Department of Computer Science

Submitted in part fulfilment for the degree of BEng

Classification of Human Movement from Videos

Nicolas Pinedo

26th April 2019

Supervisor: Dr. Adrian Bors

Table of Contents

Table of Figures

Table of Tables

Table of Equations

Executive Summary

The aim of this work is to produce an algorithm that can classify types of human movement from videos. Past work in this area is reviewed and analysed. The types of videos that are worked with consist of boxing, clapping, waving, jogging, running and walking. Producing this algorithm allows a direct analysis of how different parameters and sub-methods can affect the overall classification accuracy. The implemented method uses histograms of oriented gradients that are calculated from optical flow images, and incorporates different image processing techniques such as morphing, thresholding, cropping and blurring. Feature vectors are constructed and passed to a multi-class support-vector machine for classification. The results give insight into what techniques were useful, how accuracy changes with the parameter values, and where such an algorithm can improve. The end-result gave the algorithm a true positive rate of 95% - a good result.

There were no legal, social, ethical, professional or commercial issues concerning this work. All references and datasets used are publically available for academic purposes such as this.

1 Introduction

Computer Vision has seen significant growth over the last few decades, enabling the automatic classification and recognition of things in the real world. These *things* can be people, objects, actions or just about anything that can be captured by a camera. The field has developed using a variety of techniques and methods, producing different algorithms for computer vision. This paper attempts to describe and implement such an algorithm in the context of human movement classification, and determine what can improve or impair its classification accuracy.

There are a variety of applications for classifying human movement from videos, many of which are at the cutting edge of computer vision research. Autonomous cars require careful attention to pedestrians and their movement; classifying a pedestrian's movement can help a system understand their direction and speed which is useful for the calculation of potential collisions, ultimately increasing its reactionability. Surveillance systems could benefit from such classifications, for example; detection of pedestrians waiting by traffic light crossings, detection of people commiting crimes, or gait detection. A more intricate gait classification system can also be useful for biometric identification [1] or in healthcare for recognition of gait-related health problems, such as Parkinson's disease and Hemiplegia [2]. Movement classification can also be useful for human-computer interaction, allowing computer systems to be controlled by real life actions through a camera - such systems have no need for physical controls, increasing their compactness and accessibility for people that are unable to use standard physical controls.

The following report is structured into four main sections; a review of past work, a description of the method, results and evaluation of the method, and a conclusion and further work section.

2 Literature Review

2.1 Datasets

Human movement sequences suited to this research need only be simplistic; the aim is to discuss the principle methods of algorithms not the scope of activities or the complexity of the scene in which the classification succeeds. Newer datasets tend to include a wider and more complex range of activities, such as the Kinetic-600 dataset [3] which sources its video sequences directly from YouTube. This dataset has approximately 500,000 entries, including activities like flipping pancakes and making tea. Some scenes are occluded and have non-uniform backgrounds. The KTH dataset [4] from 2004 contains only six types of human activity; boxing, hand clapping, handwaving, jogging, running, walking. The actions are each performed by 25 different people in 4 different scenarios, resulting in 100 samples for each of the six activities. The scenes are purposely choreographed to show their respective actions clearly. The four different scenarios are organised to have the actions performed at varying angles to the camera's perspective but with neutral, homogeneous backgrounds. Examples are given in Figure 1. For these reasons, KTH is a suitable dataset and will be used for this research.

The Weizmann dataset [5] is of a similar format to the KTH dataset; a small set of classes captured with uniform backgrounds - these classes include walking, running, jumping, galloping, skipping, bending. Unfortunately, this dataset contains only 10 videos for each class, and therefore will not be enough to sufficiently train *and* test on; however, it could be useful for further testing for the running and walking classes, once a classifier has been trained on the KTH dataset.

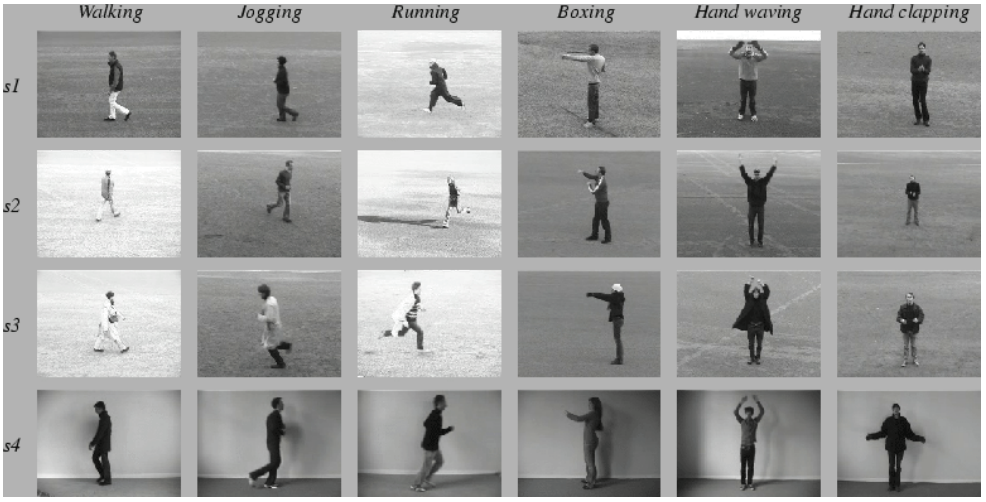

Figure 1: KTH dataset examples of each movement class (walking, jogging, running, boxing, handwaving, hand clapping) and scenario (denoted as s1, s2, s3 and s4) [4].

2.2 Existing Methods

2.2.1 Histogram of Oriented Gradients

There exists a number of methods involving histograms of features, the Histograms of Oriented Gradients (HOG) method is an effective one which is easy to implement. HOG can be used for simple character classification [6] or for more complex problems such as human detection [7]. HOGs have fast performance in comparison with other methods using orientation histograms, such as Scale Invariant Feature Transformation (SIFT) [6,7]. This method is generally used for single images but could be modified to account for image sequences.

A window of pixels is defined at a constant size, the HOG cell, whereby the image is divided into a grid of these cells. Within each cell, gradient vectors are calculated for each pixel and organised into a histogram [8]. The gradients can represent the gradient directions *or* the edge directions [6]. The histogram format allows the numerous gradient vector values to be quantised into a smaller number of categorical values (bins), creating a more general descriptor rather than raw gradient vectors from each image pixel [8]. This compression is also computationally beneficial because there is less data to consider. The histograms can be normalised so that they are invariant to illumination variation [6] - an advantage of HOG. Histograms are normalised in blocks, where a block is a standardised number of cells [8]. For example, a block size of 2x2

contains 4 cells. The normalisation is performed on a combined histogram made up from the individual histograms for each cell in the block [8]. HOG features are outputted as vectors, which makes a Support-Vector Machine (SVM) a common choice of classifier [6, 7].

