UNIVERSITY OF KWAZULU-NATAL

COLLEGE OF AGRICULTURE, ENGINEERING AND SCIENCE

Scale Coding a Bag of Words for Real-Time Video-based Action Recognition

By:

Divina Govender

215023704

Supervisor:

Prof. Jules-Raymond Tapamo

In fulfillment of the academic requirements of the degree of Master of Science in Engineering, School of Engineering, University of KwaZulu-Natal

February 7, 2021

EXAMINER'S COPY

Preface

The research described in this thesis was performed at the University of KwaZulu-Natal under the supervision of Prof. Jules-Raymond Tapamo. I hereby declare that all materials incorporated in this thesis are my own original work except where the acknowledgement is made by name or in the form of reference. The work contained herein has not been submitted in part or whole for a degree at any other university.

Divina Govender

As the candidate's Supervisor I agree to the submission of this dissertation.

Jules-Raymond Tapamo

Declaration 1 - Plagiarism

I, Divina Govender, declare that

1. The research reported in this thesis, except where otherwise indicated, is my original research.

2. This thesis has not been submitted for any degree or examination at any other university.

3. This thesis does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.

4. This thesis does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

   a. Their words have been re-written but the general information attributed to them has been referenced

   b. Where their exact words have been used, then their writing has been placed in italics and inside quotation marks, and referenced.

5. This thesis does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the thesis and in the References sections.

Divina Govender

Declaration 2 - Publications

Details of contribution to publications that form part and/or include research presented in this dissertation:

1. D. Govender, J.R. Tapamo, ”Factors Affecting the Cost to Accuracy Balance for Real-Time Video-Based Action Recognition”, *Lecture Notes in Computer Science*, vol. 12249, pp. 807-818, October 2020, doi: 10.1007/978-3-030-58799-4_58.

2. D. Govender and J. R. Tapamo, ”Spatio-Temporal Scale Coded Bag-of-Words”, *Sensors*, vol. 20, no. 21, pp. 6380, 2020, doi: 10.3390/s20216380.

Divina Govender

Acknowledgments

I would like to thank my supervisor, Prof. Jules-Raymond Tapamo, for his invaluable guidance, support and feedback through each stage of my research.

I would also like to thank my family for their unwavering encouragement and support.

Abstract

The Bag-of-Words (BoW) framework has been widely used in action recognition tasks due to its compact and efficient feature representation. Various modifications have been made to this framework to increase its classification power. Increased classification power often results in increased complexity and reduced efficiency, highlighting the compromise between computational cost and accuracy. This cost-to-accuracy balance must be carefully considered to achieve real-time video-based action recognition.

Inspired by the success of image-based scale coded BoW representations, this research investigates the potential of scale coded BoW representations for implementable real-time video-based action recognition. Scale coding a BoW involves encoding extracted multi-scale information into BoW representations by partitioning spatio-temporal features into sub-groups based on the spatial scale from which they were extracted. For video-based action recognition, a spatio-temporal Bag-of-Words (SC-BoW) was proposed and experimentally evaluated.

Results showed SC-BoW representations to improve performance by 2% - 7% with a low added computational cost. Notably, SC-BoW with dense trajectory features outperformed more complex deep learning approaches. Thus, scale coding is a low-cost and low-level encoding scheme that increases the classification power of the standard BoW without compromising efficiency. Furthermore, the incorporation of scale-information incurs a negligible additional computational cost, presenting an ideal cost-to-accuracy balance. Thus, scale-coding a BoW has great potential to be suitable for accurate real-time video-based action recognition. Future work includes scale-coding deep features to exploit the desirable cost-to-accuracy trade-off of SC-BoW.

Contents

1 Introduction **1**

1.1 Background . . . 1

1.2 Motivation and Problem Definition . . . 2

1.3 Goals and Objectives . . . 3

1.4 Contributions . . . 3

1.5 Dissertation Outline . . . 4

2 Literature Review **6**

2.1 Introduction . . . 6

2.2 Bag-of-Words . . . 7

2.2.1 Standard Scale Invariant Bag-of-Words Model . . . 8

2.2.2 Analysis of Bag-of-Words . . . 9

2.3 Bag-of-Words for Video Data . . . 9

2.3.1 Feature Extraction . . . 10

2.3.2 Visual Vocabulary Generation . . . 13

2.3.3 Feature Pre-processing . . . 14

2.3.4 Feature Encoding . . . 15

2.3.5 Pooling and Normalization . . . 16

2.4 Bag-of-Words Variants . . . 17

2.4.1 Convolutional Bag-of-Features . . . 18

2.4.2 Scale Coding . . . 19

2.5 Visual Tracking . . . 22

2.5.1 KCF Tracker . . . 23

2.6 Conclusion . . . 26

3 Computational Cost to Accuracy Balance 28

3.1 Introduction 28

3.2 Computational Cost vs Accuracy Balance 29

3.3 Factors Affecting Cost vs Accuracy Balance 30

3.3.1 Sampling 30

3.3.2 Optical Flow 32

3.3.3 Saliency 33

3.3.4 Flexibility 33

3.4 Experimentation 34

3.4.1 PC Specifications 34

3.4.2 Datasets 34

3.4.3 Dense Trajectories 35

3.4.4 Evaluation of Factors 38

3.5 Results and Discussion 39

3.5.1 Descriptors 39

3.5.2 Refresh Rate 42

3.5.3 Optical Flow 42

3.6 Conclusions 43

4 Spatio-Temporal Scale Coded Bag-of-Words 44

4.1 Introduction 44

4.2 Criteria for Real-Time Action Recognition 45

4.3 General Approach 46

4.3.1 Object Detection 47

4.3.2 Visual Tracker 49

4.3.3 Multi-Scale Sampling 53

4.3.4 Scale Coding 56

4.4 Experimentation 60

4.4.1 PC Specifications and Datasets 60

4.4.2 Object Detection 60

4.4.3 Visual Tracker 61

4.4.4 Bag of Words 62

4.4.5 Classification 62

4.5 Results and Discussion . . . 65
4.5.1 Scale Partitioning Strategies . . . 65
4.5.2 Scale Coding Schemes . . . 65
4.5.3 Computational Cost . . . 66
4.5.4 Comparison to Existing Methods . . . 67
4.6 Conclusions . . . 67

5 Scaled Spatio-Temporal Representations 70
5.1 Introduction . . . 70
5.2 Adapted Spatio-Temporal Scale Coding . . . 71
5.3 Scaled Spatio-Temporal Pyramids . . . 72
5.4 Experimentation . . . 74
5.4.1 Datasets . . . 74
5.4.2 Scale Coding Dense Trajectory Features . . . 75
5.5 Results and Discussion . . . 77
5.5.1 Performance Analysis . . . 77
5.5.2 Number of Scale Partitions . . . 81
5.5.3 Comparison to Existing Methods . . . 82
5.6 Conclusions . . . 83

6 Conclusions and Future Work 85
6.1 Scale Coding Best Practice . . . 86
6.2 Future Work . . . 87

A Additional Results 98
A.1 Reduced HMDB51 Dataset . . . 98
A.2 KTH Dataset . . . 100

List of Tables

2.1 The output vector dimension for vector quantization, Fisher Vector and VLAD encoding 15

3.1 Dimensions of the descriptors extracted for dense trajectory features . . 37

3.2 The effects of Trajectory Shape [1], HOG [2], HOF [2] and MBH [3] descriptors on the cost vs accuracy balance 40

3.3 Evaluation of the effect of the refresh rate on the cost vs accuracy balance for KTH and HMDB51 datasets using the Dense Trajectory features . . 41

4.1 Evaluation of cost-vs-accuracy for the KCF [4], DCF [5] and MOSSE [6] visual trackers 50

4.2 Parameters for the DCF-DSST Visual Tracker 61

4.3 Performance of the Spatio-Temporal Scale Coded BoW on the KTH dataset 64

4.4 Performance of the Spatio-Temporal Scale Coded BoW on the HMDB51 dataset with Dynamic Scale Parameter Definition 64

4.5 The Computational Cost of Each Task in the Spatio-Temporal Scale Coded BoW for the KTH and HMDB51 Datasets 66

4.6 Comparison to Existing Action Recognition Methods on the KTH dataset 69

4.7 Comparison to Existing Action Recognition Methods on the HMDB51 dataset 69

5.1 Computational Analysis of SC-BoW given a 4k-word vocabulary and 6000 extracted features 77

5.2 Comparing Performance of Dense Trajectories with and without scale coding for the KTH and reduced HMDB51 datasets 78

5.3 The Effect of the Number of Scale Partitions on Accuracy 81

5.4 Comparison to Existing BoW Action Recognition Methods on the KTH dataset 84

List of Figures

2.1 The pipeline for video-based action recognition using the BoW Framework. 10

2.2 Space-Time Interest Point detection of a walking person. Taken from [7]. 11

2.3 Visualization of the extraction of Dense Trajectories. Taken from [1]. . . 12

2.4 Absolute Scale Coding. Taken from [8]. 20

2.5 Relative Scale Coding. Taken from [8]. 21

3.1 HOG, HOF and MBH descriptors are computed along the trajectory in a $N \times N$ pixel neighborhood divided into $n_T \times n_\sigma \times n_\sigma$ cells. 37

3.2 The spatio-temporal grids adapted from [1]. Top row from left to right: $h1 \times v1 \times t1$, $h3 \times v1 \times t1$, $h2 \times v2 \times t1$. Bottom row from left to right: $h1 \times v1 \times t2$, $h3 \times v1 \times t2$, $h2 \times v2 \times t2$. 38

3.3 The computed optical flow for each spatial scale 43

4.1 The general pipeline for video-based action recognition with scale coded BoW. 46

4.2 Visualization of constructing the samples to update and train the scale filter [9] 52

4.3 Generation of HOG features: For each block, the HOG feature vectors for the highlighted $k \times k$ cell are sum pooled and divided by the $\ell 2$-norm to form a normalized HOG feature histogram. 56

4.4 Temporal pyramid for scale coded BoW. 60

4.5 Successful definition of the initial bounding box, B_0, for video samples from the a) 'brush hair' class of HMDB51 [10] b) 'hand waving' class of KTH [11]. 61

4.6 5-fold cross validation of the training set for hyper-parameter tuning. . . 63

5.1 The general pipeline for formation of scale coded BoW representations. 71

5.2 Scaled Spatio-Temporal Pyramids: (**a**)The first representation involves computing a SC-BoW for each cell. (**b**)The second representation adds scale as a 4th dimension and involves computing a standard BoW for each cell. 74

5.3 Scaled Spatio-Temporal grids: on each scale, 6 spatio-temporal pyramids are constructed as indicated. 76

5.4 Plot comparing the class accuracies obtained on the KTH dataset for dense trajectories and scale coded dense trajectories. 79

5.5 Plot comparing the class accuracies obtained on the reduced HMDB51 dataset for dense trajectories and scale coded dense trajectories. 79

5.6 Confusion matrix plot of the KTH dataset (R = 1) where (a) dense trajectory features were extracted. (b) scale coded dense trajectory features were extracted. 80

5.7 Confusion matrix plot of the reduced HMDB51 dataset (R = 6) where (a) dense trajectory features were extracted. (b) scale coded dense trajectory features were extracted. 80

5.8 The effect of number of scale partitions on accuracy for the KTH and HMDB51(reduced) datasets. 82

5.9 The effect of number of scale partitions on computational cost for the KTH dataset. 83

A.1 Confusion matrix plot of the reduced HMDB51 dataset with percentage accuracy. Dense trajectory features were extracted with a sub-sampling refresh rate of R = 6 frames. 98

A.2 Confusion matrix plot of the reduced HMDB51 dataset with percentage accuracy. Scale coded dense trajectory features were extracted with a sub-sampling refresh rate of R = 6 frames. 99

A.3 Confusion matrix plot of the KTH dataset. Dense trajectory features were extracted with a sub-sampling refresh rate of R = 6 frames. 100

A.4 Confusion matrix plot of the KTH dataset with percentage accuracy. Dense trajectory features were extracted with a sub-sampling refresh rate of R = 6 frames. 100

A.5 Confusion matrix plot of the KTH dataset with percentage accuracy. Dense trajectory features were extracted with a sub-sampling refresh rate of R = 1 frames. 101
A.6 Confusion matrix plot of the KTH dataset with percentage accuracy. Scale coded dense trajectory features were extracted with a sub-sampling refresh rate of R = 1 frames. 101

List of Algorithms

1 KCF Tracker [5] 24

2 Person Object Isolation after Object Detection 48

4 Definition of the Spatial Scales 53

3 DCF-DSST Tracker: Iteration at frame t 54

5 Dynamic Scaling Parameter Definition 59

List of Abbreviations

BoE	Bag-of-Expressions
BoF	Bag of Features
BoVW	Bag of Visual Words
BoW	Bag-of-Words
CBoF	Convolutional Bag of Features
CBoW	Contextual Bag-of-Words
CNN	Convolutional Neural Network
DCF	Dual Correlation Filter
DFT	Discrete Fourier Transforms
DoG	Difference of Gaussian
DSST	Discriminative Scale Space Tracker
FV	Fisher Vector
GMM	Gaussian Mixture Model
HOF	Histogram of Optical Flow
HOG	Histogram of Orientated Gradients
KCF	Kernalized Correlation Filter
MBH	Motion Boundary Histogram
MOSSE	Minimum Output Sum of Squared Error
PCA	Principle Component Analysis
RBF	Radial Basis Function
R-CNN	Region-based Convolutional Neural Network
RF	Random Forests
SC-BoW	Spatio-Temporal Scale Coded Bag-of-Words

SIFT	Scale Invariant Feature Transforms
SSDs	Single Shot Detectors
STIPs	Space-Time Interest Points
SVM	Support Vector Machine
VLAD	Vector of Locally Aggregated Descriptors
YOLO	You Only Look Once

Chapter 1

Introduction

1.1 Background

Action recognition aims to autonomously classify what is being done by observable agents in a scene. Video-based action recognition is one of the most challenging problem areas in computer vision. Action recognition is dependent on a multitude of high-level visual clues (e.g., human pose, interacting objects, and scene class) - the identification of which is already a complicated task. Furthermore, significant intra-class variations occur due to changes in illumination, viewpoints, scale, and movement speed. Additionally, there is no precise definition of the temporal extent of the action (when the action begins and ends) - a problem not faced in static object-based action recognition. Finally, the high dimension and poor quality of video data add to the challenge of developing robust and efficient recognition algorithms [12].

After video capturing, the first step of any action recognition task is to extract features from the given data. For video-based action recognition, popular features include Dense Trajectories [1, 13, 14], Space-Time Interest Points (STIPs) [7], and Scale Invariant Feature Transforms (SIFT) [15, 16]. Various approaches use the Bag of Words (BoW) framework to compactly present the extracted features for classification [1, 12, 17].

The general approach of this framework involves clustering locally extracted features to form a vocabulary ("bag") of visual "words" [18]. Extracted features are compared to this vocabulary to generate a frequency histogram. The histogram holds the count of occurrences of each visual "word" in the image/video. This histogram is the BoW representation of the image/video.

The BoW is a favored approach due to its simplicity, flexibility, and computational efficiency. As a result, it has great potential for real-time video-based action recognition. However, the standard framework lacks structure and discards all large-scale spatial information which reduces classification power. Thus, many BoW variants have been developed; aiming to increase classification power while leveraging the simplicity and computational efficiency of the original framework [8, 12, 18–20]. This highlights the computational cost vs. accuracy trade-off that needs to be carefully considered for real-time video-based action recognition. The BoW framework and its variants are formally discussed in Section 2.2.

1.2 Motivation and Problem Definition

Successful implementation of a real-time video-based action recognizer increases the potential for real-time autonomous surveillance; this has various useful real-world applications. For example: autonomously finding and reporting instances of theft in public spaces, real-time surveillance, intelligent surveillance, and smart shopping. Currently, state-of-the-art action recognition algorithms revolve around two-stream Convolutional Neural Networks (CNNs) [21]. However, these high performing recognition schemes often have high computational demands, making them unsuitable for real-world application. Furthermore, deep learning algorithms require large amounts of labeled training data (which may not be readily available in real-world circumstance), long training times and have high hardware demands. To successfully apply action recognition in real-world circumstances, algorithms that increase performance with low computational demands are valuable.

The simple and compact representation offered by the BoW framework is beneficial in action recognition tasks, which involve large sets of extracted features even for relatively small datasets [1]. However, the traditional BoW model lacks discriminating power when dealing with large amounts of data. This is a notable setback in the field of action recognition, which requires a large number of features to be extracted for unsupervised visual vocabulary creation [22]. Furthermore, the standard BoW framework discards all large-scale spatial information, including relative locations, scales, and orientations of the features [23]. Although this makes the framework robust to varying appearances caused by the factors mentioned above, Khan *et al.* showed that scale variances within an image hold vital information and that purely scale-invariant representations are sub-optimal [20]. The possibility of extending scale encoded BoW representations to the spatio-temporal domain (for visual tracking and video-based action recognition) is to be explored. This simple approach has the potential to overcome the cost-to-accuracy trade-off, moving society closer to widespread practical realization and implementation of autonomous surveillance systems.

1.3 Goals and Objectives

The ultimate goal of this research is to investigate scale coding a BoW as a possible avenue for accurate real-time action recognition of video data.

Notably, the study has the following sub-objectives:

- Conduct a literature review of the existing action recognition algorithms and tools, focusing on the Bag of Words framework and its subsequent variants.
- Investigate the relationship between algorithm performance and computational cost.
- Propose an efficient framework for video-based action recognition that incorporates scale information into the final BoW representation of a given video.
- Experimentally evaluate the proposed framework on standardized datasets (specifically the KTH and HMDB51 datasets) and compare performance to existing action recognition algorithms to determine the validity and relevance of the proposed algorithm in the current research space.

1.4 Contributions

This work contributes to the research space in the following manners:

- Identified the factors affecting the computational cost vs. accuracy balance for video-based action recognition. Furthermore, the practical effects of these factors are observed through experimental evaluation. This evaluation provides a basis for assessing and improving the efficiency of existing algorithms by stating the factors to consider when developing or adjusting existing algorithms for real-time action recognition of videos.

- Proposed a novel video-based action recognition algorithm by extending the scale encoded BoW for image-based action recognition to the spatio-temporal domain. This was found to improve the initial accuracy of extracted features.
- Proposed and evaluated parameter definition frameworks for the scale encoded BoW, namely: static and dynamic scale parameter definition.
- The scale encoded BoW framework was adjusted to be easily incorporated into any existing video-based action recognition algorithm that uses a BoW for classification. Furthermore, a novel approach to spatio-temporal pyramids was proposed; extending the concept to 4 dimensions to include scale information. An impressive improvement in accuracy was recorded. Most notably, the added computational cost of this framework is negligible. The relationship between accuracy and the key parameters of this novel framework were also investigated.

1.5 Dissertation Outline

- Chapter 2 presents a literature review covering the Bag of Words framework and its existing variants for action recognition. An analysis of this framework and its variants is conducted, highlighting the trade-off between computational cost and accuracy. Pertinent visual tracking methodologies are also reviewed.
- In Chapter 3, factors influencing the cost vs. accuracy trade-off are identified. The practical effects of these factors are experimentally evaluated using Dense Trajectory features and the KTH and a reduced HMDB51 dataset.
- In Chapter 4, a novel approach that extends Khan *et al.*'s scale coded BoW to the spatio-temporal domain is proposed and experimentally evaluated.
- Chapter 5 presents a scale coded BoW framework that can be easily incorporated into any existing video-based action recognition algorithm that uses a BoW for classification. Furthermore, the relevance of scale information is investigated; the

adjusted scale coded BoW framework is used to incorporate scale information into the more robust dense trajectory feature set.

- Chapter 6 concludes this work and outlines future work directions.

Chapter 2

Literature Review

2.1 Introduction

Video-based action recognition is currently one of the most active spaces in computer vision research. Initial approaches focused on holistic representations of an action. This involved identifying a set of 2-D [24, 25] or 3-D [26] points located at the joints of the person object. These points were tracked, thereby tracing the motion of the action via space-time trajectories. Detection and tracking individual body parts still remains a difficult task to accomplish for realistic video data. Thus, the research space shifted focus onto extracting local spatio-temporal features [1, 7, 15, 16] for video classification. Formation and extraction of spatio-temporal features extracts patterns of the changing 3-D plane as opposed to handling the action as a rigid 3-D object. As a result, the challenge of detecting and modeling the human body object was avoided. To compactly represent extracted features for classification, the BoW framework was often used.

Current state-of-the-art video-based action recognition algorithms are dominated by deep learning methodologies: CNNs are used to effectively extract spatio-temporal information from videos [21, 27] and for classification [17, 28, 29]. Current trends revolve around two-stream CNNs [27, 30] which fuses together two CNNs for spatial and temporal information respectively. However, deep learning architectures are complex, have high computational demands and require large amounts of training data. As a result, machine learning algorithms can be more advantageous for small datasets such as the KTH dataset. This was confirmed by Aslan *et al.* [31] who achieved 95.33 % on the KTH dataset using a BoW and machine learning methods. Contrarily, the deep learning algorithm by Baccouche *et al.* [32] achieved 94.39 % on the same dataset. This was further proved in the conclusions of Chapter 5.

In this literature review, action recognition with the BoW framework and its variants are explored. Deep learning approaches were not thoroughly reviewed as the focus of study is on investigating video-based action recognition with BoW representations. Additionally, visual trackers are investigated in Section 2.5. Scale coding is reliant on predefined bounding boxes. To maintain bounding boxes though subsequent frames, an efficient and accurate visual tracker is required.

2.2 Bag-of-Words

The BoW framework, initially developed for text categorization applications, has been adopted in visual tracking and action recognition tasks to form the Bag of Visual Words (BoVW) framework [12, 18, 33]. This is alternatively referred to as the Bag of Features (BoF) framework due to the clustering of extracted local features [34]. In recent years, use of this framework and its variants have dominated the research space of visual-based action recognition [18, 20, 33, 35, 36]. Although algorithms such as CNNs [23] and Fisher Vectors (FVs) [37] have higher classification power, BoW is favored due to its flexibility, simplicity, compact feature representation and computational efficiency [36]. This is useful in action recognition tasks which involves large sets of extracted features [1]. Additionally, a fast, compact feature representation algorithm makes the BoW framework favorable for real-time applications.

The general process of this framework can be summarized in 3 steps:

1. **Construction of Visual Vocabulary**:
   Features from all training set images are extracted and then clustered to a visual vocabulary typically using K-Means Clustering or a Gaussian Mixture Model (GMM) [12]; each cluster represents a visual word or term.

2. **Histogram Generation**:
   Features extracted from a particular image are assigned to the closest words in the visual vocabulary using relation algorithms such as Nearest Neighbors.

The counts of each word that appear in the input data are recorded to create a normalized histogram. This represents the term vector. This term vector is the BoW representation of the image.

3. **Classification**:

   Based on the output term vector, the input data is classified. A commonly used classifier is the Support Vector Machine (SVM) with a non-linear kernel [22].

The output term vector discards all large-scale spatial information, including relative locations, scales, and orientations of the features [38]. In this manner, the framework is robust to varying appearances (caused by the aforementioned factors) of images of the same class. Additionally, this allows for compact and computationally efficient data representations.

2.2.1 Standard Scale Invariant Bag-of-Words Model

This section expands on the mathematical description of the standard bag-of-words pipeline [8]. Features are extracted from some image or video frame via multi-scale sampling. For a given bounding box B, the set of extracted features are defined as:

$$F(B) = \{\mathbf{f}_i^s | i \in \{1, ..., N\}, s \in \{1, ..., M\}\}, \tag{2.1}$$

Where:
i indexes the N feature sites defined by the sampling grid in bounding box B
s indexes the M scales extracted at each feature site

The histogram $h(B)$ representing a given bounding box B as per the BoW framework is given by:

$$h(B) \propto \sum_{i=1}^{N} \sum_{s=1}^{M} c_{BOW}(\mathbf{f}_i^s) \tag{2.2}$$

Where:
c_{BOW} is some coding scheme that maps the input feature space to the representation space

With the standard BoW, the Euclidean distance between each extracted feature $\mathbf{f}_i^s \in F(B)$ and visual word $\mathbf{w}_k, k \in \{1, ..., q\}$ in the visual vocabulary $W = \{\mathbf{w}_1, ..., \mathbf{w}_q\}$ is computed. The features are then matched to the nearest visual word (nearest neighbor assignment). The index of the visual word assigned to an extracted feature $\mathbf{f}_i^s$ is given by:

$$\omega_i^s = \underset{k \in \{1,\ldots,q\}}{\operatorname{argmin}}\ d(\mathbf{f}_i^s, \mathbf{w}_k) \tag{2.3}$$

Where:

$d(\mathbf{a}, \mathbf{b})$ is the function computing the Euclidean distance between $\mathbf{a}$ and $\mathbf{b}$

The coding function for the standard BoW framework is summarized by Equation (2.4)

$$c_{BOW}(\mathbf{f}_i^s) = \mathbf{e}(\omega_i^s) \tag{2.4}$$

Where:

$\mathbf{e}(i)$ is a 1-D vector of length q with only one non-zero element at index i which is equal to 1. The index i corresponds to the assigned codeword for a given extracted feature $\mathbf{f}_i^s$.

