We approximate the DTW distance as a sum of multiple weighted Euclidean distances, which are known to be amenable to indexing and efficient retrieval. We call this approximated DTW distance as Fast Surrogate DTW distance.

For a given set of sequences, there are similarities between the top alignments (least cost alignments) of different pairs of sequences. This work explores these similarities by learning a small set of global principal alignments from the training data. To compute the global principal alignments, first, we compute the top alignments for all pairs of samples from the given training data. Then the global principal alignments are computed from these top alignments. We use 2D-PCA for computing the global principal alignments from the training data. In our experiments, we observe that we can represent the internal structure of the alignments using these global principal alignments. Instead of computing the optimal alignments for a given new query, we use only these precomputed global principal alignments for computing the Fast Surrogate DTW distance. Since we avoid the computation of alignments, the proposed Fast Surrogate DTW is computationally efficient compared to naive DTW distance.

Over the last decade, SVM has emerged as the most popular approach in classification. This is mainly due to its state-of-the-art performance on a wide variety of computer vision problems [57, 73, 80] like image classification, recognition and retrieval. In SVM, kernels act as the similarity measure.

A wide variety of similarity measures are available in the literature for computing various types of similarities. Unfortunately, most of these similarity measures may not yield positive definite kernels and thus cannot be used along with SVM. Positive definiteness of the kernels guarantees global optima in SVMs. Although these similarity measures do not yield positive definite kernels, they have popular applications in many fields. For example, DTW distance does not yield a positive definite kernel, but it works well on many time series classification problems with Nearest Neighbour (NN) classifier [175]. In general, SVM performs well compared to NN classifier if the appropriate choice of kernel exists [167]. Due to its superiority, it is reasonable to use DTW distance with SVM. However, due to its non-metric property, DTW distance does not yield a positive definite kernel.

Non-positive definite (indefinite) kernels are previously used in SVM [38, 169] using different techniques. Even though the kernel over DTW distance is indefinite, few kernels are defined over DTW distance [12, 54, 146, 183]. In [54], the authors corrected the negative definite kernel matrix by its square. There are also some attempts where minor changes are incorporated in the definition of DTW distance for defining a positive definite kernel [33]. Hayashi *et al.* [56] projected the time series into a Euclidean space such that the resulting representation approximates the DTW distance. Although we can define the kernels over DTW distance [12, 54, 146, 183] with some approximations/modifications in the original formulation, the resulting kernels are computationally expensive. This limits the use of kernels over DTW distance for large datasets. Since the computational complexity of computing the DTW distance between two time series of length n and m respectively is $O(nm)$, the resulting kernels over DTW distance have quadratic complexity. This is computationally prohibitive for large time series containing thousands of data points. Note that such large length time series commonly arises in many domains like speech, bioinformatics, and text processing. We propose a linear kernel over DTW distance in the next part of this work. The proposed kernel is defined using Gaussian functions over Fast Surrogate DTW distance.

Since the Gaussian is a non-linear function, the kernel defined over Fast Surrogate DTW distance is also a non-linear kernel. We compute its linear approximation using explicit feature maps [98, 125, 126, 166] and we refer to the proposed linear approximation kernel as Fast Surrogate DTW kernel. An explicit feature map is a popular technique for speeding up the non-linear kernels. It approximates the original large dimensional (infinite-dimensional) feature map of non-linear kernels by a small finite-dimensional feature map. This small dimensional feature map gives a linear approximation of the original non-linear kernel. These linear kernels can be used with linear SVM, which are computationally faster and can be computed in linear time compared to non-linear SVMs. Explicit feature maps are quite popular for different non-linear kernels like intersection [98], Hellinger's, χ^2 and RBF kernels [126, 166]. However, this technique is not widely used in the time series community. In this work, we follow this line of work and propose a novel approximate explicit feature map for the Fast Surrogate DTW kernel. We compute its explicit feature map using the global principal alignments. In [140], the authors proposed a dynamic time warping algorithm for the computation of DTW distance, which is linear in both time and space. The algorithm finds a nearly optimal warp path between two time series. However, it cannot be transformed into a valid kernel. Our proposed Fast Surrogate DTW kernel is computationally efficient compared to other kernels over DTW distance.

We evaluate the proposed Fast Surrogate DTW distance and Fast Surrogate DTW kernel over a wide variety of problems. For Fast Surrogate DTW distance, we demonstrate its utility for retrieving word images from the popular George Washington database and Indian language datasets. We show its superiority by comparing it with naive DTW distance, Euclidean distance, and metric DTW. The proposed Fast Surrogate DTW distance makes DTW based document image retrieval computationally feasible. The proposed technique is computationally efficient compared to DTW distance and metric DTW with a minor drop in retrieval performance. On multi-language datasets, we show a speed-up of more than $40\times$ compared to DTW distance and metric DTW. The performance of Fast Surrogate DTW distance is as good as DTW distance and computationally performs equally as simple Euclidean based matching. For Fast Surrogate DTW kernel, we compare our proposed kernel with the GA kernel [33] and Gaussian DTW kernel [12] over popular machine learning datasets. The Fast Surrogate DTW kernel is computationally efficient compared to GA kernel [33] and Gaussian DTW kernel [12] with a minor drop in classification performance.

The major contributions of this chapter are: (i) A novel framework for approximating the optimal alignment for given sequences using a set of global principal alignments computed from the training data, (ii) A Fast Surrogate DTW distance using global principal alignments, which is a linear approximation of the popular DTW distance, (iii) In addition, we also propose a Fast Surrogate DTW kernel using Fast Surrogate DTW distance, which is a linear kernel over DTW distance.

3.2 Related Works

Euclidean distance is an efficient similarity measure for computing the similarity between time series or sequences. However, the main limitation of Euclidean distance is that it cannot be applied if the sequences have variable lengths. Also, it cannot capture the local dependencies between neighboring states of the sequences. A successful method to compare time series or sequences is dynamic time warping (DTW) [12, 139, 146]. For a given two sequences, the DTW yields an optimal alignment from all possible alignments. This optimal alignment is being used to find the corresponding regions between the given sequences. It can also be used to find the similarity between the given sequences. The DTW measure is the difference between the two sequences after they have been mapped together with the optimal alignment.

Given two sequences, $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_m)$ of lengths n and m respectively, an alignment π of length $|\pi| = p$ is a pair of increasing p-tuples (π_1, π_2) such that

$$1 = \pi_1(1) \leq \ldots \leq \pi_1(p) = n,$$
$$1 = \pi_2(1) \leq \ldots \leq \pi_2(p) = m$$

with unitary increments and no simultaneous repetitions. That is, for $\forall 1 \leq i \leq p-1$,

$$\pi_1(i+1) \leq \pi_1(i) + 1, \;\; \pi_2(i+1) \leq \pi_2(i) + 1$$
$$(\pi_1(i+1) - \pi_1(i)) + (\pi_2(i+1) - \pi_2(i)) \geq 1.$$

Intuitively, an alignment π between X and Y describes a way to associate each element of X to one or possibly more elements in Y, and vice-versa. Let $A(X, Y)$ be the set of possible alignments between X and Y. The DTW distance between the sequences X and Y is given as the minimum distance over all possible alignments.

$$DTW(X, Y) = min_{\pi \in A(X,Y)} \sum_{i=1}^{|\pi|} \varphi(X_{\pi_1(i)}, Y_{\pi_2(i)}) \tag{3.1}$$

where, $\varphi(X_{\pi_1(i)}, Y_{\pi_2(i)})$ is the ground distance between the given sequences indexed by the alignment π. In general, Euclidean distance is used for computing the ground distance. The resulting alignment is the optimal path between the given two sequences. The DTW algorithm computes the optimal alignment as follows: For a given pair of sequences $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$, where $x_i, y_i \in \mathbb{R}^k$, of length n and m respectively, it first constructs a cost matrix C of dimension $n \times m$. The $(i, j)^{th}$ element of the cost matrix C is the Euclidean distance between the elements x_i and y_j. The X-axis and Y-axis of the matrix C represent time domain for the sequences X and Y. Each warp path in the cost matrix represents an alignment between the sequences X and Y. The sum of elements through the alignment gives the cost of the alignment. The DTW finds an optimal warp path that has the minimum cost.

For comparing two sequences X and Y of length n and m, respectively, the time and space complexity of DTW is $O(nm)$. The quadratic complexity mainly involves finding the optimal alignment from a large set of possible alignments. The quadratic complexity is particularly prohibitive for large length time series. The time series containing a few thousand measurements requires much memory and is computationally unattractive. In practice, heuristic constraints speed up DTW [65, 139], instead of searching the entire space for the optimal path, they restrict the search space into a constraint window. If constraints are used, the DTW algorithm finds the optimal path, which passes only through the constraint window. It speeds up the DTW computation by a constant factor, but the quadratic complexity remains. In addition, using these methods, the optimal path cannot be found if it does not pass through the constraint window. The constraints work well only if the optimal path is expected closer to the diagonal.

DTW distance is popularly used with nearest neighbor (NN) classifier. Since SVMs often outperforms NN classifiers [167], it is desirable to use DTW distance with SVMs. For using DTW in SVMs, we need to define a valid kernel over DTW distance. Several kernels have been proposed over DTW distance [12, 54, 146, 183]. For example, in [12], the authors proposed the Gaussian DTW (GDTW) kernel based on DTW distance and defined as,

$$\kappa_{gdtw} = exp\Big(\underset{\pi \in A(X,Y)}{-min} \frac{1}{|\pi|} \sum_{i=1}^{|\pi|} \|X_{\pi_1(i)} - Y_{\pi_2(i)}\|_2^2 \Big) \tag{3.2}$$

Similar to Gaussian kernel, in [146], the authors defined another kernel over DTW, which is given as

$$\kappa_{DTW_1} = \underset{\pi \in A(X,Y)}{max} \frac{1}{|\pi|} \sum_{i=1}^{|\pi|} e^{\frac{-1}{\sigma^2} \|X_{\pi_1(i)} - Y_{\pi_2(i)}\|_2^2} \tag{3.3}$$

In [35], Cuturi *et al.* proposed Group Alignment (GA) kernel. The kernel is not based on the optimal alignment but takes advantage of the scores obtained from all the possible alignments. The GA kernel is defined as,

$$\kappa_{GA}(X, Y) = \sum_{\pi \in A(X,Y)} \prod_{i=1}^{|\pi|} \kappa(X_{\pi_1(i)}, Y_{\pi_2(i)}) \tag{3.4}$$

where, $\kappa(X_i, Y_j) = e^{-\varphi(X_i, Y_j)}$. GA kernel becomes positive definite if $\frac{\kappa}{\kappa+1}$ is positive definite. Rather than considering the optimal alignment (simple minimum of the objective function), the kernel considers softmax of the scores of all possible alignments. Intuitively, the kernel considers the optimal alignment and all the alignments closer to it. In the paper, the authors argue that two sequences are similar if they have a single common alignment with a high score and share a wide set of other alignments. Since the GA kernel considers all the alignments or alignments near the diagonal, it performs well compared to other DTW kernels. Using the constraint windows [124] for DTW distance, Triangular Global Alignment (TGA) kernel is proposed in [33] for speeding up the GA kernel. This reduces the computational cost by a constant factor and the resulting path may be suboptimal. All the above kernels [33, 35, 146] are non-linear kernels and have quadratic complexity. Its quadratic complexity mainly involves finding the optimal alignment or near-optimal alignments.

In general, SVMs with non-linear kernels over large data sets are computationally slower compared to linear kernels. It requires efficient solvers to optimize the given problem. A linear SVM (SVM with linear kernel) is given by the inner product $F(X) = \langle w, X \rangle$ between the data sample X and a weight vector w. On the other hand, a non-linear SVM is given by the expansion $F(X) = \sum_{i=1}^{M} \alpha_i \kappa(X, X_i)$, where κ is the given non-linear kernel, X_is are support vectors and M is the number of support vectors. In most of the cases, evaluating the inner product $\langle w, X \rangle$ is more efficient than evaluating the kernel $\kappa(X, X_i)$. This makes the linear SVM atleast M times faster compared to non-linear SVM, which is a significant gain especially on large datasets. The training time is also effected in the similar way. A successful way to speed up the non-linear SVM is with the help of explicit feature maps [166]. For a given non-linear kernel κ, there exists a feature map ϕ such that $\kappa(X, Y) = \langle \phi(X), \phi(Y) \rangle$. However, in most of the cases, the feature map ϕ is of infinite dimension. In explicit feature map, it finds a finite dimension approximation $\phi^{'}$ of the feature map ϕ such that $\kappa(X, Y) \simeq \langle \phi^{'}(X), \phi^{'}(Y) \rangle$. Given a positive definite kernel $\kappa(X, Y)$ and a data density $p(\mathcal{X})$, where $X, Y \in \mathcal{X}^D$ for some input space $\mathcal{X}$, the approximation finds the feature map $\phi^{'}$, which minimizes the following functional

$$E(\phi^{'}) = \int_{\mathcal{X}^D \times \mathcal{X}^D} (\kappa(X, Y) - \langle \phi^{'}(X), \phi^{'}(Y) \rangle)^2 p(X) p(Y) dX dY \tag{3.5}$$

where, the components $\phi^{'}_k(X)$ $(k = 1, 2, \ldots)$ are eigenfunctions of the kernel κ. In general, the data density $p(\mathcal{X})$ is approximated by a finite sample set and correspondingly eigenfunctions are replaced with eigenvectors. The eigenfunctions (vectors) play an important role in the approximation of the kernels.

Metric DTW distance DTW distance computes all the linear and non-linear similarities between the given sequences; however, it is not a metric. There is a widely used metric DTW distance, which is used in kernel settings, especially with SVM. The metric DTW considers all the possible alignments compared to DTW distance, which uses only optimal alignment. Since it uses all the alignments to compute the distance, it performs better than DTW distance. However, it is computationally costly. The metric DTW is defined as follows

$$\kappa_{DTW} = exp\Big(- \sum_{\pi \in A(X,Y)} \frac{1}{|\rho|} \sum_{i=1}^{|\pi|} \| x_{\pi_1(i)} - y_{\pi_2(i)} \|_2^2 \Big) \tag{3.6}$$

where $A(X,Y)$ is the set of all alignments between X and Y.

3.3 Approximation of DTW distance

In general, DTW distance has quadratic complexity in the sequence length. This section presents a novel framework for computing the linear approximation to the DTW distance. We refer to this proposed linear approximation as Fast Surrogate DTW distance.

For a given data, there are similarities between the optimal alignments of different pairs of sequences. For example, if we take two different classes, the top alignments (least cost alignments) between the samples from these classes always have some similarities. We plot 3 top alignments between two sequences in Figure 3.1(a); these are the top 3 least-cost alignments. We show the similarity between the top alignments over a subset of samples from the Libra dataset in Figure 3.1(b). Here, we plot the top alignments between the samples from two classes. The figure shows that all the top alignments are passing through a small window. There is no single alignment that is completely different from other top alignments. Since the top alignments follow some structure, for a given new pair of sequences, we can approximate their optimal alignment using these alignments. Based on this idea, we compute a set of candidate top alignments from the training data and use these alignments to approximating the optimal alignment between new test sequences. This avoids the computation of optimal alignments. We call these candidate top alignments as global principal alignments. For the new test sequences, we use these global principal alignments for computing their DTW distance. Now, the DTW distance becomes the sum of the Euclidean distances over the global principal alignments. This gives a linear approximation of the DTW distance. These alignments capture all the similarities in the sequential data.

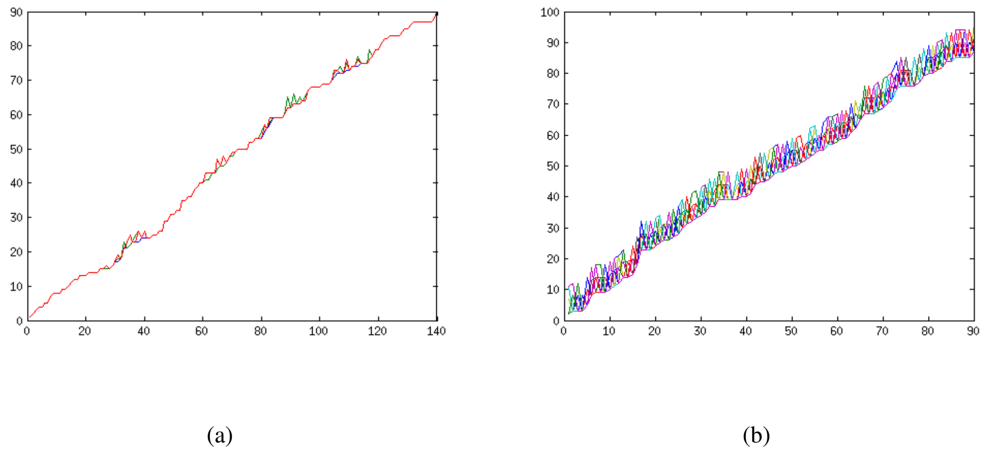


Figure 3.1: (a) The top 3 alignments between two time series X and Y of length 90 and 140 respectively. These alignments have least cost compared to other alignments. These alignments are not completely different from each other. (b) Optimal warp paths between the sequences of a subset of Libra dataset.

This work introduces two methods for computing the global principal alignments from the given data. In the first method, we compute the global principal alignments from the top alignments of the given data. For a given pair of sequences, their top alignments are computed from the cost matrix. Since there are similarities between the top alignments for different pairs of sequences, there exist similarities between their corresponding cost matrices. Instead of computing global principal alignments from the top alignments, we compute a representative cost matrix for the given data in the second method, and global principal alignments are computed from this cost matrix.

3.3.1 Approximation using Top Alignments (ATA)

We use principal component analysis (PCA) for modeling the global principal alignments for the given data. The objective of our work is to compute the global principal alignments using the set of all alignments such that the computed alignments should be well enough for approximating the DTW distance between any new pair of sequences. This is similar to the concept of PCA, where for a given data, it finds the principal directions in which the variance is maximum. PCA explores the dependencies between the variables in the given data. Using these dependencies, it converts the set of correlated variables into a set of uncorrelated variables called principal components (eigenvectors). These principal components capture the maximum variability in the given data. It better represents the given data. The global principal alignments in our work resemble the principal components in PCA. The alignments are better represented in 2D space. Since PCA works only on one-dimensional (1D) data, so we cannot directly apply it over top alignments to compute the global principal alignments. If we want to apply it over two-dimensional (2D) data, it must be transformed into 1D data. This creates a large dimensional covariance matrix, and computing the eigenvalues and eigenvectors for this matrix will be computationally expensive. Similar to PCA, a two-dimensional principal component analysis (2DPCA) [179] is proposed to overcome these issues. 2DPCA is based on 2D matrices rather than 1D vectors. In this work, we use 2DPCA for computing the global principal alignments. To compute the global principal alignments for a given data, we first represent each alignment using a 2D matrix and then global principal alignments are computed by applying 2DPCA over these matrices.

To apply 2DPCA, we need to represent each alignment using a 2D matrix. An alignment π between two time series of lengths n and m can be represented using an $n \times m$ grid. Equivalently, it can also be represented using an $n \times m$ binary matrix (alignment matrix), where the elements are either 0 or 1. This representation is given in Figure 3.2. The entries in the matrix through which the alignment passes are 1, and for other entries, it is 0. Since for every pair of sequences there exist many alignments, it corresponds to many binary matrices. For a given pair of sequences X and Y, denote their possible alignments as $\pi_1, \pi_2, \ldots, \pi_l$, where l is the total possible number of alignments. Assume that these alignments are arranged according to their cost, i.e., π_1 is the optimal alignment with the least cost and π_k is the top k^{th} alignment between X and Y. For the sequences X and Y, we define their alignment matrices as follows

$$B_{X,Y} = \cup_{k=1}^{l} B_{X,Y}^{k}$$

where, $B_{X,Y}^{k}$ is the alignment matrix correspond to the top k^{th} alignment for the time series X and Y, and l is the total number of possible alignments.

For a given dataset, we first construct these alignment matrices for every pair of sequences. Since there exist possibly many alignments for every pair of sequences and only top alignments have significance in approximating the DTW distance, we take only top t alignments for computing the global principal alignments. For a given dataset $\mathcal{D}$, we construct its alignment matrices as follows,

$$P_t = \cup_{X,Y \in \mathcal{D}} (\cup_{j=1}^{t} B_{X,Y}^{j})$$

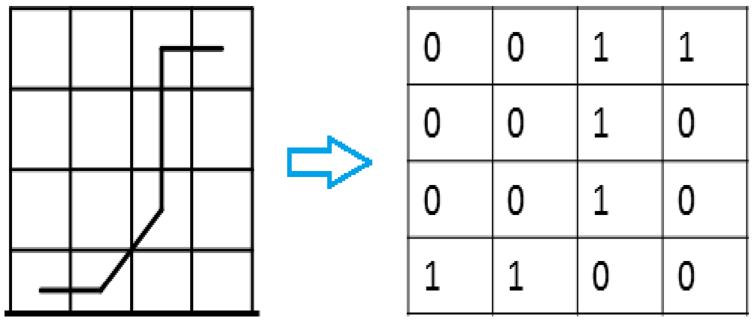


Figure 3.2: Alignment matrix representation. The entries in the matrix where the alignment passes is 1, and for others entries it is 0.

It contains only top t alignment matrices between every pair of sequences. The set P_t contains the best possible alignment matrices for the given dataset. We compute the global principal alignments from this set of alignment matrices using 2DPCA. If the dataset contains alignments of variable length, the set P_t contains matrices of variable dimension. In this case, we cannot apply the above procedure for computing the global principal alignments. To overcome this, we first scale the alignments to a fixed size. Then, the alignment matrices are computed from these scaled alignments. Now, the resulting alignment matrices have the same dimension. Finally, we apply 2DPCA over these matrices and find the eigenvectors. These eigenvectors give the global principal alignments for the given dataset.

3.3.2 Approximation using Cost Matrix (ACM)

For computing the global principal alignments in ATA, we need to compute the top alignments for every pair of samples. It is computationally unattractive for large datasets. To avoid this, we propose another technique that computes a global cost matrix from the given data, from which we derive the global principal alignments.

We construct the global cost matrix for the given data as follows. We first compute the cost matrix for every pair of sequences from the given data and normalize each cost matrix by the maximum cost of that matrix. The mean of all these cost matrices is taken as the global cost matrix for the given data. If the dataset contains alignments of variable length, the cost matrices will have variable dimensions. In this case, we cannot compute the global cost matrix as discussed above. To overcome this, we first scale the alignments to a fixed size and then cost matrices are computed from these scaled alignments. Now, the resulting cost matrices will have the same dimension. The top alignments computed from the global cost matrix give the global principal alignments for the given data. These alignments are sufficient enough for comparing any two given sequences.

Since we avoid learning the global principal alignments from a large set of top alignments, the approximation technique ACM is computationally faster than the approximation ATA. However, as we are computing the global principal alignments from the single global cost matrix, it may not capture all the correlations in the given data. Due to this, the approximation ACM performs slightly inferior compared to the ATA in terms of accuracy.

Let $G_{\mathcal{D}}$ be the set of all global principal alignments computed from the given dataset $\mathcal{D}$. For the sequences X and Y, the Fast Surrogate DTW distance over $G_{\mathcal{D}}$ is defined as

$$\begin{aligned} Fast_{DTW}(X,Y) &= \sum_{\pi \in G_{\mathcal{D}}} \sum_{k=1}^{|\pi|} (X_{\pi(k)} - Y_{\pi(k)})^2 \\ &= \sum_{\pi \in G_{\mathcal{D}}} Euclid_{\pi}(X,Y) \end{aligned} \tag{3.7}$$

where $|\pi|$ is the length of the alignment π. $Euclid_{\pi}(X,Y)$ is the Euclidean distance between X and Y over the alignment π. Notice that, the DTW distance between two samples is the Euclidean distance (ground distance) over the optimal alignment.

Computational time: From the Eq 3.7, we can observe that Fast Surrogate DTW distance is the sum of the Euclidean distances over the global principal alignments. It means Fast Surrogate DTW distance can be computed in a linear time compared to the quadratic complexity of naive DTW distance. In summary, the performance of the proposed method is as good as DTW distance and computationally is on par with simple Euclidean distance.

3.4 Dynamic Time Warping Kernels

This section presents our proposed Fast Surrogate DTW kernel, which is a linear approximation of the non-linear DTW kernel. Like Fast Surrogate DTW distance, the Fast Surrogate DTW kernel is also defined over the global principal alignments. Let $G_{\mathcal{D}}$ be the set of all global principal alignments computed from the given dataset $\mathcal{D}$. For the sequences X and Y, the Fast Surrogate DTW kernel over $G_{\mathcal{D}}$ is defined as

$$\kappa_{Fastdtw}(X,Y) = \sum_{\pi \in G_{\mathcal{D}}} e^{-\beta \sum_{k=1}^{|\pi|} (X_{\pi(k)} - Y_{\pi(k)})^2} \tag{3.8}$$

The Fast Surrogate DTW kernel defined in Eq 3.8 is the sum of RBF kernels over the global principal alignments. Since it is the sum of RBF kernels, the resulting kernel is non-linear. We compute its linear approximation using explicit feature maps. From the work on explicit feature map for RBF kernel [166], the explicit feature map of dimension n for RBF kernel is given as follows

$$\phi_{RBF}(X) = \frac{1}{\sqrt{n}}[e^{-i\langle \omega_1, X \rangle}, \ldots, e^{-i\langle \omega_n, X \rangle}] \tag{3.9}$$

where $\omega_1, \ldots, \omega_n$ are sampled from the Gaussian density. Since the Fast Surrogate DTW kernel defined in Eq 3.8 is the sum of RBF kernels over the global principal alignments, its explicit feature map is given as follows,

$$\begin{aligned} \phi_{FastDTW}(X) = \frac{1}{\sqrt{n}}[&e^{-i\langle \omega_1, X_{\pi_1} \rangle}, \ldots, e^{-i\langle \omega_n, X_{\pi_1} \rangle}, \ldots, \\ &e^{-i\langle \omega_1, X_{\pi_m} \rangle}, \ldots, e^{-i\langle \omega_n, X_{\pi_m} \rangle}] \end{aligned} \tag{3.10}$$

where $\pi_1, \ldots, \pi_m$ are the global principal alignments and m is the total number of global principal alignments. The linear approximation of the Fast Surrogate DTW kernel is given as

$$\kappa^{lin}_{FastDTW}(X, Y) = \langle \phi_{FastDTW}(X), \phi_{FastDTW}(Y) \rangle \tag{3.11}$$

3.5 Experiments

This portion evaluates various components of the proposed Fast Surrogate DTW distance and Fast Surrogate DTW kernel. We demonstrate the utility of the Fast Surrogate DTW distance for document word image retrieval problem. For Fast Surrogate DTW kernel, we evaluate using SVM classifier over popular machine learning datasets.

3.5.1 Efficient Word Image Retrieval using Fast Surrogate DTW Distance

3.5.1.1 Datasets and Evaluation Protocols

In this sub-section, we discuss the datasets and the experimental settings we follow in our experiments. We show the results on the popular George Washington (GW) database. In the experimental section, to demonstrate the utility of the proposed method across different languages, we also exemplify the results on various Indian language datasets. These datasets contain two other Indian languages

Dataset	# Classes	# Images
D1	125	16145
D2	268	30164
D3	100	14306
George Washington Database (GW)	1471	4894

Table 3.1: Details of the datasets considered in the experiments.

(Hindi (D1) and Telugu (D2)), and English (D3) with a significant change in structure. One of the Indian languages have a headline and the other does not. One of them is an Aryan language, while the other is Dravidian. Details of the dataset are given in Table 3.4. For the datasets D1, D2 and D3, the ground truth is created using the methodology discussed in [69].

Multiple query images are generated to evaluate the quantitative performance. The criteria for the selection of query images are (i) query images need to occur multiple times in the database, (ii) are mostly functional words (iii) also should have no stop words. The performance is measured by mean Average Precision (mAP). The mAP is the mean of the area under the precision-recall curve for all the queries. For every pair of sequences, we take $t = 10$, i.e., we choose the top 10 alignments. We take the number of global principal alignments for all the datasets based on their size. The number of global principal alignments for each of the datasets is given in Table 3.5. Since we compute the global principal alignments using 2D-PCA, we need fixed-size alignment matrices. Since the size of the images is not fixed in our datasets, we scale each word image into a fixed size. All experiments were carried out on a single core of a 2.1 GHz AMD 6172 processor with 12 GB RAM.

3.5.1.2 Feature Extraction

In this work, we use the split profile features [131] for representing the word images. We first divide the image horizontally into two parts for computing these features. From these images, we extract the following features, (i) vertical profile i.e., the number of ink pixels in each column, (ii) location of lowermost ink pixel, (ii) location of uppermost ink pixel, and (iv) number of ink to background transitions. The profile features are calculated on binarized word images obtained using the Otsu thresholding algorithm.