2.2.2 Mid-Level Feature Analysis

The work of Fathi and Mori [9] presents a method for human action recognition using 'mid-level' motion features, differing from its predecessors which use either small-scale or large-scale features [9]. This method starts by extracting a *figure-centric* representation from the image sequence, cropping frames around the figure of interest using a detection/tracking algorithm; this produces spatio-temporal volumes to be processed. Low-level motion features are extracted and used as weak classifiers, which are then used to construct the mid-level motion features. AdaBoost [9] is used to construct stronger classifiers by selecting a subset of the weak classifiers obtained from the figure-centric volume, combining them and applying weights [9]. Each mid-level feature represents a particular volumetric neighbourhood within the entirety of the figure-centric volume. A final classifier is constructed using a similar method with a second run of AdaBoost, 'merging' the mid-level features together. Given that AdaBoost is a binary classifier, the results must be converted to represent a set of multiple discrete classes; this is done using the 'Haming decoding' method. The recognition results showed that the method was equal with its contemporaries for certain datasets and outperformed them with others.

2.2.3 Spatio-Temporal Relationship Match

Test videos can be classified by a direct comparison to corresponding training data. This is possible using a kernel function to detect similarities, that is, the spatio-temporal relationship match as implemented by Ryoo and Aggarwal [10]. In their design the kernel function takes two feature vectors (from two seperate videos) and produces a value, denoting likelihood. The function measures similarity using *relationship* histograms, where each bin should contain pairs of feature points. For each video one spatial relationship histogram and one temporal relationship histogram is constructed. The intersection of two videos' spatial histograms and two temporal histograms produces pairs of feature points that are present in both videos, thus showing the number of shared features.

2.2.4 Static and Motion Feature Analysis

The framework designed by Liu et al. [11] contains multiple distinct steps for movement classification, analysing features from both static frames and motion sequences:

Firstly, feature extraction - for both static and motion features. The static feature detection is carried out for each temporally sampled frame using three different interest point detectors [11]. Motion features are detected by a spatiotemporal interest point detector [12] which uses a filter in the spatial dimensions and another in the temporal dimension. This detector has 3D outputs which can then be reduced using PCA [11]. Detected motion features can also include those in the background, irrelevant to the desired classification; this may be due to other movement in the scene or camera shaking [11]. Pruning helps reduce this noise - this is done using two methods: one that reduces the number of frames based on the mean and variance of features; another that discards features based on the location of other features [11], 'centering' them upon the desired part of the scene to classify. Next, static feature pruning is performed. Motion information is used to estimate the region of interest (ROI) to determine which static features could be relevant. PageRank [13] is used to analyse the interactions between features, determining which features are most significant. The framework concludes by reconstructing the visual vocabulary; grouping features in order to reduce the size of the vocabulary to improve performance and avoid over-fitting. Results found that the hybrid of motion and static features outperformed their separate counterparts.

2.2.5 Three-Dimensional Convolutional Neural Network

Convolutional Neural Networks (CNNs) have been a big success in image classification for quite some time, as demonstrated by the work of Lawrence et al. [14] on face recognition in 1997. CNNs are usually made up of a combination of three main layers; Convolution, Pooling, and Classification. The convolution layers use kernels to extract features. Kernels slide across each pixel in the image. The values of the kernels are *learnt* by the CNN during training and can perform operations such as edge detection, blurring or sharpening, for example [15]. Pooling layers have the purpose of dimensionality reduction [15]: The image is sectioned into equal-sized pixel neighbourhoods; a max, sum or average operation is applied to all neighbourhoods [15]. The classification layers, at the end of the network, function as a traditional multi-layer perceptron neural

network - classifying the convolved and pooled features [15]. CNNs usually consist of multiple convolution and pooling layers.

In recent years this method has also seen significant development in working with three-dimensional scenes (coined '3D CNNs') [16]. In the context of a video, the same techniques of a 3D CNN can be used, where the third dimension represents time [17]. 2D frames of a video can be put together to form a 3D spatio-temporal representation that can then be analysed by a CNN. The frames can be sampled from certain points and intervals across the temporal dimension, then connected into the CNN appropriately. Karpathy et al. [17] categorised these connectivity patterns as follows: Early Fusion - multiple frames are immediately combined across a continuous time window. Late Fusion - using two non-consecutive frames, the frames are only combined at the last layer of the network. Slow Fusion - multiple sets of continuous frames are combined iteratively throughout the network. It was found that the slow fusion method consistently outperformed early and late fusion [17].

2.2.6 Recurrent Neural Network

Standard neural networks retain no 'memory', that is to say, they do not use previous classifications to inform new classifications. In the context of a video, they would not use information from a previous frame to inform the classification of the next frame. Recurrent neural networks (RNNs) do retain memory, but generally on a short-term basis [18]. RNNs do this by looping - repeating with prior knowledge. The lag between the initial computation of the prior knowledge and the present iteration heavily affects the network's ability to relate them; this is generally due to an increase in the inefficiency of gradient descent because of error backflow which explodes or vanishes [18]. RNNs therefore are not overall a suitable method for video classification, as videos can contain several frames that contribute to the classification of a scene. There are some situations where an RNN could be an appropriate method, such as classification of very simple (and short) motions from low-frame-rate videos, but this is a very specific example.

Contrary to the standard RNN, there is a special type of RNN that is designed to overcome the problems of error backflow. This novel RNN is called a Long Short-Term Memory Network (LSTM). There are minor variations on the design of this architecture, but all carry the same concepts [19]. In each single network iteration, there are a number of layers: a *forget gate layer*, removes irrelevant information

or scales it down; an *input gate layer*, updates with new relevant information chosen from a set of candidate values; and an *output layer*, normalises and outputs relevant information [19]. Because of the long-term memory capability of LSTMs they are suitable for, and have been used for, video classification [20].

Gated Recurrent Unit Networks (GRUs) are similar to LSTMs, in which they are a type of RNN and are designed to mitigate the error backflow problems of a traditional RNN [21]. Standard GRUs have only two layers; reset and update. The work of Chung et al. [21] showed that neither LSTMs or GRU conclusively outperformed the other.

2.3 Summary

This review of existing work provided an understanding of past video classification methods and how they have developed. There is a wide variety of approaches, all of which have their strengths and weaknesses - each of them useful for different problems. With time, the classes that can be distinguished by computer vision systems are becoming ever more complex. The degree of accuracy that some of these methods can attain is very high - most averaging above 90% accuracy for simpler datasets like KTH. Some state of the art systems can classify unchoreographed footage, with occlusions, multiple persons and other non-uniform properties - though these results tend to be less accurate. Understanding what specific techniques and parameters can improve a simpler method, can be important for the further development and improvement of more complex video classification systems.