2.2.2 Analysis of Bag-of-Words

The standard BoW representation of an image or video discards all large-scale spatial information (relative locations, scales, and orientations of extracted features) [38]. This strategy justifies the compact and computationally efficient nature of the BoW framework. However, the exclusion of this information reduces the classification power of the framework. Shi *et al.* [37] showed that FVs outperform BoW representations as FVs offer a more complete representation of the dataset by including information regarding the samples' distribution with respect to words in the vocabulary. Similarly, Loussaief *et al.* [23] found that CNNs outperform the BoW framework in classification tasks due to their superior ability to extract features that contain relevant information from the input data. However, FVs and CNNs are dense and high-dimensional, making them computationally expensive. To address this concern, many variants of the BoW framework exist. The implemented modifications in BoW variants work to overcome the weaknesses of the standard BoW model.

2.3 Bag-of-Words for Video Data

For video-based action recognition, Peng *et al.* [12] defined the general pipeline for the classification algorithm using the BoW framework. This is summarized in Figure

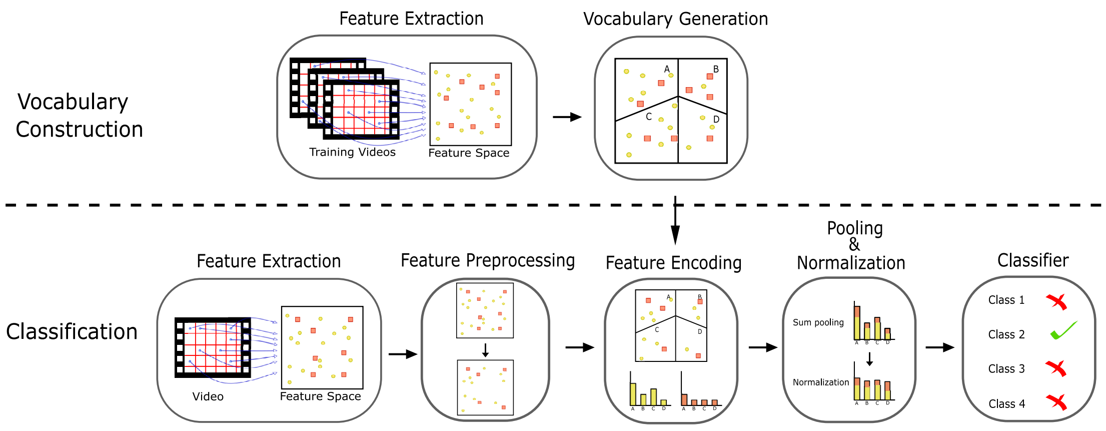


Figure 2.1: The pipeline for video-based action recognition using the BoW Framework.

2.1 [12]. To increase classification power of the standard BoW framework, various efforts and modifications have been made at each step of this pipeline. Peng *et al.* [12] concluded that each step contributes to overall performance.

2.3.1 Feature Extraction

Feature extraction consists of a detector and descriptor. Detectors select key point locations and scales in a video by maximizing some function or using a sampling strategy. Descriptors describe various attributes about the selected location. Commonly extracted descriptors are Histogram of Orientated Gradients (HOG) [2] and Histogram of Optical Flow (HOF)[2]. These descriptors are computed by quantizing pixel gradient information (HOG) and optical flow information (HOF) into histograms.

Current feature extraction approaches for video-based action recognition can be divided into two categories: hand-crafted features and deep learning mechanisms. Hand-crafted feature approaches involve manually processing raw pixels to formulate feature sets. Deep learning methods automatically obtain features [21, 27].

The accuracy of the BoW algorithm is dependent on the way extracted spatial and temporal information is represented i.e. how motion features are crafted. Thus, earlier works focused on postulating powerful spatio-temporal features. Popular hand-crafted features include: Dense Trajectories [1, 13, 14], Space-Time Interest Points (STIPs) [7] and Scale Invariant Feature Transforms (SIFT) [39].

Space-Time Interest Points

Laptev extended spatial interest points for still image-based classification to the spatio-temporal domain for video-based classification to form STIPs [7]. Detection of STIPs is illustrated in Figure 2.2. STIPs involve the detection of spatio-temporal corners defined by local areas of large variation in all three dimensions: the 2-D spatial plane, (x, y), and the 1-D temporal plane, t. These spatio-temporal corners are identified as the local maxima of a corner detection function computed for all pixels across the spatial and temporal scales. The corner detection function is derived by extending the Harris-corner operator [40] to time. This is defined as

$$\begin{aligned} H &= det(\mu) - ktrace^3(\mu) \\ &= \lambda_1\lambda_2\lambda_3 - k(\lambda_1 + \lambda_2 + \lambda_3)^3 \end{aligned} \tag{2.5}$$

Where:
μ is the spatio-temporal second-moment matrix
k is a constant. Laptev [40] set $k \approx 0.005$.
λ_1, λ_2, and λ_3 are the eigenvalues of μ. Spatio-temporal regions with significant eigenvalues mark an interest point.

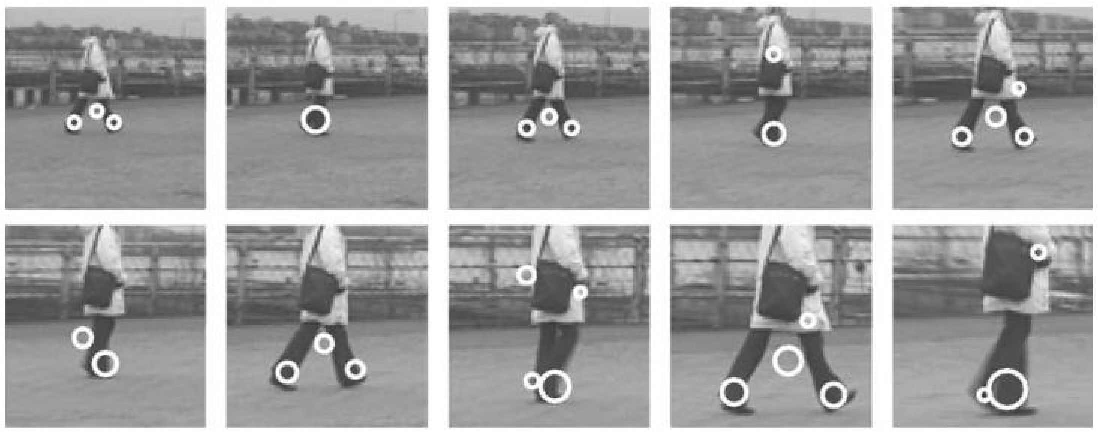


Figure 2.2: Space-Time Interest Point detection of a walking person. Taken from [7].

Dense Trajectories

The extraction of dense trajectories involves densely sampling points at multiple spatial scales. Thereafter, sampled points are tracked via an optical flow algorithm to form spatio-temporal trajectories. This is visualized in Figure 2.3. For feature formation, dense trajectories fuse together five descriptors: trajectory shape, HOG, HOF and Motion Boundary Histograms (MBH) in the horizontal (MBHx) and vertical (MBHy) planes [1]. Features are compactly represented for classification using the BoW framework.

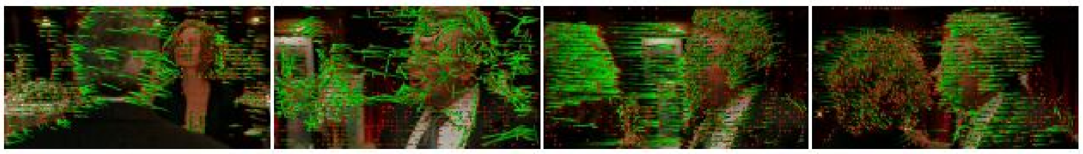


Figure 2.3: Visualization of the extraction of Dense Trajectories. Taken from [1].

Scale-Invariant Feature Transform

SIFT is a popular feature to extract for visual classification tasks as it is robust to scale, rotation, viewpoint, and illumination changes, is distinctive and efficient. Formation of a set of SIFT features for a given image/video frame is summarized in four steps [39]:

1. **Scale-space Peak Selection**
   Difference of Gaussian (DoG) images are computed in order to efficiently search over all scales and image locations to identify potential interest points. These interest points are identified as the local extrema of the DoG image.

2. **Key-point Localization**
   The stability of all potential interest points is evaluated by fitting points to a detailed model. Points that lie along edge or have low contrast are deemed to be unstable. Unstable points are sensitive to noise and do not contain useful information, thus, are removed.

3. **Orientation Assignment**
   Orientations are assigned to key-points based on local image gradient directions. As a result, the key-point descriptor orientation is represented relative to the image orientation. This makes SIFT features invariant to image rotations.

4. **Key-point Descriptor**
   In the previous steps, relative location, scale, and orientation was assigned to each valid key-point. These parameters were assigned based on locally extracted

image information; achieving invariance to the aforementioned parameters. In this step, a descriptor for the local image region about each key-point must be computed such that the descriptor is distinctive and invariant to the changes in illumination and viewpoint. A 16×16 window around the key-point is divided into 4×4 cells. An 8 bin HOG vector is computed for each cell. The resulting HOG vectors for each cell in the window is concatenated and normalized. To achieve illumination independence, a threshold of 0.2 is set. Any bin value in the HOG feature vector greater that 0.2 is set to 0.2. Thereafter, the feature vector is once again normalized.

2.3.2 Visual Vocabulary Generation

In this step, features are extracted from training data to form a codebook or visual vocabulary. Two common approaches to this step are feature space partitioning and feature probability distribution.

Feature Space Partitioning

Extracted features are clustered together; the center of each cluster represents a "code-word" or "visual word". Commonly used vector quantization techniques for this step include k-means clustering [41], hierarchical clustering [42] and Random Forests (RF) [43, 44]. The simplicity of k-means makes it one of the most popular choices for clustering tasks.

Consider a set of training features, $\{\mathbf{f}_1, \ldots, \mathbf{f}_q, \ldots, \mathbf{f}_Q\}$, where $\mathbf{f}_q \in \mathbb{R}^D$. The aim is to partition the feature space into K clusters, $\{\mathbf{d}_1, \ldots, \mathbf{d}_k, \ldots, \mathbf{d}_K\}$, where $\mathbf{d}_k \in \mathbb{R}^D$. For each feature, $\mathbf{f}_q$, there is a set of binary indicator variables $r_{qk} \in \{1, 0\}$. For a feature vector, $\mathbf{f}_q$, assigned to the cluster k, the variables are set as: $r_{qk} = 1$ and $r_{qj} = 0$ for $j \neq k$. To perform k-means clustering, values for r_{qk} and $\mathbf{d}_k$ must be found such that the objective function is minimized. The objective function is defined as

$$\Gamma(\{r_{qk}, \mathbf{d}_k\}) = \sum_{q=1}^{Q} \sum_{k=1}^{K} r_{qk} ||\mathbf{f}_q - \mathbf{d}_k||_2^2 \tag{2.6}$$

Uijlings *et al.* [45] evaluated the accuracy and efficiency of these quantization algorithms with densely extracted HOG, HOF, and MBH descriptors. There is an obvious trade-off between efficiency and accuracy; k-means clustering achieves the highest accuracy

(77.46 %) at the slowest computational speed (108 fps), whereas RF achieves the lowest accuracy (74.06 %) at the fastest computational speed (1910 fps) [45].

Feature Probability Distribution

The probability distribution of extracted features is captured using a generative model. A typical method for this step is the Gaussian Mixture Model (GMM) [12, 41]. GMM differs from k-means as it conveys both the mean information of codewords and the shape of their distribution. As a result, GMM encodes more information into the representation. As seen with FV encoding [37], this can result in better performance.

For GMM, the generative model is defined as

$$p(\mathbf{f}, \theta) = \sum_{k=1}^{K} \pi_k G(\mathbf{f}, \mu_k, \Sigma_k) \tag{2.7}$$

Where:
$\mathbf{f}$ is in the feature set $\mathbf{F} = \{\mathbf{f}_1, \ldots, \mathbf{f}_Q\}$.
K is the number of Gaussian models in the mixture.
π_k is the density of the $k-$th model in the mixture.
$\theta = \{\pi_1, \mu_1, \Sigma_1, \ldots, \pi_k, \mu_k, \Sigma_k\}$ represents the model parameters.
$G(\mathbf{f}, \mu_k, \Sigma_k)$ is the D-dimensional Gaussian distribution. Maximum likelihood estimation is used to learn optimal parameters for a given features set, $\mathbf{F}$ [41].

2.3.3 Feature Pre-processing

This step is responsible for dimensionality reduction of extracted features - a vital procedure for stable learning of features (unsupervised learning is done by k-means and GMM). This is important in the case of low-level local descriptors which are usually strongly correlated and highly dimensional. A popular method for this process is Principal Component Analysis (PCA) [12, 41]. PCA is a statistical procedure that uses orthogonal transforms to map extracted features to a set of principle components. The number of principle components is usually less than the number of original feature components; resulting in dimension reduction.

2.3.4 Feature Encoding

The encoding scheme, c_{BOW}, is responsible for the representation of extracted features. The feature descriptors and codewords are used to form a coding matrix, with a length equal to the number of codewords. The feature descriptors determine the occurrence count of each codeword in the video; generating a coding vector (frequency histogram). The length of this vector is equal to the number of codewords in the codebook. The coding algorithm used affects the detection of codewords in the video [46].

Three popular encoding methods are vector quantization (also known as hard voting), Fisher Vector (FV) encoding [37] and Vector of Locally Aggregated Descriptors (VLAD) [47]. Vector quantization methods were briefly discussed in Section 2.3.2. For a vocabulary of K words and a D-dimensional feature descriptor, the dimension of the output encoded vector of the aforementioned encoding schemes is summarized in Table 2.1.

Table 2.1: The output vector dimension for vector quantization, Fisher Vector and VLAD encoding

Encoding Scheme	Dimension
Vector Quantization [41]	K
Fisher Vector [37]	$2KD$
VLAD [47]	KD

Vector quantization is the standard way of producing the BoW representation of a feature set and produces the most compact output vector from the three encoding schemes. As mentioned in Section 2.3.2, k-means is the most commonly used vector quantization method. Although more efficient, vector quantization methods like k-means are outperformed by FV and VLAD encoding [45]. VLAD can be viewed as the non-probabilistic/hard version of FV encoding [12, 47]. Thus, VLAD is simpler and more efficient than FV encoding.

2.3.5 Pooling and Normalization

Pooling

The responses of each codeword are integrated into one value by pooling [46] to obtain a global representation $\mathbf{p}$ of a video. Furthermore, pooling is used to achieve representations that are more compact, have increased robustness to noise and clutter and are invariant to image transformations [48]. Common pooling methods include Sum Pooling, Average Pooling and Max Pooling [12].
Consider the $r \times q$ binary matrix $\mathbf{I}$; r is the number of feature extraction locations sites and q is the number of codewords in the visual vocabulary $W = \{\mathbf{w}_1, ..., \mathbf{w}_q\}$. A single n - dimensional binary column vector $\mathbf{v}$ is extracted from $\mathbf{I}$. Each element in $\mathbf{v}$ corresponds to the presence (1) or absence (0) of a visual codeword at a particular feature extraction point. The pooling operation f reduces $\mathbf{v}$ to a scalar value $f(\mathbf{v})$.

1. **Sum Pooling** [12]
$$f_s(\mathbf{v}) = \sum_{i=1}^{n} \mathbf{v}_i \tag{2.8}$$
2. **Max Pooling** [48]
$$f_m(\mathbf{v}) = \max_i \mathbf{v}_i \tag{2.9}$$
3. **Average Pooling** [48]
$$f_a(\mathbf{v}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{v}_i \tag{2.10}$$

An analysis of max pooling and sum pooling concluded that max pooling performs better with sparse features [48].

Normalization

Normalization ensures that the global representation achieved by pooling, $\mathbf{p} = \{p_1, ..., p_q\}$, is invariant to the number of feature descriptors. There are three common types of normalization techniques [49]:

1. **ℓ1-Normalization** [50]

   The normalized form of $\mathbf{p}$ by the ℓ1-norm, $\bar{\boldsymbol{p}}_{\ell 1}$, is defined as
$$\bar{\boldsymbol{p}}_{\ell 1} = \frac{\mathbf{p}}{\sum_{k=1}^{q} |p_k|} \tag{2.11}$$

2. **ℓ2-normalization** [51]

   The normalized form of $\mathbf{p}$ by the ℓ2-norm, $\bar{\boldsymbol{p}}_{\ell 2}$, is defined as

$$\bar{\boldsymbol{p}}_{\ell 2} = \frac{\mathbf{p}}{\sqrt{\sum_{k=1}^{q} {p_k}^2}} \tag{2.12}$$

3. **Power Normalization** [51]

   The power normalization of $\mathbf{p}$ is defined as $\bar{\boldsymbol{p}}_p = \{p_1^p, p_2^p, ..., p_q^p\}$, such that:

$$p_k^p = sign(p_k)|p_k|^{\alpha} \tag{2.13}$$

   Where:

   $k \in \{1, 2, ..., q\}$.

   $\alpha \in [0, 1]$ is the parameter for normalization.

   The chosen normalization technique can affect the overall performance of a classifier. Wang *et al.* [14] found that normalizing using the RootSIFT [52] approach as opposed to ℓ2-normalization resulted in a 0.5% improvement.

4. **RootSIFT: Hellinger distance** [52]

   The RootSIFT normalized form of $\mathbf{p}$ is defined as $\bar{\boldsymbol{p}}_r = \{p_1^r, p_2^r, ..., p_q^r\}$, such that:

$$p_k^r = \sqrt{\frac{p_k}{\sum_{k=1}^{q} |p_k|}} \tag{2.14}$$

   Where:

   $k \in \{1, 2, ..., q\}$.

2.4 Bag-of-Words Variants

The thematic approach of BoW variants involves encoding additional information about the input data into the final BoW representation, increasing the classification power of the framework whilst leveraging the compactness and simplicity of the standard model.

Laptev *et al.* [2] encoded spatio-temporal information into BoW representations for video-based action recognition by extending spatial pyramids used in image-based classification [53]. Li *et al.* [18] proposed a bag-of-words representation that encodes contextual information of extracted features into the final representation of the image. This is referred to as the Contextual Bag-of-Words (CBOW) and was shown to outperform the standard BoW framework in visual categorization tasks. Additionally, the CBOW model was used by Zeng *et al.* [36] to develop a robust visual tracking algorithm; addressing issues of occlusion and drifting. The tracking scheme utilizes two bag-of-words that carry appearance information on the target object and its background. This displays the flexibility of the BoW model; with slight modifications, it can be successfully applied to a variety of visual classification tasks. Nazir *et al.* [54] extended the BoW to a Bag of Expressions (BoE) by encoding neighborhood relationship information between words in the spatio-temporal domain. This achieved state-of-the-art accuracy on the KTH dataset.

Another common approach to encode relevant information into BoW representations is to combine the framework with accurate, computationally expensive video classification algorithms (such as FVs [17, 37], CNNs [23, 29] and dense sampling strategies [1, 14, 55]).

2.4.1 Convolutional Bag-of-Features

As opposed to relying on compression techniques for model simplification and parameter reduction, Passalis *et al.* [34] combined CNNs with the BoF framework to form the Convolutional Bag-of-Features (CBoF). It serves as a neural extension of the BoF framework and maintained excellent classification accuracy. This new CNN architecture uses Radial Basis Function (RBF) neurons to quantize image information for classification [34].

A major problem faced with CNNs is the increasing complexity, thus computational demand, when used in challenging classification problems. Thus, various efforts been made towards reducing the complexity of CNNs through compression techniques [17, 19, 56, 57]. These techniques can be applied to BoW variants incorporating deep neural networks to further increase computational efficiency for possible real-time applications. Although dimensionality reduction techniques address concerns of memory demand and computational cost, CNNs are slow to train and require large amounts of training data which is not readily available. Furthermore, for the algorithm to effectively perform, the CNN must be trained with data relevant to the intended application. As a result, the flexibility of the standard BoW algorithm is compromised when combined with deep learning mechanisms.

2.4.2 Scale Coding

As previously mentioned in Section 2.2.2, the standard BoW framework discards all scale information; locally extracted features that contain multi-scale information are transformed into a single scale-invariant histogram representation. This provided a simple and efficient solution to overcoming the inaccuracies caused by changes in scale. However, Khan *et al.* argued that scale variances within an image hold vital information and that purely scale-invariant image representations are sub-optimal [8]. This holds true as BoW models that contain scale information (both absolute and relative) in the final representation of an image were found to outperform scale-invariant representations [20]. The math behind scale coding is explored in the below sections as per Khan *et al.* [8].

General Approach

Scale coding a bag-of-words involves partitioning the generated visual vocabulary into small-, medium- and large-scale visual words. For a given image, three histograms are produced to represent the frequency of small-, medium-, and large-scale words respectively. The final representation is formed by concatenating the three histograms. Khan *et al.* proposed two scale coding approaches: absolute scale coding and relative scale coding [8].

Absolute Scale Coding

This approach encodes the scale information originally extracted from a given image into the final image representation. As illustrated in Figure 2.4, the scale of extracted features is independent of the size of the bounding box and is defined as per the original size of the image.

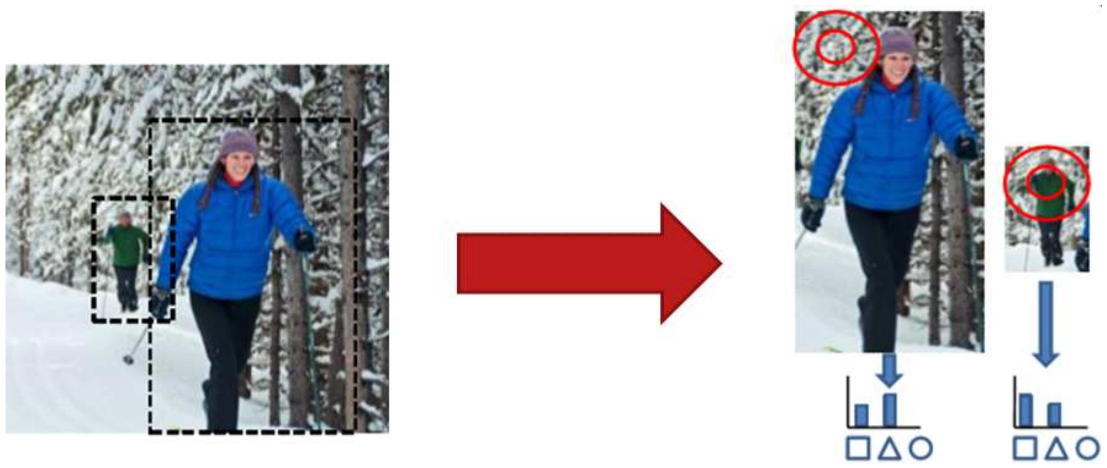


Figure 2.4: Absolute Scale Coding. Taken from [8].

For absolute scale coding, the set of M feature scales $S = \{1, ..., M\}$ are partitioned into three sub-groups as follows:

$$S^s = \{s | s \geq s^s, s \in S\}$$
$$S^m = \{s | s^s < s \leq s^l, s \in S\}$$
$$S^l = \{s | s^l < s, s \in S\}$$

Where s^s and s^l are the predefined cutoff thresholds. The final representation for the bounding box B consists of the concatenated partitions described above.

$$h^t(B) \propto \sum_{i=1}^{N} \sum_{s \in S^t} c(f_i^s) \tag{2.15}$$

Where:
S^t is the set of sub-groups which the total set of extracted feature scales S are divided into. $t \in \{s, m, l\}$ for small, medium- and large-scale features.

Relative Scale Coding

In this representation, extracted scale information is encoded relative to the bounding box of the object. This is illustrated by Figure 2.5. The difference between relative scale coding and absolute scale coding is that the scale of each feature s multiplied by the relative scaling factor, α, defined as

$$\alpha = \frac{B_w + B_h}{\bar{w} + \bar{h}} \tag{2.16}$$

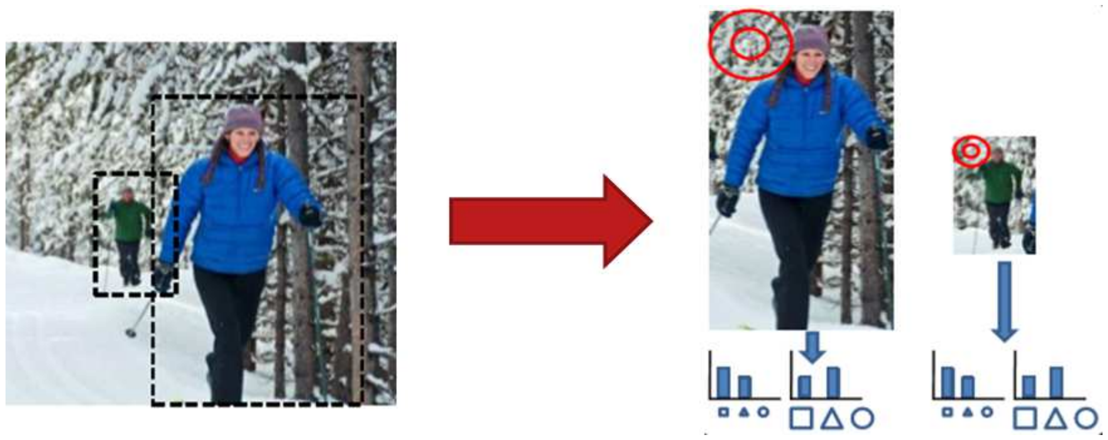


Figure 2.5: Relative Scale Coding. Taken from [8].

Where:

B_w and B_h are the width and height of the bounding box B.