	D1	D2	D3	GW
# Samples	16145	30164	14306	4894
# Global Alignments	60	100	60	40

Table 3.2: Number of global principal alignments for the datasets used in the experiments. Here, the number of global alignments are based on the size of the dataset.

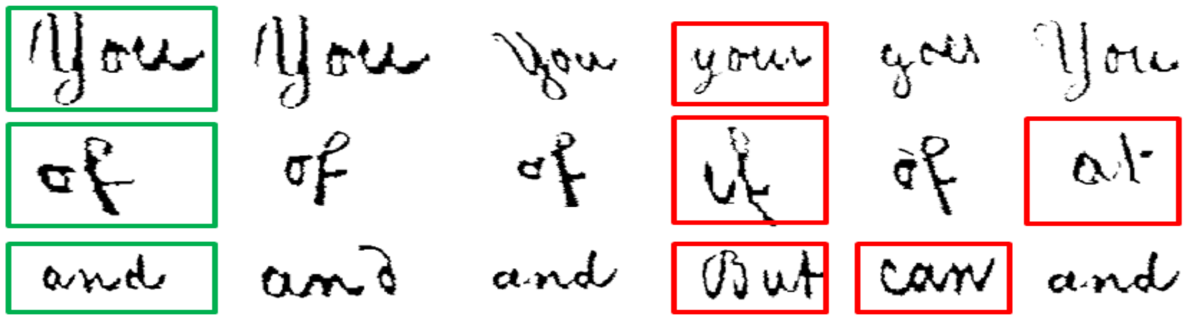


Figure 3.3: Few sample results. Top-5 retrieval results for given query images. First column shows the query image. For each query, its top 5 retrieval images are shown from left to right. In each row, query image is marked in green colour and its corresponding wrong retrieval images are marked in red colour.

	DTW distance	Proposed	Metric DTW	Euclidean
D1	0.9038	0.8927	0.9127	0.8059
D2	0.8382	0.8204	0.8438	0.7123
D3	0.8193	0.8072	0.8319	0.7329
GW	0.5173	0.5019	0.5361	0.3271

Table 3.3: Performance of the proposed technique as compared to the DTW distance, metric DTW and Euclidean distance. Here, the mAP score is compared for all the methods.

In the first experiment, we evaluate the performance of the proposed Fast Surrogate DTW distance by comparing it with naive DTW distance, Euclidean distance, and metric DTW. The comparative results over the given datasets are given in Table 3.3. We compare the mAP score over all the datasets. The results show that the Fast Surrogate DTW distance is comparable with naive DTW distance over all the datasets. It performs significantly better compared to Euclidean distance. Metric DTW performs well compared to other methods over all the datasets as it considers all the alignments to compute the distance. Sometimes, only optimal alignment may not be sufficient for computing the similarity. In that case, considering all the alignments or a few top alignments enhances the performance. The minor drop in performance of Fast Surrogate DTW distance compared to DTW distance is due to the scaling involved in resizing the images and the approximation of optimal alignments using global principal alignment

To explore the speed up of the proposed Fast Surrogate DTW distance, we compare its retrieval time with naive DTW distance, Euclidean distance, and metric DTW over all the datasets. The results are given in Figure 3.4. Here, the retrieval time is shown over a log scale. Fast surrogate DTW distance is significantly faster than DTW distance and metric DTW and is comparable to Euclidean distance. This is mainly due to the length of the final feature representations obtained from the global principal alignments. We project the feature vectors into more than one principal direction in the Fast Surrogate DTW distance. This results in larger dimensional feature representations compared to the original representation. The speed-up of our proposed technique mainly comes from the global principal alignments, which avoids the computation of optimal alignments. In DTW distance, for a given query, we need to compute the optimal alignments for all the images in the database to retrieve its similar images. In the Fast Surrogate DTW distance, we use precomputed global principal alignments for computing the distance. Due to

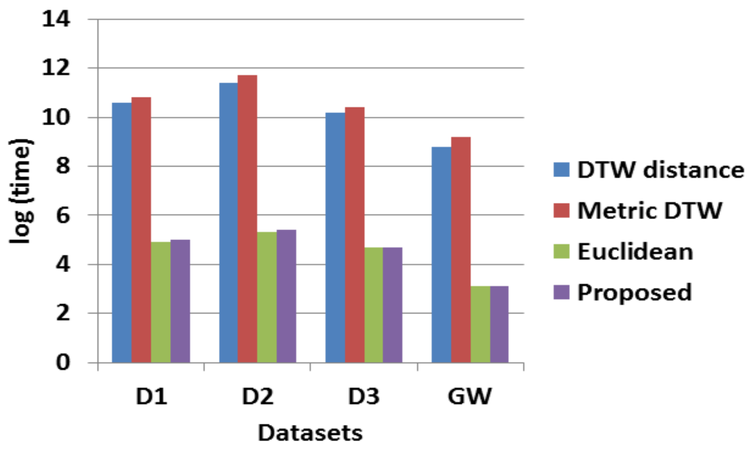


Figure 3.4: Retrieval time for a given query image for all the four methods over the given datasets. Here, the retrieval time is shown over log scale.

this, the Fast Surrogate DTW is computationally efficient compared to other methods. Note that being computationally attractive, our method achieves comparable performance to DTW based retrieval.

To evaluate the applicability of Fast Surrogate DTW distance across various languages, we show experimental results on printed Indian language datasets, namely D1 and D2. In general, these languages need rich features for better representation. However, in this experiment, we use only simple profile features and do not apply any learning techniques to obtain better representation. The results are given in Table 3.3. From the results, we can observe that, the Fast Surrogate DTW distance for the two Indian language datasets is comparable with the DTW distance. The proposed approach is language-independent as it performs equally well in other Indian languages.

3.5.1.3 Qualitative Results

We show the qualitative performance of Fast Surrogate DTW distance on the George Washington database in Figure 3.3. Here, we show some of the example queries and their top-5 retrieved word images. We have marked the query image in green and incorrect retrieval images in red. Note that the current work does not use any learning feature models or any other post-processing techniques, and the main aim of this work is to show the scalability of DTW-based retrieval methods.

Database	dimension	length	classes	# samples
Libra	2	45	15	945
Auslan	22	45-136	95	2465
JV	12	7-29	9	640
HC	3	60-182	20	2858
PEMS	963	144	7	440

Table 3.4: Details of the datasets considered in the experiments Libra, Auslan, Japanese Vowels (JV) and Handwritten Characters (HC). Here, Libra dataset contains fixed length time series, whereas all other datasets describes multivariate time series.

3.5.2 Evaluation of Fast Surrogate DTW kernel

3.5.2.1 Datasets and Evaluation Protocols

This evaluation aims to demonstrate the efficiency of the proposed Fast Surrogate DTW kernel on a wide range of time-series datasets. We evaluate our proposed kernel over popular machine learning datasets Libra, Auslan, Japanese vowels, handwritten Characters, and PEMS database of freeway traffic. Except for Libra dataset, all the other datasets contain multivariate time series of variable length. In addition to these small datasets, we also evaluate our Fast Surrogate DTW kernel on large-scale time-series datasets obtained from UCR Time Series Data Mining Archive. Since the proposed method is only applicable for fixed-length sequences, we scale the variable-length sequences to a fixed length using standard scaling techniques. We compare the proposed Fast Surrogate DTW kernel with GDTW [12] kernel and GA kernel [33] for all these datasets. The details of these datasets are given in Table 3.4.

In our proposed approximation kernel, for every pair of time series, we take $t = 10$, i.e., we choose the top 10 alignments. In both the proposed approximation techniques ATA and ACM, we consider an equal number of global principal alignments, and it is based on the size of the dataset. The number of global principal alignments for the given datasets is given in Table 3.5. For both the techniques ATA and ACM the kernel computational time is the same. In all our experiments, for comparing computational time with other kernels, we refer to our proposed kernel as the Fast Surrogate DTW kernel. All the kernels are implemented in Matlab. All experiments are carried out on a single core of a 2.1 GHz AMD

	Libra	Auslan	PEMS	HC	JV
# Samples	945	2465	440	2858	640
# Global Principal Alignments	14	24	8	24	12

Table 3.5: Number of global principal alignments for the datasets used in the experiments. Here, the number of global principal alignments are based on size of the dataset.

	Libra	Auslan	PEMS	HC	JV
ATA	12.2±0.29	43.1±2.3	25.2±0.8	45.2±2.8	17.1±0.3
ACM	**7.7±0.21**	**28.9±1.2**	**19.5±0.3**	**36.3±1.8**	**12.3±0.2**

Table 3.6: Training time (Hrs) for ATA and ACM over the given datasets. Only top 1 alignment is considered in this experiment.

6172 processor with 12 GB RAM. For comparison, the runtime is measured using the system clock with minimal background processes running.

3.5.2.2 Accuracy of Fast Surrogate DTW Kernel

We compare the accuracy of the Fast Surrogate DTW kernel with GA kernel and GDTW kernel over the given datasets in Figure 3.5. For the Fast Surrogate DTW kernel, we present the results obtained from both the approximation techniques ATA (Method-1) and ACM (Method-2). We present the results for both the equal length time-series data (Libra) and variable-length time series data (Auslan, PEMS, HC, JV). Our proposed methods achieve better accuracy on the Libra dataset than the GDTW kernel and perform equally well compared to the GA kernel. Our results are superior to the GDTW kernel on other datasets and comparable to the GA kernel. This shows that our proposed kernel performs equally well, if not better than GA kernel over equal length time series. The drop in the performance for variable-length time series is due to the scaling step we are performing for converting the data to equal length. In the scaling step, we lose some information from the data, which causes the degraded performance. However, our proposed kernel performs well compared to GDTW kernel, which is exponential of DTW distance over all the datasets.

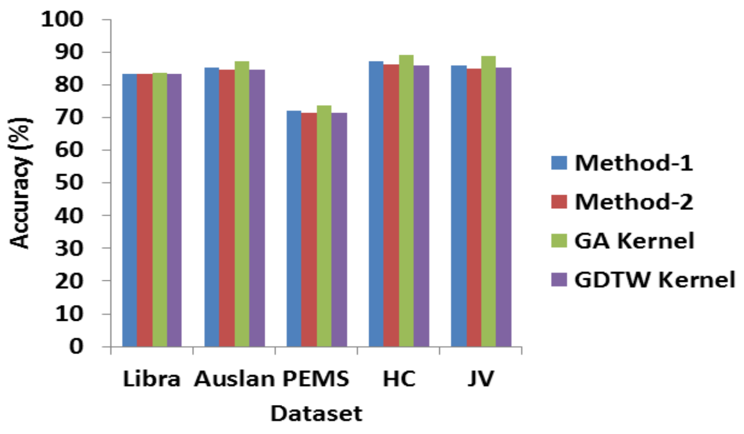


Figure 3.5: Comparison of the proposed approximation methods (Method- 1 and Method- 2) with the GA kernel and GDTW kernel over Libra, Auslan, PEMS, HC (Hand written Characters) and JV (Japanese Vowels) datasets.

3.5.2.3 Computation Time

This section explores the efficiency of the Fast Surrogate DTW kernel over small and large datasets. We compare Fast Surrogate DTW kernel with GA kernel [33] and GDTW kernel [12] in all the experiments. The first experiment compares the training time for the approximations, ATA, and ACM over the given datasets. The results are given in Table 3.6. It shows that the approximation ACM is computationally faster compared to ATA over all the datasets. In ATA, we must compute the top alignments for every pair of sequences. As a result, ATA is computationally slower than ACM in training. We compare the computational time of the proposed Fast Surrogate DTW kernel with the GA kernel and GDTW kernel over all the datasets in Figure 3.6. The results show that the Fast Surrogate DTW kernel is computationally faster than the GA kernel and GDTW kernel over all the datasets. For the Libra dataset, the smallest dataset in our experiments, both the Fast Surrogate DTW kernel and GA kernel are performing equally well. However, for other datasets, which have more sequences than to Libra, Fast Surrogate DTW kernel has a significant gain in performance compared to GA kernel. Also, the proposed Fast Surrogate DTW kernel outperforms GDTW kernel over all the datasets. On the handwritten characters dataset, the Fast Surrogate DTW kernel is nearly ten times faster than GDTW kernel. The speed-up of our proposed kernel mainly comes from the explicit feature maps, which give the linear approximation for the Fast Surrogate DTW kernel.

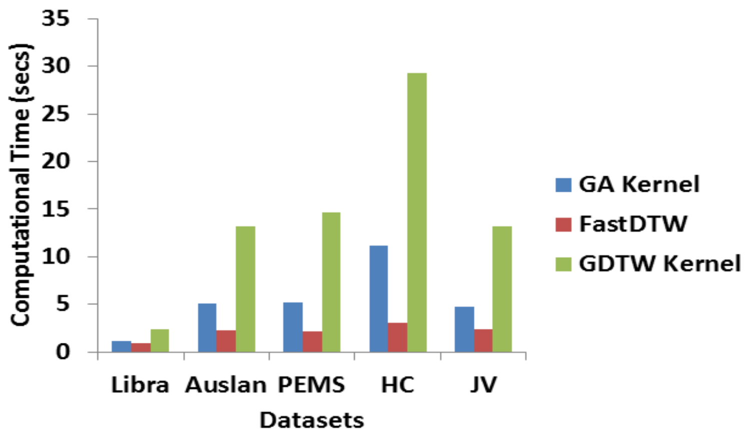


Figure 3.6: Comparison of computational time for proposed Fast Surrogate DTW kernel with GA kernel and GDTW kernel over the given datasets. Here, computational time is given seconds.

	Libra		Auslan		PEMS		HC		JV	
	Unimodal	Bimodal	Unimodal	Bimodal	Unimodal	Bimodal	Unimodal	Bimodal	Unimodal	Bimodal
ATA	83.5±0.9	**83.6±0.8**	85.3±1.0	**86.9±1.2**	72.1±0.4	**72.5±0.5**	87.1±1.3	**87.4±1.5**	86.1±0.7	**87.1±0.9**
ACM	83.5±0.8	**83.6±0.7**	84.5±0.9	**86.2±1.1**	71.4±0.3	**72.1±0.5**	86.2±1.0	**87±0.9**	84.9±0.6	**86.9±0.9**

Table 3.7: Comparison between Unimodal and Bimodal distributions over Libra, Auslan, PEMS, HC (Handwritten Characters) and JV (Japanese Vowels) datasets. GA kernel does not use Bimodal distribution.

We test the proposed kernel over the UCR time series data mining archive to further explore the Fast Surrogate DTW kernel over large length sequences. This dataset contains time series of lengths from 1000 to 0.1 Million. In Figure 3.7, we compare the computational time of the Fast Surrogate DTW kernel with the GA kernel over a varying length of time series. For better comparison, we show the computational time on a log scale. Both Fast Surrogate DTW kernel and GA kernel are performing equally well for small-length sequences. The Fast Surrogate DTW kernel's performance is least affected by the increased length of sequences. On the other hand, for the GA kernel, the computational time increases significantly with the increase in the length of sequences. For time series of length 0.1 Million, the Fast Surrogate DTW kernel can be computed in around one minute, whereas for GA kernel, it took nearly one hour. This suggests that the Fast Surrogate DTW kernel is more computationally efficient than the GA kernel and is suitable for large datasets and large time series.

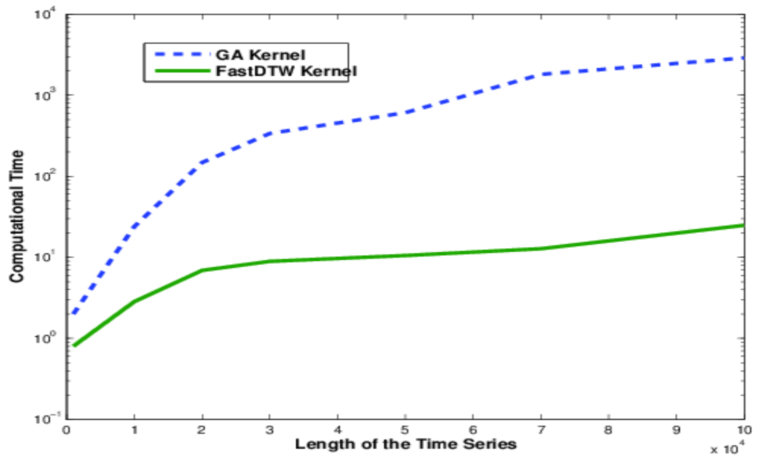


Figure 3.7: Comparison of Fast Surrogate DTW kernel with GA kernel over a varying length of time series. Here, the computational time is given in minutes and is shown on log scale.

3.5.2.4 Ablation Studies

The alignments may not always follow the unimodal distribution. PCA works well if the data follows the unimodal distribution. If the data does not follow unimodal distribution, then the principal components may not capture the maximal variance in the data. In such cases, if we directly apply PCA over the alignments, we may not get the best candidate global principal alignments. The results of these studies are given in Table 3.7. We present the results for both unimodal and bimodal distribution. We apply the PCA over each distribution separately in bimodal distribution, and global principal alignments are computed from these distributions. From the results, we can observe that global principal alignments computed from the bimodal distribution perform better compared to unimodal distribution. In unimodal, the resulting global principal alignments cannot capture all the correlations in the given data. This suggests that we get better performance if the global principal alignments are computed based on the given data distribution. The global principal alignments computed from the unimodal and bimodal distributions are performing equally well for the Libra dataset. In this case, the global principal alignments computed from the unimodal distribution are sufficient enough for encoding the correlations in the data.

Role of the number of Global Principal Alignments Since we select the number of global principal alignments based on the size of the dataset, we show the effect of the number of global principal alignments on the accuracy in Table 3.8. To show the role of the number of global principal alignments,

	Libras		Auslan		PEMS		HC		JV	
	Accuracy	Test time	Accuracy	Test time	Accuracy	Test time	Accuracy	Test time	Accuracy	Test time
Top 1 Alignment	82.2	**0.9**	83.8	**2.3**	71.0	**2.2**	86.2	**3.1**	84.9	**2.4**
Top 3 Alignments	82.5	**0.9**	84.1	**2.3**	71.5	**2.2**	86.1	**3.1**	85.5	**2.4**
Top 5 Alignments	83.1	**0.9**	84.4	**2.3**	71.8	**2.2**	86.3	**3.1**	85.6	**2.4**
Top 8 Alignments	83.4	**0.9**	85.0	**2.3**	72.1	**2.2**	86.7	**3.1**	85.8	**2.4**
Top 15 Alignments	83.5	**0.9**	85.1	**2.3**	72.1	**2.2**	87.0	**3.1**	86.1	**2.4**
GDTW Kernel	83.4	2.4	84.6	13.2	71.6	14.7	86.1	29.3	85.4	13.2
GA Kernel	**83.6**	1.1	**87.3**	5.1	**73.8**	5.2	**89.3**	11.2	**88.7**	4.8

Table 3.8: Performance of the Fast Surrogate DTW kernel over varying number of global principal alignments. Here, the performance is measured in-terms of accuracy (%) and test time (seconds). Fast Surrogate DTW kernel is compared with GA kernel and GDTW kernel.

we compare Fast Surrogate DTW kernel with the GA kernel and GDTW kernel over a varying number of global principal alignments. From the results, we can observe that the accuracy improves as we increase the number of alignments. This suggests that we need to consider more global principal alignments to get better results. The results also show that if we further increase the number of alignments, there is no improvement in accuracy.

3.6 Summary

In this work, we proposed a linear surrogate approximation to the non-linear DTW distance. The main aim of this work is to explore the hidden structure of the alignments for approximating the non-linear DTW kernel by a linear kernel. In addition, we also propose an explicit feature map for the proposed approximation kernel. For computing the explicit feature map, we exploit the hidden internal structure of the alignments. The optimal alignment between the given time series is represented using global top alignments. The proposed representation reduces the computational cost of DTW kernel by a huge factor.

Chapter 4

Efficient Query Specific DTW Distance for Document Retrieval

In this chapter, we present a novel application built using the Fast Surrogate DTW distance proposed in Chapter 3. These applications widen the scope of our work and also validates the robustness of the proposed Fast Surrogate DTW distance. In this chapter, we address the problem of faster indexing in classifier-based retrieval methods using Fast Surrogate DTW distance. We introduce the Query specific Fast Surrogate DTW distance for faster indexing, which has linear time complexity. We validate the proposed Query specific Fast Surrogate DTW on Direct Query Classifier (DQC) [128] for faster indexing in document image datasets. In addition to this, we also present an application built using the Fast Surrogate DTW distance in the deep learning framework. In this chapter, we propose, FastDTWNet, where the Fast Surrogate DTW distance is applied as a feature extractor in CNN.

4.1 Document Retrieval

Retrieving relevant documents (pages, paragraphs, or words) is a critical component in information retrieval solutions associated with digital libraries. The problem has been looked at in two settings: recognition based [110, 147] like OCR and recognition free [131, 177]. Most present day digital libraries use optical character recognizers (OCR) to recognize digitized documents and thereafter employ a text-based solution for information retrieval. Though OCRs have become the de facto preprocessing for the retrieval, they are realized as insufficient for degraded books [160], incompatible for older print styles [47], unavailable for specialized scripts [74] and very hard for handwritten documents [72]. Commercial OCRs may provide highly unacceptable results in practice even for printed books. The best commercial OCRs can only give word accuracy of 90% on printed books [177] in modern digital libraries. This means that every 10th word in a book is not searchable. Recall of retrieval systems built on such erroneous text is thus limited. Recognition-free approaches have gained interest in recent years. Word spotting [131] is a promising method for recognition free retrieval. In this method, word images are represented using different features (eg. Profile, SIFT-BOW), and the features are compared with the help of appropriate distance measures (Euclidean, Earth Movers [135], DTW [117]). Word spotting has the advantage that it does not require prior learning due to its appearance-based matching. These techniques have been popularly used in document image retrieval.

Konidaris *et al.* [160] retrieve words from an extensive collection of printed historical documents. A search keyword typed by the user is converted into a synthetic word image which is used as a query image. Word matching is based on computing the L_1 distance metric between the query feature and all the features in the database. Here the features are calculated using the density of the character pixels and the area that is formed from the projections of the upper and lower profile of the word. The ranked results are further improved by relevance feedback. Sankar and Jawahar [74] have suggested a framework of probabilistic reverse annotation for annotating a large collection of images. Word images were segmented from 500 Telugu books. The word images are matched using the DTW approach [130]. Hierarchical agglomerative clustering was used to cluster the word images. Exemplars for the keywords are generated by rendering the word to form a keyword image. Annotation involved identifying the closest word cluster to each keyword cluster. This involves estimating the probability that each cluster belongs to the keyword. Yalniz and Manmatha [177] have applied word spotting to scanned English and Telugu books. They can handle noise in the document text using SIFT features extracted on salient corner points. Rath and Manmatha [130] used projection profile and word profile features in a DTW based matching technique.

Recognition-free retrieval was attempted in the past for printed and handwritten document collections [63, 74, 99, 177]. Since most of these methods were designed for smaller collections (few handwritten documents as in [99]), computational time was not a major concern. Methods that extended this to a larger collection [6, 13, 162] used primarily (approximate) nearest neighbor retrieval. For searching complex objects in large databases, SVMs have emerged as the most popular and accurate solution in the recent past [99]. For linear SVMs, training and testing have become very fast with the introduction of efficient algorithms and excellent implementations [145]. However, there are two fundamental challenges in using a classifier based solution for word retrieval (i) A classifier needs a good amount of annotated training data (both positive and negative) for training. Obtaining annotated data for every word in every style is practically impossible. (ii) One could train a set of classifiers for a given set of frequent queries. However, they are not applicable for rare queries.

In [128], Ranjan *et al.* proposed a one-shot classifier learning scheme (Direct query classifier). The proposed one-shot learning scheme enables the direct design of a classifier for novel queries without having access to the annotated training data, i.e., classifiers are trained for a set of frequent queries and seamlessly extended for the rare and arbitrary queries, as and when required. The authors hypothesize that word images, even if degraded, can be matched and retrieved effectively with a classifier based solution. A properly trained classifier can yield an accurate ranked list of words since the classifier looks at the word as a whole and uses a larger context (say multiple examples) for matching. The results for this method are significant since (i) It does not use any language specific post-processing for improving the accuracy. (ii) Even for a language like English, where OCRs are relatively advanced and engineering solutions were perfected, the classifier based solution is as good, if not superior to the best available commercial OCRs .

In the direct query classifier (DQC) scheme [128], the authors used DTW distance for indexing the frequent mean vectors. Since the DTW distance is computationally slow, the authors do not use all the frequent mean vectors for indexing. For comparing two word images, DTW distance typically takes one second [131]. This limits the efficiency of DQC. To overcome this limitation, the authors used Euclidean distance for indexing. The authors use the top 10 (closest in terms of Euclidean distance) frequent mean vectors for indexing. Since the DTW distance better captures the similarities compared to Euclidean distance for word image retrieval, this restricts the performance of DQC.

The Fast Surrogate DTW [109] distance can be used for efficient indexing in DQC classifier. However, it gives sub-optimal results. For best results, it needs query specific global principal alignments. In this work, we introduce query specific DTW distance, which enables the direct design of global principal alignments for novel queries. Global principal alignments are computed for a set of frequent classes and seamlessly extended for the rare and arbitrary queries, as and when required, without using language specific knowledge. This is a distinct advantage over an OCR engine, which is challenging to adapt to varied fonts and noisy images and requires language-specific knowledge to generate possible hypotheses for out-of-vocabulary words. Moreover, an OCR engine can respond to a word image query only by first converting it into text, which is prone to recognition errors. In [85, 155], deep learning frameworks are used for word spotting. In [5], an attribute based learning model PHOC is presented for word spotting. In the training phase, each word image is to be given with its transcription. Both word image feature vectors and its transcriptions are used to create the PHOC representation. An SVM is learned for each attribute in this representation. Our approach bears similarities with the PHOC representation based word spotting [5]. In the sense that both the approaches are designed for handling out-of-vocabulary queries. Our work takes advantage of granular description at ngrams (cut-portion) level. This somewhat resembles the arrangement of characters used in the PHOC encoding. However, training efforts for PHOC are substantial, with a large number of classifiers (604 classifiers) being trained and require complete data for training, which is huge for large datasets. In our work, the amount of training data is restricted to only frequent classes, which is very less compared to PHOC. Further, PHOC requires labels in the form of transcriptions, whereas in our work, the labels need not be transcriptions. Also, PHOC is language-dependent [50], and challenging to apply in different languages. The method proposed in this work is language independent, it can be applied to any language.

4.2 Direct Query Classifier (DQC)

In [128], Ranjan *et al.* proposed Direct Query Classifier (DQC), a one-shot learning scheme for dynamically synthesizing classifiers for novel queries. The main idea is to compute an SVM classifier for the query class using the classifiers obtained from the frequent classes of the database. The number of possible words in a language could be huge and it would be difficult to build a classifier for each word. However, all these words come from a small set of *ngrams*. The words corresponding to the frequent queries are expected to contain the n-grams that cover the full vocabulary. Exemplar SVM classifiers are computed for the frequent queries (word classes) and then appropriately concatenated to create novel classifiers for the rare queries. However, this process has its challenges due to

(i) Variations due to nature of script and writing style,

(ii) Classifiers for smaller ngrams could be noisy.

The authors address these limitations by building the SVM classifiers for most frequent queries and using classifier synthesis only for rare queries. This improves its overall performance. They use Query Expansion (QE) for further improving the performance. An overview of the direct query classifier is given in the following sections.

4.2.1 DP DQC: Design of DQC using Dynamic Programming

Given a set of classifiers for frequent classes $\mathcal{W}_w = \{w_1, w_2, \ldots, w_N\}$ and a query vector X_q, the query classifier w_q is designed as a piecewise fusion of parts (n grams) from the available classifiers from $\mathcal{W}_w$. Let p be the number of portions to be selected for computing the query classifier w_q. These portions are characterized by the sequence of indices $a_1, \ldots, a_{p+1}$. The classifier synthesis problem is formulated as that of picking up the optimal set of classifiers $\{c_i\}$ and the set of segment indices $\{a_i\}$ such that $\{a_i\}$ form a monotonically increasing sequence of indices. This involves the following optimization:

$$\underset{\{a_i\},\{c_i\}}{max} \sum_{i=1}^{p} \sum_{k=a_i}^{a_{i+1}} w_{c_i}^k X_q^k \tag{4.1}$$

where w_{c_i} corresponds to the weight vector of the c_i^{th} classifier that we choose and the inner summation applies the index k in the range (a_i, a_{i+1}) to use the k^{th} component $w_{c_i}^k$ from the classifier c_i. The index i in the outer summation refers to the cut portions, and p is the total number of portions we need to consider.