3 Method

3.1 Overview

The method in this paper consists of three distinct parts; the optical flow is estimated from sample frames, whereby HOG features are extracted, which are then used to classify the videos into one of the six human movement classes (boxing, hand clapping, handwaving, jogging, running, walking), using a multi-class SVM. Such computations are performed on a number of sample frames for each video. The sample frames are each selected from the video using a random number generator, but made to be distinct from one another.

HOG is used as the base method because of its simplicity. Simple methods such as this do not require huge datasets to get good results, as CNNs and LSTMs do. The time constraints of this research are also suited to simpler methods, allowing more time to focus on optimisation. The HOG method is also flexible, allowing a range of preprocessing techniques to be tested and different sub-methods to be used in conjunction with it. HOG features can be extracted from static frames, from optical flow estimation images, from regions of interest. Different classifiers and classification techniques can also be used.

The aim of this method is to classify human movement from videos to a reasonable degree of accuracy; in addition, give insight as to how the method can be improved in pursuing this task. Previous work involving the KTH dataset achieved a true positive rate of over 90% [9] - this will be considered as a reasonable degree of accuracy. The complete runtime of the program (sampling, optical flow estimation, image processing, feature extraction, classifier training and testing) is desired to be under an hour.

3.2 Tools and Resources

MATLAB has been used to implement all stages of this method. MATLAB is a common programming language for image processing [22, 23]. It offers many *in-built* functions for matrix manipulation, which is very useful for image processing. Its Statistics and Machine Learning Toolbox (SMLTb) gives access to a variety of machine learning functions, which allows for many options at the classification stage of the method. MATLAB and the SMLTb are

readily installed on the departmental computers at university and accessible to me all day, all week, at no cost. The standardised Integrated Development Environment (IDE) also makes MATLAB a preferred option, containing all the workspace is one organised and interactable window. Overall MATLAB was a reasonable and well-thought-out choice.

Other programming tools were considered, such as Python. Python is an attractive option for its simplicity and ease of use, its versatility also allows complex fields like image recognition to be implemented effectively. However, python gets much of its versatility from external libraries which would require the installation of image processing and machine learning libraries on each separate machine that is used for such work. There is also no standardised IDE for Python.

3.3 Optical Flow

3.3.1 Method

In an image sequence, optical flow is a measurement of pixels' movement between images. The measurements are given as two-dimensional vectors, taking into account direction, orientation and magnitude of each pixel in a frame. The basis of optical flow comes from the following formula, where a voxel point (x,y,t) with intensity I(x,y,t) moves by (Δx, Δy, Δt) between two images:

$$I(x,\ y,\ t) = I(x + \Delta x,\ y + \Delta y,\ t + \Delta t) \quad (1.1)$$

Using a taylor series expansion and some substitutions, this formula can result in the following equation with partial derivatives [24], where Vx and Vy are the x and y components of the optical flow:

$$\frac{\delta I}{\delta x} V_x + \frac{\delta I}{\delta y} V_y + \frac{\delta I}{\delta t} = 0 \quad (1.2)$$

These formulae have been taken from existing papers [24, 25] but adapted to be easier to read.

The optical flow is calculated using the *Lucas-Kanade* method (LK). This optical flow method assumes that the displacement of pixels between frames is small-scale, whether contextually in time or space, and there are significant intensity contrasts which change smoothly over time [26]. The videos in the KTH dataset comply with these assumptions; the people in the video clearly contrast with the uniform backgrounds; and given that these contain human-speed

movements with a reasonable frame rate, the displacement of pixels is sufficiently smooth and small-scale between frames. LK measures the displacement of pixel neighbourhoods instead of individual single pixels, then solves the above equation with a least squares solution for each neighbourhood.

3.3.2 Post-processing

For each sample frame the direction, orientation and magnitude of the optical flow is estimated. These are combined together to form two matrices of motion magnitude (velocity) for both x (horizontal) and y (vertical) directions. Upon calculation, these include a signed direction - movement towards the origin is negative, movement away from the origin is positive. This distinction is removed by taking the absolute value of each element in the matrix. The distinction is not relevant to classifying these movement classes (as proven in testing).

A gaussian image filter is applied in order to filter out noise in each optical flow estimation image (OFI). This prevents random noise from being considered as movement in the frame, thus not affecting the consequential classification. The gaussian filtering is done with a sigma value of 1.

Each OFI is morphed with a close function - dilation followed by erosion, using the same structuring element. This attempts to ‘close’ holes in an image; in this context, it groups movement pixels together into shapes in order to create a more general model and reduce overfitting to noisy and miniscule groups of pixels in the OFIs.

The OFIs are checked to determine whether they contain a significant amount of motion - some sample frames can contain no motion whatsoever and are therefore not beneficial to movement classification. An example of this is the moment after a person runs off-screen. This is done by taking the mean of the OFI matrix and testing if it exceeds a certain threshold; if it does not, another frame is randomly selected and tried for the same conditions. A maximum threshold is also tested, to ensure frames with a saturation of movement are not taken further - an example of this is when the camera shakes and most pixels in the frame are regarded as moving, unrelated to the person. The threshold values are selected after manually creating a dataset of ‘mean motion values’ (MMVs) against a categorical variable for whether the frame contained any motion, or too much (appeared fuzzy and unusable) - ultimately decision boundaries for the MMV are selected and used as the

threshold values for sample frame selection. The decision boundaries are selected in such a way that the classification error is minimised, in regards to this MMV dataset.

The OFIs are cropped to an identical size about a weighted centre of motion. This creates a window around the person in the frame, attempting to ignore movement external of the person - referred to as ‘figure-centric’ by Fathi and Mori [9]. This was done using centre of mass functions on the image matrices. Cropping to a person-centric window is also beneficial in terms of building a more general model; the hand movements (boxing, clapping, waving) tend to be performed in the centre region of the camera’s field of vision for the KTH dataset, so when considering the entire frame, the classifier may (incorrectly) classify based on the global position of the movement. The cropped frames force the model to classify the human movement based on local and relevant motion. As the cropped frames are identical in size, this results in HOG vectors of equal lengths for the SVM to classify; the HOG vectors can then be predefined to certain dimensions and not need to change size each iteration - which is computationally more efficient.

For each sample frame, the same processing of optical flow is performed on the immediately consecutive frame. They act like two separate frames until the final stages of feature extraction. Effectively, this doubles the number of sample frames used, but they are to be eventually combined into one feature set.