$\bar{w} + \bar{h}$ are the mean width and height of every bounding box in the training set.

Thus, the relative feature scale is defined as:

$$\hat{s} = \alpha s = \frac{B_w + B_h}{\bar{w} + \bar{h}} s \tag{2.17}$$

Similarly to absolute scale coding, the relative set of scale sub-groups $\hat{S}^t, t \in \{s, m, l\}$ is partitioned as follows:

$$\hat{S^s} = \{\hat{s} | \hat{s} \geq s^s, \hat{s} \in S\}$$
$$\hat{S^m} = \{\hat{s} | s^s < \hat{s} \leq s^l, \hat{s} \in S\}$$
$$\hat{S^l} = \{\hat{s} | s^l < \hat{s}, \hat{s} \in S\}$$

Similar to absolute scale coding (equation 2.15), the final relative scale coded representation of the image involves concatenating the scale partitions as per equation 2.18

$$h^t(B) \propto \frac{1}{|\hat{S^t}|} \sum_{i=1}^{N} \sum_{s \in \hat{S^t}} c(f_i^s) \tag{2.18}$$

Where:

$|\hat{S^t}|$ is a normalization factor equivalent to the number of elements in the set $\hat{S^t}$ (in this case 3). Normalization is required to as the number of scales that fall into each scale sub-group varies with the size of the bounding box in relative scale coding.

Scale coding is heavily reliant on accurate definition of the bounding boxes of objects. These were pre-defined for testing and training in previous experimentation [8, 20]. For real-time video-based action recognition using the scale coding approach, a robust real-time visual tracker is required.

2.5 Visual Tracking

A visual tracker is required to isolate a target object, specifically a person in the case of action recognition, in the spatio-temporal domain. This is done by approximating the trajectory of the target object in the spatial plane as it moves through a scene [58]. This is a challenging task due to abrupt object motion, partial occlusion, deformation, motion blur, illumination variation, background clutter and scale variations.

A typical visual tracker consists of three components [36]:

1. **Object Appearance Model**
   This evaluates the probability that a candidate is the target object. For complex scenes, the appearance model is required to carry large amounts of information in order to distinguish the tracked object from the background. Furthermore, it should be able to adapt to variations over time. Thus, the appearance model largely contributes to overall performance of the visual tracker.

2. **Motion Model**
   This describes the movement of the object over the temporal plane. Both linear [59, 60] and non-linear [9, 61, 62] models have been postulated.

3. **Search Strategy**
   This determines the most likely position of the next object state. This strategy determines size of the search space, thus largely contributes to the computational cost of the visual tracking algorithm. Implemented search strategies include searching globally or locally using exhaustive or iterative methods.

Visual tracking algorithms can be broadly divided into two categories: generative [63, 64] and discriminative tracking [4, 9, 62]. Generative tracking involves learning an object appearance model and locating the target object by searching the areas that are most similar to the appearance model. Discriminative tracking treats visual tracking as a classification problem. The decision boundary separating background and the target object is established by training a discriminative classifier (e.g. SVM) with sample patches of both the target object and background. Although more robust to background clutter and activities, this approach generally requires a large dataset to achieve good performances. The visual tracking research space has largely been driven by the emergence of correlation filters [4–6, 9, 62, 65, 66] which have proven to be accurate and computationally efficient; making them suitable for real-time applications.

The general approach of correlation filter tracking involves predicting an optimal image filter such that the filtration with the input image patch produces a desired response. This response is typically Gaussian shaped centered at the location of the target object. Visual tracking involves applying this learned filter to a given image patch and evaluating the maximum value of the response to determine the new location of the target object. The filter is trained with shifted instances of the target image patch and is updated each frame; making the tracker robust to slight target position changes. Majority of the top-performing trackers are based on discriminative approaches involving Kernelized Correlation Filters (KCF) [5, 62, 66].

2.5.1 KCF Tracker

The KCF tracker is widely used due to it's simplicity and superior computational efficiency. It still remains as one of the fastest tracking algorithms. The procedure is summarized in Algorithm 1.

The KCF tracker models tracking as a classification problem $f(x) = \mathbf{w}^T x$; aiming to find a function that minimizes the error over the input samples x_i and their regression scores y_i. In order to allow for more powerful non-linear regression functions, the "Kernel Trick" [67] is used. This involves mapping the inputs, x_i, of a linear problem to a non-linear feature space by some mapping function $\phi(x)$. This is an attractive approach as the optimization problem remains linear; allowing for simple evaluation. The mapped classifier, $f(\phi(\mathbf{x}_i), \omega)$ is trained by minimizing the ridge regression error defined as

$$\omega = \underset{\omega}{\operatorname{argmin}} \sum_i |f(\mathbf{x}_i) - y_i|^2 + \lambda \|\omega\|^2 \tag{2.19}$$

Algorithm 1: KCF Tracker [5]

```
**Inputs:**
        $x$ : training image patch of size $M \times N$
        $y$ : regression target (Gaussian shaped)
        $z$ : test image patch of size $M \times N$
        $p_0$: initial target position
**Output:**
        $p_t$: detected target position
**Training:**
1 Compute the Gaussian kernel correlation between $x$ and itself, $k^{xx}$, using
  equation 2.22
2 Compute the DFT of the solution coefficients in the dual space, $A$, using
  equation 2.21
**Detection:**
3 Compute the response, $f(z)$, using equation 2.23
4 Maximize the response, $f(z)$ , to find new position of target, $p_t$
**Update:**
5 Update the model using equations 2.24 and 2.25
```

Where:

$\mathbf{x}_i$ is the cyclic shifted versions of the $M \times N$ input image patch, x, centered around the target, $i \in \{0, 1, ..., M-1\} \times \{0, 1, ..., N-1\}$.

$y_i \in [0, 1]$ is the matching score generated by a Gaussian function.

$\lambda \geq 0$ is the regularization parameter.

The solution to Equation (2.19) can be expressed as a linear combination of the input sample patches and is derived as:

$$\omega = \sum_i \alpha_i \phi(\mathbf{x}_i) = \sum_i \alpha_i \kappa(\mathbf{x}, \mathbf{x'}) \tag{2.20}$$

Where:

α_i represents the solution in the mapped feature space (known as the dual space).

$\kappa(\mathbf{x}, \mathbf{x'})$ is the kernel function defined as $\phi^T(\mathbf{x})\phi(\mathbf{x'}) = \kappa(\mathbf{x}, \mathbf{x'})$.

The solution in the dual space is efficiently computer using equation 2.21

$$A = \frac{\boldsymbol{Y}}{K^{xx} + \lambda} \tag{2.21}$$

Where:

Discrete Fourier Transforms (DFT) are indicated by capitalization (A is the DFT of α, K^{xx} is the DFT of k^{xx}, $\boldsymbol{Y}$ is the DFT of $\mathbf{y}$).

k^{xx} is the kernel correlation of $\mathbf{x}$ with itself.

$\mathbf{y}$ is a vector with the elements y_i

Using a Gaussian kernel, $k(\mathbf{x}, \mathbf{x'}) = exp(-\frac{1}{\sigma^2}\|\mathbf{x} - \mathbf{x'}\|^2)$, the kernel correlation is computed in the Fourier domain by element-wise products. This is given by:

$$k^{xx'} = exp(-\frac{1}{\sigma^2}(\|\mathbf{x}\|^2 - \|\mathbf{x'}\|^2 - 2\mathcal{F}^{-1}(\bar{X} \odot X'))) \tag{2.22}$$

Where:

$\mathcal{F}^{-1}$ denotes the inverse DFT.

$\odot$ denotes the operation of element-wise products.

$\bar{X}$ denotes the complex conjugate of X.

The first frame is trained with the initial position of the target, p_0. Subsequent frames are trained on the detected position of the target, p_t. The matching scores of all cyclic shifted versions of some candidate $M \times N$ image patch $\mathbf{z}$ is computed and p_t is set as the position that returns the maximum matching score i.e. the maximum of the response $f(\mathbf{z})$. The matching score function is given by:

$$f(\mathbf{z}) = \mathcal{F}^{-1}(\hat{A} \odot K^{\hat{x}z}) \tag{2.23}$$

Where:
$\hat{A}$ are the learned coefficients.
$\hat{x}$ is the learned object appearance model.
$K^{\hat{x}z}$ is the DFT kernel correlation between elements of $\hat{x}$ and $\mathbf{z}$.
The learned coefficients and object appearance model are updated every frame by Equations (2.24) and (2.25) respectively.

$$\hat{A_t} = \eta A_t + (1-\eta)A\hat{t-}1 \tag{2.24}$$

$$\hat{X_t} = \eta X_t + (1-\eta)\hat{X_{t-1}} \tag{2.25}$$

Where:
η is the fixed learning rate. This is typically 0.02.

Multi-Channel KCF

The standard KCF tracker can be easily extended to handle multiple channels by adjusting Equation (2.22) :

$$k^{xx'} = exp(-\frac{1}{\sigma^2}(\|\mathbf{x}\|^2 - \|\mathbf{x'}\|^2 - 2\mathcal{F}^{-1}(\sum_c \bar{X}_c \odot X'_c))) \tag{2.26}$$

Where:
$c \in \{1, 2, ..., C\}$. C is the total number of channels.
The input vector $\mathbf{x} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_c]$ concatenates the vectors of each channel, $\boldsymbol{x}_c$.

2.6 Conclusion

BoW Variants exist to overcome the weaknesses of the standard frame work - specifically, it's weak classification power. The weak classification power of BoW is as a result of the exclusion of valuable information (e.g. scale, orientation) in final representations. This highlights the compromise between classification power and computational complexity; the latter being a vital consideration for real-time applications.

Many BoW variants incorporate complex, information-rich strategies such as optical flow, dense sampling and deep neural networks in attempt to encode more information into BoW representations; leveraging the strong classification power of these strategies while maintaining the simplicity of the standard BoW framework. A notable encoding strategy is scale coding which incorporates extracted scale information into the final BoW representation of an image. Scale coding overcomes the paradigm of computational efficiency versus accuracy; it is a relatively simple approach that outperforms more complex state-of-the-art approaches. Furthermore, scale coding maintains the flexibility of the BoW model and can be extended and combined with other approaches (e.g. CNNs) to increase performance. However, scale coding has only been implemented for image-based action recognition and requires pre-defined bounding boxes of objects.

Chapter 3

Computational Cost to Accuracy Balance

3.1 Introduction

Many efforts have been made to reduce the complexity of existing high performing algorithms for real-time video-based action recognition. Shi *et al.* [68] replaced dense sampling with random sampling, thereby reducing the number of sampled patches to process and increasing efficiency. In a further work, Shi *et al.* [69] increased accuracy by increasing the sampling density. This was accomplished by using a Local Part Model and performing sampling at lower spatial resolutions. Yeffet *et al.* [70] extended Local Binary Patterns to the spatio-temporal domain (forming Local Trinary Patterns) to efficiently encode extracted information for action recognition. Zhang *et al.* [71] replaced optical flow with Motion Vectors to increase efficiency. To boost performance, the knowledge learned with optical flow CNN were transferred to motion vectors for effective real-time action recognition.

Similarly, many efforts have been made to increase the classification power of simple, computationally efficient algorithms, like the BoW framework, for accurate real-time applications. As covered in Section 2.2, numerous variants of the BoW framework have been developed; aiming to increase its classification power whilst leveraging its simplicity and computational efficiency [12, 18–20]. This highlights the computational cost to accuracy trade-off that must be carefully considered to achieve real-time video-based action recognition.

This chapter focuses on identifying the fundamental mechanisms that affect the cost-vs-accuracy balance for video-based action recognition. By analyzing the BoW and its variants, factors to consider when trying to develop and/or adjust existing algorithms for real-time action recognition of videos are identified. The dense trajectories feature framework [1] and the KTH [11] and a reduced HMDB51 [10] datasets were used to experimentally evaluate the practical effects of these factors on the cost-vs-accuracy balance.

3.2 Computational Cost vs Accuracy Balance

Upon reviewing the existing algorithms for action recognition, it was noted that improving algorithm performance often resulted in increased computational demands and complexity. Correspondingly, lowering the computational complexity for more efficient computation often resulted in decreased algorithm performance. Thus, there exists a trade-off between an algorithm's computational cost and accuracy. This cost-vs-accuracy paradigm can be stated as:

Remark 1 *The more information extracted from input data, the higher the classification power of the algorithm and the higher the computational cost. Thus, a balance between classification accuracy and computation necessitates an implementable and efficient algorithm for action recognition.*

In the ideal case, high performance is achieved at low computational costs. This balance for video-based classification tasks, is defined as

$$\beta = a \times f_p \tag{3.1}$$

Where:
a is the accuracy achieved by the algorithm. $0 \leq a \leq 1$.
f_p is the number of frames processed per second by the algorithm or computation task.

The higher the value of β, the more ideal the cost-vs-accuracy balance of the algorithm or computation task.

3.3 Factors Affecting Cost vs Accuracy Balance

This section identifies the factors affecting the computational cost vs accuracy paradigm identified in Remark 1. These factors are often addressed or manipulated by BoW variants in attempt to increase accuracy with low additional computational costs.

3.3.1 Sampling

Sampling is a vital step in any computer vision classification problem. Through sampling, information, in the form of features, can be extracted from an image or video frame. These feature extractions serve as inputs into the classification framework. Patch sampling is critical in the BoW framework as the number of patches sampled directly correlates to the algorithm's performance [55]; this complies with Remark 1. The ideal sampler should focus on regions that provide the most information for classification [16]. By focusing sampling on information rich patches and discarding insignificant patches, more information can be obtained at a lower computational cost.

Sampling methods can broadly be defined into two categories: dense sampling [13, 55, 72] and sparse sampling. Sparse sampling methods can be roughly divided into three categories: key-point-based sampling, random sampling and saliency-based methods [15]. Key point-based sampling identifies the informative content of images as the region surrounding local key points. Random sampling randomly selected local patches in an image for feature extraction. Saliency-based sampling samples points only in informative image regions; all sampled points hold important information for classification.

Dense sampling strategies are relatively popular due to its simplicity. Denser sampling yields more information about input data; resulting in better accuracy than sparse sampling methods but a higher computational cost. The general approach involves densely sampling points in each video frame and tracking sampled points via optical flow-based algorithms. A popular action recognition algorithm combining dense sampling and the BoW framework is Dense Trajectories [1, 13, 14]. Ideally, each pixel should be sampled, however this will result in large spatial complexity. Instead, patches are extracted from an image by a sliding window of a fixed size. The size of this window is referred to as the sampling step-size. The step-size must be chosen such that spatial complexity is reduced whilst keeping image information. In cases where information on the type of scene is not provided and computational resources are limited, random sampling is the best performing alternative to dense sampling [15, 68].

A popular action recognition algorithm combining dense sampling and the BoW framework is Dense Trajectories [1, 13, 14]. The algorithm involves densely sampling points in the spatial domain and tracking those points across the temporal domain; forming a trajectory. Various descriptors are then calculated around this trajectory. Although this approach has yielded competitive results [1, 13, 14], it is unsuitable for real-time applications due to the large number of features generated; video frames are densely sampled at multiple scales and five descriptors are computed for each sampled point. To reduce computational demand and maintain a reasonable performance, the sampling strategy can be changed; random sampling can be used [15, 68] and the number of frames sampled can be reduced. Reznicek *et al.* [73] investigated the minimum number of frames required to successfully recognize an action using the same BoW structure and classifier as [1]. It was found that an average of 18 frames is required to obtain a performance accuracy that is competitive with state-of-the-art action recognizers. This was achievable in 0.72s. In some action cases, an increase in the number of video frames processed resulted in a decrease in performance accuracy whilst the opposite held true in other action cases. Additionally, defining the optimum temporal extent over which to process a given number of frames is a vital factor - this is difficult to realize in real-time computations. Furthermore, there exists significant differences between the sizes of video sequences to recognize different actions. Thus, there are many additional challenges to overcome in this approach. Comparably, Wang *et al.* [1] found that a trajectory length of 15-20 frames was ideal and that performance begins to decrease after 20 frames. It can be concluded that an optimum number of video frames to process exists. This parameter should be dynamically determined based on the input video data as the optimum number of frames is different for every video/action class. Since there is a degree of randomness, artificial intelligence algorithms would be a good fit to tune this parameter. However, these are time consuming and would counteract the speed increase brought about by reducing the number of video frames to process.

In summary, random sampling provides the most favorable accuracy to computational demand trade-off. Other factors to consider when determining the most effective sampling strategy include the sample patch size, the sampling rate, and the number of video frames to be sampled.

3.3.2 Optical Flow

Optical flow models the motion of pixels between subsequent frames. It is used to capture temporal information of videos and includes both foreground object motion and background camera motion.

Considering dense trajectories, modifying the sampling strategy can greatly increase the computational efficiency of the algorithm. However, calculation of dense optical flow for computation of the trajectory shape, HOF and MBH descriptors was found to be the most time-consuming process of the Dense Trajectories algorithm; making up 52% of total computation time [1]. Although the optical flow algorithm used [74] presents a good compromise between accuracy and speed, the speed is not sufficient for real-time implementation.

To reduce computation time, optical flow can be replaced with a less computationally expensive approach. This approach was adopted by Zhang *et al.* [71] who replaced optical flow with Motion Vectors in their action recognition algorithm as Motion Vectors require no additional computation to extract. This resulted in a processing speed that was 27 times faster. However, Motion Vectors are coarser than optical flow and contain a large amount of noise; the inclusion of noise reduces performance. Thus, Motion Vectors were enhanced using knowledge transfer techniques; increasing accuracy of the algorithm whilst providing a fast enough processing speed for real-time action recognition [71]. This, once again, highlights the compromise between computational cost and performance accuracy as per Remark 1.

Computation of optical flow is extremely computationally expensive and is thus unsuitable for real-time applications. It can be replaced with less computationally expensive alternatives (such as Motion Vectors); often resulting a decreased accuracy. Another option is to avoid computation of optical flow by relying on descriptors that do not require optical flow for computation.

3.3.3 Saliency

A possible approach to achieve a favorable computation cost to accuracy balance is to only extract important (salient) information from input data. By doing this, computational resources are not wasted on extracting and processing unimportant information. However, determining which areas and features are important requires additional computation and can result in an overall decrease of computational efficiency [75] or a negligible increase [76].

The type of information extracted from sampled points is dependent on the descriptor used to describe the attributes about the selected location. Dense trajectories extract five descriptors: trajectory shape, HOG, HOF and Motion Bound Histograms in the x- (MBHx) and y- (MBHy) planes. HOG is computed using the gradient intensities of frame pixels, thus, represents static appearance information. HOF is computed using optical flow, thus, captures local motion information. MBH is computed using the derivative of optical flow. As a result, MBH removes camera motion and represents local foreground object motion information. The descriptors are formed by quantizing computed information into orientation bins. The magnitude is used as weighting.

3.3.4 Flexibility

The flexibility of the standard BoW algorithm is often compromised when combined with more powerful and complex algorithms. This is observed with CBoF [34] which combined CNNs with the BoW framework. In addition to increasing computational demands, CNNs are slow to train and require large amounts of training data which is not readily available. Furthermore, the CNN must be trained with data relevant to the intended application for effective performance. As a result, the algorithm can only be used for the intended application; compromising flexibility.

Although flexibility does not directly affect the cost vs accuracy balance, it allows for further extension and easy integration into a variety of applications. This increases the potential of improving the algorithm to obtain the ideal cost-to-accuracy balance.

3.4 Experimentation

The effects of each factor presented in Section 3.3 is investigated using the popular Dense Trajectories [1]. This section details the experimental setup. Experimental parameters are set as per [1].

3.4.1 PC Specifications

Experimentation was conducted on a PC with the following specifications: Intel Core i5 4th Gen., 1.7GHZ, 8G RAM. A single CPU core was used for computation.

3.4.2 Datasets

Wang *et al.* evaluated Dense Trajectories on 9 datasets [1]. Due to computational resource constraints, only 2 of the 9 datasets were used for experimental evaluations - the KTH[1] [11] dataset and reduced HMDB51[2] [10] action dataset.

The KTH dataset contains six human action classes: walking, jogging, running, boxing, waving and clapping. Each action is performed by 25 subjects in four different environments (outdoors, outdoors with scale variation, outdoors with different clothes and indoors). Since the background is homogeneous and static in most sequences, the dataset serves as a minimum baseline for evaluation of the action recognition algorithm. As found in [11], samples are divided into testing and training sets based on the subjects. The testing set consists of subjects 2, 3, 5, 6, 7, 8, 9, 10, and 22 (9 subjects total) and the training set is made up of the remaining 16 subjects. The average accuracy is taken over all classes.

The HMDB51 dataset contains 51 human action classes and is collated from a variety of sources. For this experimentation, a reduced dataset of 11 randomly selected action classes was used: brush hair, cartwheel, catch, chew, clap, climb stairs, smile, talk, throw, turn, wave. This is a challenging dataset that presents additional problems (e.g camera motion, occlusion and background activities) to overcome. It serves as an evaluation of how robust the action recognition algorithm is to the aforementioned challenges. As per the original setup [10], each action class is organized into 70 videos for

[1]http://www.nada.kth.se/cvap/actions/

[2]http://serre-lab.clps.brown.edu/resource/hmdb-a-large-human-motion-database/

training and 30 videos for testing. Performance is measured by the average accuracy over all classes.

3.4.3 Dense Trajectories

Dense trajectories involve densely sampling points in the spatial domain and tracking those points across the temporal domain; forming a trajectory. Five descriptors are extracted: trajectory shape, HOG, HOF and Motion Boundary Histograms in the horizontal (MBHx) and vertical (MBHy) planes [1].

Points are densely sampled with a sampling step-size of $W = 5$ on each spatial-scale separately. There are a maximum of 8 spatial scales; the number of spatial scales is dependent on the resolution of the video. Spatial scales are separated by a factor of $1/\sqrt{2}$ [1]. Sampled points in homogeneous image areas can not be tracked and are therefore removed based on the criterion presented by [77]. The threshold, T, is defined as:

$$T = 0.001 \times \max_{i \in I} \min(\lambda_i^1, \lambda_i^2) \tag{3.2}$$

Where:
$(\lambda_i^1, \lambda_i^2)$ are the eigenvalues of the i^{th}sampled point in the image I. As per Wang *et al.* [1], a value of 0.001 was used since it presented a good compromise between saliency and density.

Points are re-sampled and compared to the threshold T (see Equation (3.2)) every R frames. R is the refresh/frame sub-sampling rate. Sampled points are tracked separately on each spatial scale using a dense optical flow algorithm [74] to form trajectories. This is defined as:

$$P_{t+1} = (x_{t+1}, y_{t+1}) = (x_t, y_t) + (M * \omega_t)|_{(x_t, y_t)} \tag{3.3}$$

Where:
P_i is a point in frame I_i.
ω_t is the dense optical flow field for each frame, I_t, and is found with respect to the next frame, I_{t+1}.
M is a 3×3 median filter kernel applied to the optical flow field. The same optical flow implementation[3] as [1] is used.

[3]https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html

Descriptor Computation

From tracked points (see Equation (3.3)), five descriptors are extracted: HOG, HOF, MBH in the x (MBHx) and y (MBHy) planes and the trajectory shape, TS, defined as

$$TS = \frac{\Delta P_t, ..., \Delta P_{t+L+1}}{\sum_{j=t}^{t+L-1} ||\Delta P_j||} \tag{3.4}$$

Where:

ΔP_t is the displacement vector for a given point, $P_t = (x_t, y_t)$. $\Delta P_t = (P_{t+1} - P_t) = (x_{t+1} - x_t, y_{t+1} - y_t)$.

L is the length of the trajectory, counted in frames. This is set to $L = 15$ frames as per [1]. Since a point is represented by two values (an x and y co-ordinate), the final descriptor size is 30 ($L \times 2 = 15 \times 2$).

In addition to trajectory shape, Wang *et al.* [1] embeds additional appearance and motion information into the final representation by computing the remaining descriptors (HOG, HOF and MBH) from a $N \times N\ pixel \times L\ frame$ spatio-temporal volume surrounding the trajectory. This volume is divided into $n_T \times n_\sigma \times n_\sigma$ cells as illustrated in Figure 3.1. Dividing the spatio-temporal volume into cells embeds structural information [1] into the final representation. The trajectory is described by computing HOG, HOF and MBH descriptors for each cell and concatenating these descriptors to form the final representation. The parameters defining the structure of the spatio-temporal volume were set as per the default values [1]: $N = 32$, $n_\sigma = 2$ and $n_T = 3$.

The information extracted by the HOG, HOF and MBH descriptors are quantized into 8 orientation bins ($n_{bins} = 8$) using the magnitude as weighting. An additional zero bin is added for HOF to account for pixels with optical flow magnitudes falling below the threshold defined in Equation (3.2) [2]. Thus, HOF has 9 bins ($n_{bins} = 9$). The final dimension of each histogram descriptor is $n_T \times n_\sigma \times n_\sigma \times n_{bins}$. Thereafter, each histogram descripotr is normalized by its $\ell 2$-norm (see Equation (2.12)). The dimensions of all extracted descriptors are summarized in Table 3.1.