In [99], Malisiewicz *et al.* proposed the idea of exemplar SVM (ESVM), where a separate SVM is learnt for each example. Almazan *et al.* [67] use ESVMs for retrieving word images. ESVMs are inherently highly tuned to their corresponding example. Given a query, it can retrieve highly similar word images. This constrains the recall unless one has significant variations of the query word. Another demerit of ESVM is the large overall training time since a separate SVM needs to be trained for each exemplar. One way to reduce training time is to make the negative example mining step offline and select a common set of negative examples [100]. Gharbi *et al.* [104] provide another alternative for fast training of exemplar SVM in which the hyperplane between a single positive point and a set of negative points can be seen as finding the tangent to the manifold of images at the positive point.

Given a query q, the similar vectors in the dataset are identified by adopting the ESVM formulation proposed by Gharbi et al. [104], which yields an approach equivalent to Linear Discriminant Analysis. It involves a fast computation of the weight vector by adopting a parametric representation of the negative examples approximated as a Gaussian modeled on the complete set of training points. The normal to the Gaussian at the query point q is computed using the covariance matrix to yield the weight vector w_q as follows:

$$w_q = \Sigma^{-1}(\mu_q - \mu_0) \tag{4.2}$$

where Σ and μ_0 are the covariance and mean computed over the entire dataset. Since Σ and μ_0 are common for entire data, finding w_q requires finding the mean vector μ_q of the class to which the query q belongs to. Let us define the set of class mean vectors for the frequent classes as $\mathcal{W}_\mu = \{\mu_1, \dots, \mu_N\}$. The mean vector μ_q for the class of the query q is computed by making use of appropriate cut portions from the mean vectors of the frequent classes. Optimizing (4.1) for variable length cut portions entails high computational complexity. Therefore, instead of matching variable-length n grams the method divides X_q into p number of fixed length portions.

1. The class mean vectors of the most frequent 1000 classes are concatenated.

2. Now, each query cut portion X_q^k is searched in the concatenated mean vector using subsequence dynamic time warping [106]

3. The most similar segment in the concatenated mean vector is taken as the corresponding portion of the query class mean μ_q^k.

4. The concatenation of these query class mean cut portions μ_q^k synthesizes the query class mean $\mu_q = [\mu_q^1, \dots, \mu_q^p]$.

Since DTW is computationally slow, applying subsequence DTW in this case is computationally expensive.

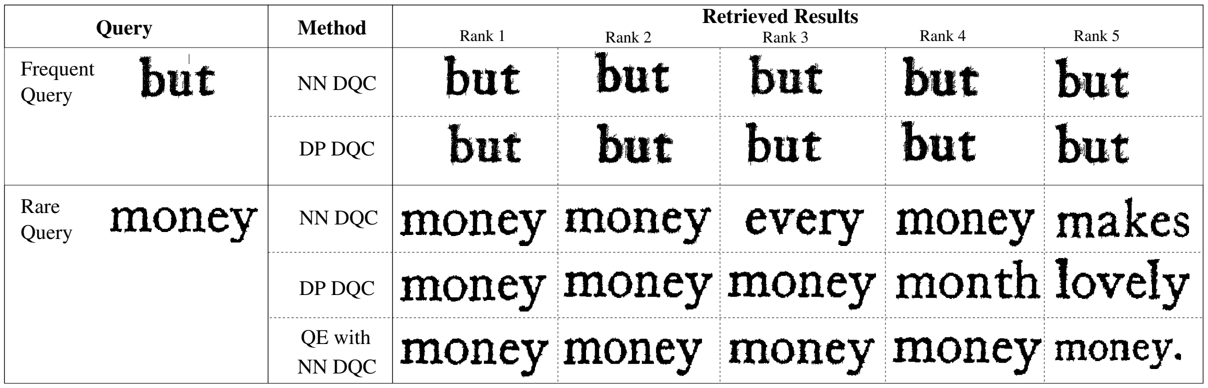
4.2.2 NN DQC: Design of DQC using Approximate Nearest Neighbour

A speed-up is obtained using approximate nearest neighbor search instead of DTW.

- Instead of concatenating the class mean vectors, each class mean vector is now divided into the same p number of fixed length portions. An index is built over frequent class means cut portions using FLANN.

- Each cut portion of X_q is compared with frequent class means cut portions using nearest neighbor search with Euclidean distance.

- The best matching cut portions of the mean vectors are used to synthesize the mean vector for the query class.



Query		Method	Retrieved Results				
			Rank 1	Rank 2	Rank 3	Rank 4	Rank 5
Frequent Query	but	NN DQC	but	but	but	but	but
		DP DQC	but	but	but	but	but
Rare Query	money	NN DQC	money	money	every	money	makes
		DP DQC	money	money	money	month	lovely
		QE with NN DQC	money	money	money	money	money.

Figure 4.1: The figure shows a few query words and their corresponding retrieval results. The first column shows the query image, and the corresponding images in each row are its retrieval results. The first two rows show frequent query results. The first row show the results for NN DQC, and the second row show the results for DP DQC. Row 3 to Row 5 show the retrieval results for a rare query. Row 3 shows the results for NN DQC and, row 4 show the results for DP DQC and Row 5 show the results for query expansion.

However, using nearest neighbour (NN DQC) instead of subsequence DTW based scheme (DP DQC) compromises the optimality of the classifier synthesis.

Few qualitative examples for the two versions of DQC are given in Figure 4.1. We have shown the retrieval results for frequent queries and rare queries. For each case, we have compared the retrieval results for NN DQC and DP DQC. For rare queries, we have also shown the results for Query expansion (QE).

4.3 Query Specific Fast DTW Distance

In Fast Surrogate DTW distance (section 3.3), the global principal alignments are computed from the given data. Here, no class information is used while computing the alignments, and also, these alignments are query independent, i.e., query information is not used while computing the global principal alignments. In this section, we introduce Query specific DTW distance, which is computed using query specific (global) principal alignments. The proposed query specific DTW distance has been found to perform much better when used with the direct query classifier.

Let $\mathcal{X}$ be the given data and all the samples are scaled to a fixed size. Let $\{C_1, C_2, \ldots, C_N\}$ be the most frequent N classes from the data and $\mu_1, \ldots, \mu_N$ be their corresponding class means. The matching process using the query specific principal alignments is as follows:

(i) Divide each sample from the frequent classes into a fixed number p of equal size portions. Let $x_{i_1}, \ldots, x_{i_{|c_i|}}$ be the samples (sequences) from the i^{th} class c_i, where $|c_i|$ is the number of samples in class c_i. The cut portions for the class means μ^i are denoted as $\mu_1^i, \ldots, \mu_p^i$, where each cut portion is of length d. Similarly, divide the query X_q into the same number p of fixed length portions.

(ii) For each class, compute the global principal alignments for each cut portion separately. These alignments corresponds to the cut specific principal alignments for the class. For i^{th} class and j^{th} cut portion the cut specific principal alignments are computed from $\{x_{i_1}^j, \ldots x_{i_{|c_i|}}^j\}$ and these are denoted as G_i^j. These alignments are computed for all the cut portions for each class.

(iii) The final step computes the cut specific principal alignments for the given query X_q as follows. For each cut portion of X_q we compute the DTW distance (Euclidean distance over the cut specific principal alignments) with the corresponding cut portions of all the class means using their corresponding cut specific principal alignments. The distance between the j^{th} cut portion of X_q i.e., X_q^j and the j^{th} cut portion of the i^{th} class mean i.e., μ_i^j is denoted as

$$Dis_i^j = \sum_{\pi \in G_i^j} \mathit{Euclid}_\pi(X_q^j, \mu_i^j) \tag{4.3}$$

For each cut portion of X_q, we compute the minimum distance mean cut portion over all the class mean vectors. The corresponding cut specific principal alignments of the closest matching mean cut portions are taken as the cut specific principal alignments of the query cut portion. Also, the corresponding class mean cut portion is taken as the matching cut portion for constructing the query mean. Let the j^{th} cut portion of the query best match with the j^{th} cut-portion of the class with index c.

$$c = \underset{i}{\mathit{arg\,min}}\ Dis_i^j \tag{4.4}$$

Here the minimum distance is computed over all the frequent classes. We thus have

$$G^j_{X_q} \longleftarrow G^j_c \qquad \text{and} \qquad \mu^j_q \longleftarrow \mu^j_c \tag{4.5}$$

Here $G^j_{X_q}$ is the cut specific principal alignments for the j^{th} cut portion of X_q.

Together, all these query mean cut portions give the query class mean. The query class mean μ_q is given as $\mu_q = (\mu^1_q, \mu^2_q, \ldots, \mu^p_q)$. This query class mean μ_q is then used as in Equation 4.2 to compute the LDA weight w_q (query classifier weight).

The query specific (qs) DTW distance between the query X_q and a sample X from the data is given as

$$\underset{qs}{dtw}(X_q, X) = \sum_{i=1}^{p} dtw_{G^i_{X_q}}(X^i_q, X^i) \tag{4.6}$$

where p is the number of cut portions.

Fig 4.2 shows all the processing stages of the nearest neighbour DQC. To summarize, we generate query specific principal alignments on the fly by selecting and concatenating the global principal alignments corresponding to the smaller *n grams* (cut portions). Our strategy is to build cut-specific principal alignments for the most frequent classes; these are the word classes that will be queried more frequently. These cut-specific principal alignments are then used to synthesize the query specific principal alignments (see Figure 4.3). The results demonstrate that our strategy gives good performance for queries from both the frequent word classes and rare word classes.

In general, the alignments trained on one dataset may not work well on another dataset. This is mainly due to the print and style variations. To adapt to different styles, we use query expansion (QE), a popular approach in the information retrieval domain in which the query is reformulated to improve the retrieval performance further. An index is built over the given sample vectors from the database, and using an approximate nearest neighbor search, the top 10 similar vectors to the given query are computed. These top 10 similar vectors are then averaged to get the new reformulated query. This reformulated query is expected to capture the variations in the query class better. In our experiments, this further improves the retrieval performance. Approximate nearest neighbors are obtained using FLANN [96].

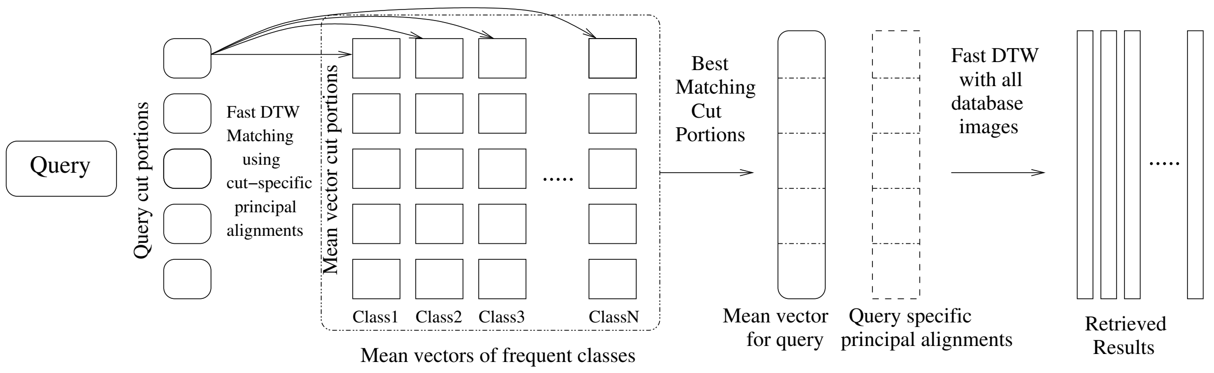


Figure 4.2: Overall Scheme for NN DQC. In an offline phase, the mean vectors for the frequent word classes are computed and their cut-specific principal alignments are computed. To process a query word image, it is divided into cut portions and Fast Surrogate DTW matching is used to get the best matching cut-portions from the frequent class mean vectors with the cut-portions of the query image. These best matching cut-portions are used to construct the mean vector for the query class and the query specific principal alignments. Fast SurrogateDTW [109] matching between the query image and the database images is done using the query specific principal alignments.

4.4 Results and Discussions

In this section we validate the DQC classifier using query specific Fast Surrogate DTW distance for efficient indexing on multiple word image collections and also demonstrate its quantitative and qualitative performance.

4.4.1 Datasets and Evaluation Protocols

In this subsection we discuss datasets and the experimental settings that we follow in the experiments. Our datasets, given in Table 4.1, comprise scanned English books from a digital library collection. We manually created ground truth at the word level for the quantitative evaluation of the methods. The first collection (D1) of words is from a reasonably clean book. The Second dataset (D2) is larger in size and is used to demonstrate the performance in the case of heterogeneous print styles. The third data set (D3) is a noisy book and is used to demonstrate the utility of the performance of our method in degraded collections. We have also given the results over the popular George Washington dataset. For the experiments, we extract profile features [130] for each of the word images. In this, we divide the image horizontally into two parts, and the following features are computed: (i) vertical profile i.e., the

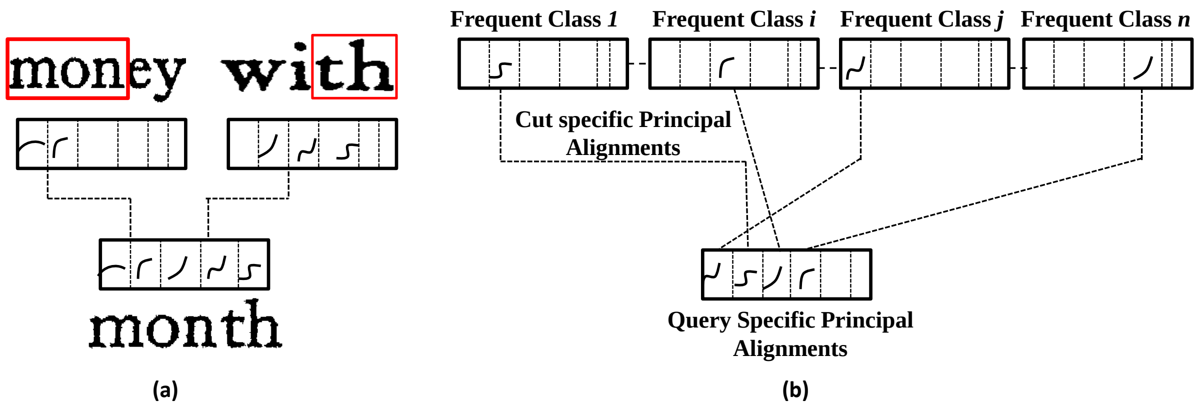


Figure 4.3: Synthesis of query specific principal alignments. (a) Cut specific principal alignments corresponding to "ground" and "leather" are joined to form the principal alignments for "great". Note that the appropriate cut portions are automatically found. (b) In a general setting, query specific principal alignments gets formed from multiple constituent cut specific principal alignments computed for frequent classes.

number of ink pixels in each column, (ii) location of lowermost ink pixel, (ii) location of uppermost ink pixel and (iv) number of ink to background transitions. The profile features are calculated on binarized word images obtained using the Otsu thresholding algorithm. The features are normalized to [0, 1] so as to avoid dominance of any specific feature.

To evaluate the quantitative performance, multiple query images were generated. The query images are selected to have multiple occurrences in the database are mostly functional words and do not include the stop words. The performance is measured by mean Average Precision (mAP), which is the mean of the area under the precision-recall curve for all the queries.

4.4.2 Experimental Settings

For representing word images, we prefer a fixed length sequence representation of the visual content, i.e., each word image is represented as a fixed length sequence of vertical strips. A set of features f_1,. . ., f_L are extracted, where $f_i \in \mathbb{R}^M$ is the feature representation of the i^{th} vertical strip and L is the number of vertical strips. This can be considered as a single feature vector $F \in \mathbb{R}^d$ of size $d = LM$. We implement the query specific alignment based solution as discussed in Section 4.3. For the query

Dataset	Source	Type	# Images	# Queries
D1	1 Book	Clean	14510	100
D2	2 Books	Clean	32180	100
D3	1 Book	Noisy	4100	100

Table 4.1: Details of the datasets considered in the experiments. The first collection (D1) of words is from a book which is reasonably clean. The second dataset (D2) is obtained from 2 books and is used to demonstrate the performance in case of heterogeneous print styles. The third data set (D3) is a noisy book.

expansion based solution, we identify the five most similar samples to the query using an approximate nearest neighbor search and compute their mean.

Each dataset contains certain words which are more frequent than others. The number of samples in the frequent word classes are more compared to the rare classes. The retrieval results for frequent queries give better performance because the number of relevant samples available in the dataset are more. It is worth emphasizing that for the method proposed in this work (QS DTW), the degradation in the performance for rare queries is much less compared to other methods.

4.4.3 Results for Frequent Queries

Table 4.2 compares the retrieval performance of the direct query classifier DQC with the nearest neighbour classifier using different options for distance measures. The performance is shown in terms of mean average precision (mAP) values on three datasets. For the nearest neighbour classifier, we experimented with five distance measures: naive DTW distance, Fast Surrogate DTW distance [109], query specific DTW (QS DTW) distance, Salvador *et al.* [140] and Euclidean distance. We see that QS DTW performs comparably with DTW for all the datasets. It performs superior compared to Salvador *et al.*. Fast Surrogate DTW distance [109] performs significantly better than Euclidean distance.

For DQC, we experimented with four options for indexing the frequent class mean vectors: subsequence DTW [128] (sDTW), approximate nearest neighbour NN DQC [128] (aNN), Fast Surrogate DTW, and QS DTW. We use the cut-portions obtained from the mean vectors of the most frequent 1000 word classes for (i) computing the cut-specific principal alignments in case of QS DTW, (ii) computing the

Dataset	Retrieval Results (mAP) for Frequent Queries								
	Using Nearest Neighbour Classifier					Using DQC (Exemplar SVM)			
	DTW	Fast Surrogate DTW [109]	QS DTW	Euclidean	Salvador *et al.* [140]	sDTW	aNN	Fast Surr DTW	QS DTW
D1	0.94	0.92	0.92	0.81	0.91	0.98	0.98	1	**1**
D2	0.91	0.89	0.9	0.75	0.87	0.96	0.95	0.97	**0.99**
D3	0.83	0.79	0.81	0.67	0.76	0.91	0.92	0.93	**0.96**

Table 4.2: Retrieval performance of various methods for frequent queries.

closest matching cut-portion (i.e., one with the smallest distance, which can be Euclidean or DTW) with a cut-portion from the query vector, in case of aNNor Fast Surrogate DTW.

However, since sDTW has computational complexity $O(n^2)$, we restrict the number of frequent words used for indexing to 100. The QS DTW distance improves the performance of the DQC classifier. This is mainly due to the improved alignments involved in the QS DTW distance. The query specific alignments better capture the variations in the query class. Moreover, unlike the case of sDTW distance the QS DTW distance has linear complexity and therefore we are able to index all the frequent mean vectors in the DQC classifier. Thus, the proposed method of QS DTW enhances the performance of the DQC classifier [128].

The experiments revealed that the QS DTW gets the global principal alignments from the mean vector of the same (query) class for frequent queries. Since the alignments are coming from the query class, it gives a minimum distance only for the samples which belong to its class. Therefore, the retrieved samples largely belong to the query class. The performance is therefore improved compared to sDTW distance. In contrast, the Fast SurrogateDTW distance [109] computes the global principal alignments using all samples in the database, without exploiting any class information. The computed global principal alignments, therefore, include alignments from classes that may be different from the query class. For this reason, it performs inferior to the proposed QS DTW distance.

Dataset	Retrieval Results (mAP) for Rare Queries									
	Using Nearest Neighbour Classifier					Using DQC (Exemplar SVM)				
	DTW	Fast Surrogate DTW [109]	QS DTW	Euclidean	Salvador *et al.* [140]	sDTW	aNN	Fast Surr DTW	QS DTW	QE
D1	0.82	0.77	0.83	0.69	0.75	0.91	0.90	0.91	0.95	**0.98**
D2	0.81	0.74	0.80	0.65	0.74	0.89	0.90	0.90	0.94	**0.95**
D3	0.73	0.66	0.71	0.59	0.62	0.80	0.78	0.80	0.91	**0.96**

Table 4.3: Retrieval performance of various methods for rare queries.

4.4.4 Results for Rare Queries

The faster indexing offered by using QS DTW with DQC allows us to use the mean vectors of all the 1000 frequent classes. This gives us much improved performance of the DQC on rare queries compared to sDTW [128], which uses mean vectors from 100 frequent classes. Table 4.3 shows the retrieval performance of DQC with a nearest neighbour classifier using different options for distance measures. The performance is shown in terms of mean average precision (mAP) values on rare queries from three datasets. For the nearest neighbour classifier, we experimented with five distance measures: naive DTW distance, Fast Surrogate DTW distance [109], query specific DTW (QS DTW) distance, Salvador *et al.* [140] and Euclidean distance. We see that QS DTW performs comparably with DTW distance for all the datasets. It performs better than the Fast Surrogate DTW distance [109], Salvador *et al.* [140] and significantly better than Euclidean distance.

For DQC, we observe that QS DTW improves the performance compared to sDTW. This improvement of QS DTW over sDTWis more for rare queries compared to that for frequent queries. This shows that QS DTW can be used for faster indexing for both frequent and rare queries.

For rare queries, the query specific DTW distance outperforms Fast SurrogateDTW [109] distance. This happens because the Fast SurrogateDTW computes the global principal alignments from the database and its performance depends on the number of samples. Also, these alignments are query independent, i.e., they do not use any query information for computing the global principal alignments. For a given query, it needs enough samples from the query class to get novel global principal alignments. However, in any database the number of samples for frequent classes dominates the number of samples for rare classes. The global principal alignments for frequent queries are likely to dominate the rare queries. Therefore, the precomputed global principal alignments in Fast SurrogateDTW may not capture all the correlations for rare query classes. In the proposed QS DTW distance, the global principal alignments are learned from the *ngrams* (cut-portions) of frequent classes. These *ngrams* are in abundance and also shared with rare queries, thus there are enough *ngram* samples for learning the cut-specific alignments. The computed query specific alignments for the cut-portions outperform the alignments obtained from Fast SurrogateDTW.

It is worth mentioning that Salvador *et al.* [140] is an approximation method attempt to compute the DTW distance efficiently. It does not consider cut portion similarities, which may be influenced by various printing styles. Hence, these approaches are not applicable in our setting where the dataset can

Average of DTW distance				
For given query	For Reformulated Query			
	n=2	n=5	n=7	n=10
2.67 ±0.19	2.69±0.23	**2.52±0.13**	2.58±0.21	2.94±0.29

Table 4.4: The table gives the average sum of DTW distance for the given query and the reformulated query with varying number of samples n from the query class.

have words printed in varied printing styles and thus can result in marked degradation of performance for rare queries. Since query specific DTW finds the approximate DTW distance using cut specific principal alignments, it can exploit properties that cannot be used by other DTW approximation methods.

To summarize, the experiments demonstrate that the proposed query specific DTW performs well for both frequent and rare queries. Since it is learning the alignments from *ngrams*, it performs comparable to sDTWdistance for rare queries. For some queries, it performed better than the DTW distance.

4.4.5 Results for Rare Query Expansion

The results for QS DTW enhanced with query expansion (QE) using the five best matching samples are also given in Table 4.3. It is observed that QE further improves the performance of our proposed method. To show the effectiveness of query expansion, we have computed the average of the DTW distance between the given query and all database samples that belonged to the query class. Likewise, we computed the average of the DTW distance for the reformulated query. Table 4.4 shows a comparison of the averaged DTW distance for the given query and the reformulated query using 2, 5, 7, and 10 most similar (to the query) samples from the database. From the results, we can observe that compared to the given query, the reformulated query using five best matching samples gives the lowest averaged DTW distance to the samples from the query class. This means the reformulated query is a good representative of the given query. However, using nine best matching samples for reformulating the query leads to a higher average of DTW distances. This means some irrelevant samples to the query come in the top similar samples.

Dataset	mAP using nearest neighbour				mAP using DQC		
	DTW	Fast Surrogate DTW [109]	QS DTW	Euclidean	sDTW	Fast Surr DTW	QS DTW
GW	0.51	0.50	0.52	0.32	0.62	0.63	**0.70**

Table 4.5: Retrieval performance on the George Washington (GW) dataset. The DQC makes use of top 800 frequent classes for indexing the cut-portions.

Length of cut portion	D1	D2	D3
1	0.81	0.78	0.7
10	**0.86**	**0.83**	0.74
20	0.86	0.82	**0.75**
30	0.82	0.77	0.72

Table 4.6: The table shows change in retrieval performance with change in the length of cut portion over all the datasets (D1, D2, D3).

4.4.6 Results on George Washington Dataset

The George Washington (GW) dataset [7] contains 4894 word images from 1471 word classes. This is one of the popular datasets for word images. We applied our proposed method of DQC using QS DTW for word retrieval on the GW dataset. Table 4.5 provides comparative results for seven methods. Experiments are repeated for 100 random queries and the average over these results are reported in the table. We can observe that for the DQC the proposed QS DTW gives better performance than DTW. We can also observe that for the nearest neighbour classifier, QS DTW distance is performing slightly superior to the DTW distance and Fast SurrogateDTW distance. The superiority is because of the principal alignments, which are query specific.

4.4.7 Setting the Hyperparameters

The proposed method has few hyperparameters, like the length of the cut portion and the number of cut specific principal alignments. For tuning these parameters, we randomly choose 100 queries for each dataset and validate the performance over these queries. Queries included in the validation set are not used for reporting the final results.

Number of cut specific	mAP for different datasets		
Principal Alignments	D1	D2	D3
10	0.92	**0.89**	**0.81**
20	**0.93**	0.89	0.81
30	0.91	0.88	0.78

Table 4.7: Retrieval performance on the 3 datasets D1, D2 and D3 for varying number of cut specific principal alignments.

In Table 4.6, we report the effect of varying the cut portion length on retrieval performance. The mAP score is less for smaller cut portion length. In this case, the learned alignments are not capturing the desired correlations. This happens because the occurrence of smaller cut portions is very frequent in the word images. For length more than 30, the mAP is again decreased. This is because the occurrences of larger cut portions are rare. Cut portion lengths in the range of 10 to 20 give better results. In this case, the cut portions are good enough to yield global principal alignments that can distinguish the different word images.

We assessed the effect of varying the number of cut-specific principal alignments on the retrieval performance of the three datasets and the results are given in Table 4.7. It is seen that the performance degrades for all the datasets when the number of alignments is chosen as 30. This can be attributed to some redundant alignments getting included in the set of principal alignments. Increasing the number of alignments from 10 to 20 improves performance for dataset D1, but has no effect on the performance for datasets D2 and D3. Therefore, we can conclude that restricting the number of principal alignments in the range of 10 to 20 would give good results. In all our experiments, we set the number of cut-specific principal alignments as 10.

4.4.8 Computation Time

Table 4.8 gives the computational time complexity for the methods based on DTW. The main computation involved in the use of QS DTW is that of computing the cut specific principal alignments for the frequent classes. Figure 4.4 shows the time for computing the cut specific principal alignments for the 3 datasets. The computation of these cut specific principal alignments can be carried out independently

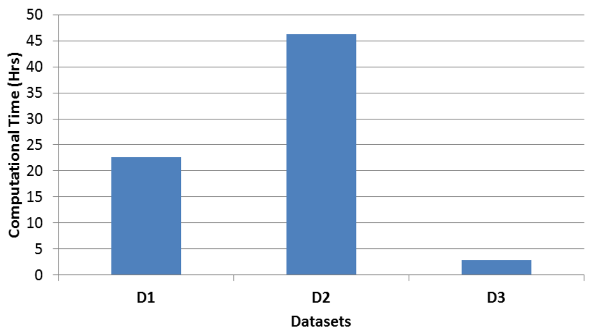


Figure 4.4: Computation time for computing the cut specific principal alignments for all the datasets. It includes the computation of cut specific principal alignments for all the frequent classes over all the cut portions.

for all the classes. Since we can compute these principal alignments in parallel with each other, the proposed QS DTW scales well with the number of samples compared to Fast Surrogate DTW [109].

Unlike the case of QS DTW, where the global principal alignments are computed for the small cut portions, in Fast Surrogate DTW, the principal alignments are computed for the full word image representation. Further, in Fast Surrogate DTW, the principal alignments are computed from the entire dataset, unlike the case of QS DTW in which the principal alignments are computed for the individual classes. For these reasons, Fast Surrogate DTW is computationally slower compared to the QS DTW.

For a given dataset, computing the cut specific principal alignments for the frequent classes is an offline process. When performing retrieval for a given query, DQC involves computing the query mean by composing together the nearest cut portions from the mean vectors of frequent classes. Further, the query specific principal alignments are not explicitly computed, but rather constructed using the cut-specific principal alignments corresponding to the nearest cut portions. Once the query specific principal alignments are obtained, computation of QS DTW involves computing the Euclidean distance (using the query specific principal alignments) with the database images.

Methods	sDTW	Fast Surrogate DTW [109]	Salvador *et al.* [140]	QS DTW
Computational Complexity	$O(n^2)$	$O(n)$	$O(n)$	$O(n)$

Table 4.8: Computational complexities of DTW-based methods for distance computation. Here n is the length of the cut-portion of the feature vector.