The sample selection, optical flow and post-processing sequence is illustrated by a block diagram in Figure 2.1.

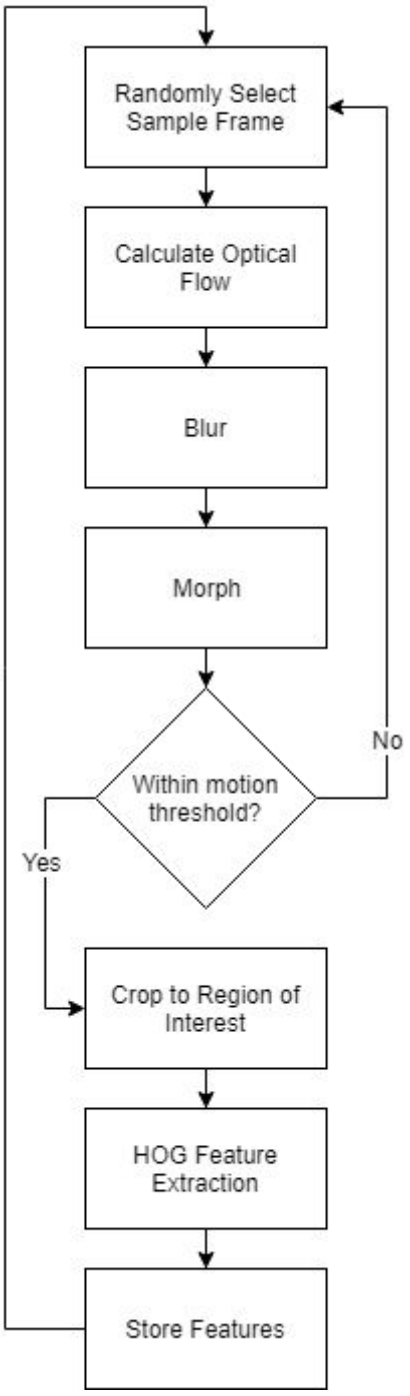

Figure 2.1: Block diagram of processing performed on video sample frames, until HOG feature extraction and storage.

3.4 Feature Extraction

3.4.1 Histogram of Oriented Gradients

For each sample frame's preprocessed OFI, HOG features are extracted. They are extracted with the following parameters:

Name	Value	Description, as taken from the MATLAB documentation [27].
CellSize	[10 10]	Size of HOG cell
BlockSize	[2 2]	Number of cells in HOG block
BlockOverlap	[1 1]	Number of overlapping cells between adjacent blocks
NumBins	9	Number of orientation histogram bins
UseSignedOrietnation	false	Selection of orientation values. When true, orientation values are evenly spaced between -180 and 180 degrees; when false, they are evenly spaced between 0 and 180 degrees.

Table 1: Values and descriptions of parameters that were used for HOG feature extraction.

Each parameter is to be tested with multiple values and finally set to a value which optimizes the algorithm accuracy, according to the comparison metrics discussed in section 4.1.

HOG features are extracted separately for both x and y directions of the optical flow, forming two separate HOG feature vectors. The two are then concatenated. This same technique is used on the immediately consecutive frame (including its separate x and y directions): the HOG feature vectors extracted from each of the two frames are then again concatenated to form the final feature vector of that particular sample cluster. The motive behind this is to enable the feature vector to describe the x and y direction optical flow between three consecutive frames (not just two), giving insight on how the *optical flow* changes over time in each direction.

3.4.2 Principal Component Analysis

The HOG features are outputted as vectors for each of the frame samples, which will result in a very large amount of data. This large amount of data results in a lengthy computation time for training a classifier. To reduce the amount of data but retain sufficient accuracy, Principal Component Analysis (PCA), a common dimensionality reduction method, is used. PCA works by using eigenvectors to find a linear basis of reduced dimensionality for the data, whereby the variance is maximised [28].

PCA is implemented to keep the components of the feature vectors that explains at least 95% variability. This is a common implementation, as shown by the MATLAB documentation for PCA [29]. PCA significantly reduces the time to compute the classifier, from several hours to under an hour (see section 4.1 for specific results).

3.5 Classification

The HOG features, outputted as 1 by 5184 vectors (1 by 2592 each for the x and y directions), for each sample results in a 28800 by 5184 feature matrix. 90% of these records are used as training data, the other 10% as test data. The classification is done using the fitcecoc MATLAB function, which gives multi-class classification for a variety of learners. An SVM learner is ultimately used to classify the feature vectors, with the key parameters listed in Table 2.

‘The ‘Coding’ parameter allows preset coding designs to be selected for the classifier. The ‘onevsone’ design is selected - this means a classifier is learnt for each pair of class labels. The ‘FitPosterior’ parameter is a flag for the transformation of scores to posterior probabilities. Posterior probabilities are not necessary, so the flag is set to false. The ‘Learners’ parameter gives the option to select different binary learners to train the classifier with. The svm binary learner is used. Cross validation is not used as to reduce the runtime of the training process.

Fundamentally, this SVM classes each separate frame of a video. The most frequent (mode) class predicted for a group of frames from one video is taken as the final class for that video. This greatly increases the accuracy of the model, by filtering out incorrect classifications of single frames.

The classification process is illustrated by a block diagram given in Figure 2.2.

Name	Value	Description, as taken from the MATLAB documentation [30].
Coding	‘onevsone’	Coding design. ‘onevsone’ means for each binary leaner, one class is positive, another is positive, and the software ignores the rest. This design exhausts all combinations of class pair assignments.
FitPosterior	false	Flag indicating whether to transform scores to posterior probabilities.
Learners	‘svm’	Binary learner templates.
ObservationsIn	‘rows’	Predictor data observation dimension.
CrossVal	‘off’	Flag to train cross-validated classifier.

Table 2: Values and descriptions of parameters used for classifier training function (fitcecoc).