Bag of Words

Following [1], a vocabulary of 4000 words is separately constructed for each descriptor. K-means clustering is used to cluster a subset of 100 000 randomly selected training

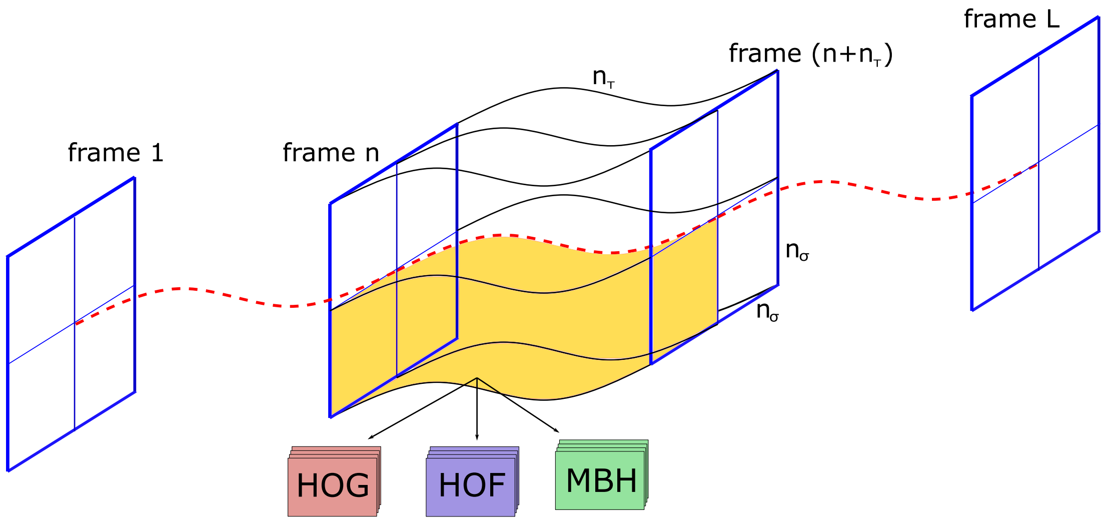


Figure 3.1: HOG, HOF and MBH descriptors are computed along the trajectory in a $N \times N$ pixel neighborhood divided into $n_T \times n_\sigma \times n_\sigma$ cells.

Table 3.1: Dimensions of the descriptors extracted for dense trajectory features

Descriptor	Dimension
Trajectory Shape	30
HOG	96
HOF	108
MBHx	96
MBHy	96
Total	**426**

features.

The feature descriptors are assigned to the "visual word" with the closest Euclidean distance. The count of visual word occurrences is represented in a histogram. Structure is added through spatio-temporal pyramids. Six different pyramids are used (see Figure 3.2). A BoW is constructed for each cell of a given pyramid. Thereafter, a global representation of the pyramid is found by summing the BoW histogram for each cell and normalizing by the RootSIFT Norm [52], defined by equation 2.14: Each spatio-temporal pyramid-descriptor pair forms a separate channel; there are 30 channels in total (6 pyramids x 5 descriptors). A multi-class, multi-channel non-linear SVM

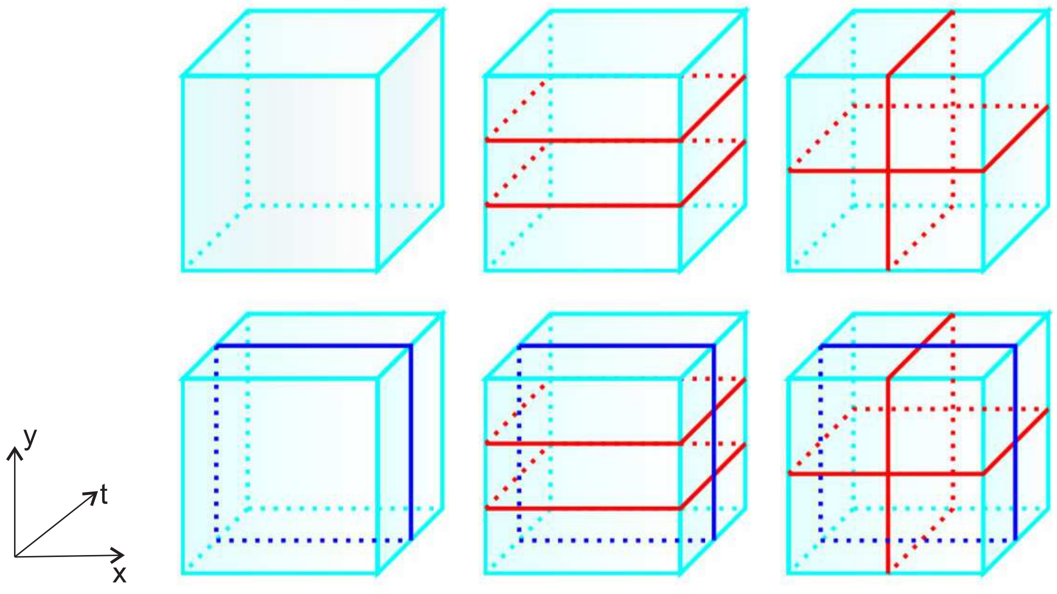


Figure 3.2: The spatio-temporal grids adapted from [1]. Top row from left to right:$h1 \times v1 \times t1$, $h3 \times v1 \times t1$, $h2 \times v2 \times t1$. Bottom row from left to right: $h1 \times v1 \times t2$, $h3 \times v1 \times t2$, $h2 \times v2 \times t2$.

with an RBF-χ^2 kernel [11] is used for classification. For multi-class classification, a one-against-rest approach is used. For multiple channels, the kernel is defined as [78]:

$$K(x_i, x_j) = exp(-\sum_c \frac{1}{A_c} D(x_i^c, x_j^c) \tag{3.5}$$

Where:

$D(x_i^c, x_j^c)$ is the χ^2 distance between each training video x_i and x_j in each channel c.

A_c is the average of the χ^2 distances between training samples in channel c.

3.4.4 Evaluation of Factors

To evaluate 'Saliency', the computational cost and accuracy achieved by each descriptor on each dataset is observed. In this manner, the information which is most important can be determined. Computational cost was measured by the number of frames computed per second - the higher this value, the lower the computational cost. Results are summarized in Table 3.2.

To evaluate 'Sampling', the accuracy and computational cost of the algorithm at different refresh rates(R) is observed. A refresh rate of 1 frame was used as a baseline for comparison. Following Remark 1, a refresh rate of R = 1 should yield the highest accuracy since the most information is obtained if the video is re-sampled every frame. Uijlings *et al.*[45] found that a refresh rate of R = 6 frames has the best cost-to-accuracy trade-off for dense computation of HOG, HOF and MBH based on the UCF50 dataset[79]. Thus, performance with R = 1 was compared to performance with R = 6. Results are summarized in Table 3.3.

To evaluate 'Optical Flow',its performance at each scale is considered in order to determine if the information extraction abilities of this task is worth the computational cost (it is responsible for 52% of the computation time [1]).

3.5 Results and Discussion

3.5.1 Descriptors

Observing the results in Table 3.2, the MBH descriptor has the best overall performance and the highest computational cost; agreeing with Remark 1. For the HMDB51 dataset, MBH outperforms all other descriptors by over 10%. This is due to the fact that the HMDB51 data samples include camera motion. Since MBH is computed using the derivative of optical flow, it is robust to camera motion unlike the other descriptors. However, in static scenery (KTH dataset), MBH only outperforms HOF by 1.7%. Since HOF is more computationally efficient than MBH, it can be concluded that MBH is unsuitable for real-time action recognition of static camera data (e.g CCTV footage). Overall, trajectory shape has the best cost-to-accuracy balance.

Theoretically, HOG should have the lowest computational cost since it is the only descriptor that does not rely on optical flow computation. However, HOG has a longer computation time than HOF in experiments. This is due to the fact that optical flow is part of the tracking algorithm therefore less additional computation is required for HOF compared to HOG; HOG requires additional calculation of the gradient between pixel points.

Table 3.2: The effects of Trajectory Shape [1], HOG [2], HOF [2] and MBH [3] descriptors on the cost vs accuracy balance

Dataset		**Trajectory Shape**	**HOG**	**HOF**	**MBH**
KTH	Computation Time (FPS)	8.16	1.11	1.71	0.61
	Accuracy as per [1] (%)	89.8	87.0	93.3	95.0
	β	7.33	0.97	1.60	0.58
HMDB51	Computational Cost (FPS)	2.48	0.11	0.14	0.07
	Accuracy as per [1] (%)	28.0	27.9	31.5	43.2
	β	0.69	0.03	0.04	0.03

Table 3.3: Evaluation of the effect of the refresh rate on the cost vs accuracy balance for KTH and HMDB51 datasets using the Dense Trajectory features

Dataset		**R = 1 frame**	**R = 6 frames**
KTH	Computation Time (FPS)	0.25	0.3
	Accuracy (%)	94.0	91.0
	β	0.24	0.27
HMDB51 (reduced)	Computation Time (FPS)	0.03	0.03
	Accuracy (%)	78.48	81.21
	β	0.02	0.02

3.5.2 Refresh Rate

The accuracy achieved on the KTH dataset for R = 1 (94.0%) is lower than expected (94.2% [1]). This might be due to the fact that Dense Trajectory computation was recreated in Python 3.7 with the latest version of OpenCV, whereas the original code[4] was created in C++ with OpenCV-2.4.2. As a result, the recreated code has a different handling of the algorithm; explaining the difference in accuracy rates. As expected, there is a decrease in accuracy (3%) when sub-sampling every 6 frames as opposed to every 1 frame. This results in a 20% (0.05 FPS) increase in computation speed.

Comparatively, there is a negligible difference in computational cost for the more challenging HMDB51 dataset (for R = 6 frames, computation time was 0.001 FPS faster). Most notably, accuracy increases by 2.73% when sub-sampling every R = 6 frames. This contradicts theoretical predictions as a decrease in accuracy was expected.

The contradicting results obtained with the KTH and HMDB51 dataset show that the effects of 'Sampling' on the cost vs accuracy balance is data dependent. Unlike the KTH dataset, the HMDB51 dataset includes unimportant information such as camera motion, occlusion and background clutter. Sub-sampling every frame increases chances that this unimportant information will be captured, which can decrease accuracy. This highlights that 'Saliency' of extracted features is the most vital factor to consider. Based on these results, Remark 1 is adjusted:

Remark 2 *The more **salient** information extracted from input data, the higher the classification power of the algorithm and the higher the computational cost. Thus, a balance between classification accuracy and computation necessitates an implementable and efficient algorithm for action recognition.*

3.5.3 Optical Flow

It can be seen that optical flow becomes noisier at smaller spatial scales (Figure 3.3) which adversely affects performance. Removing multi-scale sampling would reduce the overall computational cost of the Dense Trajectories algorithm and result in better performance of the optical flow. However, Khan *et al.* [8, 20] proved that scale pro-

[4]http://lear.inrialpes.fr/people/wang/dense_trajectories

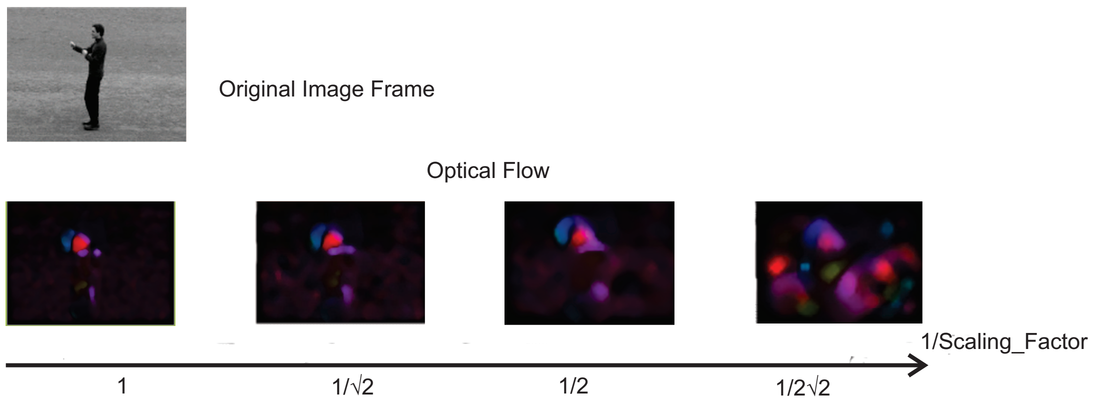


Figure 3.3: The computed optical flow for each spatial scale

vides vital information; a BoW incorporating extracted scale information into final representations outperforms its scale-invariant counterpart. Since optical flow is computationally expensive and unreliable at smaller scales, discarding its calculation has a potential to be suitable for real-time applications.

3.6 Conclusions

For successful real-time video-based action recognition, the trade-off between performance accuracy and computational cost must be carefully considered. In order to identify the fundamental mechanisms affecting this cost-vs-accuracy balance, a simple and widely used framework (the BoW framework) and its variants were analyzed. Four factors affecting this balance were identified: 'Sampling' strategy, 'Saliency' of features, 'Optical Flow' and overall algorithm 'Flexibility'. Experimental evaluation found 'Saliency' of extracted features to have the most notable affect on the cost vs accuracy balance. Capturing unimportant information (e.g. occlusion, background activities etc.) can decrease performance accuracy and wastes computational resources. The saliency of extracted features is dependent on the 'Sampling' strategy.

Exploring the effects of the identified factors on the cost-to-accuracy balance of complex state-of-the-art deep learning architectures remains as future work. This will help move the research space closer to creating algorithms that are simple yet powerful enough for practical realization of real-time video-based action recognition.

Chapter 4

Spatio-Temporal Scale Coded Bag-of-Words

4.1 Introduction

The improved accuracy of a scale-encoded BoW [8] over its scale-invariant counterpart showed scale to be salient information to extract for action recognition. As per the conclusions of Chapter 3, the 'Saliency' of extracted information is the most vital factor in achieving an ideal cost-to-accuracy balance. Thus, encoding extracted scale information into BoW representations of videos has the potential to overcome the cost-vs-accuracy paradigm for video-based action recognition.

Scale coding involves encoding the spatial information of extracted patches into the final BoW representation of an image. This is a relatively simple approach yet has outperformed more complex approaches in image-based action recognition [8]. The simplicity of the scheme allows for it to be easily incorporated into other methods; offering the same flexibility as the standard BoW approach. Khan *et al.* [20] combined scale-encoded BoW representations with CNNs to increase performance accuracy. Thus, scale coding increases accuracy while maintaining the flexibility and computational efficiency of the original BoW framework. As a result, this algorithm has high potential for successful real-time action recognition. However, this approach has only been implemented for image-based action recognition. Further work must be done to extend scale coding such that it can be use in video-based action recognition. The scale coded BoW framework for image-based action recognition was discussed in Section 2.4.2.

Various well-performing image classification algorithms that operate in a 2-dimensional space have been extended to the third dimension of time for classification of video data. Scovanner *et al.* [80] extended Scale-Invariant Feature Transforms (SIFT) [15, 16] to form the 3D-SIFT descriptor for classification of video data represented by the BoW approach. Other 3D extensions include extended SURF (ESURF) [81], local trinary patterns [70] and HOG3D [82]. In a similar manner, extending scale encoded BoW representations to the spatio-temporal domain for real-time action recognition of videos will be investigated in this chapter.

4.2 Criteria for Real-Time Action Recognition

In this section, the criteria to evaluate whether the proposed action recognition approach operates in real-time is defined.

Real-time processing is defined as the completion of pre-defined tasks within a set time frame [83]. The type of tasks and the length of this time frame is application dependent [84]. In action recognition applications, the system is considered to perform in real-time when the classification of the action from input data is relatively imperceptible to the user. Thus, for video-based action recognition, the criteria for real-time processing can be defined as:

$$t_p \leq t_c \tag{4.1}$$

Where t_p is the processing time per frame and t_c is the capturing time per frame.

Equation 4.1 can alternatively be stated as:

$$f_p \geq f_c \tag{4.2}$$

Where:

f_p is the number of frames processed per second by the algorithm

f_c is the number of frames captured per second

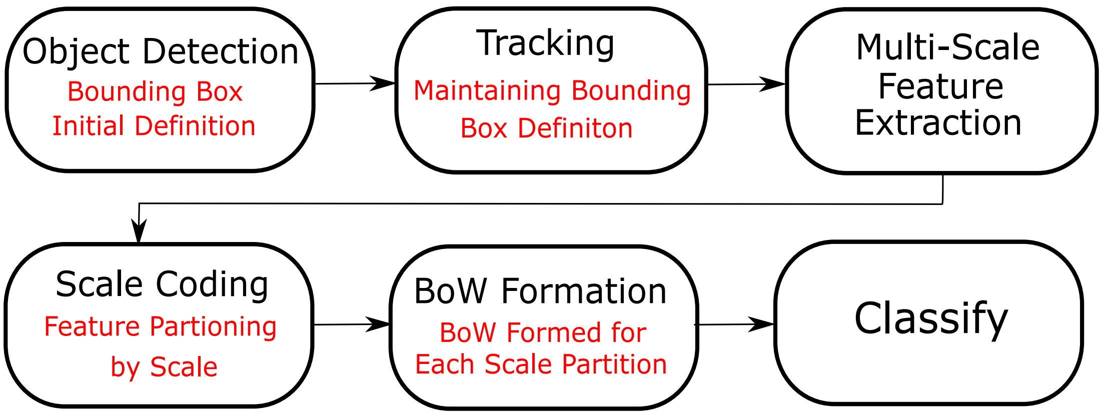

Object Detection
Bounding Box Initial Definition
Tracking
Maintaining Bounding Box Definiton
Multi-Scale Feature Extraction
Scale Coding
Feature Partioning by Scale
BoW Formation
BoW Formed for Each Scale Partition
Classify

Figure 4.1: The general pipeline for video-based action recognition with scale coded BoW.

4.3 General Approach

The general pipeline of scale-coding a bag of words for video-based action recognition is summarized in Figure 4.1. Scale coding is heavily reliant on pre-defined bounding boxes for computation. Obtaining the bounding boxes of objects in real-time poses many additional challenges. These include: correctly and completely bounding the object and identifying which objects are important for classification..

The initial bounding box, B_0, of the target object is identified by an object detection algorithm. This bounding box is then maintained through subsequent frames via a visual tracking algorithm. For effective bounding box definition, the visual tracking algorithm should be robust to occlusion, background activities and scale variations. It should also be able to operate in real-time so that it does not hinder the overall efficiency of the action recognition algorithm.

For multi-scale feature extraction, the target object is cropped from its background. Features are extracted from this cropped image, I_t, on multiple scales. The area of the cropped image, I_t, is determined by the bounding box B_t which would be defined by the visual tracking algorithm. This process is done every R frames. R represents the sampling refresh rate, and t is the frame number.

The extracted features are represented as a scale coded BoW. This is passed to a classifier for action classification.

4.3.1 Object Detection

For action recognition, the target object is a person. Thus, the object detection algorithm needs to not only isolate an object from its background, but also ensure that the target object is a person. Furthermore, the dimensions of initial bounding box B_0 defined by the object detector, must comply with a set minimum value, dim_{min}.

Unlike the field of action recognition where machine learning algorithms (like BoW) still remain competitive with state-of-the-art approaches, state-of-the-art object detection schemes are dominated by deep learning approaches. Since the performance of the visual tracker largely depends on the initial bounding box, deep learning-based object detectors are considered as they yield the most accurate results. As previously mentioned, deep learning schemes require large sets of labeled training data and long training times, making them difficult to implement in real-world circumstances. However, considering the widespread availability of pre-trained object detection models[1] with person object detection, these setbacks can be avoided.

Three common deep learning-based object detectors are Region-based CNNs (R-CNNs) and variants [85], Single Shot Detectors (SSDs) and the You Only Look Once (YOLO) detector [86]. R-CNN approaches are two-stage detectors. The first stage consists of the proposition of candidate bounding boxes. The second stage involves passing these candidate regions to a CNN for classification. As a result, R-CNN approaches are extremely accurate. However, they are extremely slow. Since the proposed action recognition algorithm aims to operate in real-time, R-CNNs are an unsuitable choice for object detection. SSDs and the YOLO detector use a one-stage detection approach and model detection as a regression problem. Although less accurate than R-CNN approaches, these are much more efficient. YOLO is the most efficient approach and is able to operate in real-time. Furthermore, it was found to achieve the best accuracy compared to other real-time detectors [86]. Thus, the YOLO object detector will be used for the object detection step. Various iterations have been developed. The best performing iteration was YOLOv3 [87] which operates at 22ms. Since the object detection algorithm is only run on the first frame to identify the initial bounding box, B_0, this is an acceptable speed.

[1]https://github.com/onnx/models

Person Object Isolation

Once objects in the frame are detected, additional processing is done to ensure the detected objects represent a person and that the bounding box, B_0, complies with the minimum width and height, dim_{min}. This is summarized in Algorithm 2.

```
Algorithm 2: Person Object Isolation after Object Detection
  Inputs:
        $bbox$: An array holding the bounding box dimensions for all detected
        objects
        $labels$: The class labels of the object bounding boxes in $bbox$
        $dim_{min}$ : Minimum dimensions for bounding box
  Output:
        $B_0$: Initial bounding box isolating the person in the frame

1  if 'person' not in labels then
2      return None
3  else
4      $i \leftarrow$ index of 'person' in $labels$
5      $B_0 \leftarrow bbox[i]$
6  end
7  if width of $B_0 < dim_{min}$ then
8      $diff \leftarrow dim_{min}$ - width of $B_0$
9      $p_0 \leftarrow (x - \frac{diff}{2}, y)$        // adjust the center of $B_0$, $p_0 = (x, y)$
10
11 end
12 if height of $B_0 < dim_{min}$ then
13     $diff = dim_{min}$ - height of $B_0$
14     $p_0 \leftarrow (x, y - \frac{diff}{2})$        // adjust the center of $B_0$, $p_0 = (x, y)$
15
16 end
17 return $B_0$
```

4.3.2 Visual Tracker

As mentioned in Section 2.5, most top-performing visual trackers are based on Correlation Filters [4–6, 62, 65, 66]. The most widely used tracker is the KCF tracker [4, 5] which is popular due to its computational efficiency and ability to accurately adapt to small translation changes. However, the standard KCF tracker is unable to adapt to target object scale changes. This is vital not only for practical visual tracking applications but also for the proposed action recognition algorithm; a scale encoded BoW relies on extraction of accurate scale information. As highlighted by the conclusions of Chapter 3, the inclusion of unimportant feature information reduces the performance of the action recognition algorithm. Thus, a visual tracker that is able to adapt to both translation and scale changes of the target object is required. Furthermore, this tracking algorithm should be able to operate in real-time such that it does not compromise the overall efficiency of the action classification algorithm. An appropriate visual tracking model is the Discriminative Scale Space Tracker (DSST) [65].

The DSST learns two independent models to handle translation and scale variances respectively. Both models involve the training of filters for detection. The translation model uses Minimum Output Sum of Squared Error (MOSSE) filters [6]; this is known as the MOSSE tracker. The MOSSE tracker localizes the target object in a new frame by minimizing the sum of squared errors to determine the optimal filter. Once the location of the translated target object is obtained, scale variations are detected by learning a separate a 1-dimensional correlation filter.

Translation Model

Henriques *et al.* [5] presented the linear Dual Correlation Filter (DCF) tracker which is the equivalent of the MOSSE tracker. The DCF tracker is based on the KCF tracker [4]. It is a multi-channel extension of kernelized linear correlation filters. Henriques *et al.* [5] evaluated their proposed multi-channel KCF and DCF tracker as well as the MOSSE tracker based on a benchmark dataset compiled by Yi *et al.*[88]. To evaluate which tracker yields the best cost-to-accuracy balance, Equation (3.1) was used to compute β. These values are summarized in Table 4.1.

MOSSE is observed to have the best cost-to-accuracy balance due to its exceptional

Table 4.1: Evaluation of cost-vs-accuracy for the KCF [4], DCF [5] and MOSSE [6] visual trackers

Tracker	**Mean Precision (%)** [5]	**Mean FPS** [5]	β
KCF	73.2	172	125.90
DCF	72.8	292	212.58
MOSSE	43.1	615	265.07

computational efficiency. However, its accuracy is much less favorable than the KCF and DCF trackers. Since the proposed spatio-temporal scale encoded BoW is reliant on accurate bounding box definition, the DCF tracker would be the best choice. Thus, for the translation model of the DSST tracker, DCF will be used.

DCF is based on the multi-channel extension of the standard KCF tracker (see Section 2.5.1). Instead of a Gaussian kernel, a linear kernel is considered: $\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$. Similarly to the multi-channel KCF (Equation (2.26)), the kernel correlation for the linear kernel is defined as

$$k^{xx'} = \mathcal{F}^{-1}(\sum_c \bar{X}_c \odot X'_c) \tag{4.3}$$

Where:
The input vector $\mathbf{x} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_c, ..., \boldsymbol{x}_C]$ concatenates the vectors of each channel, $\boldsymbol{x}_c$. $c \in \{1, 2, ..., C\}$. C is the total number of channels.
The Discrete Fourier Transform (DFT) of is indicated by a capital letter (the DFT of x_1 is X_1).
The complex conjugate form of X is indicated by a bar, i.e $\bar{X}$.

This defines the DCF tracker. It is trained in the dual space, α. The algorithm for obtaining the new position of the translated object, p_t, remains the same as Algorithm 1 except the kernel correlation computation in step 1 is done using Equation (4.3).

Scale Model

Scale variations are evaluated by learning a 1-dimensional correlation filter. This scale filter, h, is applied to an image patch, I_t, centered at the location of the target object determined by the translation model, p_t; generating a correlation response. This scale filter is trained to produce a 1-dimensional Gaussian shaped response such that the target scale is located at the maximum of this response. Similarly to the translation mode, a multi-channel filter is used.