For given two samples x and y of length N, Salvador *et al.* [140] computed the DTW in the following way. First, these two samples are reduced to a smaller lengths (1/8 times), and the naive DTW distance is applied over the reduced length samples to find the optimal warp path. Next, both the optimal path and the reduced length samples from the previous step are projected to be higher (2 times) resolution. Instead of filling all the entries in the cost matrix in the higher resolution, only the entries around a neighborhood of the projected warp path, governed by a parameter called radius r, are filled up. This projection step is continued till the original resolution is obtained. The time complexity of the approximated DTW proposed in [140] is $N(8r + 14)$, where r is the radius. Its performance depends on the radius r. The higher the value of r, the better the performance is. The time complexity of QS DTW/Fast Surrogate DTW is $N * p$, where p is the number of principal alignments. In general, $p << 8r + 14$, for getting the similar performance in both the methods.

4.5 Recent Advances in DTW Distance

In this section, we discuss few recent advances in DTW distance.

4.5.1 DTW Distance as a Differentiable Loss Function for Time Series

In the domain of sequence data analysis, compared to the Euclidean distance, the DTW distance reveals the true similarity between target samples. This is mainly due to invariance against shifting and scaling in the time axis (Doppler effect). However, it has limited applicability for the problems where we need the gradient of the corresponding similarity function. These gradients are needed for the problems like clustering and multistep-ahead prediction. The DTW distance is not differentiable with respect to its input variables. Due to the non-differentiable behavior of DTW distance, Euclidean distance is preferred for the problems where we need its corresponding gradient.

4.5.1.1 Soft-DTW

In [34], a differentiable learning loss between time series is proposed, which turns the popular DTW discrepancy into a full fledged loss function. It widens the applicability of DTW distance. Consider two multivariate time series $x = (x_1, \ldots, x_n) \in \mathbb{R}^{p \times n}$, $y = (y_1, \ldots, y_m) \in \mathbb{R}^{p \times m}$ and their cost matrix $C(x, y)$, the γ-soft-DTW is defined as

$$dtw_\gamma = min^\gamma\{\langle A, C(x,y)\rangle, A \in A_{n,m}\} \tag{4.7}$$

where, $A_{n,m}$ is the set of all alignment matrices between x and y. The min operator min^γ with a smoothing parameter $\gamma \geq 0$ is defined as

$$min^\gamma\{b_1, \dots, b_n\} = \begin{cases} min_{i \leq n} b_i, & \text{if } \gamma = 0 \\ -\gamma log \sum_{i=1}^{n} e^{-\frac{b_i}{\gamma}}, & \gamma > 0 \end{cases} \tag{4.8}$$

If $\gamma = 0$ then the soft-DTW is same as the naive DTW distance.

4.5.1.1.1 Differentiation of soft-DTW If $\gamma = 0$, then dtw_0 is the minimum over a finite set of linear equations of C. Therefore, dtw_0 is locally differentiable w.r.to the cost function C [137]. The gradient of dtw_0 w.r.to x is computed using the chain rule

$$\nabla_x dtw_0(x,y) = \Big(\frac{\partial C(x,y)}{\partial x}\Big)^T A^{op} \tag{4.9}$$

where, $\dfrac{\partial C(x,y)}{\partial x}$ is the jacobian of C w.r.to x and A^{op} is the optimal alignment matrix between x and y.

For $\gamma > 0$, the gradient of soft-DTW is given as

$$\nabla_x dtw_\gamma(x,y) = \Big(\frac{\partial C(x,y)}{\partial x}\Big)^T \mathbb{E}_\gamma[A] \tag{4.10}$$

where, $\mathbb{E}_\gamma[A] = \dfrac{1}{\sum_{A \in A_{n,m}} e^{-\langle A, C(x,y)/\gamma\rangle}} \sum_{A \in A_{n,m}} e^{-\langle A, C(x,y)/\gamma\rangle} A$ is the average alignment matrix under the Gibbs distribution $p_\gamma = e^{-\langle A, C(x,y)\rangle/\gamma}$. The gradient computation given in Eq 4.10 is inspired from the derivative of Edit distance [119, 143].

For the problems like clustering and multistep-ahead prediction, soft-DTW loss produces better results compared to Euclidean loss [34]. Soft-DTW loss can be applied for the problems where we use Euclidean loss. In [26], a discriminative differentiable dynamic time warping (D3TW) is proposed to solve video alignment and segmentation problems in a weakly supervised setting.

4.5.2 Dynamic Time Warping Networks (DTWNet)

Over the past few years, there has been increasing interest in deep learning methods, and they have produced state-of-the-art results over various problems. Although DTW is one of the important similarity measures for temporal data and is useful in data analysis, it is not explored much in the deep learning domain. In [23], the DTW distance is successfully used in the deep learning framework. The authors [23] proposed the DTWNet, where the DTW distance is applied as a feature extractor. Unlike the convolution, the DTW handles the Doppler effect and has the non-linear transformation property [23]. In DTWNet, the DTW kernels are learned using standard stochastic gradient descent on the warping path. For an input sequence $x \in \mathbb{R}^l$, the objective is to learn a kernel $k \in \mathbb{R}^m (m \leq l)$ such that it has best warping path with x, i.e. $k = \text{argmin}_z \; dtw(x, z)$. The kernel k is initialized randomly. For the problems like data decomposition, it outperforms standard convolutional kernels.

In DTWNet, the DTW computation is applied in the DTW layers. The DTW is computed between the input and the kernels. The DTWNet contains multiple DTW layers, each of which contains multiple DTW kernels, which extract important features from the input. In sliding window, similar to the convolutional kernels, the DTW kernel produces multiple distance outputs. Since the entries along the warping path (optimal path) contribute to the final DTW distance, in DTWNet, the differentiation is performed only along this warping path. It makes the differentiation of DTW distance faster. For two sequences of length l and m, the gradient along the warping path takes $O(l+m)$ time. The soft-DTW [34] takes $O(lm)$ time for computing the gradient.

4.6 FastDTWNet

In DTWNet [23], the gradient is computed along the warping path. For two sequences of length l and m, the warping path computation takes $O(lm)$ time. This slows down the training process in DTWNet. In this section, we propose FastDTWNet. Compared to DTWNet, FastDTWNet employs Fast Surrogate DTW distance, which increases the training speed. In Fast Surrogate DTW distance, we approximate the warping path using global principal alignments. In FastDTWNet, instead of computing the warping path between the input and kernels, we approximate it using the global principal alignments (defined in Chapter 3). Now, the gradient is computed along these global principal alignments. It will increase the training speed in FastDTWNet. We show the speed up on the popular Libra dataset. For both the DTWNet and FastDTWNet, we take 5 DTW layers. We compare the training time for DTWNet and FastDTWNet in

	Training Time (mins)	Accuracy
DTWNet (Actual warping path)	51	97.8
FastDTWNet (Approximated warping path)	32	97.6

Table 4.9: Comparison of training time for DTWNet using Actual Warping path and Approximated Warping path (FastDTWNet). In Actual Warping path, the warping path is computed using dynamic programming. In Approximated Warping path, the warping path is computed using global principal alignments.Training time is given in minutes.

Table 4.9. From the results, we can observe that the warping path approximated using global principal alignments speeds up the training in DTWNet. In terms of accuracy, the DTWNet using global principal alignments (FastDTWNet) is performing equally well compared to the DTWNet using the actual warping path.

Chapter 5

CCK: A Surrogate Kernel Approximation of Canonical Correlation Analysis (CCA)

In our previous chapter (Chapter 3), we presented a surrogate kernel over DTW (Fast Surrogate DTW kernel), enabling us to use DTW with a linear SVM. In this chapter, we present a surrogate kernel CCK over canonical correlation analysis (CCA). It enables us to use CCA in kernel framework, which further improving its performance. The kernel function works well for action recognition as it embeds the temporal context in the videos. We have also shown that multiple features can be seamlessly integrated into the surrogate kernel to enhance the recognition performance further. It seamlessly integrates the advantages of lower dimensional representation of videos with a discriminative classifier like SVM. This chapter presents a novel application built using the proposed surrogate kernel CCK in the deep learning setting. We propose CCKNet, where CCK is used as a feature extractor in CNN.

5.1 Introduction

Recent advances in action recognition are propelled by (i) the use of local as well as global features [87, 170], which have significantly helped in object and scene recognition by computing them over 2D frames [64, 170] or over a 3D video volume [37] (ii) the use of factorization techniques over video volume tensors [83,94] and defining similarity measures over the resulting lower dimensional factors [18]. In this work, we try to take advantage of both approaches by defining a canonical correlation kernel computed from tensor representation of the videos. This also enables seamless feature fusion by combining multiple feature kernels.

Wang *et al.* [170] demonstrated the successful use of multiple features defined over relatively densely extracted tracks. Motivated by the success of dense features for object recognition [39, 171], we do a further dense feature extraction on a regular grid of pixels which helps us to obtain a rich and robust set of descriptors. However, we avoid any explicit tracking across frames. Though there have been many previous attempts in using spatio-temporal descriptors in the past [156], our focus is to explore the utility of well-understood 2D image descriptors. Our method captures the temporal information as well as correlation across the frames while computing the low-dimensional representations of tensors.

Spatio-temporal shape and texture of the action videos are well represented in a number of low-rank representations computed out of the tensors with various factorization techniques [94, 95]. This tensor computation has been successfully applied in many vision tasks including action classification in the past [83, 94, 95]. Recognition is often carried out on the prominent components obtained by the factorization or dimensionality reduction of the videos. Kim *et al.* [83] represent the video as a tensor and compute the similarities using canonical correlation [18] with specially selected discriminative correlation coefficients. These tensorial representation methods [83, 94, 95] often use pixel values directly as observations to build the data tensor. However, [81] uses SIFT for tensor representation. One of the ingredients of the success of our method is the use of rich feature descriptors in the tensorial representation of the video. We also define a canonical correlation kernel that seamlessly integrates the advantages of lower dimensional representation of videos with a discriminative classifier like SVM. This also enables us to weigh the features differently to improve the recognition performance.

Wolf and Shashua [173] extended the notion of canonical correlation analysis (CCA) to introduce a kernelized version of the same in KCCA. This is considerably different from what we do in this work. Rather than kernelizing CCA, we are interested in defining a kernel that can be used in many situations where canonical correlation is used. However, by computing similarities over projections on nonlinear manifolds we find that the correlation analysis with the help of KCCA also provides useful information for action recognition. We simultaneously consider the correlations computed over linear as well as nonlinear manifolds of the video data tensor. While the individual correlation coefficients can not be used as a valid measure for comparing two action videos in a kernel setting, their sum becomes a valid measure. Our kernel is simple to compute and visualize, starting from canonical correlation analysis. However, this enables a host of useful tricks. (i) we can use SVM along with CCA/KCCA based feature extraction (ii) we can simultaneously compute similarities over multiple features in a single framework (iii) we can optimally fuse the advantages of the bag of words representation (as in [171]) and tensor based methods (as in [83, 95]) for action recognition. Multiple feature kernels are often combined using multiple kernel learning (MKL) [86] framework. MKL techniques [172] have been used in action recognition for combining different contextual features. Our kernel yields superior results with simple (say pixel values) features, it allows comparing videos without hand coding or tracking. An illustration of the framework of the proposed method is shown in Figure 5.1. The method has 3 steps, a given video is represented using a 3D tensor in the first step. In the second step, this 3D tensor is decomposed into three 2D tensors/matrices. Finally, for the given two videos, CCK is computed from their corresponding 2D tensors, which involves CCA and KCCA. This results in a set of correlation coefficients. The sum of these correlations gives the CCK for the given two videos.

Our approach is evaluated on four popular action video data sets. Our single feature representation (with pixel values as features) outperforms most competitive methods, which use more powerful features. We also show that recognition performance improves by the intuitive and seamless fusion of multiple feature kernels. Our proposed canonical correlation kernel is explained in Section 5.3. Experimental results are discussed in detail in Section 5.4.

5.2 Related Work

A broad spectrum of features and representations has been used for action recognition in the past. Initial attempts like Motion Intensity/History Images represented the whole video as a single image and used traditional feature extraction for recognizing actions. Such features typically capture global motion information in a compact manner. On the other extreme, local information captured using features like SIFT [156], HOG [64, 170], LBP [78] and MBH were also used for describing video frames and found to be useful for action recognition. The need for defining a set of distinct descriptors for video (from images) was realized, and many features like SIFT and HOG got extended to video by defining them over a volume rather than over a 2D grid [156]. However, a successful direction has been to track the features over frames and to compute the descriptors over this track to represent the action content [101, 103]. By making these tracks denser, Wang *et al.* [170] obtained excellent results on popular data sets. We argue that such dense and feature rich representations can result in superior results when used along with the learned representations from video volumes.

A video can be represented as a third order data tensor denoted as $\mathcal{V} \in \mathbb{R}^{X \times Y \times Z}$. Where, X and Y are spatial dimensions, which gives spatial information and Z is a time axis, which gives temporal information of a video. This representation is bulky and noisy for deriving effective representations for action recognition. A wide variety of decomposition techniques [83,94,95] were used for deriving lower

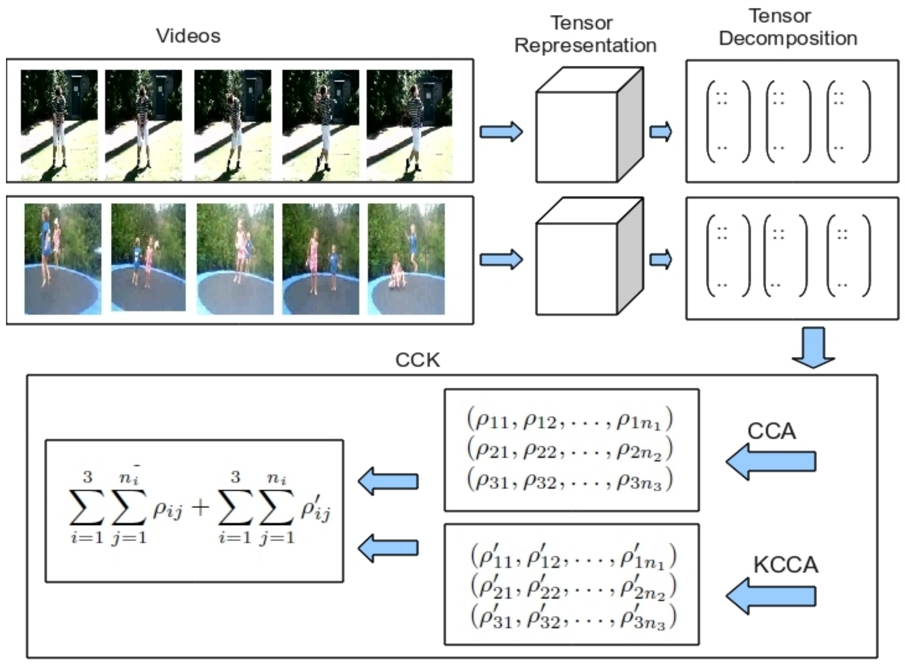

Videos
Tensor Representation
Tensor Decomposition
CCK
CCA
KCCA
$(\rho_{11}, \rho_{12}, \ldots, \rho_{1n_1})$
$(\rho_{21}, \rho_{22}, \ldots, \rho_{2n_2})$
$(\rho_{31}, \rho_{32}, \ldots, \rho_{3n_3})$
$(\rho'_{11}, \rho'_{12}, \ldots, \rho'_{1n_1})$
$(\rho'_{21}, \rho'_{22}, \ldots, \rho'_{2n_2})$
$(\rho'_{31}, \rho'_{32}, \ldots, \rho'_{3n_3})$
$\sum_{i=1}^{3}\sum_{j=1}^{n_i}\rho_{ij} + \sum_{i=1}^{3}\sum_{j=1}^{n_i}\rho'_{ij}$

Figure 5.1: An illustration of the proposed method. In the first step, videos are represented using 3D tensors. Next, these 3D tensors are flattened to obtain three 2D matrices. Finally, CCK is computed as the sum of correlations between the flattened matrices obtained from CCA and KCCA.

dimensional representations from these data tensors. A successful method [83] is to start with projections of these tensors over multiple dimensions resulting in matrices and deriving algorithms that work on these matrices. Often matrix projections on the spatial and temporal axes are represented as points on a manifold [94]. Canonical correlation analysis represents an action video with the help of a vector of a discriminatively selected subset of correlation coefficients defined over a linear manifold [18]. The kernalized version of CCA [173] measures the correlation on a nonlinear implicit manifold. Alternate techniques for representing the action videos include those based on tangent bundles [94], product manifolds [95], etc. In tangent bundles [94], data tensors are factorized to a set of tangent spaces on a Grassmann manifold. In product manifold [95], each tensor is considered as a point on a product manifold and the tensor is factorized using a modified High Order Singular Value Decomposition. Naoki *et al.* [3] represent the gait dynamics trained from multiple training videos by a standard manifold.

Once the video is represented in a lower dimension, a similarity measure is defined to compare two videos. This similarity/dissimilarity measure could be simple Euclidean or cosine similarity as in canonical correlation. Our canonical correlation kernel is based on the principal angles between the

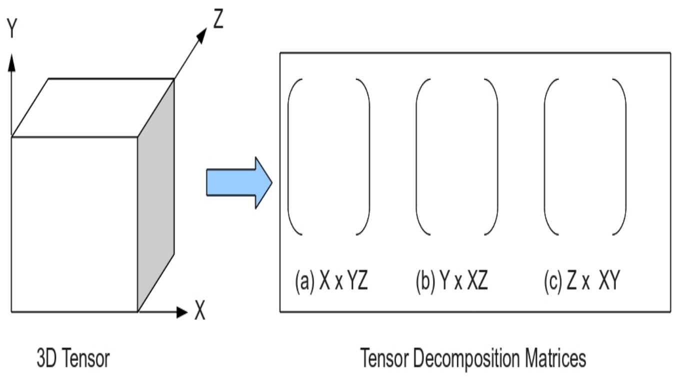


Figure 5.2: Flattening of Tensor representation. A third order tensor is flattened into 3 second order tensors (2D matrices) X×YZ, Y×XZ and Z×XY. Here, X, Y are spatial axes and Z is a time axis. The first tensor matrix (X×YZ) (a) is obtained by keeping X fixed and flattening YZ into a single dimension. The other decomposition matrices are also obtained in a similar way.

points on a manifold [18] resulting from the tensor decomposition of two videos. This is a simple, yet powerful generalization of the similarity computation in a canonical correlation analysis.

Canonical correlation analysis [18] has been successfully applied for image set comparison in robust object recognition, and later extended for action recognition [83, 94]. It gives the linear relationships between two set of random variables (or two matrices) in terms of correlations. Given two matrices, CCA finds two projections one for each of the matrices such that the correlation between projections of the matrices is maximized. The correlation constants (correlations) between the projections of the matrices gives a similarity measure between original matrices. For any given two matrices, its canonical correlations can be computed in two ways. One is from the singular values of given matrices and other is using the eigendecomposition of the matrix which is obtained by pre and post multiplying the cross-covariance matrix by the inverse square root of the covariance matrix of given matrices. In this work, we have followed the SVD based solution [18]. For the matrices P and Q of dimension $n \times d_1$ and $n \times d_2$ $(n > min\{d_1, d_2\})$, we denote their orthonormal matrices as $\bar{P}$ and $\bar{Q}$ of dimension $n \times rank(P)$ and $n \times rank(Q)$. The d correlation constants $(\rho_1, \rho_2, \ldots, \rho_d)$, where $d = min\{rank(P), rank(Q)\}$, for the matrices P and Q, can be computed as the singular values of the matrix $\bar{P}^T\bar{Q}$.

Our definition of the canonical correlation kernel is motivated by the need to use tensorial representation framework along with a discriminative classifier like SVM. Diverse set of features can also be used for tensorial representation.

5.3 Canonical Correlation Kernels

We start with a tensor representation of the video volume. To begin with, let us consider that elements of the tensor are the pixel (or intensity) values. However, this representation can easily scale to other dense representations, where more powerful feature descriptors (e.g., SIFT) are used to encode the local information. Our objective is to define an effective similarity measure that can scale for multiple features. We achieve this with the help of a canonical correlation kernel defined based on the canonical correlation analysis [18], which has already been used in action recognition [83]. Our kernel is far more effective than the discriminatively selected (using boosting) correlation coefficients used in [83] for comparing videos, as can be seen in Section 5.4.

The use of CCA for defining a kernel for the action recognition task is motivated by multiple factors: (i) Canonical correlation allows us to define a kernel, which can be used in a maximum margin discriminative framework like SVM, and used for a seamless combination of multiple features and representations. For example, in Section 5.4, we show that Bag of Words based methods can be used along with the tensor based ones (ii) TCCA based method had shown some success in recognizing actions in the past [83]. (iii) similarity of the videos can be computed by projecting them over a linear manifold in CCA and a nonlinear manifold in KCCA in the same framework. (iv) correlation coefficients measure the similarity more intuitively compared to the popular distance functions like Euclidean. We find this to be very effective for comparing videos. Note that the principal angle based similarity computation (cosine similarity) is widely used in the text domain and has proven to be more appropriate than the L^p norms in a wide range of problems.

5.3.1 Canonical Correlation Kernel (CCK)

Given two random vectors $\mathbf{x}$ and $\mathbf{y}$, canonical correlation analysis measures the similarity by finding the correlation of these two vectors after a set of linear transformations. Assume $\mathbf{x}$ gets linearly transformed with $\mathbf{u}$ as $\mathbf{x}' = \mathbf{u}^t\mathbf{x}$ and $\mathbf{y}$ as $\mathbf{y}' = \mathbf{v}^t\mathbf{y}$, then canonical correlation is defined as the maximum possible correlation over *all* possible transformations $\mathbf{u}$ and $\mathbf{v}$. For a video recognition problem, this implies that the video is getting transformed (or features are getting extracted) such that the correlation in the most appropriate feature space is maximized. Thus, we simultaneously learn the most appropriate features and the similarity in that feature space. The method remains same irrespective of whether $\mathbf{x}$ and

$\mathbf{y}$ are vectors, matrices or tensors. Correlation coefficients ρ_i measures the similarity in the projected space as the cosine of the angles between the linear manifold.

A limitation of the above is the use of linear transformation while extracting features. Wolf and Shashua [173] addressed this problem by kernelizing the canonical correlation by defining the transformation in an implicit feature space. For example, a kernel $\kappa(\mathbf{x}, \mathbf{u}) = \phi(\mathbf{x})^T\phi(\mathbf{u})$ does the feature extraction by finding a manifold instead of a linear subspace. This generalizes the classical canonical correlation and provides a mechanism for extracting a richer set of features. In our implementation, we use an exponential kernel (for KCCA) as $\kappa(\mathbf{x}, \mathbf{y}) = e^{-\beta d(\mathbf{x},\mathbf{y})}$. Here also, similarity is measured using correlation coefficients ρ'_i computed for nonlinear manifolds. For our experiments, we use the similarities computed over both linear as well as nonlinear manifolds i.e., ρ_i and ρ'_i, simultaneously.

Our video representation is essentially a third order tensor. While working with pixel values, we scale the video to a smaller size, as explained in the experimental section. While using feature representations (e.g. SIFT descriptors), we sample the image/frame further and compute the features at smaller set of grid points. A tensor thus obtained from pixel values or feature descriptors is first flattened to obtain a matrix. Here, we use three kinds of flattening corresponding to the spatial and time axis. This is done similarly to many of the previous works [83, 94, 95]. This flattening is explained in Figure 5.2. For a given video of size $l \times m \times n$, where n is the number of frames and $l \times m$ is the size of each frame, the flattening corresponding to the time axis leads to a matrix of size $d_1 \times n$. Each column in the matrix corresponds to a video frame and the number of columns is same as the number of frames in the video/tensor. Here d_1 is the length of the feature descriptor obtained from each frame; for pixel values, this is equal to $l \cdot m$. The computation of feature descriptors is explained in Section 5.3.2. Similarities between two videos is then computed with the help of canonical correlations coefficients between the corresponding flattened matrices, which are the principal angles between the subspaces.

Given two videos $\mathcal{V}_1$ and $\mathcal{V}_2$ denote their i^{th} $(i = 1, 2, 3)$ flatten matrices as V_1^i, V_2^i, we define the canonical correlation kernel corresponding to the i^{th} flatten matrices as the sum of all the correlation coefficients obtained from both CCA and KCCA over V_1^i and V_2^i. i.e.,

$$K^{'}(V_1^i, V_2^i) = \sum_{j=1}^{d} \rho_j + \sum_{j=1}^{d} \rho'_j \tag{5.1}$$

where, $d = min\{rank(V_1^i), rank(V_2^i)\}$ and ρ_i, ρ'_i are the correlation coefficients obtained from CCA and KCCA over V_1^i and V_2^i. We flatten the third order tensor video into three second order matrices (tensor). Our final canonical correlation kernel between two videos is the sum of canonical correlation kernels obtained from three flattening matrices. i.e.,

$$CCK(\mathcal{V}_1, \mathcal{V}_2) = \sum_{i=1}^{3} K^{'}(V_1^i, V_2^i) \tag{5.2}$$

5.3.2 Feature Kernels

CCK can be computed for pixel values and other extracted features. In both cases, the procedure of computation remains the same. For every feature, we first compute its feature matrices corresponding to three flattenings then CCK is computed over these matrices. For a given feature, its feature matrix corresponding to time axis flattening over a video of n frames is an $d \times n$ matrix. The i^{th} column in this matrix represents the feature descriptor obtained from the i^{th} frame and d is the descriptor length. In addition to the pixel values, we use the features SIFT [93], HOG (histograms of oriented gradients) [36], MBH (motion boundary histogram) [37] and HOF (histograms of optical flow) [87]. Among these descriptors, HOG and HOF have shown to give good results for action recognition [171]. HOG captures static local information whereas HOF captures motion information. Dalal *et al.* [37] proposed MBH for human detection, it captures the relative motion between the pixels. In our experiments, these feature descriptors are extracted as follows,

- For **SIFT**, we divide a frame into fixed grids, where the size of each grid is 4×4, and SIFT is extracted at each grid location. The final descriptor for the corresponding frame is obtained by concatenating all the SIFT descriptors.

- For **HOG**, we compute HOG descriptor using a window of size 4×4 and a bin size 9. Concatenation of all the local histograms is taken as the final representation.

- For **MBH**, similar to SIFT, we divide a frame into grids of size 4×4 and MBH is computed at each of the grid locations. We take bin size of 8 and patch size 32.

- For **HOF**, we divide the frame into grids of size 6×6 and HOF descriptors are computed at each grid locations with a neighborhood size of 24×24 and concatenation of all these descriptors is taken as the final feature descriptor. Here, we quantize the orientations into 9 bins.

All the descriptors are normalized to zero mean and unit variance.

CCK is reasonably insensitive to various factors such as temporal misalignment, scale variations and background variations. Temporal misalignment comes from the affine invariance property of CCA. Since CCK is computed after normalizing the videos, scale variation is also taken care of. When the background changes significantly, insensitivity depends on the features used. In our experience, CCK defined over HOF and MBH is practically insensitive to this.

5.3.3 Classifier

We use a support vector machine (SVM) classifier. Feature kernels are combined [48] by giving equal weights to all the kernels or by giving high weight to one kernel and zero weightage to all other kernels. One can also use multiple kernel learning (MKL) [27] for combing the feature kernels. If $\kappa_j(\cdot,\cdot)$ is the canonical correlation kernel computed using j^{th} descriptor, then the final kernel is obtained as the linear combination of all the kernels

$$\kappa(\mathbf{x},\mathbf{y}) = \sum_j d_j \kappa_j(\mathbf{x},\mathbf{y}) \tag{5.3}$$

If all the d_js are equal then the final kernel κ will be the simple average of given kernels. In all our experiments, we use libsvm [25] package for the SVM classifier.

5.4 Experiments

This section presents a detailed evaluation of our proposed kernel (CCK). CCK is based on canonical correlation analysis and can be used to compare the videos for action recognition. We evaluate various components of the proposed kernel to justify our choices. We compare our results with the previously published works.

We report our results on four publicly available standard action datasets: Cambridge gesture, KTH human action, Youtube and UCF sports action datasets. Some sample frames from the datasets are shown in Figure 5.3. For multiclass classification, in all our experiments, we use a one-vs-rest SVM classifier and select the class with the highest score. We perform the experiments on raw video representation using pixel values as well as on feature representations. For combining the feature representations, we use the simple weighted scheme as discussed in Section 5.3.3. The kernel matrices obtained over the given datasets are positive definite in all our experiments. For the given four datasets, average accuracy over all the classes is reported as the performance measure.