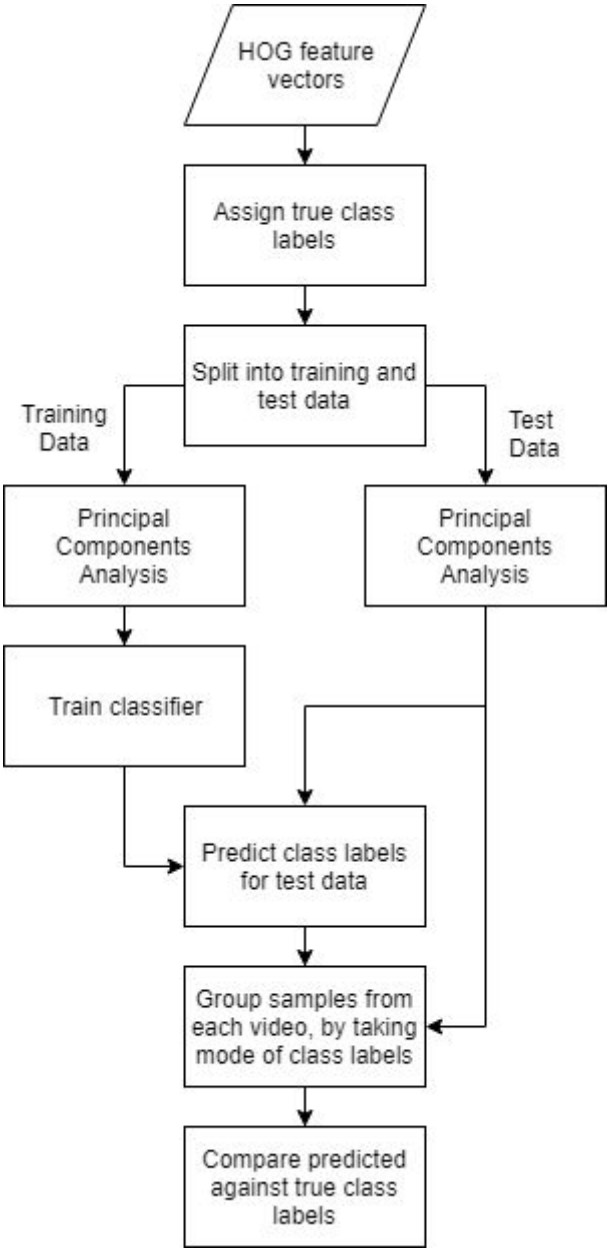

Figure 2.2: Block diagram of classification process, starting from HOG feature vectors.

4 Results

4.1 Comparisons

4.1.1 Metrics

The method underwent several iterations and versions to attempt to improve its accuracy. The main metric for comparing each sub-method was based on the output confusion matrix, focusing on the True Positive Rate (TPR) of the whole model. The TPR is calculated as the average of the leading diagonal of the confusion matrix. The computational complexity was also considered and compared for such implementations. In addition to the TPR, some comparisons of accuracy were made with the classification loss. The loss was calculated using classification error, described by formula 2, as written in the MATLAB documentation [31], where:

- w_j is the weight for observation j.
- e_j = 1 if the predicted class of observation j differs from its true class, and 0 otherwise.

$$L = \frac{\sum_{j=1}^{n} w_j e_j}{\sum_{j=1}^{n} w_j} \qquad (2)$$

The random element to the algorithm meant that there was variation in the results; because of this, repeated testing was performed and the mean values of these metrics were considered rather than the individual.

Given the volume of data that the implementation is dealing with, and the need for repeated testing, computational complexity was an issue to consider, in terms of both space and time. A change in some parameters would cause the MATLAB session to run out of memory; for example, a large number of sample frames, or a small HOG cell size. This out of memory error would occur at the PCA stage of the method. Although the PCA stage is not essential to the method's completion or accuracy, removing it would result in problems with time complexity. Prior to the PCA implementation, training the classifier would take more than four hours - this was undesirable and difficult to work with, especially with the time

constraints of the university project. The final run, including feature extraction, completed execution in approximately 43 minutes, which is a considerable reduction.

4.1.2 Parameter Optimisation Tests

This section gives a summary of the tests done in order to select optimum parameter values with regards to the overall performance of the model. The TPR here refers to the overall TPR of all six classes. The runtime here refers to the runtime for the entirety of the program; sampling, feature extraction, training, testing. The KTH dataset was used for all test data in this section.

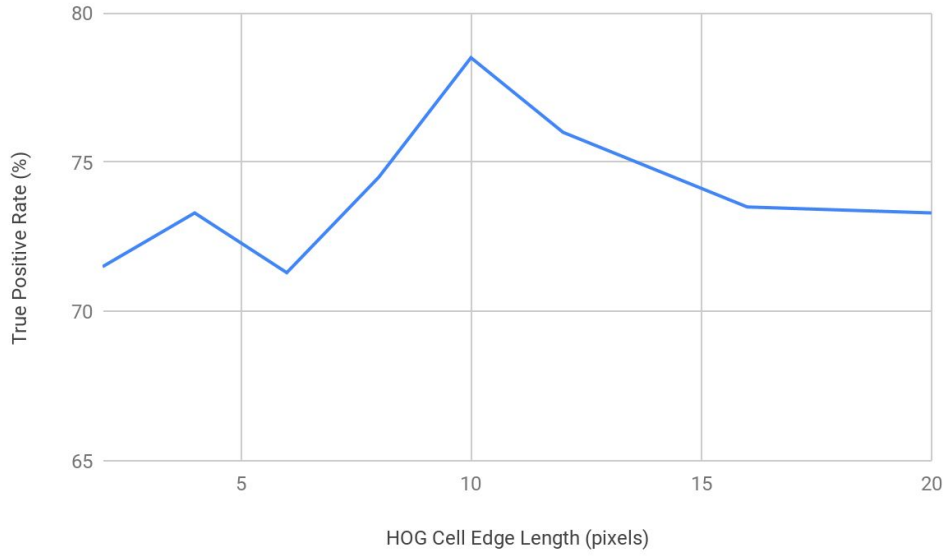

Figure 3.1: Plot of results for the TPR against HOG cell edge length. Tested with 24 samples per video and where 60% of the data was used as training data. Where a HOG cell edge length of x corresponds to an x by x HOG cell.

Multiple HOG cell sizes were tested and compared, as shown in Figure 3.1. The TPR peaked at a cell size of 10 by 10. Smaller cells did not generalise enough, larger cells were not detailed enough - this is demonstrated in Figure 3.2. These tests were carried out with 24 samples per video to allow smaller HOG cell sizes to be tested without an out of memory error occuring.