As per [65], the scale filter for channel c, is defined as

$$H^c = \frac{\bar{Y} X_c}{\sum_{k=1}^{C} \bar{X}_k X_k + \lambda} \tag{4.4}$$

Where:
Y is the DFT of the desired response of the filter. This is set to be 1-dimensional Gaussian response.
λ is the regularization parameter of the scale filter.

To find the optimal filter, the numerator and denominator parts of Equation (4.4) are trained separately. U^c is the learned numerator for the c^{th} channel. D is the learned denominator. The filter is updated every frame t using the following equations.

$$U_t^c = (1-\eta)U_{t-1}^c + \eta \bar{Y} X_c \tag{4.5}$$

$$D_t = (1-\eta)D_{t-1} + \eta \sum_{k=1}^{C} \bar{X}_k X_k + \lambda \tag{4.6}$$

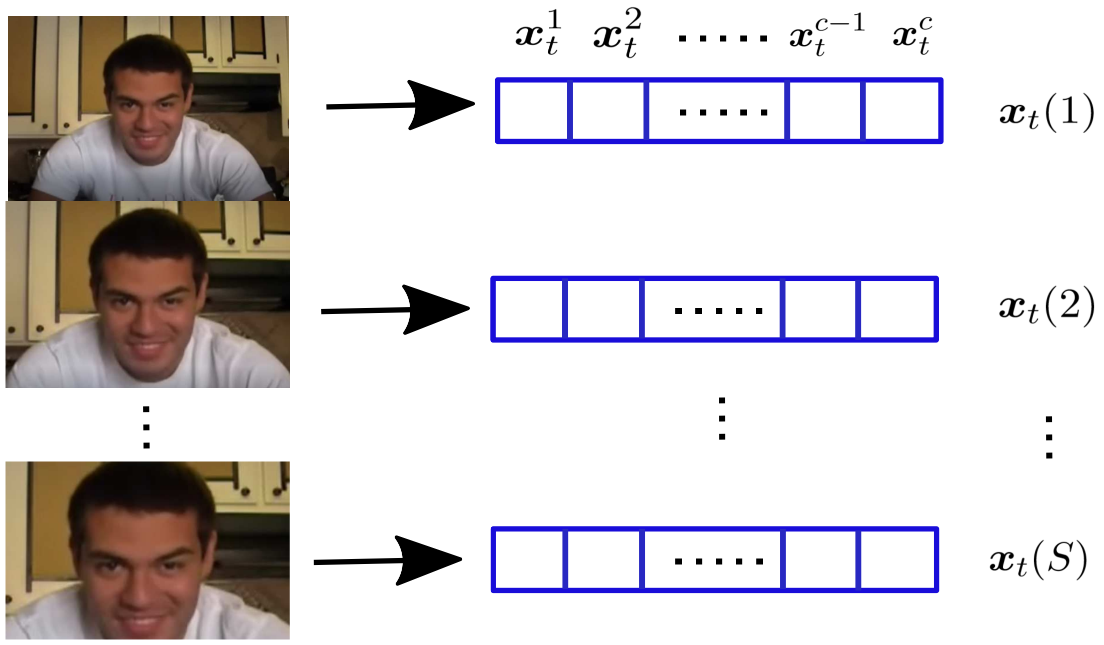
Where:
η is the learning rate parameter.
X_c is the DFT of the vector in the c^{th} channel of the training sample, $\mathbf{x}_t$.

The training sample, $\mathbf{x}_t = [x_t^1, ..., x_t^c, ..., x_t^C]$, is constructed using the features extracted at various scales from the image patch I_t at frame t. The size of the scaled image patches, $I_{t,n}$, is defined as

$$sf^n M \times sf^n N \tag{4.7}$$

Where:
sf is the scale factor.
$M \times N$ is the size of the image patch, I_t. $n \in \{-\frac{S-1}{2}, ..., \frac{S-1}{2}\}$. S is the size of the scale filter.

To populate the scale training sample set, $\mathbf{x}_t$, the value $\mathbf{x}_t(n)$ is the the C-dimensional



$\boldsymbol{x}_t^1$ $\boldsymbol{x}_t^2$ ..... $\boldsymbol{x}_t^{c-1}$ $\boldsymbol{x}_t^c$
.....
$\boldsymbol{x}_t(1)$
$\boldsymbol{x}_t(2)$
$\boldsymbol{x}_t(S)$

Figure 4.2: Visualization of constructing the samples to update and train the scale filter [9]

feature vector extracted from the image patch, $I_{t,n}$ at the scale level n. Thus the size of $\mathbf{x}_t$ is $C \times S$. This procedure is visualized in Figure 4.2. The correlation scores for frame t, $f(\mathbf{z}_t)$, are computed using the below equation.

$$f(\mathbf{z}_t) = \mathcal{F}^{-1}\left(\frac{\sum_{c=1}^{C} \bar{U}_{t-1}^c Z_t^c}{D_{t-1} + \lambda}\right) \tag{4.8}$$

Where:

z_t^c is the vector in the c^{th} channel of the test sample, $\mathbf{z}_t = [z_t^1, ..., z_t^c, ..., z_t^C]$, constructed in frame t. This is constructed in the same manner as the training sample, $\mathbf{x}_t$.

By maximizing $f(\mathbf{z}_t)$, the target scale is found. Thus, the bounding box for the frame t, B_t, is centered at p_t and is of the size $sf^{s_t}M \times sf^{s_t}N$. s_t is the target scale.

DCF-DSST

The algorithm for the visual tracker used in the proposed spatio-temporal scale encoded BoW scheme is summarized in Algorithm 3. Variables for the translation and scale models are denoted with the subscripts *trans* and *scale* respectively. Following [65], PCA-HOG [89] features are extracted from the image patch, I_t, for sample construction. As detailed in [6], the extracted features are always multiplied by a Hanning Window defined by:

$$Hann(v) = 0.5 \left(1 - \cos(\frac{2\pi v}{S-1})\right) \tag{4.9}$$

Where:

$0 \leq v \leq S - 1$.

4.3.3 Multi-Scale Sampling

For multi-scale sampling, the image patch, I_t, is cropped from the frame t. The area of I_t is defined by the bounding box B_t. The number of spatial scales extracted is dependent on the dimensions of the image patch, I_t. A maximum of 9 spatial scales can be extracted. As per the image-based scale coding approach [8], consecutive spatial scales are separated by a scale factor of $\sqrt{2}$. The definition of the spatial scale set is outlined in Algorithm 4. The features extracted for classification are the same as the features extracted by the visual tracker - HOG features with dimensionality reduction by Principle Component Analysis (PCA-HOG) [89]. These features can be efficiently computed and are robust to illumination and deformation. Features are sampled separately on each spatial scale $s \in S_{set}$.

Algorithm 4: Definition of the Spatial Scales

```
Inputs:
        $B_0$ : initial bounding box
        $dim_{min}$: minimum dimensions of bounding box allowed
        $S_{max}$: maximum number of spatial scales
Output:
        $S_{set}$: the set of spatial scales
1  $minSide \leftarrow$ min($B_0$ width, $B_0$ height)
2  $layer \leftarrow 0$
3  $s \leftarrow 1$
4  while $minSide \geq dim_{min}$ and $layer \leq S_{max}$ do
5      append $s$ to $S_{set}$
6      $s \leftarrow s \times \sqrt{2}$
7      $minSide \leftarrow minSide/\sqrt{2}$
8      $layer \leftarrow layer + 1$
9  end
10 return $S_{set}$
```

Algorithm 3: DCF-DSST Tracker: Iteration at frame t

```
Inputs :
        $I_t$ : Image patch
        $p_{t-1}$: previous frame target position
        $s_{t-1}$: previous frame target scale
        $y_{trans}$: regression target for translation model (Gaussian shaped)
        $y_{scale}$ : regression target for scale model (Gaussian shaped)
Outputs:
        $p_t$: detected target position
        $s_t$: detected target scale
Training:
1  Compute the Gaussian kernel correlation between $x$ and itself, $k^{xx}$, using
     Equation (4.3)
2  Compute the DFT of the solution coefficients in the dual space, $A_{trans}$, using
     Equation (2.21)
Translation Detection:
3  Construct the test sample, $\mathbf{z}_{t,trans}$, from $I_t$ at $p_{t-1}$ and $s_{t-1}$
4  Compute the correlation response, $f(\mathbf{z}_{t,trans})$, using Equation (2.23)
5  Maximize the response, $f(\mathbf{z}_{t,trans})$, to find target position, $p_t$
Scale Detection:
6  Construct the test sample, $\mathbf{z}_{t,scale}$, from $I_t$ at $p_t$ and $s_{t-1}$
7  Compute the correlation response, $f(\mathbf{z}_{t,scale})$, using Equation (4.8)
8  Maximize the response, $f(\mathbf{z}_{t,scale})$, to find target scale, $s_t$
Update:
9  Extract training samples $\mathbf{x}_{t,trans}$ and $\mathbf{x}_{t,scale}$ from $I_t$ at $p_t$ and $s_t$
10 Update the translation model using Equations (2.24) and (2.25)
11 Update the scale model using Equations (4.5) and (4.6)
```

PCA-HOG Feature Computation

Feature computation involves dividing I_t into a grid of $k \times k$ cells and generating a HOG feature vector for each $k \times k$ cell. Computing HOG features in this manner not only offers a more compact representation but makes the representation more robust to noise. For each cell, the gradient intensity orientation $\theta(x, y)$ and magnitude $r(x, y)$ are computed at each pixel (x, y). In the case of color images, the θ and r values are taken from color channel with the largest gradient magnitude. A bin value, $b \in \{0, .., l\}$, is assigned to each pixel based on its gradient orientation, $\theta(x, y)$. b is found by Equation (4.10) for contrast insensitive definition or by Equation (4.11) for contrast sensitive definition [89].

$$B_a(x, y) = round(\frac{l \times \theta(x, y)}{\pi}) \bmod l \tag{4.10}$$

$$B_b(x, y) = round(\frac{l \times \theta(x, y)}{2\pi}) \bmod l \tag{4.11}$$

Where:

l is the total number of bins in the histogram.

The feature vector at each pixel, (x, y) is defined as

$$F(x, y)_b = \begin{cases} r(x, y) & \text{if } B(x, y) \text{ is equal to } b \\ 0 & \end{cases} \tag{4.12}$$

Where:

$B(x, y)$ denotes either $B_a(x, y)$ or $B_b(x, y)$

The HOG feature vector, $\boldsymbol{F}_{HOG} = [f_1, ..., f_l]$, for the $k \times k$ cell is defined as

$$f_b = \sum_{x=0}^{k-1} \sum_{y=0}^{k-1} F(x, y)_b \tag{4.13}$$

The HOG feature vector used in this application is the concatenation of the contrast insensitive definition (Equation (4.10)) with $l = 9$ and the contrast sensitive definition (Equation (4.11)) with $l = 18$.

This 27-channel feature vector is block normalized by the $\ell 2$-norm (Equation (2.12)) as shown in Figure 4.3. This generates 4 normalized histograms which are then concatenated together to form a $4 \times (9 + 18) = 108$-channel feature vector. Thereafter, the dimensionality of the feature vector is reduced with negligible loss of information by applying PCA. This allows for more efficient computation without compromising performance. Thus, the feature map of the image patch, I_t, at the spatial scale, s, consists of 31-dimensional feature vectors, $\boldsymbol{F}_{HOG}(i, j)$: 18 contrast sensitive orientation channels, 9 contrast insensitive orientation channels and 4 texture channels that reflect the gradient energy of the cells surrounding (i, j) [89].

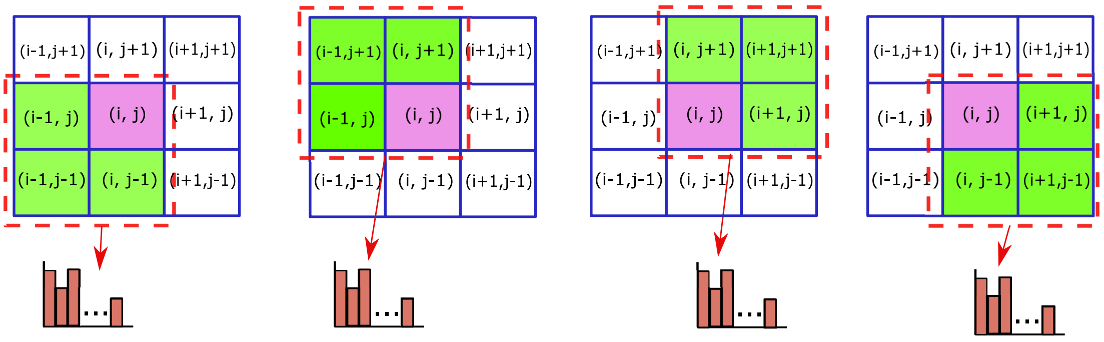


Figure 4.3: Generation of HOG features: For each block, the HOG feature vectors for the highlighted $k \times k$ cell are sum pooled and divided by the $\ell 2$-norm to form a normalized HOG feature histogram.

4.3.4 Scale Coding

Similarly to the image-based scale coding approach [8], features extracted via multi-scale sampling are partitioned into three groups based on the spatial scale from which they were extracted: small, medium and large (see Section 2.4.2). A BoW histogram is computed for each partition. The final representation is a concatenation of the histograms of each partition. However, for extension of this encoding approach to the spatio-temporal domain, the following must be handled:

- Size variations of the bounding box, B_t, over the frames
- Capturing temporal information

Absolute Scale Coding

Absolute scale coding is independent of the bounding box size. Thus, the spatio-temporal extension remains largely the same as the original approach [8]. S_{set} is defined as per Algorithm 4. The scales are partitioned into three sub-groups as follows:

$$S^s = \{s | s < s^s, s \in S_{set}\}$$
$$S^m = \{s | s^s \leq s < s^l, s \in S_{set}\}$$
$$S^l = \{s | s^l \leq s, s \in S_{set}\}$$

Where:

s^s and s^l are the predefined cutoff thresholds.

Let $\boldsymbol{F}_{t,s}$ be the extracted HOG feature map from the bounding box, B_t, at the spatial scale, $s \in S_{set}$, on frame t. BoW frequency histogram is constructed for the feature maps in each scale group. The final scale coded BoW is a concatenation of each of these histograms. This is given by:

$$h^v \propto \sum_{t=1}^{T} \sum_{s \in S^v} c_{BOW}(F_{t,s}) \tag{4.14}$$

Where: $v \in \{s, m, l\}$.

T is the total number of frames in the video sample.

c_{BOW} is the BoW coding scheme defined in Equation (2.4).

Relative Scale Coding

In this representation, extracted scale information is encoded relative to the bounding box of the object. Thus, relative scale coding is dependent on the size of the bounding box. With relative scale coding the scale of each feature, s, is multiplied by the relative scaling factor, α (Equation (4.15)) This factor is dependent on the dimensions of the bounding box. Since the size of the bounding box, B_t, varies from frame-to-frame, the relative scaling factor must be computed for each frame. This is defined as:

$$\alpha_t = \frac{B_{w,t} + B_{h,t}}{\bar{w} + \bar{h}} \tag{4.15}$$

Where:

$B_{w,t}$ and $B_{w,t}$ are the width and height of the bounding box B_t.

$\bar{w} + \bar{h}$ are the mean width and height of all bounding boxes in the training set.

Thus, the relative feature scale for each frame is given by:

$$\hat{s} = \alpha_t s = \frac{B_{w,t} + B_{h,t}}{\bar{w} + \bar{h}} s \tag{4.16}$$

This forms a new spatial scale set, $\hat{S}_{set} = \{\hat{S}_1 \cup \hat{S}_2 \cup ... \cup \hat{S}_t\}$, where $\hat{S}_t = \{\alpha_t s | s \in S_{set}\}$ Similarly to absolute scale coding, the relative set of scale sub-groups $\hat{S^v}, v \in \{s, m, l\}$ is partitioned as follows:

$$\hat{S^s} = \{\hat{s} | \hat{s} < s^s, \hat{s} \in \hat{S}_{set}\}$$

$$\hat{S^m} = \{\hat{s} | s^s \leq \hat{s} < s^l, \hat{s} \in \hat{S}_{set}\}$$

$$\hat{S^l} = \{\hat{s} | s^l \leq \hat{s}, \hat{s} \in \hat{S}_{set}\}$$

The final relative scale coded representation of the image involves concatenating the scale partitions. This is given by:

$$h^v \propto \frac{1}{|\hat{S^v}|} \sum_{t=1}^{T} \sum_{s \in \hat{S^v}} c_{BOW}(F_{t,s}) \tag{4.17}$$

Where:
$|\hat{S^v}|$ is a normalization factor equivalent to the number of elements in the set $\hat{S^v}$.

Scale Partitioning Strategies

The scale partitions are defined by the cutoff thresholds, s^s and s^l. In the original work [8], these are predefined parameters for a given dataset. Thus, all samples of the dataset have the same cutoff thresholds. However, for the proposed video-based implementation, the spatial-scale set, S_{set} is different for each video sample as is is dependent on the size of the initial bounding box B_0 (see Algorithm 4. Furthermore, additional variations in scale (bounding box size) occur within a given video sample as the visual tracking algorithm defining the bounding box over the temporal domain is scale adaptive. Thus, to accurately capture scale information, the cutoff thresholds should be dynamically set. The parameter setting scheme as per [8] will be referred to as **Static Scaling Parameter Definition**. The proposed dynamic setting of the cutoff thresholds will be referred to as **Dynamic Scaling Parameter Definition**

Where Q is the number of scale sub-groups, the number of required cutoff threshold parameters required is $Q - 1$. The cutoff thresholds, $s^v, v \in \{1, ..., Q - 1\}$, are set as per Algorithm 5.

Algorithm 5: Dynamic Scaling Parameter Definition

```
Inputs:
        $Q$ : number of scale sub-groups
        $S_{set}$: the set of all feature spatial scales
Output:
        $s^1, ..., s^v$: the cutoff thresholds
1 $R \leftarrow S_{set}$              // get total number of feature spatial scales
2
3 $S_{set} \leftarrow$sort($S_{set}$)
4 for $v \leftarrow 1$ to $Q - 1$ do
5     append $s$ to $S_{set}$
6     $i \leftarrow \frac{R}{Q} \times v$
7     $s^v \leftarrow S_{set}[i]$
8 end
```

Temporal Pyramid Structure

The proposed algorithm excludes an optical flow algorithm as the target object is identified through the visual tracking algorithm. Thus, to capture temporal information and add structure to the scale coded BoW, temporal pyramids are used (Figure 4.4). Three temporal cells are defined as

$$T_p = \{t|(\frac{T}{3} \times (p-1)) \leq t < (\frac{T}{3} \times p)\} \tag{4.18}$$

Where:
T is the total number of frames in the video sample.
$t \in \{1, ..., T\}$.
$p \in \{1, 2, 3\}$.

A scale coded BoW, SC_{BOW_p}, is constructed for each temporal cell using either the absolute or relative scale coding scheme.

$$SC_{BOW_p} = h^v \propto \sum_{t\ inT_p} \sum_{s \in \hat{S^v}} c_{BOW}(F_{t,s}) \tag{4.19}$$

Finally, the scale coded BoW for each temporal cell is pooled together and normalized.

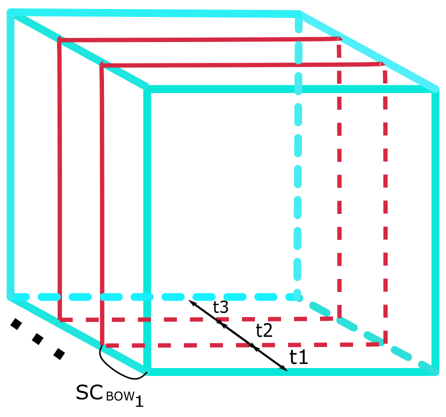


Figure 4.4: Temporal pyramid for scale coded BoW.

4.4 Experimentation

4.4.1 PC Specifications and Datasets

The PC specifications are the same as stated in Section 3.4.1. To evaluate performance of the proposed spatio-temporal scale encoded BoW, the KTH [11] and HMDB51 [10] datasets are used. The setup is the same as stated in 3.4.2 except the full HMDB51 dataset is used for evaluation as opposed to the reduced version. The class distribution in both datasets is balanced. The relatively simple KTH dataset is used as a minimum baseline in order to evaluate the ability of the proposed algorithm to perform action classification. This dataset is filmed from a static camera and contains scale and translation variations of the target object. The more challenging HMDB51 dataset is used to evaluate the robustness of the action classifier. In addition to target object translation and scale variations, this dataset includes camera motion, occlusion and background activities. For both datasets, performance is evaluated by computing the average accuracy over all the classes.

4.4.2 Object Detection

The cvlib[2] implementation is used. The YOLOv3 model used was trained on the COCO dataset [3] and is able to detect 80 objects including a person.

The minimum required dimensions of the initial bounding box, B_0, is set to $dim_{min} = 24pixels$. A 24×24 bounding box ensures at least one feature map containing 2 PCA-

[2]https://github.com/arunponnusamy/cvlib

[3]https://cocodataset.org/#home

HOG feature vectors can be extracted via multi-scale sampling. As shown in Figure 4.5, the person in the video is successfully detected and isolated by the initial bounding box, B_0.

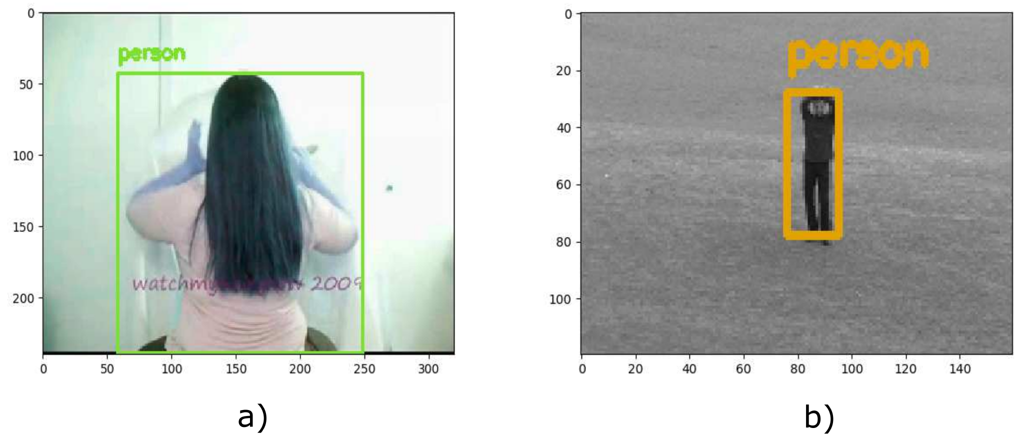


Figure 4.5: Successful definition of the initial bounding box, B_0, for video samples from the a) 'brush hair' class of HMDB51 [10] b) 'hand waving' class of KTH [11].

4.4.3 Visual Tracker

The set of parameter values for implementation of the DCF-DSST tracker are summarized in Table 4.2.

Table 4.2: Parameters for the DCF-DSST Visual Tracker

Parameter	**Description**	**Value**
Parameters for Translation Model (DCF) as per [5]		
λ_{trans}	Regularization parameter for translation model	0.0001
η_{trans}	Learning rate for translation model	0.02
Parameters for Scale Model as per [65]		
λ_{scale}	Regularization parameter for scale model	0.01
η_{scale}	Learning rate for translation model	0.025
S	Number of scales	33
sf	Scale Factor	1.02

4.4.4 Bag of Words

PCA-HOG[4] features are efficiently extracted from training samples (as covered in Section 4.3.3) using a 4×4 cell grid. K-means clustering is used to cluster a subset of 100 000 randomly selected training features to construct a BoW vocabulary of 4000 words. The scale coded BoW is formed from the extracted features as outlined in Section 4.3.4.

Both the absolute and relative scale coding schemes are evaluated. Furthermore, the effects of the proposed scale partitioning strategies (static and dynamic scaling parameter definition) are evaluated on the KTH dataset in order to determine which strategy yields the best results. For static parameter definition, the cutoff thresholds are set as: $s^s = \sqrt{2}$, $s^l = 2$.

To lower computation time, the scale coding process is parallelized such that each scale sub-group, S^s, S^m and S^l, are computed on individual threads.

4.4.5 Classification

Similarly to [8], a one-vs-rest SVM is used for multi-class classification. Three SVM kernels were considered:

1. A linear kernel defined as [78]

$$K(x_i, x_j) = x_i \bullet x_j \tag{4.20}$$

   Where:

   $\bullet$ denotes the dot product.

2. A χ^2 kernel defined as [78]

$$K(x_i, x_j) = exp(-\frac{1}{A} D(x_i, x_j)) \tag{4.21}$$

   Where:

   $D(x_i, x_j)$ is the χ^2 distance between each training video x_i and x_j.

   A is a scaling parameter.

3. An rbf kernel defined as [90]

$$K(x_i, x_j) = exp(-\gamma \|x_i - x_j\|^2) \tag{4.22}$$

[4]https://github.com/uoip/KCFpy/blob/master/fhog.py

The kernel type, A and γ values are hyper-parameters that are tuned using grid search with cross validation. This involves randomly partitioning the training set into k-folds. One fold is selected to be the validation set while the remaining $k-1$ folds are used as training. This process is repeated k times; alternating the fold selected to be the validation. The accuracy values achieved are averaged to form a single estimation. 5-fold cross validation is used (see Figure 4.6). Each combination of hyper-parameter values are evaluated and the combination returning the best score is selected.