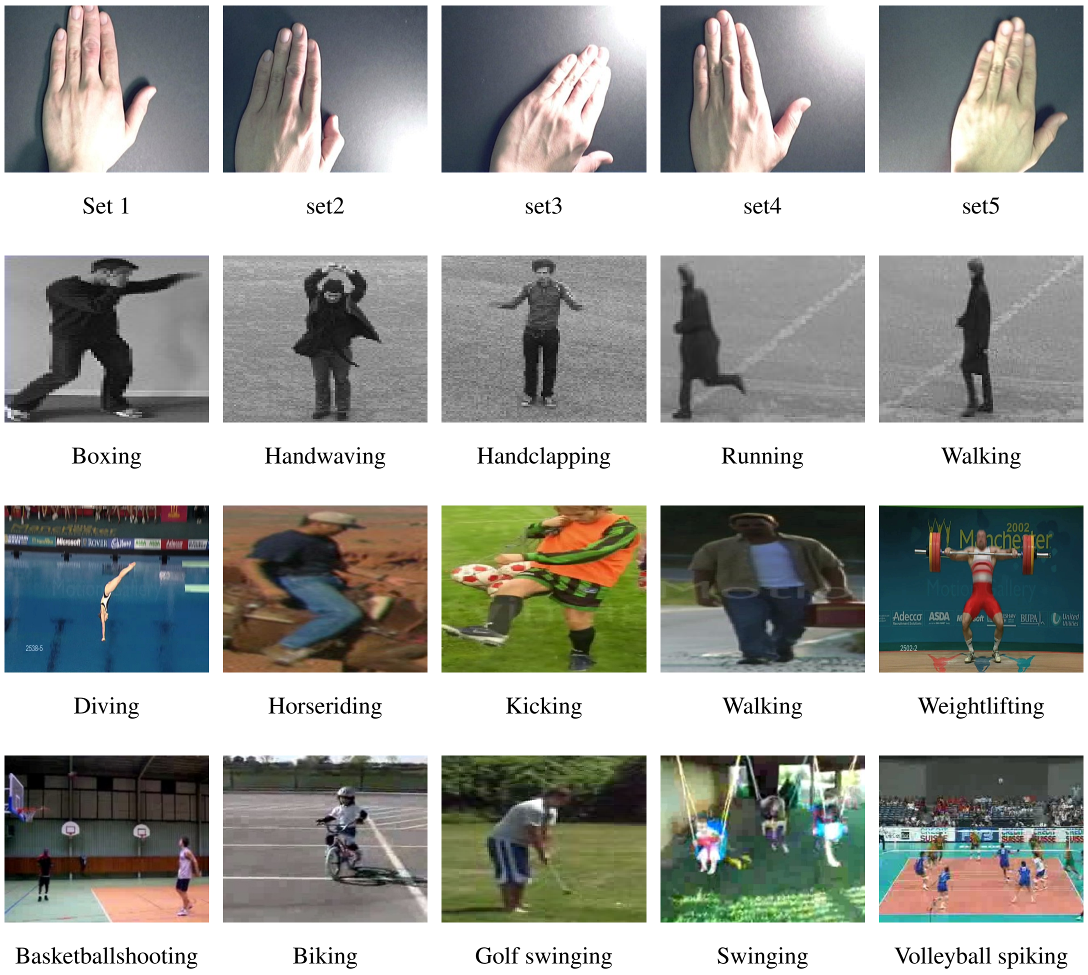


Figure 5.3: Some sample frames from video sequences on Cambridge Gesture (first row), KTH (second row), UCF (third row) and Youtube (fourth row) datasets. For cambridge, background is uniform in most of the sequences. For KTH, background is homogenous and static. UCF and youtube videos have non uniform background. Youtube dataset has large variations in camera motion

5.4.1 Datasets and Experimental Setting

Cambridge Gesture Dataset: The cambridge gesture dataset [82][1] consists of 900 video sequences belonging to nine action classes. These videos are captured using five different lighting conditions from two subjects. The data is divided into five sets (one for each illumination setting), where each set contains a total of 180 video sequences. we use 10 random videos from each class in set5 (plain illumination setting) for training and all videos from the remaining sets (set1, set2, set3 and set4) for

[1] ftp://mi.eng.cam.ac.uk/pub/CamGesData

testing as reported in [83, 94, 95]. For our experimental setup, we use tensors of size 20×20×20, where 20 frames from each sequence were obtained by uniform sampling. Each frame in the sequence is resized to 20×20. Classifier is trained using the randomly chosen 90 videos. Accuracies are reported by taking the average over ten trials.

KTH Dataset: The KTH dataset [142][2] contains 600 videos from 6 human action classes. In most of the videos, background is homogeneous and static. Each type of action is performed by 25 different actors in indoor and outdoor settings. We extract the human actions by following the procedure used in [94, 95]. Our tensor formulation is identical to [82, 94] by constructing tensors of size 20×20×32. Experiments are carried out using leave one out cross-validation, which is performed by dividing the dataset into 25 folds (each fold containing 24 videos of the same person).

UCF Sports Action Dataset: The UCF sports action dataset [97][3] contains 150 videos from 10 different sports action classes. The number of videos for each class varies from 6 to 22. This dataset has large intra-class variability. Similar to [94], we use tensors of size 32×32×64 where each frame is resized to 32×32. Similar to the KTH dataset, experiments are performed using leave-one-out cross-validation, where each video is taken as a separate fold and the remaining 149 videos are used for training.

Youtube Dataset: The youtube dataset [91][4] contains 1168 videos from 11 different classes. It is one of the challenging datasets. This is mainly due to significant camera motion, viewpoint transitions, varying illumination conditions and cluttered backgrounds in the videos. Videos for each class are divided into 25 folds based on the persons performing that action. We use tensors of size 32×32×64, where each frame is resized to 32×32. Accuracies are reported using leave one out cross validation over the predefined 25 folds.

5.4.2 Results and Discussions

The methods we compare with our proposed method, can be divided into two categories. Methods that uses tensor decomposition representation for videos [83, 94, 95] and which use the feature representations [64, 88, 170].

5.4.2.0.1 CCK on individual Features We report the individual feature accuracies using CCK in Table 5.1. We compare the results with dense trajectories [170] feature kernels, which are computed

[2]http://www.nada.kth.se/cvap/actions/
[3]http://www.cs.ucf.edu/vision/public_html
[4]http://www.cs.ucf.edu/~liujg/YouTube_Action_dataset.html

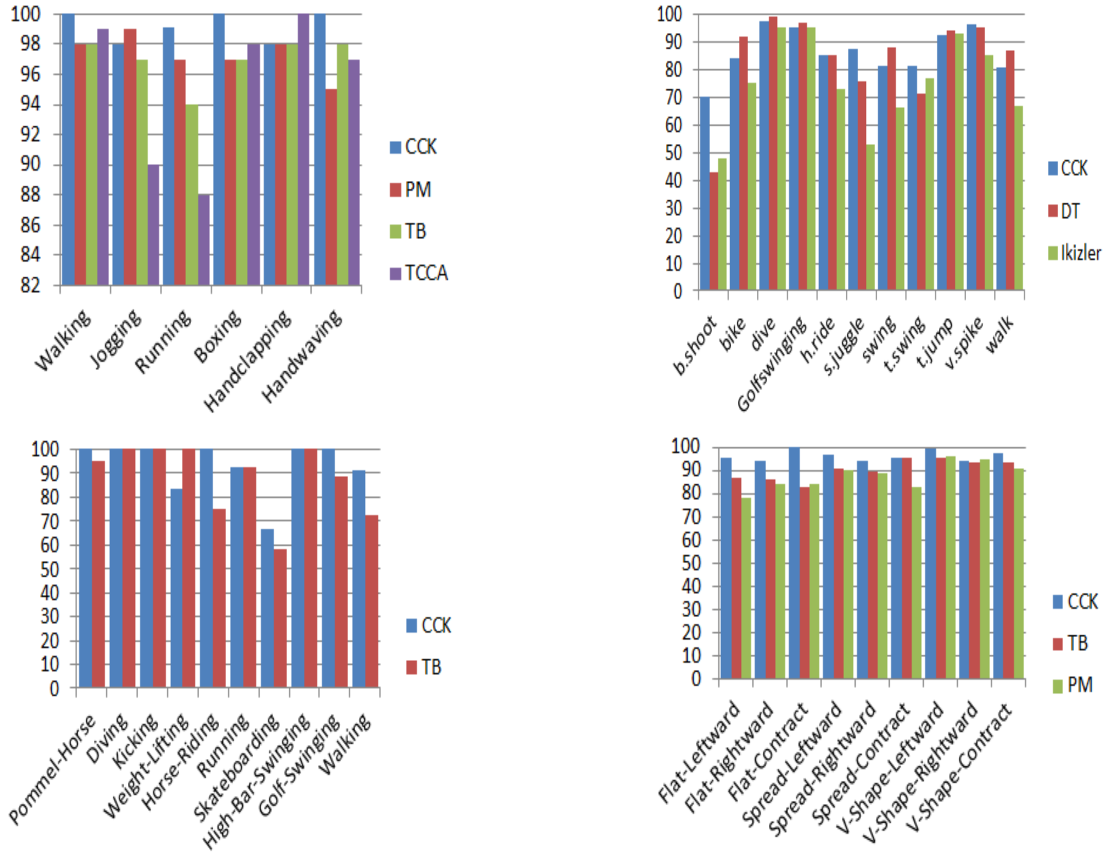


Figure 5.4: Classwise accuracies on KTH and Youtube (in first row), UCF and Cambridge (in second row). For KTH, we compare with Tangent Bundle (TB) [94], Product Manifold (PM) [95] and TCCA [83]. For Youtube, we compare with Dense Trajectories (DT) [170] and Ikizler [64]. For UCF, we compare with Tangent Bundle (TB) [94]. For Cambridge, we compare with Tangent Bundle (TB) [94] and Product Manifold (PM) [95].

using χ^2 kernel [87] and kernels are combined in a multichannel approach similar to [164]. CCK perform better than the dense trajectories for HOG and HOF over all the four datasets. This indicates the strength of CCK, the superiority comes from the temporal context it embeds from the videos. Thus, it best suits for the action recognition task over other kernels.

5.4.2.0.2 CCK using Pixel values Pixel value accuracies are reported in Table 5.2. Using pixel values alone, we achieve a significant improvement of 2.1% and 5.3% for Cambridge and UCF datasets over previous work. We report 97.5% on KTH dataset, which is an improvement of 0.5% over [94]. The CCK using pixel values is comparable to state-of-the-art [170] on Youtube dataset. This indicates

	Cambridge		UCF		KTH		Youtube	
	CCK	DT [170]	CCK	DT [170]	CCK	DT [170]	CCK	DT [170]
Pixel Values	93.1	-	93.5	-	97.5	-	82.5	-
HOG	89.0	-	**83.8**	**83.8**	**98.3**	86.5	**83.2**	74.5
SIFT	95.1	-	85.7	-	98.6	-	79.1	-
HOF	95.2	-	**81.5**	77.6	**94.3**	93.2	**80.4**	72.8
MBH	75.1	-	80.4	**84.8**	**98.9**	95.0	80.1	**83.9**
Combined	96.4	-	**93.5**	88.2	**98.9**	94.2	**86.3**	84.2

Table 5.1: Comparison of our proposed canonical correlation kernel (CCK) with DT (Dense trajectories) [170] over different feature descriptors on Cambridge, UCF, KTH and Youtube datasets. Results are reported on pixel values, HOG, SIFT, HOF and MBH. For each dataset, we report average accuracy over all the classes. Final accuracies after combining the features are also displayed. Dense trajectories [170] have not used pixel values and SIFT. Accuracies are reported in %.

that pixel values alone are good enough for CCK to get the better results. The main reason behind this is that, for tensorial representation, the actions in the videos are well represented using pixel values compared to other features. This gives the superiority of CCK.

5.4.2.0.3 CCK using multiple features We combine the feature descriptors to enhance the accuracy further. Features are combined using a simple weighted scheme as discussed in Section 5.3.3. We report the combined feature accuracies using the weighted scheme in Table 5.2, which compares our results with the previous methods. We achieve an improvement of 5.4%, 5.3%, 1.9% and 2.1% over Cambridge, UCF, KTH and Youtube datasets. This indicates that videos can be well represented using multiple features in a tensorial representation framework. One can also use MKL [84] for combining the feature descriptors.

Accuracy over the combination of canonical correlation feature kernels and DT (dense trajectory) [170] feature kernels are shown in Table 5.2. It further improved the accuracy over all the datasets. This indicates that CCK can be easily integrated with other features such as Bag of words histograms to further improve accuracy. We also compare the classwise accuracies for all the datasets with other methods in Figure 5.4. On KTH, CCK gives the best results for 5 out of 6 action classes, as compared to [95].

Method	Cambridge	UCF	KTH	Youtube
TCCA [83]	82±3.5	-	95.33	-
Product Manifold [95]	88	-	97	-
Tangent Bundle [94]	91	88	97	-
Dense trajectories [170]	-	88.2	94.2	84.2
Le *et al.* [88]	-	86.5	93.9	75.8
Ikizler-Cinbis *et al.* [64]	-	-	-	75.21
Jiang Wang *et al.* [172]	-	-	93.8	-
Proposed (Using pixel values)	93.1	**93.5**	97.5	82.5
Proposed (Using multiple features)	96.4	93.5	**98.9**	86.3
Proposed (CCK feature kernels + DT feature kernels)	**97.2**	93.5	98.9	**86.6**

Table 5.2: Comparison of our proposed method with other state-of-the-art methods. Here, we give the accuracy of our proposed kernel (CCK) over simple pixel values and using multiple features (pixel values, HOG, SIFT and MBH). Accuracies over multiple features are obtained using simple weighting scheme. Finally, combined accuracy using CCK feature kernels and of DT [170] feature kernels are also reported. Accuracies are reported in %.

On Youtube, our method got the best results over 5 classes compared to [170]. For UCF, we got best results for 9 out of 10 classes and for Cambridge, we got best results for all the classes compared to the state-of-the-art.

In summary, our superiority comes from (1) The proposed CCK which embeds temporal context in the videos into similarity measure (2) Seamless fusion of multiple features into tensor representation.

5.5 Conclusions

In this work, we have introduced the canonical correlation kernel (CCK), which enables the comparison of videos in a kernel framework. This kernel function works well for action recognition as it embeds the temporal context in the videos. We have also shown that multiple features can seamlessly integrate into CCK to further enhance recognition performance. We hope our work opens up scope for a class of action recognition algorithms which use, tensor representation, multiple feature description and use a discriminatively max margin classification.

5.6 Recent Advances in CCA

In this section, we discuss few recent advances in Canonical Correlation Analysis (CCA).

5.6.1 Deep Canonical Correlation Analysis (DCCA)

CCA is a standard statistical technique for finding linear relations between two random vectors. It gives the maximum correlation between the given random vectors in the linear space. Let $(X_1, X_2) \in \mathbb{R}^{n_1 \times n_2}$ be the two random vectors (data views). For the given data views (X_1, X_2), the CCA finds a pair of linear projections (w_1, w_2), such that (w_1X_1, w_2X_2) are maximally correlated. It can be expressed as

$$(w_1^*, w_2^*) = \underset{(w_1,w_2)}{\operatorname{argmax}}\, corr(w_1X_1, w_1X_1) \tag{5.4}$$

Let Σ_{11} and Σ_{22} are the covariances for the random vector X_1 and X_2 with a cross-covariance Σ_{12}. Define $C = \Sigma_{11}^{-1/2}\Sigma_{12}\Sigma_{22}^{-1/2}$, and let $P_k = [p_1, p_2, \ldots, p_k]$ and $Q_k = [q_1, q_2, \ldots, q_k]$, containing first k left and right singular vectors of C (p_j is the j^{th} singular vector of C from left and q_j is the j^{th} singular vector of C from right). Then the sum of singular values of C gives the total maximum correlation between X_1 and X_2. It can also be defined as

$$corr(X_1, X_2) = trace(C^TC) = \|C\|_{tr} \tag{5.5}$$

Kernel canonical correlation analysis (KCCA) is an extension of CCA which gives the maximum correlation between the two views in the non-linear projection space. If H_1, H_2 are the reproducing kernel hilbert spaces of functions on $\mathbb{R}^{n_1}$, $\mathbb{R}^{n_2}$ and κ_1, κ_2 are their associated kernels, then the optimal non-linear projections $f_1^* \in H_1$, $f_2^* \in H_2$ are given as

$$(f_1^*, f_2^*) = \underset{(f_1 \in H_1, f_2 \in H_2)}{\operatorname{argmax}} \quad corr(f_1(X_1), f_2(X_2)) \tag{5.6}$$

For the given data, KCCA allows learning of non-linear representations. However, these representations are limited by the given fixed kernel. Also, it is a non-parametric method. For the given data, the CCK [108] allows jointly learning both linear and non-linear representations. CCA and KCCA are also used in feature learning for a single data view when another view is available in representation learning [10, 19].

Over the past few years, there has been increasing interest in deep learning methods and they have produced state-of-the-art results over a wide variety of problems. In [8], a deep canonical correlation analysis (DCCA) is proposed for learning non-linear correlations between the given random vectors, which have the maximum correlation. These representations are learned using a deep learning framework. It gives the highly correlated representations compared to CCA and KCCA. Deep learning methods have been successful in learning representations of a single data view [10,19]. Deep CCA simultaneously learns two deep non-linear mappings for two data views that are maximally correlated. Let $X_1 \in \mathbb{R}^{n_1}$ and $X_2 \in \mathbb{R}^{n_2}$ are the different instances of the two data views. The objective of Deep CCA is to jointly learn two deep networks g_1 and g_2 such that $corr(g_1(X_1), g_2(X_2))$ is as high as possible. If π_1 and π_2 are the parameters for the two networks g_1 and g_2 respectively, then their optimal parameters (π_1^*, π_2^*) are given as

$$(\pi_1^*, \pi_2^*) = \underset{(\pi_1, \pi_2)}{\operatorname{argmax}}\, corr(g_1(X_1; \pi_1), g_2(X_2; \pi_2)) \tag{5.7}$$

Here, it only learns the non-linear embeddings. Since Deep CCA is learning the non-linear representations, it can be viewed as learning a kernel for KCCA. Deep CCA is closely related to the multimodal autoencoders [112] and multimodal restricted Boltzmann machines [154], where they learn a single network connected to both the modalities (views). In Deep CCA, it learns two separate deep encodings for the two modalities. The learned encodings have the maximum possible correlation.

5.6.2 Representational Similarity in Neural Networks using Canonical Correlation

In general, CCA, KCCA and DCCA are used to compute the correlation (similarity) between different data representations. Understanding the function of neural networks is a challenging task, in particular, comparing two neural network representations is a difficult task as the structure of these representations changes vastly over the training. In [105], the authors developed a projection weighted CCA for understanding and comparing network representations across a group of CNN. Using projection weighted CCA, the authors analyzed the converged solutions in CNNs. It is demonstrated that compared to narrow networks, wider networks converge to similar solutions, and the networks which have identical topology but distinct learning rates converge to a small set of distinct solutions. The projection weighted CCA is also useful in analyzing RNN representations over training. The authors proved that RNNs exhibit bottom-up convergence and these representations vary nonlinearly. Here, the LSTMs are used for the Penn Treebank (PTB) and WikiText-2 (WT2) language modelling tasks.

5.7 CCKNet

In general, the convolution operation is used as a feature extractor in CNN. However, recently these convolution operations are replaced with other similarity measures [23] for feature extraction in CNN. In [23], DTW distance is applied as a feature extractor. For the problems like data decomposition, it outperforms standard convolutional kernels [23]. In this section, we propose CCKNet, where we applied CCK [108] [108] as a feature extractor in deep learning framework. For a given input sequence $X \in \mathbb{R}^n$, the objective is to learn a kernel (filter) $K \in \mathbb{R}^m$ such that it has the maximum correlation with X. The kernel is initialized randomly and learned using standard gradient descent methods. For the input sequence X, the CCK kernel (filter) K, is computed as

$$K = \underset{z}{\operatorname{argmax}}\, CCK(X, z) \tag{5.8}$$

where,

$$CCK(X, K) = \|C\|_{tr} + \|C_\phi\|_{tr} \tag{5.9}$$

C_ϕ is computed in the feature space ϕ corresponding to the kernel κ in KCCA. CCK is the sum of the correlations obtained from both CCA and KCCA. Denote $CCK(X,K) = CCK_1(X,K) + CCK_2(X,K)$, where $CCK_1(X,K) = \|C\|_{tr}$ and $CCK_2(X,K) = \|C_\phi\|_{tr}$.

The CCKNet contains the multiple CCK layers like convolutional layers in CNN. A CCK layer consists of multiple CCK kernels that extract meaningful features from the input. Each CCK kernel generates a correlation output by performing CCK computation between the kernel and the input sequences. In sliding window, similar to the convolutional kernels, the CCK kernels produces multiple correlation outputs. These correlation outputs are combined in the subsequent layers. These outputs capture important details in the input sequence. To learn the CCK kernel using gradient descent, we need the gradient of CCK (eq 5.9). Let the singular value decomposition of C is $C = PDQ$ and C_ϕ is $C_\phi = P_\phi D_\phi Q_\phi$. Define $\Sigma_{12} = \frac{1}{m-1} X K^T$ and $\Sigma_{22} = \frac{1}{m-1} K K^T + r_1 I$, where r_1 is a regularization constant and $m = dim(K)$ (for 1-D), then the gradient of CCK_1 is computed as [8]

	Training Time (mins)	Accuracy
DTWNet	51	97.8
FastDTWNet	32	97.6
CCKNet	68	98.1

Table 5.3: Comarison of DTWNet, FastDTWNet and CCKNet. Training time is given in minutes.

$$\frac{\partial CCK_1(X,K)}{\partial K} = \frac{1}{m}(2\nabla_{22}X + \nabla_{12}K) \tag{5.10}$$

where

$$\nabla_{12} = \Sigma_{11}^{-1/2} PQ\Sigma_{22}^{-1/2} \tag{5.11}$$

and

$$\nabla_{22} = -\frac{1}{2}\Sigma_{22}^{-1/2} Q^T DQ\Sigma_{22}^{-1/2} \tag{5.12}$$

Similarly, we can compute the gradient of CCK_2.

To validate the CCKNet, we compare it with the DTWNet [23] and FastDTWNet over the libra dataset. In the DTWNet, the DTW distance is computed between the input and kernel. In the FastDTWNet, the Fast Surrogate DTW distance is used to compute the similarity between the input and kernel. The results are given in Table 5.3. For both the DTWNet and FastDTWNet, we take 5 DTW layers. For CCKNet, we take 5 CCK layers. From the results, we can observe that, in terms of accuracy, the CCKNet is performing well compared to the DTWNet and FastDTWNet. However, it is taking more time for the training. This is mainly due to the computational time needed for computing both the CCA and KCCA.

Chapter 6

Surrogate Loss Networks for Semantic Segmentation and Human Pose Estimation

Supervised training of deep neural networks involves minimizing the loss function over the given training data. However, in many domains, we are interested in performing well on metrics specific to the application. Like average precision (AP) from information retrieval, intersection-over-union (IoU) which is used in image labelling, normalized discounted cumulative gain (NDCG) which is popular in ranking and percentage of correct parts (PCP)/percentage of correct keypoints (PCKh) which are used in pose estimation for measuring the accuracy of detected joint locations. However, these metrics are inherently non-differentiable by nature due to their discrete form and non-decomposable behavior. In this chapter, we propose a novel method to automatically learn a surrogate loss function that approximates discrete and non-decomposable metrics - in particular, the IoU, PCP and PCKh loss and is hence better suited for providing stronger performance. The proposed loss can be directly learned over any base network. For IoU loss, we validated our method over semantic segmentation models [11, 92] on the PASCAL VOC and Cityscapes datasets. On the PASCAL VOC, we got an improvement of 3.79 in IoU over cross-entropy loss. For PCP loss, we validated our method over LSP dataset and for the PCKh loss, we validated our method over MPII dataset. For the PCP and PCKh loss, we applied our method over DeepPose [163] network. Our results on this work show consistent improvement over baseline methods. On LSP dataset, we got an improvement of 2.1% on the PCP measure and on MPII dataset, we got an improvement of 1.8% on the PCKh measure compared to the regression loss.

6.1 Introduction

For the last few years, Convolutional Neural Networks (CNNs) have shown to be very effective for a wide variety of problems, including classification [11], semantic segmentation [92, 133, 182], and pose estimation [24, 163]. Loss functions play an important role in the performance of CNNs. The loss function is used to guide the training process of CNNs, and training involves computing the gradient of the loss function with respect to the parameters of the model. For the given problem, the network is trained to minimize the training error, which is measured by the given loss function. The performance of the CNN model is typically evaluated using the training and test error (or equivalent accuracy). However, in many problems which have gained importance in recent times, we are interested that the final learned model performs well on metrics specific to the given application. Such metrics are, however, often non-differentiable with respect to the output of the network. Examples of such metrics include average precision (AP), intersection-over-union (IoU), normalized discounted cumulative gain (NDCG), and percentage of corrected parts (PCP)/percentage of corrected keypoints (PCK)h. Many of these metrics are discrete and non-decomposable (which are not simple sums over the network outputs), and are not readily differentiable. In such scenarios, the loss used to train the network is different (e.g., cross-entropy in semantic segmentation) from the actual performance metric (e.g., IoU in semantic segmentation). In this work, we explore the non-differentiable behavior of discrete and non-decomposable metrics - in particular, IoU, PCP and PCKh - and propose differentiable surrogate approximations using neural networks. IoU is a discrete metric as it is defined over pixel counts which are discrete in nature, while PCP and PCKh measures are non-decomposable metrics.

CNNs for semantic segmentation typically use the cross-entropy loss for training the network, given as: $L_{CE} = -\frac{1}{N}\sum_{i=1}^{c} y_i \log(y_i^{'})$, where y_i is the ground truth, $y_i^{'}$ is the output class score predicted from the network for the pixel i, c is the total number of classes, and N is the number of pixels in the image. While the vision community uses pixel-wise cross-entropy loss to train networks for a semantic segmentation task, the standard performance measure that is used for evaluation is intersection-over-union (IoU). For an object present in a given image, the IoU measure gives the overlap between the predicted region and the ground truth region of the object. The IoU measure is popularly used for semantic segmentation due to its ability to handle class imbalances [127].

Consider the example in Figure 6.1. For the given image, two segmentation results are given in Figures 6.1 (c) and 6.1 (d). The segmentation results in Figure 6.1 (c) classifies all the pixels as background.

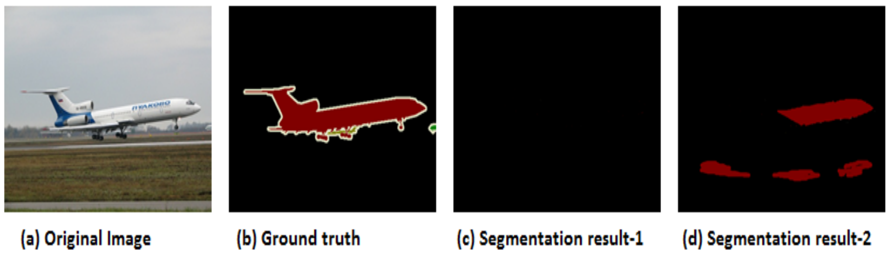


Figure 6.1: Illustration of cross entropy loss for semantic segmentation: (a) Original image (b) Ground truth segmentation (c) All pixels predicted as background (d) Few pixels correctly predicted as foreground.

The segmentation result in Figure 6.1 (d) correctly classifies few object pixels and classifies few background pixels as object pixels. These two results have the same cross-entropy loss. However, the IoU measure for the segmentation result in Figure 6.1 (c) is zero, clearly indicating a gap between the loss function used in training the network and the metric used for measuring the performance of the network. The key limitation of not using IoU directly as a loss function in CNNs is its non-differentiability [127]. While there have been a few recent attempts to address this issue (described in Section 6.2) for IoU, these methods have their limitations and are handcrafted approximations.

Human pose estimation [24, 163] is another example where there is a gap between the loss function used in training the network and the metric used for measuring the performance of the network. Human pose estimation aims to localize each human part (key points). CNNs for pose estimation typically use the regression loss for training the network, given as: $L_{RL} = \sum_1^k ||y_i - y_i'||^2$, where y_i is the ground truth location for i^{th} part, y_i' is the output location predicted by the network for the i^{th} part, and k is the number of human parts to be localized. It uses regression loss over the locations of body parts to train the network. During training, regression loss enforces the network to detect the keypoint joint locations. This helps to improve the training accuracy at the outset. However, if we observe any pose, any pixel location in a small neighbourhood around the true joint location is also considered the correct joint location. Due to this, the accuracy of pose estimation methods (networks) is measured using metrics such as percentage of correct parts (PCP) [163], PCPm [24], and percentage of correct keypoints (PCK) [127]. PCP measures the detection rate of limbs. In PCP, a limb is considered localized correctly if the distance between the predicted and actual joint locations is at most half of the limb length. It penalizes shorter limbs. PCK is the most widely used performance measure. It considers a detected joint to be localized correctly if the distance between the predicted and the true joint is within a certain threshold. The threshold is defined as 50% of the head segment length. The corresponding metric is defined as PCKh. The PCKh measure is popularly used for human pose estimation due to its ability to evaluate joint positions for varying torso diameters. Since we are considering the locations around the true joint location as correct predictions, training the network to detect exact joint locations as given in the ground truth may not yield the best test performance when measured by such metrics as PCP and PCKh. This suggests that regression loss used in contemporary deep pose networks may not be the most appropriate loss function to train such models. The PCKh measure is an example of an ideal loss function for training deep pose networks. However, both PCP and PCKh measures cannot be used as a loss function directly in deep pose networks due to their non-differentiable nature, and the gradient lacking a closed-form representation. So far to the best of our knowledge, there has been no effort to address this issue.