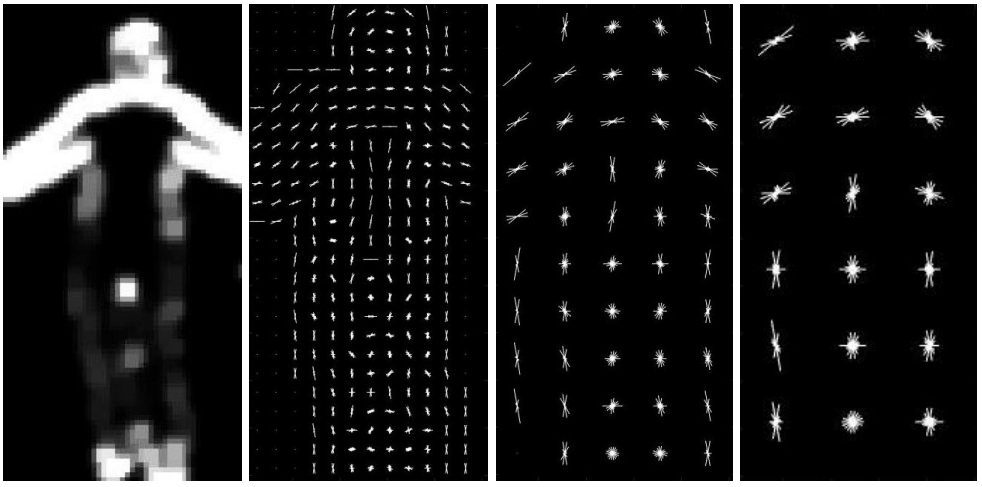

Figure 3.2: Comparison of post-processed optical flow estimation image of hand waving (far left) with corresponding HOG feature visualisation plots for HOG cell sizes of (left to right) 4 by 4, 10 by 10, and 16 by 16.

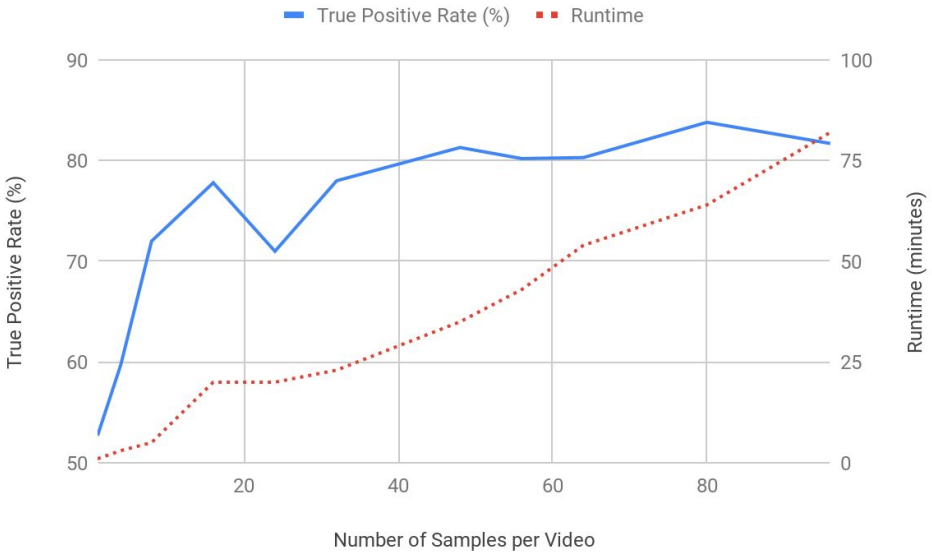

Figure 3.3: Plot of results for samples per video against the TPR. 60% of the data was used as training data - the optimum (90%) was not trained with as to reduce runtime.

The number of samples per video was tested with 10 by 10 HOG cells and with 60% of the data used as training data - the optimum (90%) was not trained with for all tests as to reduce runtime. The optimum number of samples per video tested for the TPR was 80, producing a TPR of 83.8%, however this took 64 minutes to run. 48

samples per video produced a marginally lower TPR of 81.3% in approximately 45% less time (35 minutes). These two tests were repeated with 90% of the data used as training data - this resulted in both the 48 and 80 samples tests producing the same TPRs but the same runtime disparity as with 60% training data - therefore 48 samples per video was the clear choice. Results shown in Figure 3.3.

In earlier versions of the method, a proportion of boxing test data was classed into the running or jogging classes. This is clear to see why when watching the boxing videos from the dataset. The majority of the boxers are stood still, only moving their arms outward and inward; however, in a small proportion of videos, they are moving their feet at a considerable pace on the spot - tricking the classifier into believing this a jogging or running movement. Increasing the size of the HOG cell and the number of sample frames amended this.

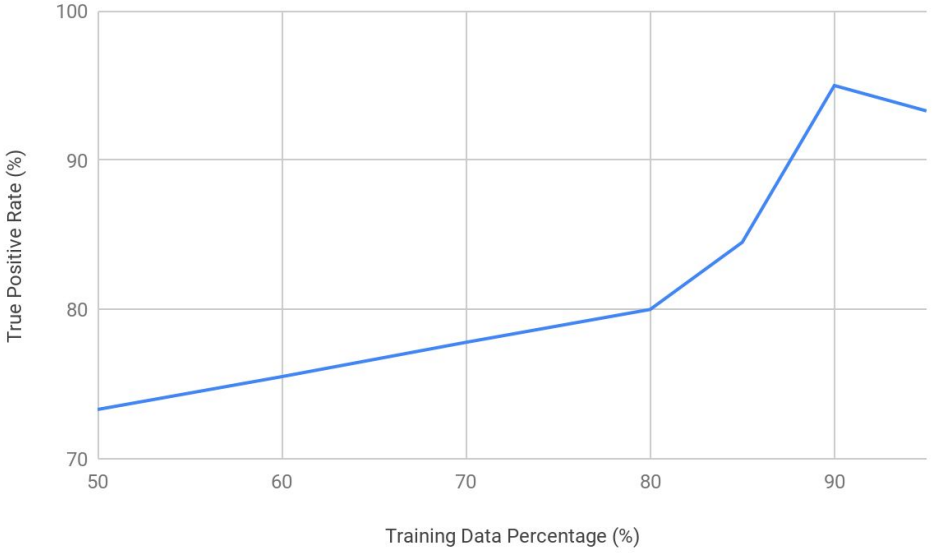

Figure 3.4: Plot of results for percentage of data used as training data against the TPR. Tested with HOG cells of 10 by 10 and 48 samples per video.

Different proportions of data used as training data was tested against the overall TPR. These tests were carried out with 10 by 10 HOG cells and 48 samples per video. The optimum percentage used as training data was 90%. This gives 540 videos to train on, 60 to test on. Generally, the more data to train on the better the accuracy of the model; however, exceeding 90% would not allow for a

suitable number of test videos to draw conclusions from - this is evident in the results where after 90% the TPR starts to decrease. Results shown in Figure 3.4.

Number of additional consecutive frames used in one sample	True Positive Rate (%)	Runtime (minutes)
0	77.7	32
1	84.5	46
2	84.3	52

Table 3.1: Results for number of consecutive frames used in one sample against the TPR and the runtime. Tested with HOG cells of 10 by 10, 85% training data and 48 samples per video.