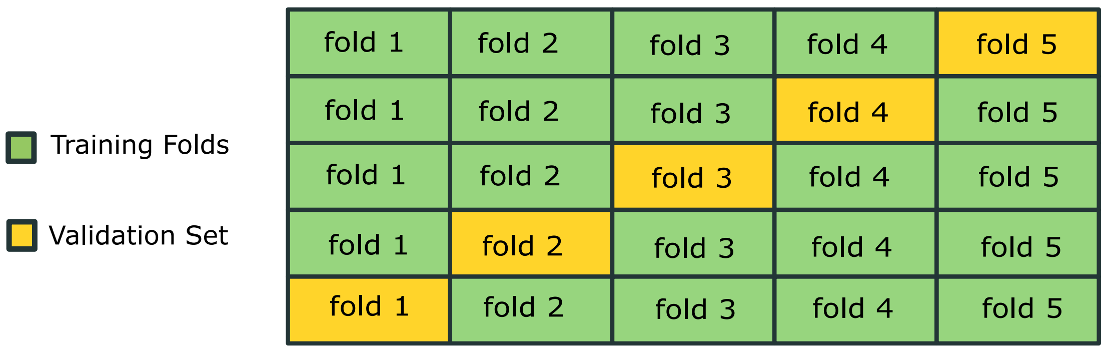


Figure 4.6: 5-fold cross validation of the training set for hyper-parameter tuning.

Table 4.3: Performance of the Spatio-Temporal Scale Coded BoW on the KTH dataset

Scale Coding Scheme	**Parameter Definition Scheme**	**Hyper-parameters**	**Accuracy (%)**
Absolute	Static	kernel = 'linear'	63.42
	Dynamic	kernel = 'linear'	63.89
Relative	Static	kernel = 'linear'	61.57
	Dynamic	kernel = 'linear'	**64.81**

Table 4.4: Performance of the Spatio-Temporal Scale Coded BoW on the HMDB51 dataset with Dynamic Scale Parameter Definition

Scale Coding Scheme	**Hyper-parameters**	**Accuracy (%)**
Absolute	kernel = 'rbf', $\gamma = 0.01$	**27.78**
Relative	kernel = 'rbf', $\gamma = 0.01$	21.24

4.5 Results and Discussion

4.5.1 Scale Partitioning Strategies

The effects of each scale partitioning strategy can be observed in Table 4.3. Dynamically defining the cutoff thresholds, s^s and s^l, results in better performance compared to the static strategy. Notably, the improvement in accuracy was much higher for the relative scale encoding scheme (+3.24%) compared to the absolute scale encoding scheme (+0.47%). This was expected as the set of extracted scales for relative scale encoding, $\hat{S}_{set}$, is dynamically created since it is dependent on the size of the bounding box, B_t, in each frame. Whereas the set of extracted scales for absolute scale encoding, S_{set}, is created on the first frame since it is only dependent on the initial bounding box, B_0. Thus, it is more difficult to predict the cutoff thresholds that will partition extracted scales into scale sub-sets of equal size for the relative scale encoding scheme. This results in a less accurate representation of extracted scale information; lowering accuracy. It can be concluded that for video-based scale coding, dynamic definition of the cutoff thresholds ensures that extracted scale information is more accurately represented and thus results in a better performance.

4.5.2 Scale Coding Schemes

Observing the results achieved on the KTH dataset (Table 4.3), the relative scale coding scheme (64.81 %) achieves a slightly higher accuracy than the absolute scale coding scheme (63.89 %). The same outcome was observed in the image-based scale coded BoW [8]; relative scale coding outperformed the absolute scheme by 0.7 %. This may be because the relative encoding scheme accounts for the size of the bounding box, thus encodes more information into the final representation compared to the absolute coding scheme. As per Remark 2, this results in a higher accuracy. However, for the HMDB51 dataset, the opposite is observed (see Table 4.4). The absolute coding scheme (27.8 %) outperforms the relative coding scheme (21.24%) by a relatively large margin of 6.56 %. This may be due to the the visual tracking algorithm inaccurately defining the bounding box, B_t, in each frame t, as the HMDB51 dataset contains occlusion, camera motion and background activities. As per the conclusions of Chapter 3, capturing unimportant information can reduce performance and wastes resources. Since the relative coding scheme is heavily dependent on the bounding box, the errors of the visual tracker have a greater negative impact on the relative coding scheme compared to the absolute coding scheme.

4.5.3 Computational Cost

The time taken to complete each task of the proposed scale coded BoW was computed for 25 randomly selected video samples in each dataset. These values were averaged and are presented in Table 4.5.

Table 4.5: The Computational Cost of Each Task in the Spatio-Temporal Scale Coded BoW for the KTH and HMDB51 Datasets

Task	**KTH**	**HMDB51**
Object Detection	1.01s	1.81s
Feature Extraction	7.15s	7.73s
Scale Coding	5.44s	4.35s
Total Processing Time t_p	**13.60 s**	**13.89 s**
Average Number of frames	335.95	417.95
Processing Frequency f_p	**24.70 fps**	**30.09 fps**
Capturing Frequency f_c	25 fps [11]	30 fps [10]

As per Equation (4.2), the proposed pipeline operates in real-time. Although by definition, the KTH dataset does not operate in real-time since the processing frequency (f_p) is slightly less than the capturing frequency (f_c), the difference between these values is small. Thus, processing is perceived to be in real-time. Furthermore, the hardware of the PC on which experiments were conducted is outdated. Thus, the proposed action recognition pipeline will easily operate in real-time on state-of-the-art PCs.

On average, formation of the SC-BoW for a vocabulary size of 4000 words operates at 78.92 fps. This is relatively efficient (given the outdated hardware on which experiments were conducted) and is done in real-time for the proposed pipeline. The computation time of SC-BoW representations can be improved by computing the BoW using the algorithm presented in [91].

4.5.4 Comparison to Existing Methods

Observing the comparisons in Table 4.6 and Table 4.7, the proposed method is outperformed by most cited existing methods. However, the spatio-temporal scale coded BoW outperforms the HOG/HOF feature-based algorithm proposed by Kuehne *et al.*[10] by 4.6% (see Table 4.7). Furthermore, the scale coded BoW outperforms the Local Part Model [68] HOG feature set by 6.78% even though the scale coded BoW is a relatively simpler approach. Thus, it can be concluded that the inclusion of scale information in the BoW framework improves the action classification of the algorithm. The performance of the proposed framework is lower than state-of-the-art real-time action recognition approaches due to the weak HOG features used.

4.6 Conclusions

The scale coded BoW for image-based action recognition [8] was extended to form the spatio-temporal scale coded BoW for video-based action recognition. This algorithm successfully performs action recognition of videos in real-time. Additionally, two strategies for setting the cutoff thresholds that define the scale set partitions were proposed: Static Scaling Parameter Definition and Dynamic Scaling Parameter Definition. Through experimental evaluation, it was found that the dynamic strategy defined the scale partitions more accurately; yielding a better performance.

A large setback of the proposed algorithm is its dependence on the the performance of the visual tracker. If the visual tracker fails; large amounts of unimportant information will be captured. As per the conclusions of Chapter 3, capturing unimportant information can reduce performance and wastes computational resources. Thus, it would be beneficial to reduce the reliance of this algorithm on the visual tracker by either:

- Removing the need for a visual tracker by using alternative methods to define the target object through subsequent frames.
- Remove dependence of the relative coding scheme on the definition of a bounding box by using alternative cues to compute relative scale (e.g depth information).

Finally, it was observed that the spatio-temporal scale coded BoW improves the performance of HOG features for action recognition. Applying this framework on more robust feature sets has potential to achieve state-of-the-art performance at a high computational efficiency. This overcomes the cost-vs-accuracy balance highlighted in Chapter 3.

Table 4.6: Comparison to Existing Action Recognition Methods on the KTH dataset

Method	**Accuracy(%)**
KTH [11]	71.72
Local Part Model [68]	93.0*
PSRM [92]	95.67*
Scale Coded BoW	**64.81***

* Algorithms that operate in real-time

Table 4.7: Comparison to Existing Action Recognition Methods on the HMDB51 dataset

Method	**Accuracy(%)**
HMDB51 [10] (Combined)	23.18
HOGHOF	20.44
HOG	15.47
HOF	22.48
Local Part Model [68] (Combined)	47.6*
HOG	21.0
HOF	33.5
HOG3D	34.7
MBH	43.0
Motion Vector CNNs [93]	55.3*
Scale Coded BoW	**27.78***

* Algorithms that operate in real-time

Chapter 5

Scaled Spatio-Temporal Representations

5.1 Introduction

The experimental results obtained in Chapter 4 showed scale coded spatio-temporal BoW (SC-BoW) representations to outperform more complex approaches on the HOG feature set for video-based action recognition. This highlighted scale coding as a potential low-cost solution to increase performance; presenting a favorable cost-to-accuracy balance. In this chapter, the potential of scale coding for real-time action recognition of videos is further investigated.

The SC-BoW approach is formalized such that it can be easily incorporated into any algorithm utilizing the BoW for feature representation. Furthermore, the spatio-temporal pyramids commonly used to add structure to BoW representations of videos [2] were extended to incorporate scale information. In order to directly observe the performance increase SC-BoW representations can achieve, the formalized scale coding scheme is applied to the more robust dense trajectories feature set [1]. The performance gain and computational cost increase incurred are analyzed in order to determine if SC-BoW representations are a promising avenue for practical implementation of real-time video-based action recognition.

5.2 Adapted Spatio-Temporal Scale Coding

The relative scale coding scheme requires bounding box definition on each frame. As a result, a visual tracker is needed. As per the conclusions of Chapter 4, reliance of the scale coding scheme on a visual tracker poses many setbacks and can reduce performance if the bounding box is inaccurately defined. Furthermore, majority of existing action recognition algorithms do not isolate the object via bounding boxes. As a result, incorporation of relative scale coding into existing algorithms is challenging as it requires an additional scheme to establish the relative scale. The absolute scale coding scheme is more flexible as it only requires multi-scale sampling and feature extraction; these steps are a staple in high-performing action recognition approaches [1, 12, 14, 68, 71]. Thus, for an adaptable scale coding scheme that can easily be incorporated into any existing algorithm utilizing BoW representations, the absolute scale coding scheme is used. The formalized scale coding scheme is summarized in Figure 5.1.

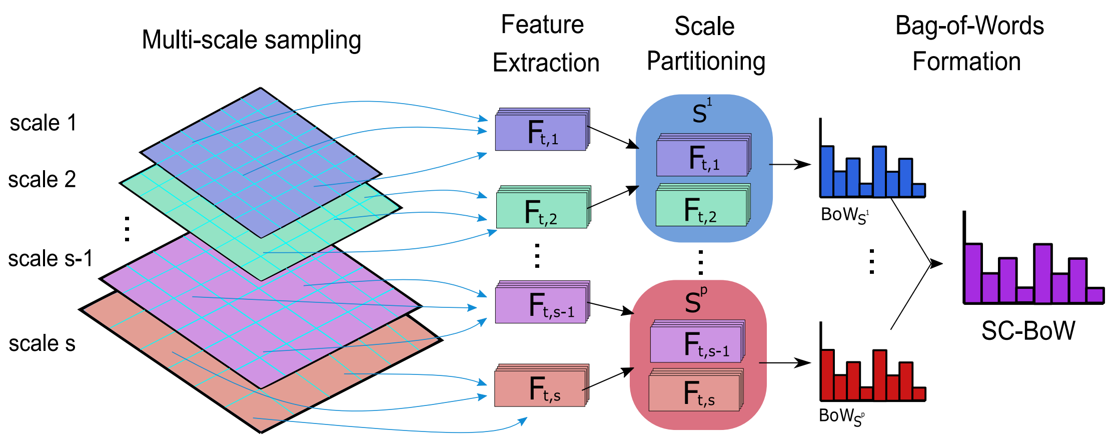


Figure 5.1: The general pipeline for formation of scale coded BoW representations.

The absolute scheme, first defined in section 4.3.4, is generalized for a variable number of partitions p. Let $\boldsymbol{F}_{i,t,s}$ be the set of features extracted from a video at the pixel $i = (x, y)$, on frame t, at the spatial scale $s \in S_{set}$. S_{set} is the set of extracted feature scales. The scales are partitioned into p sub-groups as follows:

$$\begin{aligned}
&S^1 = \{s|s < s^1, s \in S_{set}\} \\
&S^2 = \{s|s^1 \leq s < s^2, s \in S_{set}\} \\
&\vdots \\
&S^{p-1} = \{s|s^{p-2} \leq s < s^{p-1}, s \in S_{set}\} \\
&S^p = \{s|s^{p-1} \leq s, s \in S_{set}\}
\end{aligned}$$

Where:
$s^1, s^2, ..., s^{p-1}$ are the cutoff thresholds.
$1 < p \leq |S_{set}|$. $|S_{set}|$ is the cardinality of S_{set}.

The cutoff thresholds are defined by dynamic scale parameter definition (see Algorithm 5). Dynamic definition yields a better performance compared to the static scheme as per the experimental results of Chapter 4.

A BoW frequency histogram is constructed for the features in each scale sub-group. The final scale coded BoW is a concatenation of each of these histograms. This is given by:

$$h^v \propto \sum_{i=(0,0)}^{(X,Y)} \sum_{t=1}^{T} \sum_{s \in S^v} c_{BOW}(F_{i,t,s}) \tag{5.1}$$

Where:
$v \in \{1, \ldots, p-1, p\}$.
(X, Y) is the maximum pixel value in the video. X is the video width in pixels. Y is the video height in pixels.
T is the total number of frames in the video sample.
c_{BOW} is the BoW coding scheme defined in Equation (2.4).

5.3 Scaled Spatio-Temporal Pyramids

In action recognition tasks involving BoW representations, spatio-temporal pyramids [2] are commonly used to add additional structure and improve performance. Adding spatio-temporal structure to scale coded representations involves computing a SC-BoW for each spatio-temporal cell (see Figure 5.2 (a)).

The spatio-temporal cell, c, is the union of a spatial cell division (this consists of a division in the horizontal and vertical spatial plane respectively) and a temporal cell division. Consider a spatio-temporal pyramid with C_x vertical spatial cells, C_y horizontal spatial cells and C_t temporal cells. A single spatio-temporal cell is defined as:

$$\begin{aligned} c_{p_x,p_y,p_t} = &\{x|(\frac{X}{C_x} \times (p_x - 1)) \leq x < (\frac{X}{C_x} \times p_x)\} \cup \\ &\{y|(\frac{Y}{C_y} \times (p_y - 1)) \leq y < (\frac{X}{C_y} \times p_y)\} \cup \\ &\{t|(\frac{T}{C_t} \times (p_t - 1)) \leq t < (\frac{T}{C_t} \times p_t)\} \end{aligned} \tag{5.2}$$

Where:
X is the maximum pixel value in the $x-$direction.
Y is the maximum pixel value in the $y-$direction.
T is the total number of frames in the video sample.
p_x, p_y, p_t represent the cell number in the x, y and t planes respectively.

The SC-BoW is thereafter constructed as:

$$SC_{BOW_c} = h^v \propto \sum_{c} \sum_{s \in \hat{S}^v} c(F_{c,s}) \tag{5.3}$$

Where:
$F_{c,s}$ is the set of features extracted from spatio-temporal cell c.
c denotes the spatio-temporal partition c_{p_x,p_y,p_t} defined by Equation (5.2).

The SC-BoW for each spatio-temporal cell is pooled together and normalized to form the final scale coded spatio-temporal BoW. This effectively extends spatio-temporal pyramids to 4-dimensions as shown in Figure 5.2 (b).

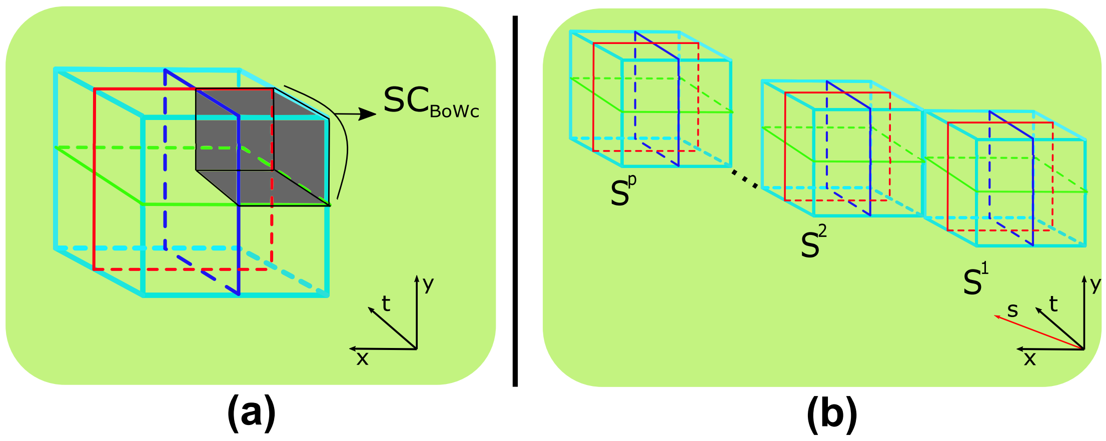


Figure 5.2: Scaled Spatio-Temporal Pyramids: (**a**)The first representation involves computing a SC-BoW for each cell. (**b**)The second representation adds scale as a 4th dimension and involves computing a standard BoW for each cell.

5.4 Experimentation

This section outlines the experimental setup for applying scale coding to dense trajectory features. Following the original dense trajectories experimental conditions [1], the algorithm is run on a single CPU core; the code is not parallelized.

5.4.1 Datasets

Performance evaluation of the scale coded dense trajectory feature set is done using the KTH[1] dataset [11] and a reduced HMDB51[2] [10] datasets. The setup of these datasets is the same as outlined in Chapter 3, Section 3.4.2. Performance is measured by the average accuracy over all classes.

Following the original setup [11], the KTH dataset is divided into testing and training sets based on the subjects. The testing set consists of subjects 2, 3, 5, 6, 7, 8, 9, 10, and 22 (9 subjects total) and the training set is made up of the remaining 16 subjects. The average accuracy is taken over all classes.

The reduced HMDB51 dataset consists of 11 action classes: brush hair, cartwheel, catch, chew, clap, climb stairs, smile, talk, throw, turn, wave. As per the original setup

[1]http://www.nada.kth.se/cvap/actions/

[2]http://serre-lab.clps.brown.edu/resource/hmdb-a-large-human-motion-database/

[10], each action class is organized into 70 videos for training and 30 videos for testing.

5.4.2 Scale Coding Dense Trajectory Features

To produce scale coded dense trajectory feature representations, the feature extraction process remains the same. Only the feature encoding stage is modified; this is the stage where the BoW representation of the video sample is formed.

Feature Extraction

Dense trajectory feature formation involves densely sampling points and tracking sampled points over the temporal domain via an optical flow algorithm to form a trajectory. Points are re-sampled every R frames. Five feature descriptors are extracted every frame: trajectory shape, HOG, HOF, MBHx and MBHy. The complete set of extracted features for a given video sample is a concatenation of these 5 descriptors. This is denoted as $\boldsymbol{F} = [\boldsymbol{F}_{TS}, \boldsymbol{F}_{HOG}, \boldsymbol{F}_{HOF}, \boldsymbol{F}_{MBHx}, \boldsymbol{F}_{MBHy}]$. The dense trajectory feature formation process was described in more detail in Chapter 3, Section 3.4.3.

The only additional information required to apply scale coding is the set of extracted feature scales. This is obtained during the sampling stage. The sampling strategy used involves sampling points from a frame t via a sampling grid. For dense sampling, a grid spacing of $W = 5$ is used [1]. The dense sampling of points is done on multiple scales. Each feature scale is separated by a factor of $\sqrt{2}$. The set of extracted scales, S_{set}, is defined as per Algorithm 4 (Chapter 4, Section 4.3.3). The inputs to Algorithm 4 are set as follows:

- The initial bounding box, B_0, encapsulates the entire video frame. Let $X \times Y$ be the resolution of the video sample. Thus, the dimensions of B_0 is $X \times Y$.
- Wang *et al.* [1] set the neighborhood size for computing descriptors to 32 pixels. Thus, the minimum dimension, dim_{min}, is set to 32 pixels. In this manner, the dimensions of the sample patch on the smallest feature scale will be greater than 32×32. This ensures at least one set of descriptors can be extracted on each spatial scale.
- Following the original setup [1], a maximum of 8 spatial scales are extracted, $S_{max} = 8$.

SC-BoW Representation

Following [1], k-means clustering is used to cluster a subset of 100 000 randomly selected training features. A vocabulary of 4000 words is separately constructed for each descriptor.

For a given video sample, the complete set of extracted features, $\boldsymbol{F}$, is partitioned into sub-groups based on the scale on which they were extracted. This process was outlined in Section 5.2.

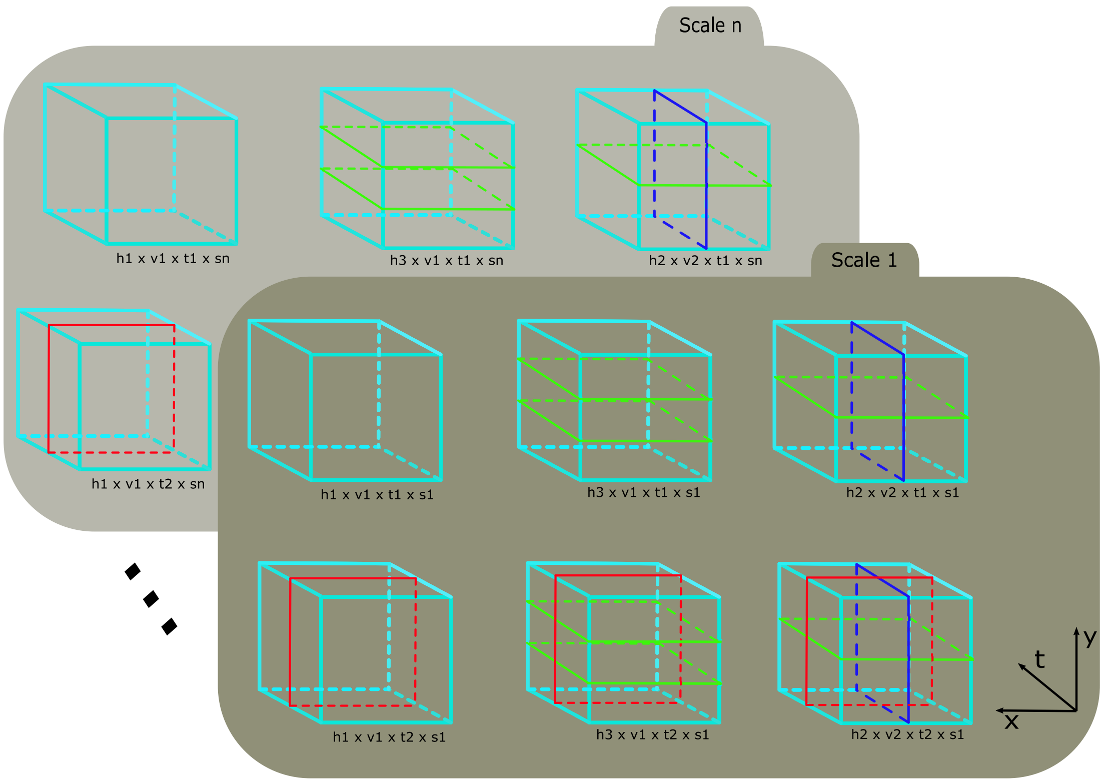


Figure 5.3: Scaled Spatio-Temporal grids: on each scale, 6 spatio-temporal pyramids are constructed as indicated.

The standard dense trajectories algorithm uses 6 spatio-temporal pyramids to add structure to BoW representations [1]. As outlined in Section 3.4.3, the pyramids are extended to 4-dimension in order to hold scale information. The 6 scaled spatio-temporal pyramid structures used are illustrated in Figure 5.3. A BoW is constructed for each scaled spatio-temporal cell. Thereafter, a global representation of the pyramid is found by summing the BoW histogram for each cell and normalizing by the RootSIFT Norm

[52].

The 6 scaled spatio-temporal pyramids are computed for each descriptor. Each pyramid-descriptor pair forms a separate channel; there are 30 channels in total (6 pyramids x 5 descriptors). This forms the scale coded representation of the dense trajectory feature set. Finally, a multi-class, multi-channel non-linear SVM with an RBF-χ^2 kernel [11] is used for classification. The multi-channel kernel is defined by Equation (3.5).

5.5 Results and Discussion

5.5.1 Performance Analysis

For the KTH dataset, experiments were run twice; setting the sub-sampling refresh rate to R = 1 frame and R = 6 frames respectfully.

The time taken to obtain SC-BoW representations from extracted dense trajectory features was recorded for 25 randomly selected video samples from the KTH dataset. These values were averaged and are presented in Table 5.1. The average number of extracted features clustered for each video sample was 6000. SC-BoW representations were formed with 3 scale partitions.