In this chapter, we propose a novel method to automatically learn a surrogate loss function for non-differentiable metrics such as IoU, PCP and PCKh that is aimed at attaining good performance. In particular, we use a neural network to approximate these metrics, thereby providing a differentiable approximation that can be directly used with CNNs. We call the corresponding proposed loss functions as the $NeuroIoU$ (for semantic segmentation), $NeuroPCP$ and $NeuroPCKh$ (for pose estimation) loss functions, which can be integrated with any deep network. To the best of our knowledge, this is the first such work that attempts to learn a loss function for this purpose. Further, there has been no such effort for human pose estimation to the best of our knowledge. For $NeuroIoU$, we validated our method by integrating the $NeuroIoU$ loss in the FCN [92], SegNet [11] and UNet [133] models, and evaluated their performance on the PASCAL VOC and Cityscapes datasets. For $NeuroPCP$ and $NeuroPCKh$, we integrated the corresponding loss in the popular DeepPose network [163] and evaluated our performance on the LSP and MPII datasets. Our results show promise, and a consistent increase in the performance across all models on all the datasets.

The remainder of this chapter is organized as follows. Section 6.2 reviews earlier related work, including prior work on surrogate loss functions for semantic segmentation. The proposed surrogate loss for semantic segmentation and human pose estimation, as well as the methodology to train CNNs using the proposed loss, are presented in Section 6.4. Experiments and results are discussed in Section 6.5, followed by concluding remarks in Section 6.6.

6.2 Related Work

Existing efforts that have attempted to use surrogate loss functions to train deep neural networks have largely focused on forms of functions that can be easily optimized [52, 89, 102, 150]. For example, in [102], McAllester *et al.* show how to compute the gradient of complex non-differentiable loss functions for linear models. In [150], this is extended to the non-linear case, in particular, to maximize average precision for ranking problems using direct loss minimization techniques. These methods cannot be directly used in a wide range of learning architectures and problems without significant effort. Besides, these methods are often problem-specific, such as [150], and cannot be used for other problems that use CNNs.

Non-decomposable loss functions have been previously used in the context of maximum-margin methods [2, 122, 168, 180]. In structured prediction problems, in which the output is multi-dimensional, these efforts have attempted to develop maximum-margin training methods capable of minimizing an upper bound on non-decomposable loss functions. Standard learning in this paradigm involves updating the parameters such that the model assigns a higher score to the ground truth output than to any other output. This is typically encoded by a constraint, enforcing that the ground truth score should be higher than that of a selected, contrastive output. The latter is defined as the result of inference performed using a modified score function that combines the model score and the task loss, representing the metric that we care about for the application. This modified scoring function encodes that we should penalize higher scoring configurations that are inferior in terms of task loss. These methods mainly try to incorporate complex discrete loss functions into maximum-margin methods [2, 122, 168, 180], and only optimize an upper bound. Our focus in this work is on deep neural network models.

6.2.0.0.1 Semantic Segmentation. State-of-the-art models for semantic segmentation largely rely on fully convolutional layers in the architectures [11, 92, 133, 182]. In these models, for a given input, we obtain class scores for each pixel, and the class scores are mapped to probabilities using a softmax layer. These probabilities also capture the likelihood of the pixels being the foreground (object/class) or background. In all these models, the cross-entropy loss function is applied over this probability map pixel-wise. The cross-entropy loss is closely tied to the overall classification accuracy. If the number of examples for foreground and background are balanced, then the cross-entropy loss works well. However, in a typical object category segmentation, these may not be balanced, thus raising questions on whether cross-entropy loss may be the best choice for loss functions in the object category segmentation task.

As already mentioned, the IoU loss cannot be used in deep networks due to its non-differentiability [127]. In recent years, a few approximations to the IoU measure have been proposed for various applications. These methods [2, 114, 121, 129, 159] attempt different techniques to optimize the IoU measure in the given problem. In [129], the IoU measure is optimized specifically for the object detection and localization problem using joint kernel maps in a SVM context. They can only optimize over bounding boxes of the object and not over full pixel-wise segmentation. In [159], structured Markov Random Field models are used for the same purpose. In [114], special purpose message passing algorithms are used for optimization. [114] provides a Bayesian framework for optimizing the IoU, in particular, an approximate algorithm is proposed using parametric linear programming. In [121], a framework for optimizing Expected IoU (EIOU) is proposed. In [2], instead of directly optimizing the model with respect to IoU, the authors select a few candidate segmentations for optimization. However, in contrast to the proposed work, all these methods do not provide a differentiable approximation to the IoU measure, which can be used for directly training contemporary deep learning models. In particular, all these methods [2, 114, 121, 129, 159] optimize the IoU measure specific to certain problems and are not generalizable across problems. In this work, we seek to obtain a differentiable approximation of IoU that can be applied across problems and architectures. We also provide us with gradients of the measure that we finally seek to maximize.

The work closest to ours in semantic segmentation is [127], which provides a differentiable approximation of the IoU measure, and hence can be used in training deep learning networks. Rahman and Wang presented a handcrafted approximation of the IoU measure in [127]. For a given image I, let $V = \{1, \ldots, m\}$ be the set of all its pixels and X be its probability map obtained from the network. Let $Y = \{0, 1\}^V$ be the groundtruth of I, where 0 represents background and 1 represents foreground (object). Then, the approximated IoU measure defined in [127] is given as $IoU - Appx = \dfrac{\sum_v X_v * Y_v}{\sum_v X_v + Y_v - X_v * Y_v}$. The authors showed that this a differentiable function. For the given image, the numerator in $IoU - Appx$ is the sum of true positive pixel probabilities. The value in the denominator can be simplified as the sum of number of object pixels in the ground truth and false positive pixel probabilities. We can observe that this expression approximates the IoU measure well if the probabilities of background pixels are near zero and probabilities of object pixels are near 1, which is not always practical. In this work, we instead propose a method to directly learn the loss function corresponding to good IoU performance for the given problem. We also compare the performance of the proposed method with Rahman and Wang's approach [127] to corroborate this claim. The proposed loss can be integrated with any existing deep semantic segmentation network.

In another related work [15], Berman *et al.* presented a method for direct optimization of the mean intersection over union loss (Jacard loss) in the context of semantic image segmentation in deep networks. It is based on the convex Lovasz extension of submodular losses. The authors developed a specialized optimization method based on efficient computation of the proximal operator of the Lovász hinge, yielding reliably faster and more stable optimization than alternatives. The loss is shown to perform better than cross entropy loss in the context of semantic segmentation. They specially designed the optimization technique for IoU loss. However, it is practically very difficult to extend their technique to other non-differentiable loss functions which do not satisfy the submodular property, like the PCP and PCKh measures in pose estimation [9, 41]. The PCP and PCKh measures do not have a closed form expression. The main focus of this work is to propose a generalized framework that enables us to use non-differentiable performance measures in CNN networks. The proposed surrogate neural network approach can be used to find a differentiable approximation for any non-differentiable performance measure.

6.2.0.0.2 Human Pose Estimation. State-of-the-art performance on the task of human pose estimation has made significant progress in recent years. This has been largely due to the success of using various CNN architectures for the task [24, 77, 161, 163, 176]. In pose estimation, the task is to find the joint locations of body parts. In a widely used recent work [163], the problem is formulated as a direct regression problem over joint locations. Here, the network output is the joint locations for the body parts. In [24, 77, 161, 176], the network output is a discrete heatmap instead of continuous regression. A heatmap predicts the probability of the joint occurring at each pixel. In these approaches [24, 77, 161, 176], the model is trained by minimizing the Mean Squared Error (MSE) between the predicted heatmap and the target heatmap. Both these types of models use either standard (MSE) or other variants of regression loss. However, to evaluate the trained model in any of these methods, the standard performance measures used include PCP [41], PCK [141], PCKh [9]. These performance measures cannot be directly used for training the network due to their non-differentiable nature. Clearly, there is a gap between the loss function and performance measure for state-of-the-art networks in pose estimation. To the best of our knowledge, the proposed approach is the first work in the direction of bridging the gap between loss functions and performance measures in deep pose networks.

6.3 Surrogate Loss Networks

This section explains our overall idea of approximating loss functions and their derivatives using the surrogate neural networks (NN)s. We present the proposed framework to train deep neural networks with approximated gradients obtained from the surrogate neural network. Then, demonstrate its application to semantic segmentation and human pose estimation.

Neural networks have established themselves as very good function approximators over the years [46, 58, 59, 66, 158] with as few as one hidden layer. In this work, we are interested in the computation of the derivatives of approximated functions (in particular, performance measure functions, which can be used to train a base network). Fortunately, neural networks can also be extended for the approximation of derivatives [58, 59]. If we need to approximate a function $f : \mathbb{R}^N \to \mathbb{R}$ using neural networks, we need the input and output data for the given function. i.e. we need $D = \{(\mathbf{x}_i, y_i) : i = 1, \cdots, n\}$, where $\mathbf{x}_i \in \mathbb{R}^N$, $y_i \in \mathbb{R}$, $y_i = f(\mathbf{x}_i)$ and n is the number of data points. Using a single hidden layer network, for a given input $\mathbf{x}$, the function is computed as the following:

$$f(\mathbf{x}) = \sum_{i=1}^{l} \mathbf{w}^{o}_{1i} S(\mathbf{x}^T \mathbf{w}^{h}_{i}) \tag{6.1}$$

where ws are the weights of the neural network, and $S : \mathbb{R} \to \mathbb{R}$ is an activation function. We note that the corresponding first-order derivatives of f can be computed w.r.t. the input $\mathbf{x}$ (also known as backprop-to-image [151, 181]) can be computed as [58]:

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_j} = \sum_{i=1}^{l} \mathbf{w}^{op}_{1i} \mathbf{w}^{h}_{ij} S' (\mathbf{x}^T \mathbf{w}^{h}_{i}) \quad j = 1, \ldots, N \tag{6.2}$$

where $\mathbf{x} = (\mathbf{x}_1, \cdots, \mathbf{x}_j, \cdots, \mathbf{x}_N)$, l is the number of neurons in the hidden layer, $\mathbf{w}^h \in \mathbb{R}^{l \times N}$ are the weights in the hidden layer and $\mathbf{w}^{op} \in \mathbb{R}^l$ are the weights in the output layer. Any function can be approximated by a neural network represented by Eqn 6.1, and its derivative can be approximated by Eqn 6.2. It only remains to determine the weights $\mathbf{w}^h$ and $\mathbf{w}^{op}$ using the given data D. We call such a network as a *surrogate network* for the given approximated function and denote it as Ψ in this work. This is done by propagating the training data through a (potentially pre-trained) CNN and obtaining its

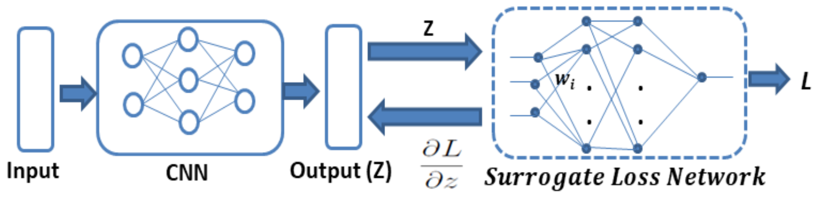


Figure 6.2: Overview of the proposed framework for training the CNN using surrogate loss network.

outputs, which act as inputs to the Ψ. Then it is trained using the expected performance measure values computed for training data.

This is done by propagating the training data through a (potentially pre-trained) CNN and obtaining its outputs, which are used as inputs to the Ψ, which is then trained using the expected performance measure values that can be computed for training data. As stated earlier, this approach leverages the fact that neural networks can approximate continuous and non-differentiable functions (please see sample results of our empirical studies in Section 6.5.2). This $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_j}$ can now be used as the derivative of the appropriate non-differentiable performance measure to train the original CNN. This can be done as follows,

6.3.1 Training CNNs with Surrogate Networks

The main criterion for a function as a loss function in a CNN is its differentiability. If a function is non-differentiable, we cannot compute its gradients for the network parameters. This restricts us from using it as a loss function since the gradients of the loss function are required during backpropagation for the updation of network weights. As discussed above, we can approximate a non-differentiable function and its gradients (derivatives) using surrogate networks. We propose that an approach to use non-differentiable functions as loss functions in CNNs is that the training of the CNN can be performed using approximated gradients obtained from a surrogate network. In this subsection, we present a framework for training a CNN (this can be generalized to any neural network, for that matter) using the gradients obtained from the surrogate network.

We propose our methodology in a manner which allows us to use this framework with any existing CNN (or neural network) architecture. We put forward a direct optimization technique for training a CNN with surrogate loss networks. In direct optimization, we train the surrogate loss network a priori.

6.3.1.1 Training using Direct Optimization

An overview of training a CNN using *direct optimization* is illustrated in Figure 6.2. In the first step, the given non-differentiable loss function L is approximated using a surrogate loss network Ψ_{Loss}. We then train the CNN using this surrogate loss network as follows. In the forward pass, given an image I_i, we first compute the CNN output $y_i^{'}$. Using the trained surrogate loss network, we then compute the loss between the network output $y_i^{'}$ and target output y_i, i.e $L(y_i^{'})$ using Eq 6.1. The parameters in the CNN are then typically updated in the backpropagation step, which requires the computation of $\frac{\partial L}{\partial y_i^{'}}$, the gradient of the loss function w.r.t the network output. We compute $\frac{\partial L}{\partial y_i^{'}}$ using Eq 6.2, where the output of the CNN is provided as input to the surrogate network, and the gradient of the refined loss function, as approximated by the surrogate network, is computed using the backprop-to-image approach (Eq 6.2). In the next step, the gradient of the loss function L w.r.t. the weights in the last layer, say fully connected (FC) layer W^{fc}, are computed using the chain rule as follows:

$$\frac{\partial L}{\partial W^{fc}} = \frac{\partial L}{\partial y_i^{'}} \frac{\partial y_i^{'}}{\partial W^{fc}} \tag{6.3}$$

The weights in the FC layer are updated using these gradients, and the weights in the preceding layers are updated using the standard chain rule. Given a pre-trained surrogate network (which is first trained based on a given CNN model), the CNN is further refined using the aforementioned backpropagation until convergence or the desired number of iterations. In summary, the availability of the gradients of the non-differentiable loss function L w.r.t. the network output $y_i^{'}(\frac{\partial L}{\partial y_i^{'}})$ through the surrogate network allows us to finetune the CNN parameters using chain rule.

We note that if the given loss function is differentiable, the gradients obtained from the surrogate network would more or less be the same as its actual gradients. In this case, the above direct optimization technique for training the CNN network would be equivalent to the usual training of the CNN. From this perspective, one can view the regular training of CNN as a particular case of our proposed approach.

6.4 Surrogate Loss Network for Semantic Segmentation and Human Pose Estimation

This section explains the proposed surrogate loss network framework for the discrete IoU and the non-decomposable PCP and PCKh metrics, used in semantic segmentation and human pose estimation respectively. We begin with defining the loss functions in both cases, the corresponding surrogate networks, and finally describe the training methodology.

6.4.1 Defining the loss functions

In order to propose a surrogate loss network for the IoU, PCP and PCKh metrics, we need to define a loss function over these performance measures. In this subsection, we begin with a discussion of a loss function over IoU and PCKh measures.

6.4.1.0.1 IoU **Loss** For a given image, the widely used IoU measure gives the similarity between the predicted region and actual region (ground truth) of the object present in the image, and is given by: $IoU = \dfrac{TP}{TP + FP + FN}$, where TP, FP and FN denote the counts of true positives, false positives and false negatives respectively. If the output of the semantic segmentation model exactly matches with the ground truth, then its IoU becomes 1, which is desired. Owing to the discrete nature of the counts, the IoU measure is inherently non-differentiable [127]. Instead of replacing the non-differentiable IoU measure with proxies such as cross-entropy (which is the standard practice in existing semantic segmentation work [11, 92, 133, 182]) or use other recently proposed measures [127], we use a neural network as a function approximator to automatically learn the surrogate IoU loss to train models for semantic segmentation. Note that to incorporate the IoU measure directly as a loss function to train models, we need to minimize $1 - IoU$. We hence define the IoU loss, denoted by $\mathcal{L}_{IoU}$, as:

$$\mathcal{L}_{IoU} = 1 - IoU = 1 - \frac{TP}{TP + FP + FN} = \frac{FP + FN}{TP + FP + FN} \tag{6.4}$$

Since IoU is not differentiable, the IoU loss, $\mathcal{L}_{IoU}$ (Eqn 6.4), is also non-differentiable. Hence, we cannot directly compute its gradients, which restricts us from using it directly to train deep networks. Instead we choose to approximate the IoU loss using a surrogate neural network, and obtain its derivative across this new network to get the required gradients (using the backpropagation-to-image trick as explained earlier, Eqn 6.2.)

6.4.1.0.2 PCKh Loss For a given predicted human pose, the PCKh metric measures the percentage of joints location predictions that is at maximum distance of 50% of the head size from the ground truth. We use the following notations for defining pose of a human present in the given image. For a given labeled image **x**, its pose is defined as $\mathbf{x}_p = (l_1, l_2, \ldots, l_k)$, where $l_i = (m_i, n_i)$ is the location of i^{th}

joint and k is the number of body joints. In a typical pose estimation CNN network [24, 163, 176, 176], we generally train a model ψ such that, for the given image **x**, it outputs the pose presented in the image, i.e. $y = \psi(\mathbf{x}, \theta) \in \mathbb{R}^{2k}$, where θ denotes model parameters. It outputs the locations of k body joints. Since PCKh gives the prediction accuracy, if the output of the pose estimation model exactly matches with the ground truth, then its PCKh becomes 100%, which is desired. During training, we normalize these accuracy values between 0 to 1. However, PCKh measure is a non-decomposable metric, and hence also non-differentiable. To incorporate the PCKh measure as a loss function to train the network, we need to minimize $1-$PCKh. We hence define the PCKh loss as:

$$L_{PCKh} = 1 - PCKh \tag{6.5}$$

where $PCKh$ is the PCKh measure for the given pose. Similar to what was discussed for the IOU loss above, we approximate L_{PCKh} using a surrogate neural network, and obtain its derivative across this new network to get the required gradients.

Now we describe our methodology to automatically learn a differentiable approximation for both IoU and PCKh loss functions using surrogate loss networks.

6.4.2 The Surrogate Loss Networks

Given the input-output data pairs $D = \{(\mathbf{x}_i, y_i) : i = 1, \cdots, n\}$, where $y_i = f(\mathbf{x}_i)$ and n is the number of data points, we propose our methodology in a manner which allows us to use this framework for any non-differentiable loss function.

6.4.2.0.1 Surrogate IoU Loss Network In semantic segmentation, the output of the CNN gives a probability score map, i.e. for each pixel, it gives the probability of being part of each of the classes considered. To approximate the IoU loss using neural networks, we need to define the input data over the continuous domain. To this end, we define the outputs of the semantic segmentation CNN in terms of probability counts TP_{pr}, FP_{pr} and FN_{pr} as below:

$$\begin{gathered} TP_{pr} = \sum_{x_i} P(x_i), \text{ where } x_i \in TP, \quad FP_{pr} = \sum_{x_i} P(x_i), \text{ where } x_i \in FP \\ FN_{pr} = \sum_{x_i} P(x_i), \text{ where } x_i \in FN \end{gathered} \tag{6.6}$$

TP_{pr}, FP_{pr} and FN_{pr} are the sums of probabilities of pixels in TP, FP and FN respectively (which provide us a continuous-domain equivalent of the corresponding TP, FP and FN counts). Given a

CNN model, we can calculate TP_{pr}, FP_{pr} and FN_{pr} using the available ground truth. Our IoU loss approximator network is hence defined by the input-output data pairs: $D = \{((TP_{pr_i}, FP_{pr_i}, FN_{pr_i}), L_{IoU_i}) : i = 1, \cdots, n\}$, where $L_{IoU_i} = 1 - \frac{TP_i}{TP_i + FP_{i+} FN_i}$ is the actual IoU loss computed from the pixel counts. We generate the dataset D to train this IoU-loss approximator network as follows:

- For the given training data, we first train a CNN, Θ_{CE}, using cross-entropy loss.
- After training, for each training image, we store the output probability map obtained from Θ_{CE}.
- For each image, we compute its TP_{pr}, FP_{pr}, FN_{pr} using Eq 6.6.

We now have the dataset D to train the approximator network. We call this surrogate neural network as $NeuroIoU$, and the loss as $NeuroIoU$ loss. We use the mean-squared error loss function, as defined below, to train this network using this data:

$$E(w) = \sum_i [NeuroIoU(TP_{pr_i}, FP_{pr_i}, FN_{pr_i}) - L_{IoU_i}]^2 \tag{6.7}$$

In semantic segmentation, it is possible that two different segmentation results may have the same IoU score, i.e. two different sets of TP, FP and FN counts (or TP_{pr}, FP_{pr}, FN_{pr}) may have the same IoU value. This affects the approximation performance. To make the approximation more robust to such issues, apart from TP_{pr}, FP_{pr}, FN_{pr}, we also include TP, FP and FN counts as inputs to the $NeuroIOU$ network. In particular, we add $\frac{TP_{pr}}{|TP|}$, $\frac{FP_{pr}}{|FP|}$ and $\frac{FN_{pr}}{|FN|}$ as inputs to the network, where, $|TP|, |FP|, |FN|$ are the counts of TP, FP and FN. Our empirical studies (detailed in Section 6.5) showed that with more inputs, the network was more robust in the approximation.

Approximation using Prediction Loss: Instead of approximating the IoU loss using pixel counts as discussed above, we can also approximate using the loss values obtained for each pixel during the prediction obtained from the original model. We use hinge loss, and these values are obtained from the classification layer. Hinge loss values have been previously used for approximating the IoU loss [15]. In this new setting, input to the $NeuroIoU$ loss are TP_H, FP_H and FN_H, which are computed as $TP_H = \sum_{x_i} h(x_i),$ where $x_i \in TP$, $FP_H = \sum_{x_i} h(x_i),$ where $x_i \in FP$ and $FN_H = \sum_{x_i} h(x_i),$ where $x_i \in FN$. $h(x_i)$ is the hinge loss for pixel x_i.

6.4.2.0.2 Surrogate PCKh Loss Network As discussed in the previous section, to approximate PCKh loss (L_{PCKh}) using neural networks, we need its input-output data D. In pose estimation, for a given image, the deep pose network models [24, 77, 163, 176] output the locations of body joints.

Algorithm 1 Training a Semantic Segmentation Network using $NeuroIoU$ Loss

```
Input: (i) Set of training images $I$ and their ground truth labels $G$; (ii) $NeuroIoU$ loss for each training image (obtained from the surrogate-IoU neural network); (iii) Initial trained CNN model $\Theta_{CE}$ trained on standard cross entropy loss
Output: Updated semantic segmentation CNN model $\Theta^{'}$
Let $\Theta^{'} = \Theta_{CE}$
Repeat until convergence
for x $\in I$ do
    (1). Compute $\Theta^{'}(\mathbf{x})$, the probability score map for x
    (2). Compute the $TP_{pr}$, $FP_{pr}$ and $FN_{pr}$ for x using Equations 6.6
    (3). Using Equations 6.9, compute $\frac{\partial NeuroIoU}{\partial TP_{pr}}$, $\frac{\partial NeuroIoU}{\partial FP_{pr}}$ and $\frac{\partial NeuroIoU}{\partial FN_{pr}}$
    (4). Using the gradients obtained from step (3), backpropagate through the $\Theta^{'}$ network and update the network weights
end for=0
```

The regression loss takes these joint locations as input and outputs its corresponding loss. Similarly, we take the joint locations as input to our surrogate neural network. To get a more generalized loss network, we need to consider the interactions between joint locations. Regression loss does not encode these interactions. In the computation of PCKh measure, a part is considered detected only if two of its joint locations are within a threshold from its true locations. This means connections between the joint locations are also very important for improved accuracy. To encode the interactions between the joint locations, we also take the midpoint of each part locations and a unit normal vector along each part as inputs to our surrogate network. The midpoint of body parts and the unit normal vector along the parts can also characterize the given pose. We also add a threshold for each part, which is used for measuring the PCKh measure. These thresholds are calculated from the training data. Our results showed that, the addition of interactions between joint locations enhance the performance of the network. Thus, the input-output data for our surrogate network is defied as $D = \{(p_i, L_{PCKh}(p_i) : i = 1, \ldots, t\}$, where p_i is the detected pose and $L_{PCKh}(p_i)$ is its PCKh loss. The pose p_i is given as $(l_1, m_1, u_1, t_1, \ldots, l_k, m_k, u_k, t_k)$, where l_j is the location of j^{th} joint. u_p is the unit normal vector for the p^{th} part and m_p is its midpoint, and t_p is its threshold. We generate the dataset D as follows:

- For the given training data, we first train a CNN, θ_{reg}, using regression loss

- During training, for each training image, we store the detected pose obtained from θ_{reg}
- For each detected pose in the training image, we compute the unit vector for each part and its midpoint

We now have the dataset D for training the surrogate network. We call this surrogate network as $NeuroPCKh$ and the loss as $NeuroPCKh$ loss. Using the similar methodology, we can also compute the surrogate loss network over the performance measure PCP. We call the corresponding loss as $NeuroPCP$.

6.4.3 Training CNN networks with Surrogate Loss

This section presents the details of the methodology to train the original CNN using the proposed surrogate loss. We describe this in the context of training deep semantic segmentation networks by minimizing the corresponding $NeuroIoU$ loss. A similar formulation is used for deep pose networks.

The $NeuroIoU$ loss takes TP_{pr}, FP_{pr} and FN_{pr} as inputs and approximates the corresponding IoU loss. Once we train the network, we use the backprop-to-image trick [181] to compute the derivative of the $NeuroIoU$ loss with respect to the outputs of the original CNN. These derivatives are computed as follows. For simplicity, we consider a single hidden layer in the surrogate network (this can be easily extended for multiple hidden layers). If S is the activation function used in the network, then for the given input $x_i = (TP_{pr_i}, FP_{pr_i}, FN_{pr_i})$, $NeuroIoU(TP_{pr_i}, FP_{pr_i}, FN_{pr_i})$ is computed as:

$$NeuroIoU(TP_{pr_i}, FP_{pr_i}, FN_{pr_i}) = \sum_{j=1}^{k} w_j^{op} S(x_i^T w_j^h) \tag{6.8}$$

where k is the number of neurons in the hidden layer, $w^h \in \mathbb{R}^{k\times 3}$ are weights in the hidden layer and $w^{op} \in \mathbb{R}^k$ are the weights in the output layer. The derivatives of the $NeuroIoU$ loss w.r.t. the CNN outputs are given as:

$$\begin{gathered}\frac{\partial Neuro_{IoU}}{\partial TP_{pr}} = \sum_{j=1}^{k} w_j^{op} w_{j1}^h S^{'}(x^T w_j^h), \frac{\partial Neuro_{IoU}}{\partial FP_{pr}} = \sum_{j=1}^{k} w_j^{op} w_{j2}^h S^{'}(x^T w_j^h) \\ \frac{\partial Neuro_{IoU}}{\partial FN_{pr}} = \sum_{j=1}^{k} w_j^{op} w_{j3}^h S^{'}(x^T w_j^h)\end{gathered} \tag{6.9}$$

where $S^{'}$ is the derivative of the activation function S.

We now have the gradients of $NeuroIoU$ with respect to TP_{pr_i}, FP_{pr_i} and FN_{pr_i}, and can retrain the initial CNN, Θ_{CE}, using these gradients. The training procedure is summarized in Algorithm 1. We consider the semantic segmentation network Θ_{CE} initially trained using cross-entropy loss. We then finetune this network using $NeuroIoU$ loss, by using the gradients of the $NeuroIoU$ loss w.r.t. the outputs of Θ_{CE} in Equations 6.9. To this end, we remove the softmax layer from the Θ_{CE} network. For a given training image **x**, we compute its probability score map $\Theta_{CE}(\mathbf{x})$ from the network. We then update the network weights using backpropagation, by using the gradients of $NeuroIoU$ loss with respect to probability score maps. These steps are continued till convergence. In each iteration, the $NeuroIoU$ loss increases the pixel probabilities for true positive and false negative pixels, as well as decreases the pixel probabilities for false positive pixels. This increases the true positive count and reduces the false positive and false negative counts for each image, thus increasing overall IoU performance. We use a similar formulation for training deep pose networks, except that the initial network is trained using regression loss in this case.