The number of consecutive frames used per sample was tested against the TPR. These tests used 10 by 10 HOG cells, 48 samples per video and 85% training data. Using only 1 frame per sample (no consecutive frames), produced significantly lower TPRs than runs with consecutive frames. The optimum number of consecutive frames to use was 1, summing to 2 frames per sample. TPRs produced from runs with greater than 1 consecutive frame were marginally lower and required longer runtimes. Results shown in Table 3.1.

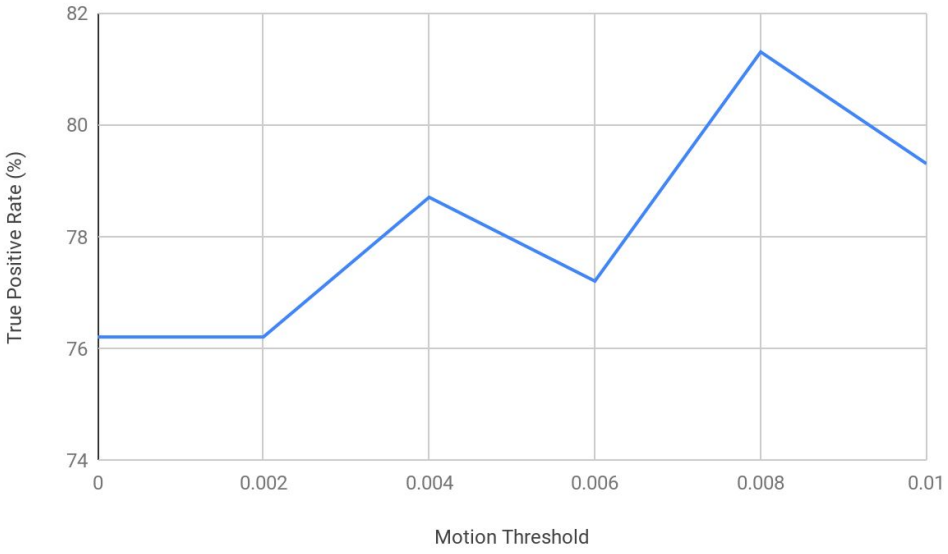

Figure 3.5: Plot of results for motion threshold (minimum) value for sample acceptance against overall true positive rate of the model. Tested with HOG cells of 10 by 10, 60% training data and 48 samples per video.

The minimum MMV (as discussed in section 3.3.2) was tested at different values against the TPR. The optimum MMV for the TPR was 0.008; lower values would allow frames with less significant amounts of motion to be processed and taken through the algorithm, larger values would allow frames with significant and classifiable amounts of motion to be discarded. This testing was done in order to verify the manual selection of the MMV values as described in section 3.3.2. Results shown in Figure 3.5.

The splitting of the optical flow into x and y directions was implemented based on its positive effect toward the TPR. Initially, the method split the optical flow into positive x-direction, negative x-direction, positive y-direction and negative y-direction to form four non-negative optical flow channels, as the method described by Fathi and Mori [9]. In following iterations of the method, the positive and negative channels were combined into one, leaving only the two channels of x and y direction. These channels are first calculated with both negative and positive values but are subsequently converted to all-positive values by taking the modulus of each element - this was implemented as it produced the highest TPR in comparison to the prior methods. A channel for the magnitude of the optical flow was also calculated, giving values irrelevant to the flow

direction; this also proved to worsen the TPR, when used alone and in conjunction with the other two channels.

The morphing stage of the optical flow post-processing was tested against the TPR. The close function marginally increased the TPR, producing very similar values for different sizes of structuring element. The optical flow image was binarised in the initial versions of the method, where morphing produced a more significant improvement to the TPR. Binarisation was later removed as it produced a worse TPR, including when morphed - it did however speed up the optical flow post-processing stage considerably which made early testing more convenient.

Learner	True Positive Rate (%)	Classification Loss	Runtime (minutes)
discriminant (analysis)	93.3	0.291	51
knn (k-nearest neighbour)	91.7	0.290	37
linear	93.3	0.294	34
naivebayes	65.0	0.504	34
svm	95.0	0.283	43
tree	83.3	0.447	35

Table 3.2: Results for each categorical value of the 'Learners' parameter for classifier training, against the TPR and classification loss.

The function for training multi-class classifiers, has a parameter 'Learners' which can be used to train a classifier with specific preset models of binary learners. Each binary learner preset was tested against the TPR and classification loss, to find which produced the best results. The 'svm' model gave the best TPR and best classification loss, with an acceptable runtime (consulting the method aims). Results shown in Table 3.2.

Coding	True Positive Rate (%)	Classification Loss	Runtime (minutes)
onevsone	95.0	0.283	43
onevsall	95.0	0.309	68
ordinal	83.3	0.331	56

Table 3.3: Results for each categorical value of the 'Coding' parameter for classifier training, against the TPR and classification loss.

The function for training multi-class classifiers, also has a parameter 'Coding' which can be used to train a classifier with specific preset models of coding design in relation to the binary learners. Each coding design preset was tested against the TPR and classification loss, to find which produced the best results. The 'onevsone' model gave the best TPR and best classification loss, and had the shortest runtime. Results shown in Table 3.3. Some of the coding designs took over 2 hours to compute, and were therefore not considered further - these were 'binarycomplete', 'denserandom', 'sparserandom' and 'ternarycomplete'.

4.2 Synopsis

This section gives the results of the final method which has been optimised by the parameter selections and techniques discussed in section 4.1.2.

The optical flow processing produced good visual results, clearly cropping to the region of interest and marking areas of the image with a velocity (represented as intensity) that is expected of that motion class - as shown in Figure 4.1. The same can be said for the visualisation of the HOG features, as shown in Figure 4.2; the orientation gradients accurately correspond to the optical flow image.

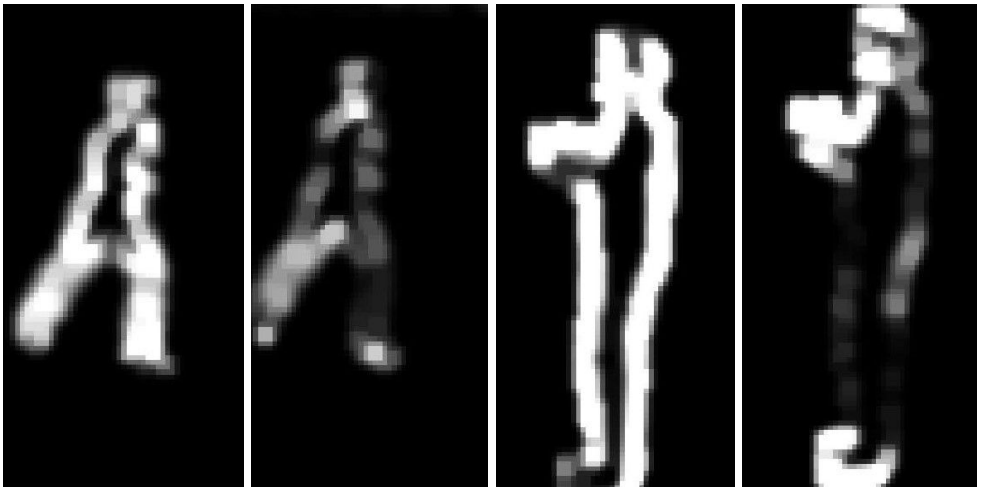

Figure 4.1: Examples of post-processed optical flow estimation images. From left to right: an x-direction optical flow image for running, a y-direction optical flow image for running, an x-direction optical flow image for boxing, a y-direction optical flow image for boxing.