SC-BoW representations were found to improve performance on the KTH by 2.76 % and on the reduced HMDB51 dataset by 3.64 % (see Table 5.2). The increase in performance was accompanied by a 5.59s increase in computational time (Table 5.1). For the standard dense trajectory algorithm, it was experimentally found that BoW formation accounts for 2% of total computation time (9013s). Thus, incorporation of

Table 5.1: Computational Analysis of SC-BoW given a 4k-word vocabulary and 6000 extracted features

Task	**Computation Time (s)**	**Computation Frequency (FPS)**
SC-BoW Formation	192.24	5.22
BoW Formation	186.65	6.10
Added Cost	**5.59**	**0.88**

Table 5.2: Comparing Performance of Dense Trajectories with and without scale coding for the KTH and reduced HMDB51 datasets

Description	**KTH (R = 1)**	**KTH (R = 6)**	**HMDB51 (reduced)**
DT (%)	94.0	91.0	81.21
DT + SC-BoW (%)	96.76	93.24	84.85
Net Change (%)	**+2.76**	**+2.24**	**+3.64**

SC-BoW increases computation time by 0.06%. This is a negligible increase in most cases. Additionally, the added computation time can be further reduced by parallelizing the code and adopting the scaled BoW generation scheme proposed by Singh *et al.* [91]. It can be concluded that SC-BoW representations increase accuracy with negligible added computational cost; presenting an ideal cost-to-accuracy balance.

It can also be seen that the performance increase achieved by SC-BoW representations is similar for different sub-sampling rates. This shows that scale coding is not influenced by the number of points sampled; it simply enhances performance of the multi-scale extracted features.

The accuracy in each class for scale coded and non-scale coded dense trajectories was recorded for both datasets. This is summarized in Figure 5.4 (KTH dataset) and Figure 5.5 (reduced HMDB51 dataset). For the KTH dataset, SC-BoW representations improved the accuracy off all classes except ”running”. Observing the plotted confusion matrices in Figure 5.6, it can be seen that ”running” had two false positives for ”jogging” and ”walking”. These actions are very similar; they are differentiated by temporal information: jogging is a faster walk, running is a faster jog. Thus, extracting additional temporal information could possibly better differentiate between these classes. This can be done by extending trajectory length to $L \geq 15$ frames or by defining additional temporal divisions in the scale spatio-temporal pyramids.

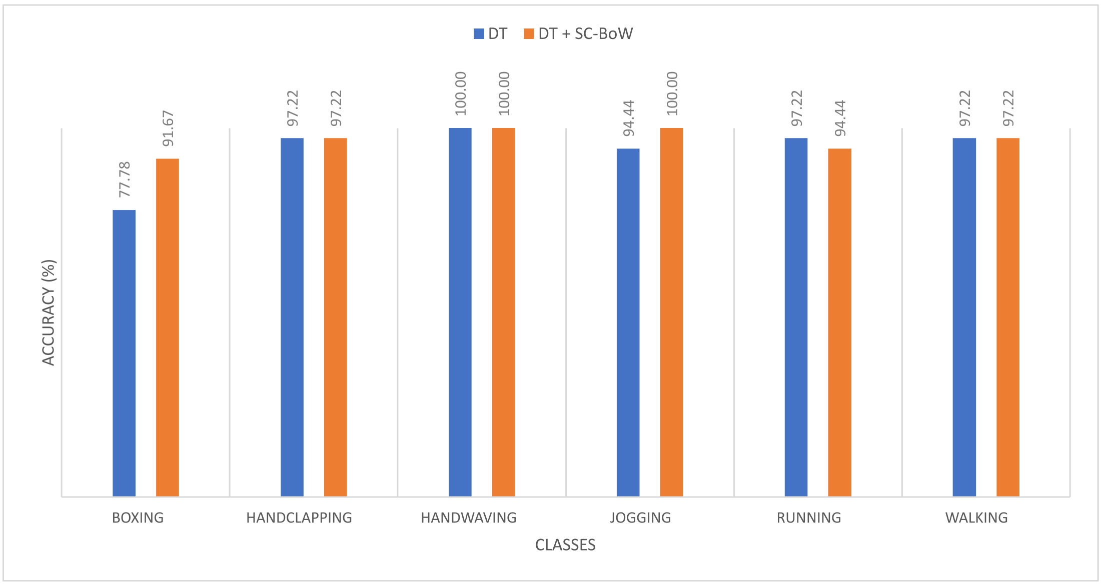


Figure 5.4: Plot comparing the class accuracies obtained on the KTH dataset for dense trajectories and scale coded dense trajectories.

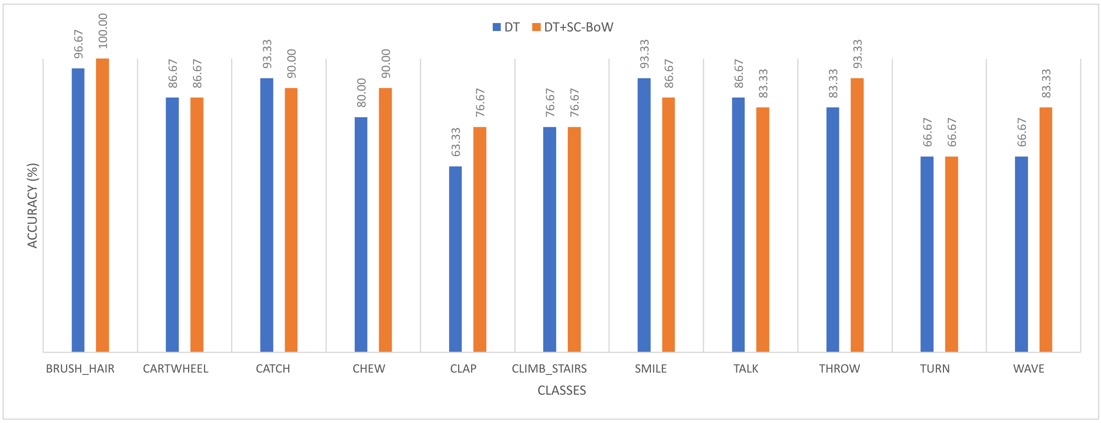
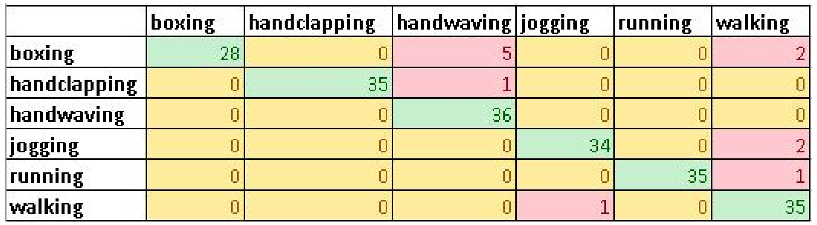
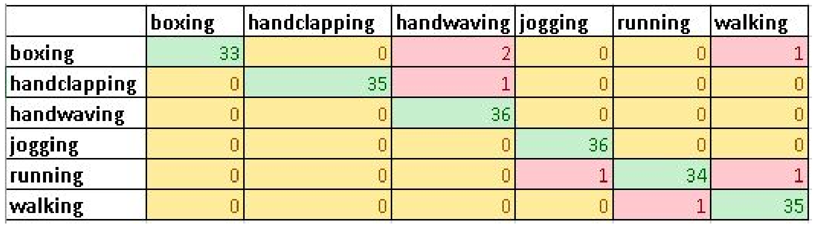


Figure 5.5: Plot comparing the class accuracies obtained on the reduced HMDB51 dataset for dense trajectories and scale coded dense trajectories.



	boxing	handclapping	handwaving	jogging	running	walking
boxing	28	0	5	0	0	2
handclapping	0	35	1	0	0	0
handwaving	0	0	36	0	0	0
jogging	0	0	0	34	0	2
running	0	0	0	0	35	1
walking	0	0	0	1	0	35

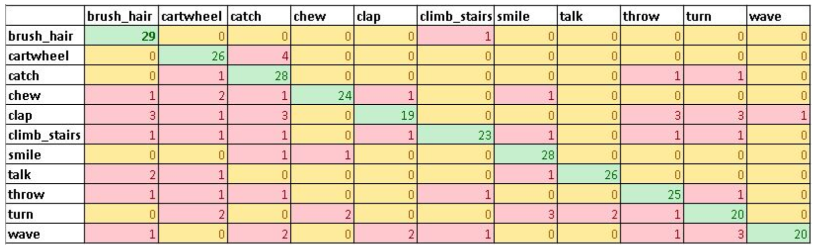
(a)



	boxing	handclapping	handwaving	jogging	running	walking
boxing	33	0	2	0	0	1
handclapping	0	35	1	0	0	0
handwaving	0	0	36	0	0	0
jogging	0	0	0	36	0	0
running	0	0	0	1	34	1
walking	0	0	0	0	1	35

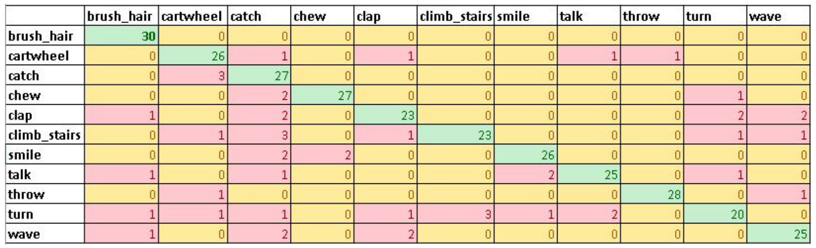
(b)

Figure 5.6: Confusion matrix plot of the KTH dataset (R = 1) where (a) dense trajectory features were extracted. (b) scale coded dense trajectory features were extracted.



	brush_hair	cartwheel	catch	chew	clap	climb_stairs	smile	talk	throw	turn	wave
brush_hair	29	0	0	0	0	1	0	0	0	0	0
cartwheel	0	26	4	0	0	0	0	0	0	0	0
catch	0	1	28	0	0	0	0	0	1	1	0
chew	1	2	1	24	1	0	1	0	0	0	0
clap	3	1	3	0	19	0	0	0	3	3	1
climb_stairs	1	1	1	0	1	23	1	0	1	1	0
smile	0	0	1	1	0	0	28	0	0	0	0
talk	2	1	0	0	0	0	1	26	0	0	0
throw	1	1	1	0	0	1	0	0	25	1	0
turn	0	2	0	2	0	0	3	2	1	20	0
wave	1	0	2	0	2	1	0	0	1	3	20

(a)



	brush_hair	cartwheel	catch	chew	clap	climb_stairs	smile	talk	throw	turn	wave
brush_hair	30	0	0	0	0	0	0	0	0	0	0
cartwheel	0	26	1	0	1	0	0	1	1	0	0
catch	0	3	27	0	0	0	0	0	0	0	0
chew	0	0	2	27	0	0	0	0	0	1	0
clap	1	0	2	0	23	0	0	0	0	2	2
climb_stairs	0	1	3	0	1	23	0	0	0	1	1
smile	0	0	2	2	0	0	26	0	0	0	0
talk	1	0	1	0	0	0	2	25	0	1	0
throw	0	1	0	0	0	0	0	0	28	0	1
turn	1	1	1	0	1	3	1	2	0	20	0
wave	1	0	2	0	2	0	0	0	0	0	25

(b)

Figure 5.7: Confusion matrix plot of the reduced HMDB51 dataset (R = 6) where (a) dense trajectory features were extracted. (b) scale coded dense trajectory features were extracted.

5.5.2 Number of Scale Partitions

To observe the effects of the number of scale partitions on performance, the achieved accuracy for each partition, $p \in \{1, \ldots, n\}$, was recorded. $p = 1$ corresponds to the original algorithm with no scale coding. n is the maximum spatial scale and is dependent on the resolution of the video sample. For the KTH dataset, $n = 4$. For the reduced HMDB51 dataset, $n = 6$.

Table 5.3: The Effect of the Number of Scale Partitions on Accuracy

Scale Partitions	**KTH (R = 1)**	**KTH (R = 6)**	**HMDB51 (reduced)**
1	94.0	91.0	81.21
2	94.91	91.39	82.2
3	**96.76**	**93.24**	**84.85**
4	90.27	89.35	82.99
5	-	-	80.91
6	-	-	77.57

Observing the results in Table 5.3 and Figure 5.8, there exists an optimum number of scale partitions. For both the KTH and reduced HMDB51 dataset, this was found to be $p = 3$. Similarly, Khan *et al.* [20] had found a negligible gain in performance beyond 3 scale partitions for image-based action recognition. However, in the video-base approach, defining scale partitions beyond the optimum number can result in performances lower than non-scale coded representations. Thus, the best practice is to form SC-BoW representations with $p = 3$ partitions. In most cases, this will achieve the optimal performance.

Additionally, it can be seen that the trend line for both sub-sampling rates on the KTH dataset is similar; further solidifying that performance gain achieved through scale coding is independent of the number of points sampled. The time take to compute an SC-BoW for each number of partitions was recorded for four videos from the KTH dataset (R = 1). Approximately 10k, 6k, 4k and 2k dense trajectory features were extracted from the four videos respectively. This is summarized in Figure 5.9. It

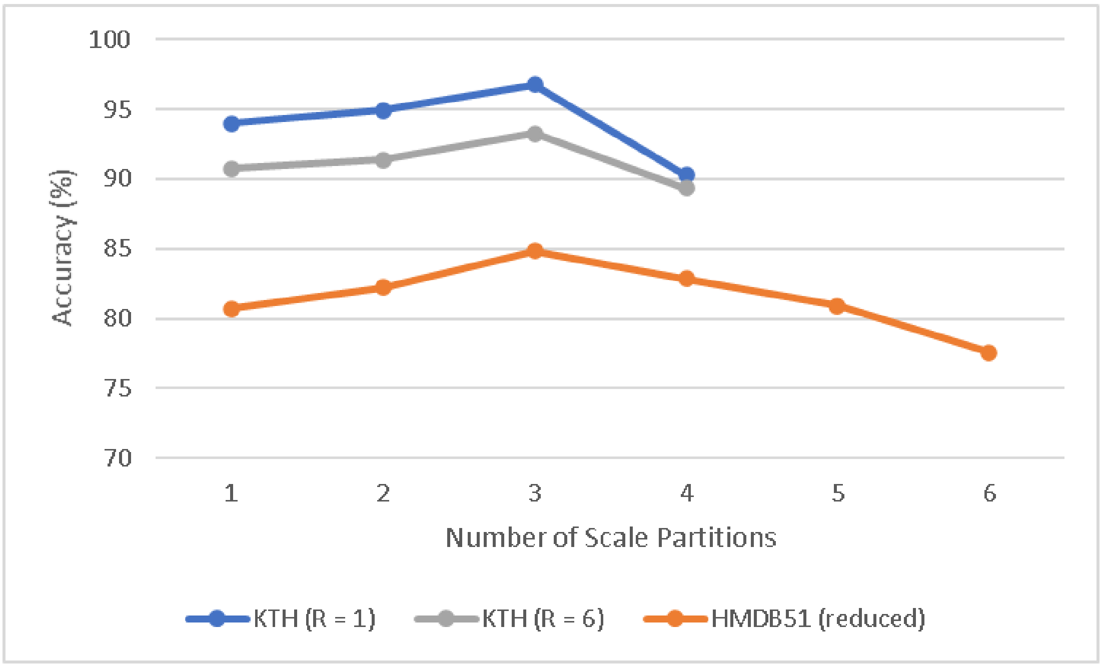


Figure 5.8: The effect of number of scale partitions on accuracy for the KTH and HMDB51(reduced) datasets.

can be seen that the relationship between number of scale partitions and computational cost is not linearly directly correlated. Even though a larger number of scale partitions requires more BoW computations, the BoW computation time for each partition varies depending on the number of features in the partition. The number of features in each scale partition is dependent on various factors: the number of sampling scales, the feature extraction algorithm and the video content.

5.5.3 Comparison to Existing Methods

Observing the values in Table 5.4, applying SC-BoW to dense trajectory features results in performance close to state-of-the-art. Notably, scale coded dense trajectories outperformed complex deep learning approaches that use the popular two-stream CNN network architecture [94]. This re-affirms the conclusions of Chapter 4; scale coding shows better performance than more complex approaches. Furthermore, this illustrates that deep learning approaches do not always outperform classical machine learning approaches. In smaller datasets, such as the KTH dataset, classical machine learning approaches are much more valuable. This is favorable in real-world circumstances where large amounts of training data are unavailable.

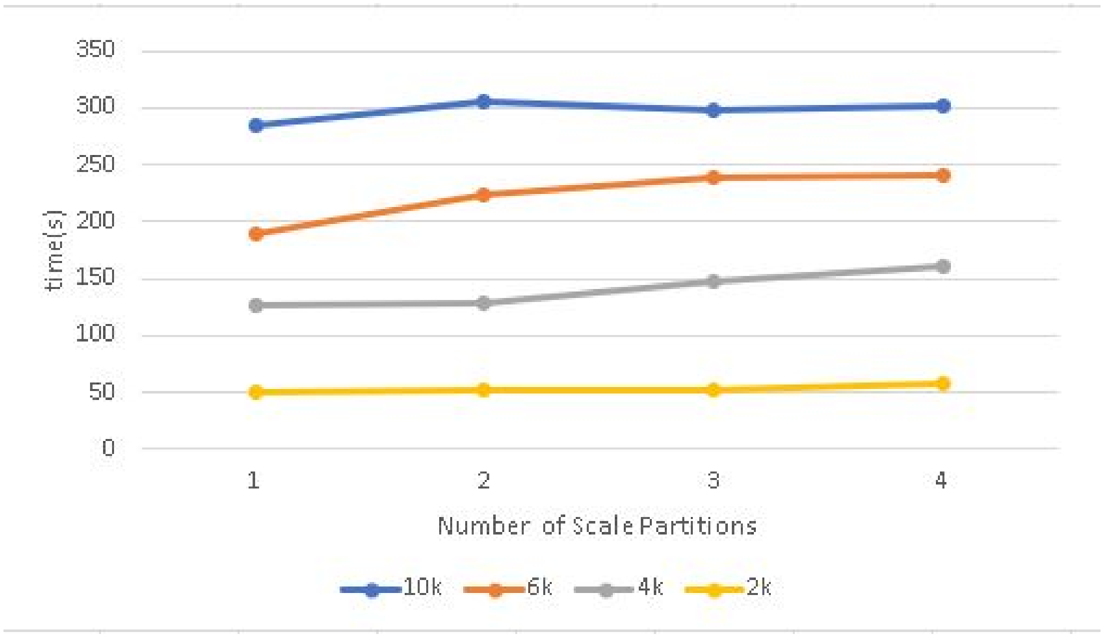


Figure 5.9: The effect of number of scale partitions on computational cost for the KTH dataset.

5.6 Conclusions

The encoding of scale information into BoW representations (SC-BoW) was formalized; simplifying and generalizing the approach such that it can be easily adapted into any existing video-based action recognition algorithms that use the BoW for feature representation. Thus, SC-BoW maintains the simplicity and flexibility of the standard BoW framework. The formalized SC-BoW approach was applied to the dense trajectory feature set for experimental evaluation.

It was found that the performance of scale coding is independent of the number of points sampled. However, performance is influenced by the number of scale partitions defined. In most cases, 3 scale partitions yield the optimum performance.

Experimental results showed that SC-BoW representations improve performance of the dense trajectory features with a negligible added computational cost; presenting an ideal cost-to-accuracy trade-off. This re-affirms that scale is salient information to extract for video-based action recognition. Furthermore, scale coded dense trajectories produced accuracies competitive with state-of-the-art approaches. Notably, SC-BoW

Table 5.4: Comparison to Existing BoW Action Recognition Methods on the KTH dataset

Method	**Accuracy (%)**
Sequential Deep Learning [32]	94.38
Spatio-temporal CNN [94]	95.86 ± 0.3
Local Part Model [68]	93.0*
BoE [54]	99.51
DT + SC-BoW	**96.76**

* Algorithms that operate in real-time

representations outperformed more complex deep learning approaches on the KTH dataset.

It can be concluded that scale coding improves the performance of BoW representations whilst maintaining the efficiency and flexibility of the standard framework. Thus, SC-BoW representations are a promising avenue for real-world action recognition problems.

Chapter 6

Conclusions and Future Work

In this research, the potential of scale coding a Bag-of-Words (BoW) for real-time video-based action recognition was explored. To develop practical real-time action recognition algorithms, the trade-off between computational cost and accuracy must be carefully considered. In the ideal scenario, high performance is achieved with minimal computational cost. To study this balance, existing BoW variants were theoretically analyzed. These variants work to increase the discriminative power of the framework, often compromising the efficiency, simplicity and flexibility of the original framework as a result. Four factors affecting the cost-vs-accuracy balance were identified: 'Sampling' strategy, 'Saliency' of features, 'Optical Flow' and overall algorithm 'Flexibility'. To determine the practical effects of the identified factors, experiments were conducted on the KTH and a reduced HMDB51 datasets. It was found that the 'Saliency' of extracted information is the most notable factor affecting the cost-vs-accuracy balance for video classification algorithms; extracting only the important information from video samples can yield the best performance at minimal computational cost. Based on the success of scale encoded BoW representations for image-based action recognition, scale was noted to be salient information to extract.

To make use of the salient extracted scale information, scale coded BoW representations were extended to form Spatio-temporal scale coded BoW (SC-BoW) representations for video classification tasks. A general pipeline that uses both relative and absolute encoded SC-BoW representations was proposed and consists of the following steps: i) Object Detection, ii) Tracking, iii) Feature Extraction, iv) Feature Partitioning, v) SC-BoW formation and vi) Classification. This pipeline was designed such that it operates in real-time. Experiments showed that SC-BoW improves performance by 4-7% on the given feature set (PCA-HOG) with low computational cost. Thus, SC-BoW has a favorable cost-to-accuracy balance; highlighting SC-BoW representations as a promising avenue for practical real-time action recognition of videos.

To further explore the potential of SC-BoW representations, the scale coding scheme was applied to the popular and high-performing dense trajectory feature set. It was found that SC-BoW representations improve performance with negligible added computational cost; presenting an ideal cost-to-accuracy trade-off. Notably, scale coded dense trajectories performed better than algorithms using the complex deep learning approaches on the KTH dataset. This showed that deep learning approaches, although powerful, are not always appropriate. Classical machine learning approaches are more beneficial in tasks requiring efficient classification with limited training data. Furthermore, SC-BoW maintains the efficiency, compact feature representation and flexibility of the standard BoW framework. Thus, scale coding a BoW has great potential to perform accurate real-time action recognition of video data in real-world circumstances where computational resources and training data are limited.

6.1 Scale Coding Best Practice

This section summarizes the recommended practice for forming scale coded BoW representations for classification of video data.

- The relative scale encoding scheme encodes more information into BoW representations, thus, results in better performance. However, the relative scheme is reliant on visual tracker performance. Failed attempts of the visual tracker results in ill-defined bounding boxes. This causes unimportant information to be captured; lowering performance and wasting resources. The absolute scale encoding scheme is more robust and flexible; making it a better option for real-world action recognition.
- Dynamic scaling parameter definition outperforms its static counterpart for both the absolute and relative scale encoding schemes. Dynamically setting the cutoff thresholds that define each scale partition was found to better represent extracted scale information.

- An optimum number of scale partitions exists; performance decreases if the number of scale partitions is set above or below this value. It was found that, in most cases, partitioning the set of spatial scales into 3 sub-groups results in the best performance.

6.2 Future Work

SC-BoW representation is a relatively low-level encoding scheme. Thus, it can be easily incorporated into any existing algorithm that uses BoW representation for a low-cost performance boost. Furthermore, it maintains the simplicity and flexibility of the original framework. Similarly to the large number of variants of the BoW that have been proposed in literature, there is room to do the same with SC-BoW.

Notable future work directions include:

- Improve the robustness of the relative scale coding scheme by defining a more reliable relative scaling scheme. For example, depth information can be used in place of bounding box size.

- Combine scale coding with deep learning algorithms. Possible approaches include: forming SC-BoW representations from features extracted by CNNs as opposed to hand-crafted features; extending the scale coded deep features for image-based action recognition approach [20] to the spatio-temporal domain for video-based action recognition; applying scale coding to the Convolutional Bag-of-Features (CBoF) [34]; incorporating extracted scale information into the current state-of-the-art two-stream CNNs algorithm.

References

[1] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu, “Dense trajectories and motion boundary descriptors for action recognition,” *International Journal of Computer Vision*, vol. 103, no. 1, p. 60–79, 2013.

[2] I. Laptev, M. Marszałek, C. Schmid, and B. Rozenfeld, “Learning realistic human actions from movies,” in *CVPR 2008-IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE Computer Society.

[3] N. Dalal, B. Triggs, and C. Schmid, “Human detection using oriented histograms of flow and appearance,” in *European conference on computer vision*, pp. 428–441, Springer.

[4] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, “Exploiting the circulant structure of tracking-by-detection with kernels,” in *European conference on computer vision*, pp. 702–715, Springer.

[5] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, “High-speed tracking with kernelized correlation filters,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 3, pp. 583–596, 2014.

[6] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, “Visual object tracking using adaptive correlation filters,” in *2010 IEEE computer society conference on computer vision and pattern recognition*, pp. 2544–2550, IEEE.

[7] I. Laptev, “On space-time interest points,” *International Journal of Computer Vision*, vol. 64, no. 2-3, pp. 107–123, 2005.

[8] F. S. Khan, J. v. d. Weijer, A. D. Bagdanov, and M. Felsber, “Scale coding bag-of-words for action recognition,” in *22nd International Conference on Pattern Recognition.*

[9] M. Danelljan, G. Häger, F. S. Khan, and M. Felsberg, “Discriminative scale space tracking,” *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, vol. 39, no. 8, pp. 1561–1575, 2017.