6.4.4 Weighted Surrogate Loss Function

In the previous subsection, we defined the proposed $NeuroIoU$ loss for semantic segmentation and $NeuroPCKh$ Loss for human pose estimation. In semantic segmentation, to leverage the additional information obtained from the cross-entropy loss, we define the weighted $NeuroIoU$ loss for semantic segmentation as:

$$L_{wtNeuroIoU} = L_{NeuroIoU} + \lambda L_{CE} \tag{6.10}$$

where $L_{NeuroIoU}$ is the $NeuroIoU$ loss and L_{CE} is the cross-entropy loss. λ is a tradeoff parameter controlling the importance of cross-entropy loss.

Similarly, for human pose estimation, to leverage the additional information obtained from the regression loss, we define the weighted $NeuroPCKh$ loss as:

$$L_{wtNeuroPCKh} = L_{NeuroPCKh} + \lambda L_{reg} \tag{6.11}$$

where $L_{NeuroPCKh}$ is the $NeuroPCKh$ loss and L_{reg} is the regression loss. λ is a tradeoff parameter controlling the importance of regression loss.

6.5 Experiments

In this section, we validate the performance of the proposed $NeuroIoU$ and $NeuroPCKh$ loss functions to train the corresponding state-of-the-art deep models for semantic segmentation and human pose estimation.

6.5.1 Datasets and Experimental Settings

For semantic segmentation, we conducted our experiments on two popular datasets: PASCAL VOC 2011 [42] and Cityscapes [30]. For both datasets, we train the network on the provided benchmark training and validation splits (the ground-truth for the test set is not publicly available and our focus is on comparison to the baseline method, which is also evaluated in the same manner for fairness). For the PASCAL VOC dataset, we resized the training images to 375×500, and for Cityscapes, we resized the images to 512×1024. We integrated the proposed $NeuroIoU$ loss into 3 contemporary semantic segmentation networks, FCN [92], Segnet [11] and UNet [133]. We take the original FCN, Segnet and UNet with cross-entropy loss as the baseline for studying the performance of our proposed $NeuroIoU$ loss, as well as compare our results to [127] on models that were validated in their work (FCN). In the experiments, we refer to the proposed method as $NeuroIoU$, the approximation in [127] as $IoU-Appx$ and the deep networks trained with standard cross-entropy loss as CE.

For human pose estimation, we conducted all our experiments on Leeds Sports Dataset [71] (LSP) and MPII datasets. These are images from sports activities and are quite challenging in terms of appearance and especially articulations. In this dataset, for each person, the full body is labeled with total of 14 joints. The MPII dataset [9] (single person) contains images in diverse scenarios that contain many real-world challenges such as crowding, scale variation, occlusion, and contact. For both datasets, we train the network on the provided benchmark training set and test on the respective test set. We integrated the proposed $NeuroPCP$ and $NeuroPCKh$ loss into DeepPose [163] network. We take the original DeepPose [163] network with regression loss as the baseline for studying the performance of our proposed $NeuroPCKh$ loss. We evaluated the performance of $NeuroPCP$ loss over LSP dataset and $NeuroPCKh$ over MPII dataset. For all our experiments, we used a fixed learning rate of 10^{-5}, momentum of 0.99 and weight decay parameter of 0.0005. In our surrogate neural network used to approximate the IoU, PCP and PCKh measures, we use 4 hidden layers and 100 nodes in each layer.

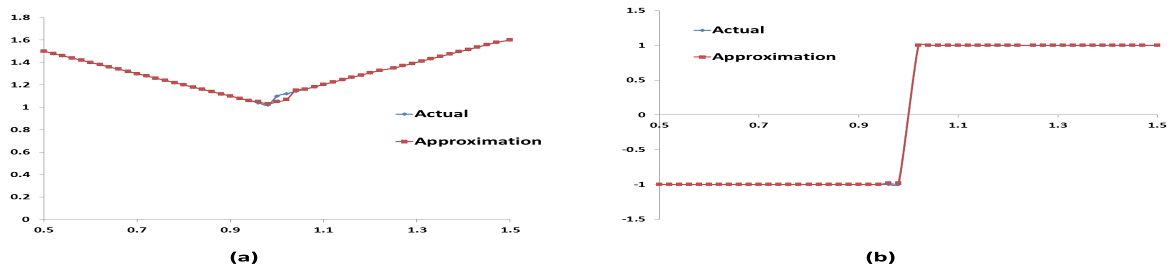


Figure 6.3: Approximation of a function (non-differentiable at $x = 1$) and its derivatives using neural networks: (a) Approximation of the function; (b) Approximation of its derivatives.

6.5.2 Function Approximation using Neural Networks

In order to understand the usefulness of the Surrogate-IoU network for approximating the IoU measure, we conducted simple experiments on the capability of neural networks to approximate a discontinuous function and its derivatives. Consider $f(x) = \begin{cases} 2-x & \text{if } x < 1 \\ x+0.1 & \text{if } x \geq 1 \end{cases}$ and its derivative $f^{'}(x) = \begin{cases} -1 & \text{if } x < 1 \\ 1 & \text{if } x > 1 \end{cases}$. We can check that $f(x)$ is not continuous and hence not differentiable at $x = 1$. Results for this approximation using a simple one hidden-layer neural network are given in Figure 6.3. Figure 6.3 (a) show the approximation of the function and Figure 6.3 (b) shows the approximation of its derivatives. We can observe that the approximations obtained almost exactly matches the original function. At $x = 1$, there is a sudden change in the derivative due to the discontinuity, but the neural network is able to approximate this sudden peak with a smooth curve. Since $f(x)$ is not differentiable at $x = 1$, we did not plot its derivative value at $x = 1$. The approximation of the derivative at $x = 1$ lies between -1 and 1.

6.5.3 Results on Semantic Segmentation

The results for FCN, Segnet and UNet on PASCAL VOC are shown in Figure 6.4. It shows the comparison of the proposed $NeuroIoU$ loss and the baseline cross-entropy loss. We can observe that the proposed $NeuroIoU$ loss consistently outperforms the cross-entropy loss in all the categories, with a performance gain for all three networks. This corroborates our claim that this approach can be integrated into any existing deep learning network for improving its performance in semantic segmentation. From

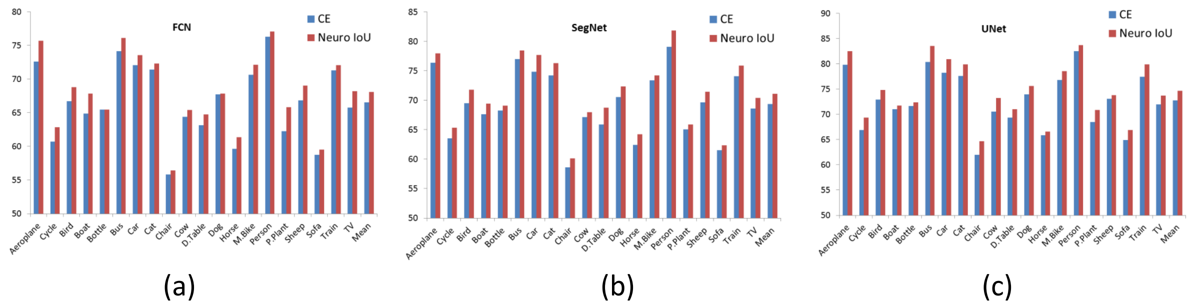


Figure 6.4: Results on PASCAL VOC: (a) FCN (b) SegNet (c) UNet. Here, CE is the network trained using cross-entropy loss, and $NeuroIoU$ is the network trained using proposed $NeuroIoU$ loss.

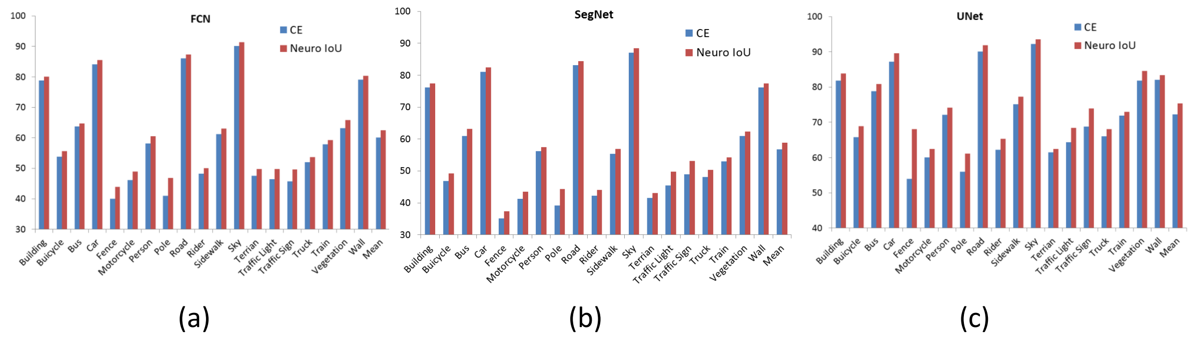


Figure 6.5: Results on CityScapes: (a) FCN (b) SegNet (c) UNet. Here, CE is the network trained using cross-entropy loss, and $NeuroIoU$ is the network trained using proposed $NeuroIoU$ loss.

the Figure 6.4, we also notice that the performance improvements are more significant for some classes, where the foreground to background pixel ratio is very small.

Similar results for FCN, SegNet and UNet on the Cityscapes dataset are presented in Figure 6.5. Once again, we observe that the proposed $NeuroIoU$ loss outperforms cross-entropy loss over all classes for all three networks. Note again that the maximum performance improvement is obtained for classes where the ratio of foreground to background pixels is very small. We also tested our method against using weighted cross-entropy loss on a subset of classes with this foreground-background imbalance from the Cityscapes dataset. One could argue that weighting cross-entropy loss suitably can account for class imbalance in the training set. Using weighted cross-entropy, we obtained an IoU of 58.4, whereas $NeuroIoU$ gave an IoU of 60.2 for the same subset. The proposed $NeuroIoU$ method does not need such explicit reweighting since it implicitly learns what is necessary for overall IoU performance.

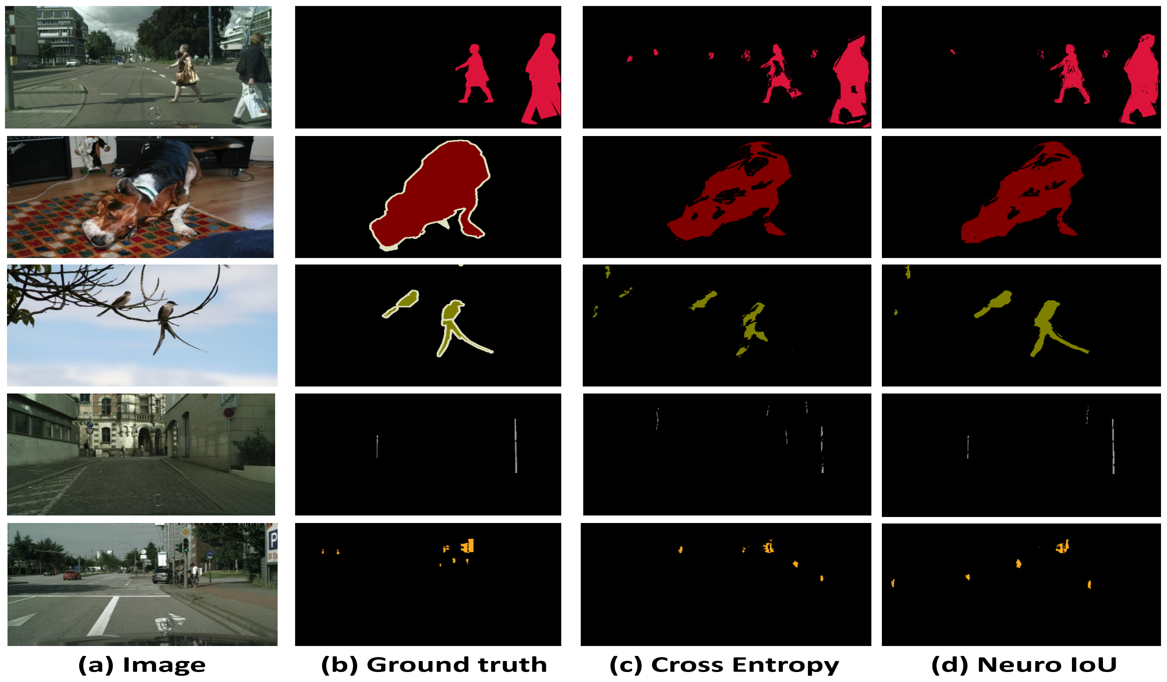


Figure 6.6: Few qualitative results. (a) Original image (b) Ground truth segmentation (c) Segmentation obtained from cross entropy loss (d) Segmentation obtained using $NeuroIoU$.

We also show some qualitative results in Figure 6.6. From the figure, we can observe that, in comparison to the cross-entropy loss, the proposed $Neuro_{IoU}$ loss tends to fill the gaps in the segmentation. It means it tends to recover some of the false negative errors made by the cross entropy loss. In the 4^{th} image (Row-4), we can observe that $NeuroIoU$ is able to localize small objects (pole), which the cross entropy-trained model could not. In the 5^{th} image (Row-5), we show a failure case for the $NeuroIoU$ loss. Here, $NeuroIoU$ is not able to find the small traffic signals present in the image. Note that these objects are also not detected by the model trained using cross-entropy loss.

6.5.3.0.1 Comparison to previous work Rahman and Wang proposed an approximation to the IoU measure in [127] (as stated in Section 6.2). The results of comparing the proposed method to this approximation are given in Table 6.1, where CE represents networks trained using cross-entropy loss. The results are given for FCN over the PASCAL VOC dataset (as in [127]). The results for [127] are obtained from our implementation of their work. The results for training the model using cross-entropy loss (CE) are obtained for a fixed learning rate. Note that these performance numbers can be improved by experimenting with different learning rates, weight decay and other parameters. Our objective is to show

	Aeroplane	Cycle	Bird	Boat	Bottle	Bus	Car	Cat	Chair	Cow
CE	70.58	59.71	67.71	64.84	63.48	75.17	72.05	70.40	53.81	64.37
IoU-Appx [127]	72.72	61.78	68.61	67.29	64.31	76.57	73.03	70.82	54.18	64.06
$NeuroIoU$	73.68	61.84	69.80	67.83	64.49	77.73	73.57	71.28	54.82	65.38
$NeuroIoU_{No-Init}$	73.65	61.80	69.80	67.82	64.38	77.73	73.57	71.27	54.77	65.37
$NeuroIoU_{Hinge}$	75.14	63.35	70.71	69.75	65.84	79.80	74.97	**73.03**	55.47	66.16
$NeuroIoU_{Hinge} - AO$	75.16	63.35	70.72	69.73	65.87	79.81	74.97	73.00	55.48	66.16
$NeuroIoU_{WtHinge}$	**75.52**	**63.69**	**71.15**	**70.05**	**66.23**	**80.16**	**75.34**	**73.45**	**55.86**	**66.49**

	D.Table	Dog	Horse	M.Bike	Person	P.Plant	Sheep	Sofa	Train	TV	Mean
CE	64.12	65.71	58.64	70.61	77.28	62.27	66.85	56.73	72.27	64.79	66.06
IoU-Appx [127]	65.63	64.17	58.70	71.20	77.39	64.14	67.72	58.38	72.20	66.07	66.94
$NeuroIoU$	65.98	65.82	60.37	72.10	78.04	66.62	69.02	59.52	73.06	67.18	67.90
$NeuroIoU_{No-Init}$	65.97	65.82	60.37	72.04	78.03	66.62	68.98	59.52	73.03	67.18	67.87
$NeuroIoU_{Hinge}$	67.63	66.74	62.73	73.68	79.89	68.03	71.62	60.85	74.75	69.27	69.46
$NeuroIoU_{Hinge} - AO$	67.62	66.79	62.74	73.68	79.86	68.04	71.65	60.86	74.73	69.27	69.47
$NeuroIoU_{WtHinge}$	**67.97**	**67.23**	**63.18**	**74.03**	**80.20**	**68.45**	**72.07**	**61.40**	**75.06**	**69.53**	**69.85**

Table 6.1: Comparison of $NeuroIoU$ with cross-entropy loss and [127] on PASCAL VOC.

the relative performance gain for different methods over CE loss under the same training conditions. The relative improvement in the performance of [127] over cross-entropy loss (CE) is what we intend to show, rather than the absolute IoU value. Here, the $NeuroIoU$ loss is used to retrain the network obtained from training using the CE loss. It is evident that $NeuroIoU$ loss outperforms this surrogate loss [127] over all classes. Compared to the cross-entropy loss, the surrogate loss in [127] underperforms for some classes. However, the proposed $NeuroIoU$ loss consistently outperforms cross-entropy loss over all classes.

We also compare other variants of $NeuroIoU$, $NeuroIoU_{No-Init}$, $NeuroIoU_{Hinge}$ and $NeuroIoU_{WtHinge}$ in Table 6.1. In $NeuroIoU_{No-Init}$, we train the network without any initialization, i.e., the model trained using cross-entropy loss is not used to initialize the network. In $NeuroIoU_{Hinge}$, IoU is approximated using hinge loss values instead of probabilities from the final layer (as in Section 6.4.2). In $NeuroIoU_{WtHinge}$, the weighted surrogate loss is used as the final loss function (as in Section 6.4.4). Here, we take λ= 0.1. From the Table 6.1, we can observe that the network trained without initial-

	RBF	Tanh	Sigmoid
IoU	67.58	67.76	**67.90**

Table 6.2: Comparison of RBF, Tanh and Sigmoid activation functions in $NeuroIoU$ loss for semantic segmentation.

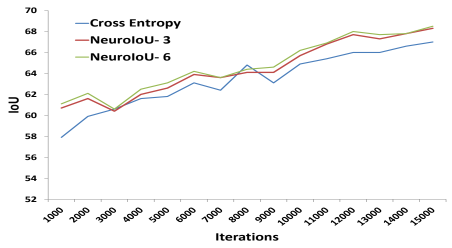


Figure 6.7: Comparison of performance of NeuroIoU with 3 inputs vs NeuroIoU with 6 inputs using FCN on PASCAL VOC. While both performed better than CE, NeuroIoU-6 was more consistent.

ization from a pre-trained model ($NeuroIoU_{No-Init}$) can obtain similar performance as the network trained using the initialization ($NeuroIoU$) obtained using cross-entropy loss. We can also observe that $NeuroIoU_{Hinge}$ performs well compared to $NeuroIoU$ and, $NeuroIoU_{WtHinge}$ outperformed all other methods. Using $NeuroIoU_{WtHinge}$, we obtained an IoU of 69.85, which is an improvement of 3.79 IoU over cross-entropy. The superiority in the performance is due to the additional information obtained from the cross-entropy loss while training the network. In all the above experiments, we use direct optimization technique for training the segmentation network using $NeuroIoU$ loss.

6.5.3.0.2 Ablation Studies: We show here the impact of various choices in the surrogate loss network on the CNN performance. We show the results for $NeuroIoU$ loss over semantic segmentation. In all the experiments, the results are given for FCN over PASCAL VOC. Here, the FCN is trained using $NeuroIoU$ loss. We first evaluate the impact of different activation functions for the $NeuroIoU$ loss in Table 6.2.

From the results, we can observe that the Sigmoid function performs slightly better compared to RBF and Tanh activation functions.

We studied the performance of the Surrogate-IoU neural network over different inputs in Figure 6.7. In $NeuroIoU - 6$, the neural network is trained over 6 inputs $TP_{pr}, FP_{pr}, FN_{pr}$, $\frac{TP_{pr}}{|TP|}$, $\frac{FP_{pr}}{|FP|}$ and $\frac{FN_{pr}}{|FN|}$, while the neural network is trained only on 3 inputs $TP_{pr}, FP_{pr}, FN_{pr}$ in $NeuroIoU - 3$.

# Hidden layers	2	4	6	8
IoU	67.57	67.90	67.90	**67.92**

Table 6.3: Comparison of performance of $NeuroIoU$ loss with varying number of hidden layers for semantic segmentation.

	$NeuroIoU$-Mapillary	$NeuroIoU$-Cityscapes	$NeuroIoU$-Map-City
IoU	61.86	62.18	**62.41**

Table 6.4: Performance of the $NeuroIoU$ trained over Mapillary and Cityscapes datasets. In $NeuroIoU$-Mapillary, $NeuroIoU$ is trained on mapillary dataset. In $NeuroIoU$-Cityscapes, $NeuroIoU$ is trained on Cityscapes. In $NeuroIoU$-Map-City, $NeuroIoU$ is trained on both mapillary and Cityscapes.

The results are given for 15000 iterations. From the figure, we notice that $NeuroIoU-6$ is better than $NeuroIoU-3$ (we observed the same trend for other datasets too), while both outperform cross-entropy loss. In this experiment, we take number of hidden layers as 4 in $NeuroIoU$.

In the next experiment, we evaluate the impact of the number of hidden layers in the $NeuroIoU$ loss for the given semantic segmentation problem. The results for comparing the performance on varying number of hidden layers are given in Table 6.3. From the results, we can observe that there is an improvement in performance with the increase in the number of hidden layers. However, the improvement is rather minimal. It means that our surrogate network can approximate the IoU measure with a fairly low number of hidden layers. In all our experiments, a surrogate network with four hidden layers gives reasonably good performance.

In all our experiments, for each dataset, we train the surrogate loss network on the same dataset. To further evaluate our surrogate loss network, we initially train it on other datasets and test it on the given dataset. In this experiment, we train our $NeuroIoU$ in different configurations and test each case in the Cityscapes. These results are given in Table 6.4. In $NeuroIoU$-Mapillary, $NeuroIoU$ is trained only on the Mapillary dataset. There is no training involved on Cityscapes dataset. In $NeuroIoU$-Cityscapes, $NeuroIoU$ is pre-trained on Cityscapes dataset. Here, there is no training involved on the Mapillary dataset. In $NeuroIoU$-Map-City, $NeuroIoU$ is initially pre-trained on the Mapillary dataset and during the training of the segmentation model, it is finetuned on the Cityscapes dataset.

	Cross-Entropy	$Neuro$-Cross-Entropy
IoU	66.06	**66.07**

Table 6.5: Performance of FCN trained with naive cross-entropy loss (cross-entropy) and cross-entropy loss approximated using the proposed surrogate loss network ($Neuro$-Cross-Entropy) on PASCAL VOC.

From the results, we can observe that $NeuroIoU$-Map-City performs the best, thanks to the additional information from the Mapillary. $NeuroIoU$-Mapillary has performance comparable to the other two settings but has slightly lower performance since it is not exposed to the Cityscapes dataset, which is used for testing.

The naive training in CNN is a particular case of our proposed approach. To demonstrate this, we try to approximate the cross entropy loss using our proposed surrogate loss network and train the given segmentation network using the trained surrogate loss network. We use (FCN) as the segmentation network. Note that, the cross entropy loss is a differentiable function. The results over PASCAL VOC are given in Table 6.5. In cross-entropy, FCN is trained using original cross entropy loss. In $Neuro$-Cross-Entropy, cross-entropy itself is approximated using the surrogate network and FCN is trained using this surrogate loss network. From the results, we can observe that $Neuro$-Cross-Entropy performing equally well compared to cross-entropy. In fact, it is slightly performing better compared to cross-entropy. It shows that CNN training with differentiable loss functions is a particular case of our proposed technique. This demonstrates that our proposed surrogate loss network is a more generalized version of existing methods for training.

6.5.3.0.3 Training using Alternate Optimization In alternate optimization, we train both CNN and surrogate loss network in parallel in an alternate manner. It is based on alternate optimization. The training procedure (for semantic segmentation) is summarized in Algorithm 2. Unlike the direct optimization, presented in Section 6.3.1.1 (where the surrogate network is trained completely before fine-tuning the CNN), we alternately train the CNN and surrogate loss networks in this technique. The results for the alternate optimization technique are presented in Table 6.1 under $NeuroIoU_{Hinge} - AO$ and compared with $NeuroIoU_{Hinge}$. From the results, we observe that both $NeuroIoU_{Hinge}$ and $NeuroIoU_{Hinge} - AO$ are performing equally well. In the alternative optimization technique, at each step of training the segmentation network, we train the $NeuroIoU$ for five iterations.

Algorithm 2 Training using Alternate Optimization

```
Input: (i) Set of training images $I$ and their ground truth labels $G$; (ii) Initial network $\Theta^{'}$ (initialized using a backbone network)
Output: Final model $\Theta^{'}$ trained using alternate optimization technique
Repeat until convergence
for x $\in I$ do
    (1). Compute $\Theta^{'}(\mathbf{x})$, the probability score map for x
    (2). Compute the actual IoU loss for $x$ using the pixel counts obtained from $\Theta^{'}(\mathbf{x})$, $L_{IoU_i} = 1 - \frac{TP_i}{TP_i + FP_{i+}FN_i}$
    (3). Compute $NeuroIoU$ loss using Equation 6.7 and its gradients w.r.to $TP_{pr}$, $FP_{pr}$ and $FN_{pr}$
    (4). Using the gradients obtained from step (3), backpropagate through the surrogate network ($NeuroIoU$) and $\Theta^{'}$ network, and update the network weights
end for=0
```

6.5.4 Results on Human Pose Estimation

The results for DeepPose network [163] using regression loss (Deep Pose) and the proposed $NeuroPCP$ loss ($Neuro_{PCP}$) on the LSP dataset are given in Table 6.6. We begin our analysis by reporting each approach's overall pose estimation performance and summarize the results in Table 6.6. As given in [163], we show results for the four most challenging limbs, lower and upper arms and legs to compare across different loss functions. We also show the average value across these limbs. The $NeuroPCP$ loss achieves the best result of 61.3% PCP, followed by the regression loss with 60.7% PCP. We obtain maximum improvement for lower arm, which has many variations compared to all other parts. We also show the results for training DeepPose [163] using alternative optimization ($Neuro_{PCP} - AO$) for the proposed $NeuroPCP$ loss in Table 6.6. In alternate optimization, we train both CNN and surrogate loss networks in parallel in an alternate manner. The results show that the model trained using alternative optimization slightly performs well compared to the $NeuroPCP$. In $NeuroPCP$, the model is trained using direct optimization. We hypothesize that the slight superior performance for the alternative optimization technique is due to the performance of the surrogate network trained for approximating PCP measure as stated earlier for semantic segmentation. In alternative optimization, we gradually increase the training set to train the surrogate network, whereas in direct optimization, we train the surrogate network only once with complete training data. This causes a difference in performance. From the results,

	Upper Leg	Lower Leg	Upper Arm	Fore Arm	Ave
Deep Pose	77.2	71.4	56.1	38.2	60.7
$Neuro_{PCP}$	77.8	71.8	56.6	39.1	61.3
$Neuro_{PCP} - AO$	77.8	71.9	56.9	39	61.4
$Neuro_{WtPCP}$	**78.1**	**73.6**	**58.5**	**40.9**	**62.8**

Table 6.6: Comparison of $NeuroPCP$ loss with regression loss on LSP dataset. In $Neuro_{PCP} - AO$, CNN is trained using alternative optimization. In $Neuro_{WtPCP}$, weighted $NeuroPCP$ loss is used as the loss function.

	Head	Sho	Elb	Wri	Hip	Knee	Ank	PCKh
Deep Pose	94.9	90.5	80.2	74.5	77.1	69.2	63.2	79.2
$Neuro_{PCKh}$	95.3	90.9	80.4	75	77.7	69.7	63.8	79.7
$Neuro_{PCKh} - AO$	95.6	90.8	80.4	75.2	77.6	69.8	64.1	79.9
$Neuro_{WtPCKh}$	**96.5**	**92.3**	**82.4**	**76.1**	**78.7**	**71.0**	**65.4**	**81.0**

Table 6.7: Comparison of $NeuroPCKh$ loss with regression loss on MPII dataset. In $Neuro_{PCKh} - AO$, CNN is trained using alternative optimization. In $Neuro_{WtPCKh}$, weighted $NeuroPCKh$ loss is used as the loss function.

we can also observe that $Neuro_{WtPCKh}$, where the weighted surrogate loss (as in Section 6.4.4) is used as the final loss function, outperformed all other variants of $NeuroPCP$. Here, we take λ=0.5. The noticeable performance gain of 2.1% in PCP measure for $Neuro_{WtPCKh}$ compared to the regression loss is due to the additional information obtained from the regression loss. It shows that the proposed surrogate loss can be added with other loss functions to further enhance its performance.