[10] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, “Hmdb: A large video database for human motion recognition,” in *IEEE International Conference on Computer Vision* (IEEE, ed.), p. 2556–2563.

[11] C. Schüldt, I. Laptev, and B. Caputo, “Recognizing human actions: a local svm approach,” in *International Conference on Pattern Recognition.*

[12] X. Peng, L. Wang, X. Wang, and Y. Qiao, “Bag of visual words and fusion methods for action recognition: Comprehensive study and good practice,” *Computer Vision and Image Understanding*, vol. 150, pp. 109 – 125, 2016.

[13] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu, “Action recognition by dense trajectories,” in *CVPR 2011 - IEEE Conference on Computer Vision.*

[14] H. Wang and C. Schmid, “Action recognition with improved trajectories,” in *Proceedings of the IEEE international conference on computer vision*, pp. 3551–3558.

[15] J. Hu, G.-S. Xia, F. Hu, and L. Zhang, “Dense vs sparse: A comparative study of sampling analysis in scene classification of high-resolution remote sensing imagery,” *arXiv preprint arXiv:.01097*, 2015.

[16] E. Nowak, F. Jurie, and B. Triggs, “Sampling strategies for bag-of-features image classification,” in *European conference on computer vision*, pp. 490–503, Springer.

[17] L. Li and S. Dai, “Action recognition with deep network features and dimension reduction,” *KSII Transactions on Internet Information Systems*, vol. 13, no. 2, 2019.

[18] T. Li, T. Mei, I.-S. Kweon, and X.-S. Hua, “Contextual bag-of-words for visual categorization,” *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, vol. 21, no. 4, pp. 381 – 392, 2011.

[19] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trainined quantization and huffman coding,” in *International Conference on Learning Representations.*

[20] F. S. Khan, J. v. d. Weijer, R. M. Anwer, A. D. Bagdanov, M. Felsberg, and J. Laaksonen, "Scale coding bag of deep features for human attribute and action recognition," *Machine Vision and Applications*, vol. 29, p. 55 – 71, 2018.

[21] A. Piergiovanni, A. Angelova, A. Toshev, and M. S. Ryoo, "Evolving space-time neural architectures for videos," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1793–1802.

[22] A. Richard and J. Gall, "A bag-of-words equivalent recurrent neural network for action recognition," *Computer Vision and Image Understanding*, vol. 156, pp. 79–91, 2017.

[23] S. Loussaief and A. Abdelkrim, "Deep learning vs. bag of features in machine learning for image classification," in *International Conference on Advanced Systems and Electric Technologies*, pp. 6 – 10.

[24] L. W. Campbell and A. F. Bobick, "Recognition of human body motion using phase space constraints," in *Proceedings of IEEE International Conference on Computer Vision*, pp. 624–630, IEEE, 1995.

[25] Y. Yacoob and M. J. Black, "Parameterized modeling and recognition of activities," *Computer Vision and Image Understanding*, vol. 73, no. 2, pp. 232–247, 1999.

[26] S. A. Niyogi, E. H. Adelson, *et al.*, "Analyzing and recognizing walking figures in xyt," in *CVPR*, vol. 94, pp. 469–474, 1994.

[27] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Advances in neural information processing systems*, pp. 568–576.

[28] S. Sehgal, "Human activity recognition using bpnn classifier on hog features," in *Proceedings of the 2018 International Conference on Intelligent Circuits and Systems, Phagwara, India, 20–21 April 2018*, pp. 286–289, IEEE, 2018.

[29] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732.

[30] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Convolutional two-stream network fusion for video action recognition," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016*, pp. 1933–1941.

[31] M. F. Aslan, A. Durdu, and K. Sabanci, "Human action recognition with bag of visual words using different machine learning methods and hyperparameter optimization," *Neural Computing and Applications*, vol. 32, no. 12, pp. 8585–8597, 2020.

[32] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt, "Sequential deep learning for human action recognition," in *Proceedings of the 2nd International Workshop on Human Behavior Understanding, Amsterdam, Netherlands, 16 November 2011* (A. A. Salah and B. Lepri, eds.), pp. 29–39, Springer, 2011.

[33] D. Filliat, "A visual bag of words method for interactive qualitative localization and mapping," in *2007 IEEE International Conference on Robotics and Automation*, pp. 3921 – 3926.

[34] N. Passalis and A. Tefas, "Learning bag-of-features pooling for deep convolutional neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5755 – 5763.

[35] F. Yang, H. Lu, W. Zhang, and G. Yang, "Visual tracking via bag of features," *IET Image Processing*, vol. 6, no. 2, pp. 115 – 128, 2012.

[36] F. Zeng, Y. Ji, and M. D. Levine, "Contextual bag-of-words for robust visual tracking," *IEEE TRANSACTIONS ON IMAGE PROCESSING*, vol. 27, no. 3, pp. 1433 – 1447, 2018.

[37] J. Sanchez, F. Perronnin, T. Mensink, and J. Verbeek, "Image classification with the fisher vector: Theory and practice," *International Journal of Computer Vision*, vol. 105, no. 3, pp. 222–245, 2013.

[38] S. O'Hara and B. A. Draper, "Introduction to the bag of features paradigm for image classification and retrieval," *Arxiv preprint arXiv:1101.3354*, 2011.

[39] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[40] C. G. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey vision conference*, vol. 15, pp. 10–5244, Citeseer.

[41] C. M. Bishop, *Pattern Recognition and Machine Learning.* Information Science and Statistics, Secaucus, NJ: Springer Science+ Business Media, 2006.

[42] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[43] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[44] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.

[45] J. Uijlings, I. C. Duta, E. Sangineto, and N. Sebe, "Video classification with densely extracted hog/hof/mbh features: an evaluation of the accuracy/computational efficiency trade-off," *International Journal of Multimedia Information Retrieval*, vol. 4, no. 1, pp. 33–44, 2015.

[46] Y. Huang, Z. Wu, L. Wang, and T. Tan, "Feature coding in image classification: A comprehensive study," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, vol. 36, no. 3, pp. 493 – 506, 2014.

[47] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3304–3311, IEEE, 2010.

[48] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *International Conference on Machine Learning.*

[49] X. Wang, L. Wang, and Y. Qiao, "A comparative study of encoding, pooling and normalization methods for action recognition," in *Asian Conference on Computer Vision*, pp. 572–585, Springer.

[50] J. Yang, K. Yu, Y. Gong, and T. S. Huang, "Linear spatial pyramid matching using sparse coding for image classification," in *CVPR*, vol. 1, p. 6.

[51] F. Perronnin, J. Sánchez, and T. Mensink, "Improving the fisher kernel for large-scale image classification," in *European conference on computer vision*, pp. 143–156, Springer.

[52] R. Arandjelović and A. Zisserman, "Three things everyone should know to improve object retrieval," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2911–2918, IEEE.

[53] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, pp. 2169–2178, IEEE.

[54] S. Nazir, M. H. Yousaf, J.-C. Nebel, and S. A. Velastin, "A bag of expression framework for improved human action recognition," *Pattern Recognition Letters*, vol. 103, pp. 39–45, 2018.

[55] P. Dai, W. Liu, L. Wang, C. Li, and Y. Xie, "Bag of features with dense sampling for visual tracking," *Journal of Computational Information Systems*, vol. 9, no. 20, pp. 8307–8315, 2013.

[56] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," in *International Conference on Learning Representation.*

[57] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4820–4828.

[58] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *Acm computing surveys (CSUR)*, vol. 38, no. 4, pp. 13–es, 2006.

[59] S. Chen, "Kalman filter for robot vision: a survey," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 11, pp. 4409–4420, 2011.

[60] H. A. Patel and D. G. Thakore, "Moving object tracking using kalman filter," *International Journal of Computer Science and Mobile Computing*, vol. 2, no. 4, pp. 326–332, 2013.

[61] F. Yang, H. Lu, and M.-H. Yang, "Robust superpixel tracking," *IEEE Transactions on Image Processing*, vol. 23, no. 4, pp. 1639–1651, 2014.

[62] T. Zhou, M. Zhu, D. Zeng, and H. Yang, "Scale adaptive kernelized correlation filter tracker with feature fusion," *Mathematical Problems in Engineering*, vol. 2017, 2017.

[63] K. Briechle and U. D. Hanebeck, "Template matching using fast normalized cross correlation," in *Optical Pattern Recognition XII*, vol. 4387, pp. 95–102, International Society for Optics and Photonics.

[64] D. Comaniciu, V. Ramesh, and P. Meer, "Real-time tracking of non-rigid objects using mean shift," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, vol. 2, pp. 142–149, IEEE.

[65] M. Danelljan, G. Häger, F. Khan, and M. Felsberg, "Accurate scale estimation for robust visual tracking," in *British Machine Vision Conference, Nottingham, September 1-5, 2014*, BMVA Press.

[66] F. Weiyan, Z. Ting, L. Cong, and Z. Xiang, "A kcf-based tracker with faster scale estimation and effective occlusion detection," in *2019 Chinese Control And Decision Conference (CCDC)*, pp. 762–766, IEEE.

[67] B. Schölkopf, A. J. Smola, and F. Bach, *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press, 2002.

[68] F. Shi, E. Petriu, and R. Laganiere, "Sampling strategies for real-time action recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2595–2602.

[69] F. Shi, E. M. Petriu, and A. Cordeiro, "Human action recognition from local part model," in *2011 IEEE International Workshop on Haptic Audio Visual Environments and Games*, pp. 35–38, IEEE.

[70] L. Yeffet and L. Wolf, "Local trinary patterns for human action recognition," in *2009 IEEE 12th international conference on computer vision*, pp. 492–497, IEEE.

[71] B. Zhang, L. Wang, Z. Wang, Y. Qiao, and H. Wang, "Real-time action recognition with enhanced motion vector cnns," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2718–2726.

[72] R. Sicre and T. Gevers, "Dense sampling of features for image retrieval," in *IEEE International Conference on Image Processing (ICIP)*, IEEE.

[73] I. Reznicek and P. Zemcik, "Human action recognition for real-time applications," in *Proceedings of the 3rd International Conference on Pattern Recognition Applications and Methods*, pp. 646–653, SCITEPRESS-Science and Technology Publications, Lda.

[74] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Scandinavian conference on Image analysis*, pp. 363–370, Springer.

[75] Q. Li, H. Cheng, Y. Zhou, and G. Huo, "Human action recognition using improved salient dense trajectories," *Computational Intelligence and Neuroscience*, vol. 2016, 2016.

[76] Z. Xu, R. Hu, J. Chen, C. Chen, H. Chen, H. Li, and Q. Sun, "Action recognition by saliency-based dense sampling," *Neurocomputing*, vol. 236, pp. 82–92, 2017.

[77] J. Shi and C. Tomasi, "Good features to track," in *IEEE Conference on Computer Vision and Pattern Recognition.*

[78] J. Zhang, M. Marszałek, S. Lazebnik, and C. Schmid, "Local features and kernels for classification of texture and object categories: A comprehensive study," *International journal of computer vision*, vol. 73, no. 2, pp. 213–238, 2007.

[79] K. Reddy and M. Shah, "Recognizing 50 human action categories of web videos," *Machine Vision and Applications*, p. 1–11, 2012.

[80] P. Scovanner, S. Ali, and M. Shah, "A 3-dimensional sift descriptor and its application to action recognition," in *ACM Conference on Multimedia.*

[81] G. Willems, T. Tuytelaars, and L. Van Gool, "An efficient dense and scale-invariant spatio-temporal interest point detector," in *European conference on computer vision*, pp. 650–663, Springer.

[82] A. Klaser, M. Marszałek, and C. Schmid, "A spatio-temporal descriptor based on 3d-gradients," in *BMVC 2008-19th British Machine Vision Conference*, pp. 275: 1–10, British Machine Vision Association.

[83] S. M. Kuo, B. H. Lee, and W. Tian, *Introduction to real-time digital signal processing*, pp. 1–43. John Wiley and Sons, Ltd, 2013.

[84] K. G. Shin and P. Ramanathan, "Real-time computing: A new discipline of computer science and engineering," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 6–24, 1994.

[85] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587.

[86] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

[87] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[88] Y. Wu, J. Lim, and M.-H. Yang, "Online object tracking: A benchmark," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2411–2418.

[89] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2009.

[90] J.-P. Vert, K. Tsuda, and B. Schölkopf, "A primer on kernel methods," *Kernel methods in computational biology*, vol. 47, pp. 35–70, 2004.

[91] D. Singh, A. Bhure, S. Mamtani, C. K. Mohan, and S. Kandi, "Fast-bow: Scaling bag-of-visual-words generation," 2018.

[92] T.-H. Yu, T.-K. Kim, and R. Cipolla, "Real-time action recognition by spatiotemporal semantic and structural forests," in *BMVC*, vol. 2, p. 6.

[93] B. Zhang, L. Wang, Z. Wang, Y. Qiao, and H. Wang, "Real-time action recognition with deeply transferred motion vector cnns," *IEEE Transactions on Image Processing*, vol. 27, no. 5, pp. 2326–2339, 2018.

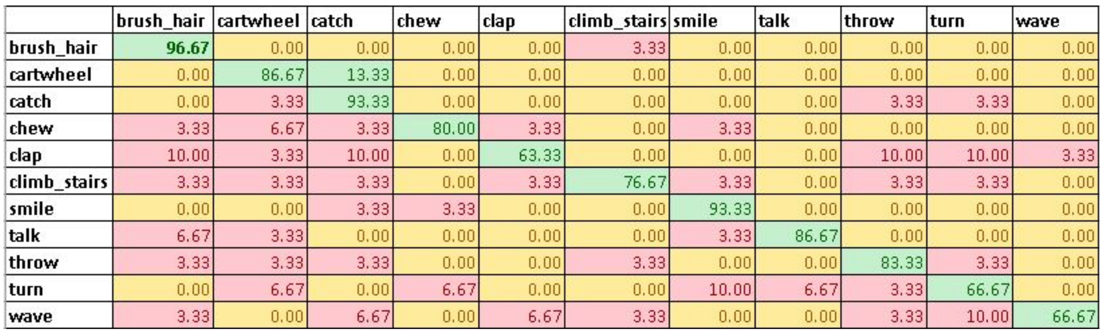
[94] S. Basha, V. Pulabaigari, and S. Mukherjee, "An information-rich sampling technique over spatio-temporal cnn for classification of human actions in videos," *arXiv preprint arXiv:2002.02100*, 2020.

Appendix A

Additional Results

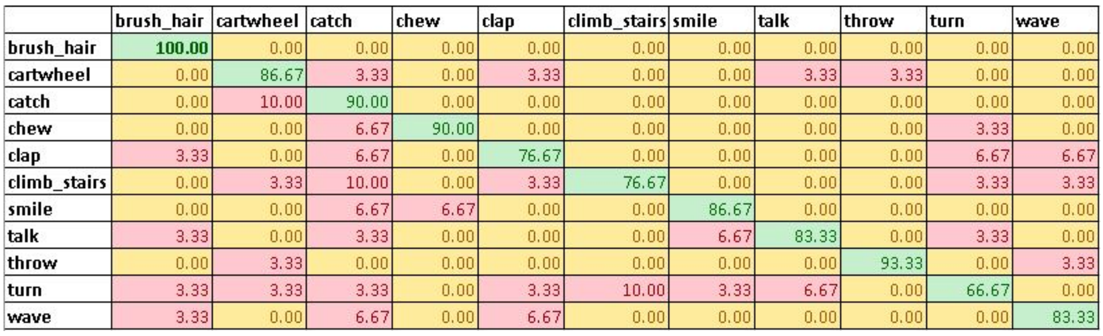
A.1 Reduced HMDB51 Dataset

This section covers other representations of experimental results obtained.



	brush_hair	cartwheel	catch	chew	clap	climb_stairs	smile	talk	throw	turn	wave
brush_hair	**96.67**	0.00	0.00	0.00	0.00	3.33	0.00	0.00	0.00	0.00	0.00
cartwheel	0.00	86.67	13.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
catch	0.00	3.33	93.33	0.00	0.00	0.00	0.00	0.00	3.33	3.33	0.00
chew	3.33	6.67	3.33	80.00	3.33	0.00	3.33	0.00	0.00	0.00	0.00
clap	10.00	3.33	10.00	0.00	63.33	0.00	0.00	0.00	10.00	10.00	3.33
climb_stairs	3.33	3.33	3.33	0.00	3.33	76.67	3.33	0.00	3.33	3.33	0.00
smile	0.00	0.00	3.33	3.33	0.00	0.00	93.33	0.00	0.00	0.00	0.00
talk	6.67	3.33	0.00	0.00	0.00	0.00	3.33	86.67	0.00	0.00	0.00
throw	3.33	3.33	3.33	0.00	0.00	3.33	0.00	0.00	83.33	3.33	0.00
turn	0.00	6.67	0.00	6.67	0.00	0.00	10.00	6.67	3.33	66.67	0.00
wave	3.33	0.00	6.67	0.00	6.67	3.33	0.00	0.00	3.33	10.00	66.67

Figure A.1: Confusion matrix plot of the reduced HMDB51 dataset with percentage accuracy. Dense trajectory features were extracted with a sub-sampling refresh rate of R = 6 frames.

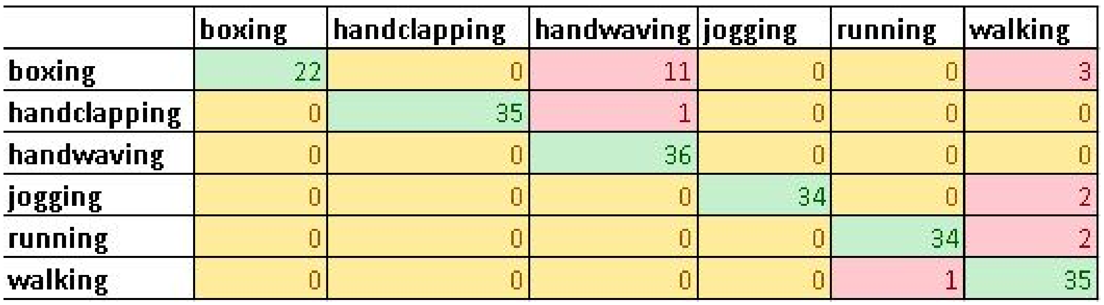


	brush_hair	cartwheel	catch	chew	clap	climb_stairs	smile	talk	throw	turn	wave
brush_hair	**100.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
cartwheel	0.00	86.67	3.33	0.00	3.33	0.00	0.00	3.33	3.33	0.00	0.00
catch	0.00	10.00	90.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
chew	0.00	0.00	6.67	90.00	0.00	0.00	0.00	0.00	0.00	3.33	0.00
clap	3.33	0.00	6.67	0.00	76.67	0.00	0.00	0.00	0.00	6.67	6.67
climb_stairs	0.00	3.33	10.00	0.00	3.33	76.67	0.00	0.00	0.00	3.33	3.33
smile	0.00	0.00	6.67	6.67	0.00	0.00	86.67	0.00	0.00	0.00	0.00
talk	3.33	0.00	3.33	0.00	0.00	0.00	6.67	83.33	0.00	3.33	0.00
throw	0.00	3.33	0.00	0.00	0.00	0.00	0.00	0.00	93.33	0.00	3.33
turn	3.33	3.33	3.33	0.00	3.33	10.00	3.33	6.67	0.00	66.67	0.00
wave	3.33	0.00	6.67	0.00	6.67	0.00	0.00	0.00	0.00	0.00	83.33

Figure A.2: Confusion matrix plot of the reduced HMDB51 dataset with percentage accuracy. Scale coded dense trajectory features were extracted with a sub-sampling refresh rate of R = 6 frames.

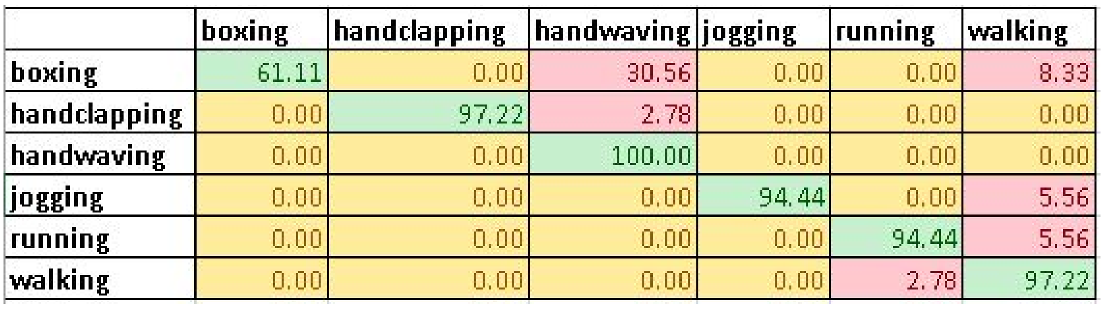
A.2 KTH Dataset

This section covers all the raw results obtained on the reduced KTH dataset.



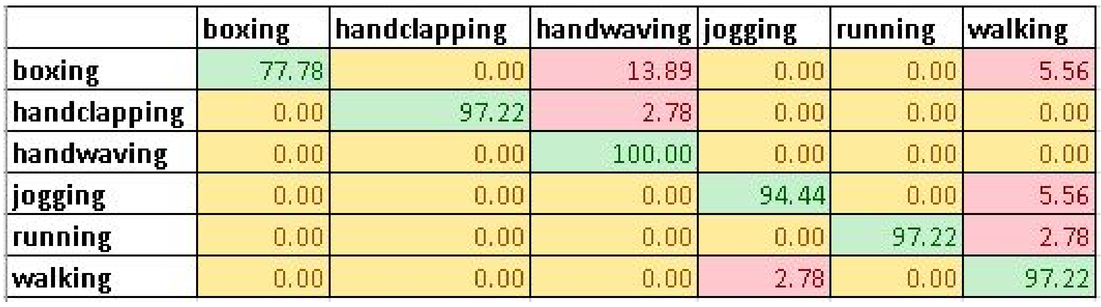
	boxing	handclapping	handwaving	jogging	running	walking
boxing	22	0	11	0	0	3
handclapping	0	35	1	0	0	0
handwaving	0	0	36	0	0	0
jogging	0	0	0	34	0	2
running	0	0	0	0	34	2
walking	0	0	0	0	1	35

Figure A.3: Confusion matrix plot of the KTH dataset. Dense trajectory features were extracted with a sub-sampling refresh rate of R = 6 frames.



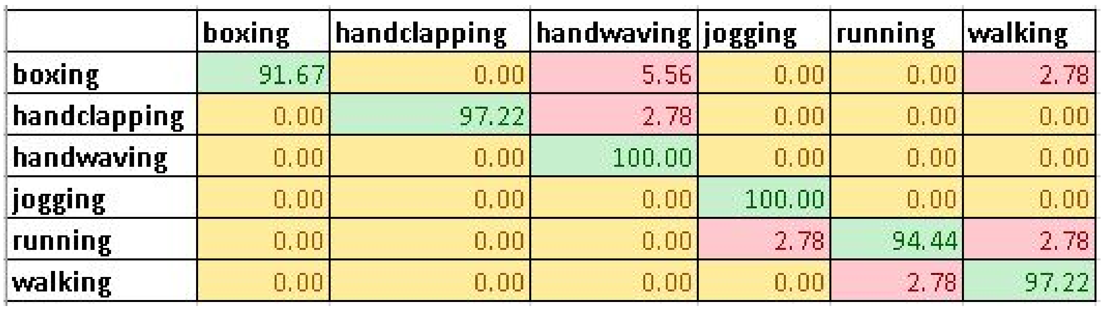
	boxing	handclapping	handwaving	jogging	running	walking
boxing	61.11	0.00	30.56	0.00	0.00	8.33
handclapping	0.00	97.22	2.78	0.00	0.00	0.00
handwaving	0.00	0.00	100.00	0.00	0.00	0.00
jogging	0.00	0.00	0.00	94.44	0.00	5.56
running	0.00	0.00	0.00	0.00	94.44	5.56
walking	0.00	0.00	0.00	0.00	2.78	97.22

Figure A.4: Confusion matrix plot of the KTH dataset with percentage accuracy. Dense trajectory features were extracted with a sub-sampling refresh rate of R = 6 frames.



	boxing	handclapping	handwaving	jogging	running	walking
boxing	77.78	0.00	13.89	0.00	0.00	5.56
handclapping	0.00	97.22	2.78	0.00	0.00	0.00
handwaving	0.00	0.00	100.00	0.00	0.00	0.00
jogging	0.00	0.00	0.00	94.44	0.00	5.56
running	0.00	0.00	0.00	0.00	97.22	2.78
walking	0.00	0.00	0.00	2.78	0.00	97.22

Figure A.5: Confusion matrix plot of the KTH dataset with percentage accuracy. Dense trajectory features were extracted with a sub-sampling refresh rate of R = 1 frames.



	boxing	handclapping	handwaving	jogging	running	walking
boxing	91.67	0.00	5.56	0.00	0.00	2.78
handclapping	0.00	97.22	2.78	0.00	0.00	0.00
handwaving	0.00	0.00	100.00	0.00	0.00	0.00
jogging	0.00	0.00	0.00	100.00	0.00	0.00
running	0.00	0.00	0.00	2.78	94.44	2.78
walking	0.00	0.00	0.00	0.00	2.78	97.22

Figure A.6: Confusion matrix plot of the KTH dataset with percentage accuracy. Scale coded dense trajectory features were extracted with a sub-sampling refresh rate of R = 1 frames.