The results for DeepPose network [163] using regression loss (Deep Pose) and the proposed $NeuroPCKh$ loss ($Neuro_{PCKh} - AO$) on MPII dataset is given in Table 6.7. From the results, we can observe that $NeuroPCKh$ loss outperforms regression loss over all parts. For the $NeuroPCKh$ loss, we get 79.7% PCKh, whereas regression loss obtained 79.2% PCKh measure, corroborating our claim. The superior performance obtained for $NeuroPCKh$ loss over regression loss is due to its ability to model

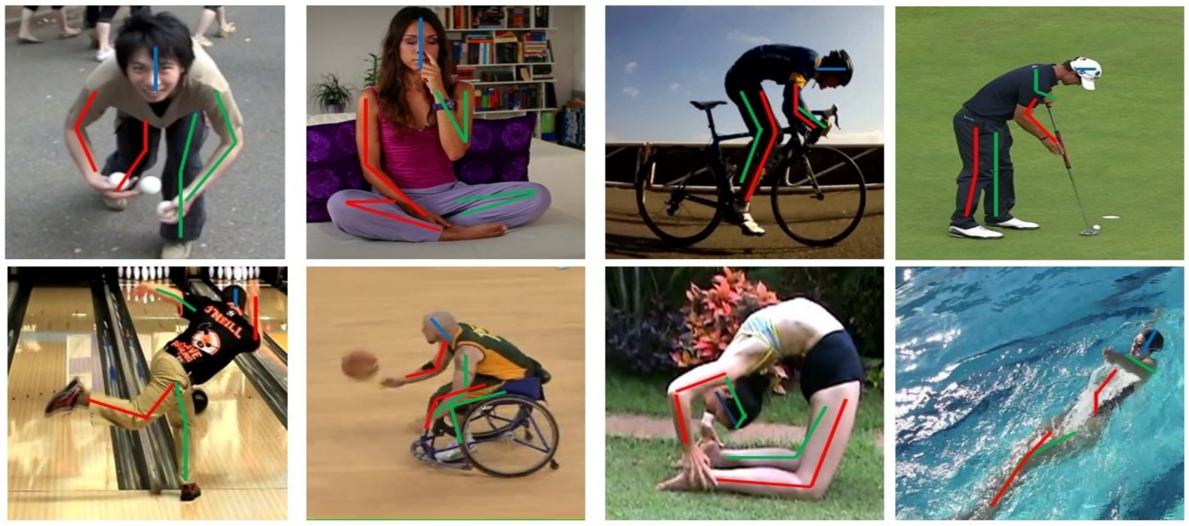


Figure 6.8: Visualization of pose results obtained using $NeuroPCKh$ loss on images from MPII. The last two results shows some fail cases caused by combinations of extreme occlusion and rare poses.

the explicit interactions between the body joints. While training the network regression loss does not capture these interactions; however, $NeuroPCKh$ loss captures these interactions from the pose information we provide using unit normal vectors while training the surrogate network. We also show the results for training the DeepPose network using alternative optimization ($Neuro_{PCKh} - AO$) for the proposed $NeuroPCKh$ loss in Table 6.6. The results show that the model trained using alternative optimization performs better than the regression loss, and the model trained using direct optimization ($NeuroPCKh$). We also compare the $Neuro_{WtPCKh}$, where the weighted surrogate loss (as in Section 6.4.4) is used as the final loss function with $NeuroPCKh$ and $Neuro_{PCKh} - AO$ in Table 6.7. From the results, we can observe that $Neuro_{WtPCKh}$ outperformed all other variants of $NeuroPCP$. Here, we take λ=0.5. Using $Neuro_{WtPCKh}$, we got an improvement of 1.8% in PCKh measure compared to the regression loss. The superiority in the performance is again due to the additional information obtained from the regression loss while training the network. We also show some qualitative results in Figure 6.6. The results are obtained using $NeuroPCKh$ loss. The figure shows that the proposed $NeuroPCKh$ loss copes well with both occlusions and difficult poses. We also show a few failure cases in the last two images. The failure cases are mainly caused by the combinations of extreme occlusion and rare poses.

6.6 Summary

In this work, we presented a new approach using surrogate neural networks for learning discrete and non-decomposable loss functions, which are inherently non-differentiable. In particular, we learned the loss functions over IoU, PCP and PCKh measures for the semantic segmentation and human pose estimation tasks. Our experimental results demonstrate that the proposed loss function can be applied to any existing network. For IoU loss, we demonstrated the effectiveness of the proposed method over semantic segmentation models on the PASCAL VOC and Cityscapes datasets. For PCP loss, we validated our method over LSP dataset and for the PCKh loss, we validated our method over MPII dataset. For both the PCP and PCKh loss, we applied our method over DeepPose [163] network. Our results on this work show consistent improvement over baseline methods, and the ablation studies presented also show the robustness of the proposed method.

Chapter 7

Conclusion

In this chapter, we conclude this thesis by discussing the contributions, impact, and comparisons of our proposed approaches with contemporary methods.

7.1 Summary

This thesis targets the problem of surrogate approximations for similarity measures to improve their performance in various applications. We have presented surrogate approximation for DTW distance, canonical correlation analysis (CCA), Intersection-over-Union (IoU), PCP, and PCKh measures.

First, we propose a linear approximation for naive DTW distance. To compute the DTW distance for any given sequence, we need to find the optimal warping path from all the possible alignments, which is a computationally expensive operation and requires quadratic complexity. We try to speed up the DTW distance computation by learning the optimal alignment from the training data. We learn a small set of global principal alignments from the given training data, and for the new test sequences, the optimal alignment is approximated using these alignments. As far as we are aware, none of the previous methods have exploited the hidden structure of the alignments. We approximate the DTW distance as a sum of multiple weighted Euclidean distances, which are known to be amenable to indexing and efficient retrieval. The proposed FastDTW distance is as good as DTW distance and computationally performs equally as simple Euclidean-based matching. Further, we also introduced a Fast Dynamic Time Warping kernel (Fast Surrogate DTW kernel), which is a surrogate linear kernel over DTW distance. We have also proposed an explicit feature map for the Fast Surrogate DTW kernel, which enables the proposed kernel to be applied with linear SVM. The explicit feature map for the FastDTW kernel is computed using the global principal alignments learned from the training data. We then presented an application using

FastDTW distance, which widens the scope of our work. In this application, we address the problem of faster indexing in classifier-based retrieval methods using FastDTW distance. We introduce the Query specific FastDTW distance for faster indexing, which has linear time complexity. In addition to this, we also present an application built using the Fast Surrogate DTW distance in the deep learning framework. We propose, FastDTWNet, where the Fast Surrogate DTW distance is applied as a feature extractor in CNN.

Our next contribution proposes a surrogate kernel approximation over canonical correlation analysis (CCA). It enables us to use CCA in the kernel framework, further improving its performance. The kernel function works well for action recognition as it embeds the temporal context in the videos. We have also shown that multiple features can be seamlessly integrated into the surrogate kernel to enhance recognition performance.

In our final contribution, we propose a method to automatically learn a surrogate loss function that approximates the IoU, PCP and PCKh loss. Semantic segmentation is a popular task in computer vision today, and deep neural network models have emerged as the popular solution to this problem in recent times. The typical loss function used to train neural networks for this task is cross-entropy loss. However, the success of the learned models is measured using Intersection-Over-Union (IoU), which is inherently non-differentiable. This gap between performance measure and loss function results in a fall in performance, which few recent efforts have also studied. In this work, we propose a novel method to automatically learn a surrogate loss function that approximates the IoU loss and is better suited for good IoU performance. To the best of our knowledge, this is the first such work that attempts to learn a loss function for this purpose. The proposed loss can be directly applied over any network.

Bibliography

[1] J. Aach and G. M. Church. Aligning gene expression time series with time warping algorithms. *Bioinformatics*, 2001.

[2] F. Ahmed, D. Tarlow, and D. Batra. Optimizing expected intersection-over-union with candidate-constrained crfs. In *ICCV*, 2015.

[3] N. Akae, A. Mansur, Y. Makihara, and Y. Yagi. Video from nearly still: an application to low frame-rate gait recognition. In *CVPR*, 2012.

[4] H. Alfeilat, A. Hassanat, O. Lasassmeh, A. Tarawneh, M. Alhasanat, H. Eyal-Salman, and S. Prasath. Effects of distance measure choice on k-nearest neighbor classifier performance: A review. *Big Data*, 2019.

[5] J. Almazán, A. Gordo, A. Fornés, and E. Valveny. Word spotting and recognition with embedded attributes. In *PAMI*, 2014.

[6] N. D. Alon Kovalchuk, Lior Wolf. A simple and fast word spotting method. In *ICFHR*, 2014.

[7] V. F. Andreas Fischer, Andreas Keller and H. Bunke. Lexicon-free handwritten word spotting using character hmms. In *PR Letters*, 2012.

[8] G. Andrew, R. Arora, J. Bilmes, and K. Livescu. Deep canonical correlation analysis. ICML, 2013.

[9] M. Andriluka, L. Pishchulin, P. V. Gehler, and B. Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *CVPR*, 2014.

[10] R. Arora and K. Livescu. Multi-view cca-based acoustic features for phonetic recognition across speakers and domains. In *International Conference on Acoustics, Speech and Signal Processing*, 2013.

[11] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *PAMI*, 2017.

[12] C. Bahlmann, B. Haasdonk, and H. Burkhardt. On-line handwriting recognition with support vector machines - a kernel approach. In *IWFHR*, 2002.

[13] A. Balasubramanian, M. Meshesha, and C. V. Jawahar. Retrieval from document image collections. In *DAS*, 2006.

[14] A. Barla, F. Odone, and A. Verri. Histogram intersection kernel for image classification. In *ICIP*, 2003.

[15] M. Berman and M. B. Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In *CVPR*, 2018.

[16] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *AAAI*, 1994.

[17] N. Bhatia and Vandana. Survey of nearest neighbor techniques. *IJCSIS*, 2010.

[18] A. Björck and G. H. Golub. Numerical methods for computing angles between linear subspaces. In *Mathematics of Computation*, 1971.

[19] M. B. Blaschko and C. H. Lampert. Correlational spectral clustering. In *CVPR*, 2008.

[20] M. Borga. Canonical correlation a tutorial, 1999.

[21] M. K. Brown and L. R. Rabiner. Dynamic time warping for isolated word recognition based on ordered graph searching techniques. In *ICASSP*, 1982.

[22] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 1998.

[23] X. Cai, T. Xu, J. Yi, J. Huang, and S. Rajasekaran. Dtwnet: a dynamic time warping network. In *NIPS*. 2019.

[24] Z. Cao, T. Simon, S. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*, 2017.

[25] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2011.

[26] C.-Y. Chang, D.-A. Huang, Y. Sui, L. Fei-Fei, and J. C. Niebles. D3tw: Discriminative differentiable dynamic time warping for weakly supervised action alignment and segmentation. In *CVPR*, 2019.

[27] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 2002.

[28] V. Cherkassky. The nature of statistical learning theory. *IEEE Trans. Neural Netw. Learning Syst.*, 1997.

[29] K. Chomboon, P. Chujai, P. Teerarassammee, K. Kerdprasop, and N. Kerdprasop. An empirical study of distance metrics for k-nearest neighbor algorithm. In *International Conference on Industrial Application Engineering*, 2015.

[30] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *CoRR*, 2016.

[31] C. Cortes and V. Vapnik. Support-vector networks. In *Machine Learning*, 1995.

[32] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory*, 1967.

[33] M. Cuturi. Fast global alignment kernels. ICML, 2011.

[34] M. Cuturi and M. Blondel. Soft-DTW: a differentiable loss function for time-series. JMLR, 2017.

[35] M. Cuturi, J.-P. Vert, O. Birkenes, and T. Matsui. A kernel for time series based on global alignments. *CoRR*, 2006.

[36] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[37] N. Dalal, B. Triggs, and C. Schmid. Human detection using oriented histograms of flow and appearance. In *ECCV*, 2006.

[38] D. DeCoste and B. Schölkopf. Training invariant support vector machines. *Machine Learning*, 2002.

[39] F. J. E. Nowak and B. Triggs. Sampling strategies for bag-of-features image classification. In *ECCV*, 2006.

[40] J. Eichhorn and O. Chapelle. Object categorization with svm: Kernels for local features. 2004.

[41] M. Eichner, M. Marin-jimenez, A. Zisserman, and V. Ferrari. 2d articulated human pose estimation and retrieval in (almost) unconstrained still images. In *IJCV*, 2012.

[42] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results.

[43] E. Fix and J. Hodges. Discriminatory analysis, non-parametric discrimination: consistency properties. Technical report, 1951.

[44] O. Friman, J. Cedefamn, P. Lundberg, M. Borga, and H. Knutsson. Detection of neural activity in functional mri using canonical correlation analysis. *Magnetic Resonance in Medicine*, 2001.

[45] K. Fukui and O. Yamaguchi. Face recognition using multi-viewpoint patterns for robot vision. 2003.

[46] K. Funahashi. On the approximate realization of continuous mappings by neural networks. In *Neural Networks*, 1989.

[47] S. N. Gatos, Basilios and G. Louloudis. ICDAR 2009 handwriting segmentation contest. In *ICDAR*, 2009.

[48] P. Gehler and S. Nowozin. Infinite kernel learning. In *Proceedings of NIPS 2008 Workshop on "Kernel Learning: Automatic Selection of Optimal Kernels"*, 2008.

[49] A. M. Geoffrion. Objective function approximations in mathematical programming. In *Mathematical Programming*, 1977.

[50] G. L. Giorgos Sfikas, Angelos P. Giotis and B. Gatos. Using attributes for word spotting and recognition in polytonic greek documents. In *ICDAR*, 2015.

[51] R. Gittins. *Canonical analysis : a review with applications in ecology*. 1985.

[52] I. J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[53] K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *ICCV*, 2005.

[54] S. Gudmundsson, T. P. Runarsson, and S. Sigurdsson. Support vector machines and dynamic time warping for time series. In *IJCNN*. IEEE, 2008.

[55] T. Hangelbroek and A. Ron. Nonlinear approximation using gaussian kernels. *Journal of Functional Analysis*, 2010.

[56] A. Hayashi, Y. Mizuhara, and N. Suematsu. Embedding time series data for classification. In *MLDM*, 2005.

[57] H. Hidalgo, S. S. León, and E. Gómez-Treviño. Application of the kernel method to the inverse geosounding problem. *Neural Networks*, 2003.

[58] K. Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. In *Neural Networks*, 1990.

[59] K. Hornik, M. Stinchcombe, H. White, and P. Auer. Degree of approximation results for feedforward networks approximating unknown mappings and their derivatives. In *Neural Computation*, 1994.

[60] H. HOTELLING. Relations between two sets of variables. *Biometrika*, 1936.

[61] P. hsuen Chen, R. en Fan, and C. jen Lin. A study on smo-type decomposition methods for support vector machines. *IEEE Transactions on Neural Networks*, 2006.

[62] L.-Y. Hu, M.-W. Huang, S.-W. Ke, and C.-F. Tsai. The distance function effect on k-nearest neighbor classification for medical datasets. *SpringerPlus*, 2016.

[63] V. G. Huaigu Cao. Vector model based indexing and retrieval of handwritten medical forms. In *ICDAR*, 2007.

[64] N. Ikizler-Cinbis and S. Sclaroff. Object, scene and actions: Combining multiple features for human action recognition. In *ECCV*, 2010.

[65] F. Itakura. Minimum prediction residual principle applied to speech. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1975.

[66] Y. Ito. Approximation capability of layered neural networks with sigmoid units on two layers. In *Neural Computation*, 1994.

[67] A. F. E. V. J. Almazán, A. Gordo. Segmentation-free word spotting with exemplar svms. In *PR*, 2014.

[68] T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In *NIPS*, 1998.

[69] C. V. Jawahar and A. Kumar. Content-level annotation of large collection of printed document images. In *ICDAR*, 2007.

[70] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *ECML*, 1998.

[71] S. Johnson and M. Everingham. Clustered pose and nonlinear appearance models for human pose estimation. In *Proceedings of the British Machine Vision Conference*, 2010. doi:10.5244/C.24.12.

[72] A. F. J. L. E. V. Jon Almazán, David Fernández Mota. A coarse-to-fine approach for handwritten word spotting in large scale historical documents collection. In *ICFHR*, 2012.

[73] Y. jye Lee and O. L. Mangasarian. Ssvm: A smooth support vector machine for classification. Technical report, 1999.

[74] C. V. J. K. Pramod Sankar. Probabilistic reverse annotation for large scale image retrieval. In *CVPR*, 2007.

[75] T. Kailath. A view of three decades of linear filtering theory. *IEEE Trans. Inf. Theory*, 1974.

[76] A. Kataria and M. Singh. A review of data classification using k-nearest neighbour algorithm. 2013.

[77] S. Ke, B. Xiao, D. Liu, and J. Wang. Deep high-resolution representation learning for human pose estimation. In *CVPR*, 2019.

[78] V. Kellokumpu, G. Zhao, and M. Pietikäinen. Human activity recognition using a dynamic texture based method. In *BMVC*, 2008.

[79] E. J. Keogh and M. J. Pazzani. Derivative dynamic time warping. In *SDM*, 2001.

[80] K. I. Kim, K. Jung, S. H. Park, and H. J. Kim. Support vector machines for texture classification. *PAMI*, 2002.

[81] T.-K. Kim and R. Cipolla. Gesture recognition under small sample size. In *ACCV*, 2007.

[82] T.-K. Kim and R. Cipolla. Canonical correlation analysis of video volume tensors for action categorization and detection. *PAMI*, 2009.

[83] T.-K. Kim, S.-F. Wong, and R. Cipolla. Tensor canonical correlation analysis for action classification. In *CVPR*, 2007.

[84] M. Kloft, U. Brefeld, S. Sonnenburg, P. Laskov, K.-R. Müller, and A. Zien. Efficient and accurate lp-norm multiple kernel learning. In *NIPS*, 2009.

[85] P. Krishnan, K. Dutta, and C. V. Jawahar. Deep feature embedding for accurate recognition and retrieval of handwritten text. In *ICFHR*, 2015.

[86] G. R. G. Lanckriet, N. Cristianini, P. L. Bartlett, L. E. Ghaoui, and M. I. Jordan. Learning the kernel matrix with semidefinite programming. *JMLR*, 2004.

[87] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *CVPR*, 2008.

[88] Q. V. Le, W. Y. Zou, S. Y. Yeung, and A. Y. Ng. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *CVPR*, 2011.

[89] Y. LeCun and F. Huang. Loss functions for discriminative training of energy-based models. In *AISTATS*, 2005.

[90] H. Li and T. Jiang. A class of edit kernels for svms to predict translation initiation sites in eukaryotic mrnas. *Journal of Computational Biology*, 2005.

[91] J. Liu, J. Luo, and M. Shah. Recognizing realistic actions from videos. In *CVPR*, 2009.

[92] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[93] D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, 1999.

[94] Y. M. Lui and J. R. Beveridge. Tangent bundle for human action recognition. In *FG*, 2011.

[95] Y. M. Lui, J. R. Beveridge, and M. Kirby. Action classification on product manifolds. In *CVPR*, 2010.

[96] D. G. L. M Muja. Fast approximate nearest neighbours with automatic algorithm configuration. In *VISS-APP*, 2009.

[97] J. A. M. Rodriguez and M. Shah. Action mach: A spatiotemporal maximum average correlation height filter for action recognition. In *CVPR*, 2008.

[98] S. Maji and A. C. Berg. Max-margin additive classifiers for detection. In *ICCV*, 2009.

[99] T. Malisiewicz, A. Gupta, and A. A. Efros. Ensemble of exemplar-svms for object detection and beyond. In *ICCV*, 2011.

[100] B. O. Masato Takami, Peter Bell. Offline learning of prototypical negatives for efficient online exemplar SVM. In *WACV*, 2014.

[101] P. Matikainen, M. Hebert, and R. Sukthankar. Trajectons: Action recognition through the motion analysis of tracked features. In *Workshop on Video-Oriented Object and Event Classification, ICCV*, September 2009.

[102] D. McAllester, T. Hazan, and J. Keshet. Direct loss minimization for structured prediction. In *NIPS*, 2010.

[103] R. Messing, C. Pal, and H. A. Kautz. Activity recognition using the velocity histories of tracked keypoints. In *ICCV*, 2009.

[104] S. P. F. D. Michael Gharbi, Tomasz Malisiewicz. A gaussian approximation of feature space for fast image similarity. In *MIT CSAIL Technical Report*, 2012.

[105] A. Morcos, M. Raghu, and S. Bengio. Insights on representational similarity in neural networks with canonical correlation. In *NIPS*, 2018.

[106] M. Müller. *Information Retrieval for Music and Motion*. 2007.

[107] C. S. Myers, L. R. Rabiner, and A. E. Rosenberg. An investigation of the use of dynamic time warping for word spotting and connected speech recognition. In *ICASSP*, 1980.

[108] G. Nagendar, S. G. Bandiatmakuri, M. G. Tandarpally, and C. V. Jawahar. Action recognition using canonical correlation kernels. In *ACCV*, 2012.

[109] G. Nagendar and C. V. Jawahar. Efficient word image retrieval using fast DTW distance. In *ICDAR*, 2015.

[110] G. Nagy. Twenty years of document image analysis in PAMI. *TPAMI*, 2000.

[111] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[112] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. Multimodal deep learning. ICML, 2011.

[113] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *ICCV*, 2015.

[114] S. Nowozin. Optimal decisions from probabilistic models: the intersection-over-union case. In *CVPR*, 2014.

[115] C. S. Ong, X. Mary, S. Canu, and A. J. Smola. Learning with non-positive kernels. In *ICML*, 2004.

[116] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: an application to face detection. In *CVPR*, 1997.

[117] R. Overill. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. 1983.

[118] F. Perronnin and C. R. Dance. Fisher kernels on visual vocabularies for image categorization. In *CVPR*, 2007.

[119] F. Petitjean, A. Ketterlin, and P. Gançarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, 2011.

[120] A. Pezeshki, M. R. Azimi-Sadjadi, and L. L. Scharf. Undersea target classification using canonical correlation analysis. *IEEE Journal of Oceanic Engineering*, 2007.

[121] B. D. Premachandran V, Tarlow D. Empirical minimum bayes risk prediction: How to extract an extra few performance from vision models with just three more parameters. In *CVPR*, 2014.

[122] M. Pritish, C. Jawahar, and K. M. Pawan. Efficient optimization for average precision svm. In *NIPS*, 2014.

[123] M. Punam and T. Nitin. Analysis of distance measures using k-nearest neighbor algorithm on kdd dataset. *IJSR*, 2015.

[124] L. Rabiner and B.-H. Juang. *Fundamentals of Speech Recognition*. 1993.

[125] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *NIPS*, 2009.

[126] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *NIPS*, 2007.

[127] M. A. Rahman and Y. Wang. Optimizing intersection-over-union in deep neural networks for image segmentation. In *ISVC*, 2016.

[128] V. Ranjan, G. Harit, and C. Jawahar. Document retrieval with unlimited vocabulary. In *WCAC*, 2015.

[129] M. Ranjbar, T. Lan, Y. Wang, S. N. Robinovitch, Z.-N. Li, and G. Mori. Optimizing nondecomposable loss functions in structured prediction. In *PAMI*, 2012.

[130] T. M. Rath and R. Manmatha. Word image matching using dynamic time warping. In *CVPR*, 2003.

[131] T. M. Rath and R. Manmatha. Word spotting for historical documents. *IJDAR*, 2007.

[132] M. Reiter, R. Donner, G. Langs, and H. Bischof. Estimation of face depth maps from color textures using canonical correlation analysis. *Czech Pattern Recognition Society*, 2006.

[133] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[134] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *ICCV*, 1998.

[135] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *IJCV*, 2000.

[136] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. 1988.

[137] V. J.-P. A. T. Saigo, Hiroto. Optimizing amino acid substitution matrices with a local alignment kernel. BMC Bioinformatics, 2006.

[138] P. Saisan, G. Doretto, Ying Nian Wu, and S. Soatto. Dynamic texture recognition. In *CVPR*, 2001.

[139] H. Sakoe. Dynamic programming algorithm optimization for spoken word recognition. 1978.

[140] S. Salvador and P. Chan. Toward accurate dynamic time warping in linear time and space. *Intell. Data Anal*, 2007.

[141] B. Sapp and B. Taskar. Modec: Multimodal decomposable models for human pose estimation. In *CVPR*, 2013.

[142] C. Schüldt, I. Laptev, and B. Caputo. Recognizing human actions: A local svm approach. In *ICPR*, 2004.

[143] D. Schultz and B. J. Jain. Nonsmooth analysis and subgradient methods for averaging in dynamic time warping spaces. *CoRR*, 2017.

[144] G. L. Scott and H. C. Longuet-Higgins. An algorithm for associating the features of two images. *Proceedings. Biological sciences*, 1991.

[145] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *ICML*, 2007.

[146] H. Shimodaira, K. ichi Noma, M. Nakai, and S. Sagayama. Dynamic time-alignment kernel in support vector machine. In *NIPS*, 2001.

[147] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *ICCV*, 2003.

[148] D. Skočaj and A. Leonardis. Appearnce-based localization using cca. In *CVWW*, 2004.

[149] A. J. Smola, Z. L. Óvári, and R. C. Williamson. Regularization with dot-product kernels. In *NIPS*, 2000.

[150] Y. Song, A. Schwing, Richard, and R. Urtasun. Training deep neural networks via direct loss minimization. In *ICML*, 2016.

[151] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[152] B. Squires. Automatic speaker recognition: An application of machine learning. In *ICML*, 1995.

[153] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 2014.

[154] N. Srivastava and R. R. Salakhutdinov. Multimodal learning with deep boltzmann machines. In *NIPS*, 2012.

[155] S. Sudholt and G. A. Fink. Phocnet: A deep convolutional neural network for word spotting in handwritten documents. In *ICFHR*, 2016.

[156] J. Sun, X. Wu, S. Yan, L. F. Cheong, T.-S. Chua, and J. Li. Hierarchical spatio-temporal context modeling for action recognition. In *CVPR*, 2009.

[157] M. J. Swain and D. H. Ballard. Color indexing. *IJCV*, 1991.

[158] T. T.-C. T. Nguyen-Thien a. Approximation of functions and their derivatives: A neural network implementation with applications. In *Applied Mathematical Modelling*, 1999.

[159] D. Tarlow and R. Zemel. Structured output learning with high order loss functions. In *AISTATS*, 2012.

[160] K. N.-I. P. S. T. S. J. P. Thomas Konidaris, Basilios Gatos. Keyword-guided word spotting in historical printed documents using synthetic data and user feedback. In *IJDAR*, 2007.

[161] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient object localization using convolutional networks. In *CVPR*, 2015.

[162] R. M. Toni M. Rath. Features for word spotting in historical manuscripts. In *ICDAR*, 2003.

[163] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. In *CVPR*, 2014.

[164] M. M. Ullah, S. N. Parizi, and I. Laptev. Improving bag-of-features action recognition with non-local cues. In *BMVC*, 2010.

[165] V. Vapnik. An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, 1999.

[166] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *PAMI*, 2012.

[167] P. Vincent and Y. Bengio. K-local hyperplane and convex distance nearest neighbor algorithms. Technical report, 2001.

[168] M. N. Volkovs and R. S. Zemel. Boltzrank: Learning to maximize expected ranking gain. In *ICML*, 2009.

[169] C. Wallraven, B. Caputo, and A. B. A. Graf. Recognition with local features: the kernel recipe. In *ICCV*, 2003.

[170] H. Wang, A. Klaser, C. Schmid, and C.-L. Liu. Action recognition by dense trajectories. CVPR, 2011.

[171] H. Wang, M. M. Ullah, A. Kläser, I. Laptev, and C. Schmid. Evaluation of local spatio-temporal features for action recognition. In *BMVC*, 2009.

[172] J. Wang, Z. Chen, and Y. Wu. Action recognition with multiscale spatio-temporal contexts. In *CVPR*, 2011.

[173] L. Wolf and A. Shashua. Kernel principal angles for classification machines with applications to image sequence interpretation. In *CVPR*, 2003.

[174] L. Wolf and A. Shashua. Learning over sets using kernel principal angles. *JMLR*, 2003.

[175] X. Xi, E. J. Keogh, C. R. Shelton, L. Wei, and C. A. Ratanamahatana. Fast time series classification using numerosity reduction. In *ICML*, 2006.

[176] B. Xiao, H. Wu, and Y. Wei. Simple baselines for human pose estimation and tracking. In *ECCV*, 2018.

[177] I. Z. Yalniz and R. Manmatha. An efficient framework for searching text in noisy document images. In *DAS*, 2012.

[178] O. Yamaguchi, K. Fukui, and K. Maeda. Face recognition using temporal image sequence. FG, 1998.

[179] J. Yang, D. Zhang, A. F. Frangi, and J.-Y. Yang. Two-dimensional pca: A new approach to appearance-based face representation and recognition. *PAMI*, 2004.

[180] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR*, 2007.

[181] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

[182] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *CVPR*, 2016.

[183] F. Zhou, F. De la Torre, and J. F. Cohn. Unsupervised discovery of facial events. In *CVPR*, 2010.