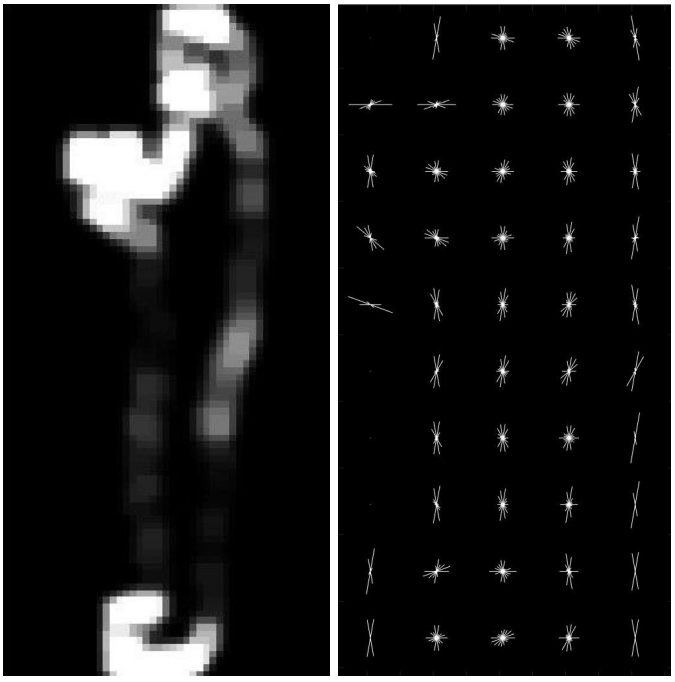

Figure 4.2: A comparison of a y-direction optical flow image for boxing and its HOG feature visualisation plot.

	Boxing	Hand clapping	Hand waving	Jogging	Running	Walking
Boxing	100.0	0	0	0	0	0
Hand clapping	0	100.0	0	0	0	0
Hand waving	0	0	100.0	0	0	0
Jogging	0	0	0	80.0	20.0	0
Running	0	0	0	10.0	90.0	0
Walking	0	0	0	0	0	100.0

Table 4.1: The confusion matrix for the final method, trained and tested on the KTH dataset.

The final method produced a TPR of 95%, when training and testing on the KTH dataset, correctly classifying 100% of the test data for boxing, hand clapping, hand waving and jogging (see Table 4.1).

	Boxing	Hand clapping	Hand waving	Jogging	Running	Walking
Running	0	0	0	70.0	30.0	0
Walking	0	0	0	10.0	0	90.0

Table 4.2: The confusion matrix for the final method, trained on the KTH dataset, tested on the Weizmann dataset.

The final method produced a TPR of 60%, when training on the KTH dataset but testing on the Weizmann dataset (see Table 4.2). The datasets differ on their definition of classes; there is no jogging class in the Weizmann dataset, and so the running class contains many actions that would be considered as jogging in the KTH dataset - as proven by the results, where 70% of Weizmann 'running' classes are classified as KTH 'jogging classes'. Considering this, the results produced by these tests are more than satisfactory. When taking jogging and running as a combined class, this overall TPR value also equates to 95% which is equal to the KTH test TPR from Table 4.1.

5 Conclusions and Further Work

The final method correctly classified videos 95% of the time. Boxing, hand clapping, hand waving and walking were classed correctly 100% of the time. Overall 57 of 60 test cases for the KTH dataset were classified correctly. The classification of running and jogging brought the overall accuracy down; running was misclassified as jogging in 1 case, jogging was misclassified as running in 2 cases. Jogging and running are the hardest classes to distinguish in the dataset, given that they are so similar in nature - even to the human eye; however, today's movement classification algorithms should be able to distinguish this, demonstrating that a HOG-based one is perhaps not the best method for such a task, or that it should be implemented in a different way. Nevertheless, the TPR exceeded the original goal of this research (90%), proving that a HOG method does have credibility for human movement classification from videos.

The KTH dataset provided a solid starting point for developing a classification algorithm for human movement from videos. Using a variety of datasets could affirm the serviceability of HOG features in video classification and give insight as to where this current method can improve. Other datasets can also provide other classes to work with and increase the scope of the algorithm, such as the UCF Sports Action dataset [32] - this has classes for diving, kicking, lifting and horse riding, for example.

This algorithm could also benefit from more data. Multiple tests were done for different ratios of training and test data. Each increase of training data resulted in better accuracy of the model - this peaked at 90% training data (exceeding this would not give enough test data) (see Figure 3.4) giving the algorithm 540 videos to train on and 60 videos to test on. The steady and significant improvement suggests that if the algorithm could train on more than 540 videos and still have a large enough pool to test on, then the true positive rate may yet increase.

The final method would take 43 minutes to run; this includes sampling from 600 videos, calculating and processing of optical flow, feature extraction, training the classifier, and testing. Although this is a significant improvement from its earlier iterations (which would take several hours), there is still room to improve further. In the future, work can be done to speed up this process, which would

make it more usable. Currently, the slowest part of the algorithm is the frame sampling and feature extraction. Due to PCA, training the classifier is a relatively quick computation. The final method used 48 sample frames per video, for 600 videos - amounting to 28,800 optical flow calculations, postprocessing and HOG feature extractions in series. Further work could identify ways to quicken these calculations, make them more efficient, or reduce the amount of sample frames needed to obtain the current accuracy of the model.

Motion features are used in this method - HOG features are extracted from *optical flow* estimates between frames. In the method proposed by Liu et al. [11], static features are also extracted in addition to motion features. Their method uses different interest point detectors, which together extract corner features and blob features from individual raw frames [11]. Static features, such as these, could be incorporated into the algorithm; any static features that are extracted could be appended to the motion-based feature vector and classified as before.

References

[1] J. Wang, et al., “A Review of Vision-based Gait Recognition Methods for Human Identification,” in *DICTA 2010. Proceedings of the Digital Image Computing: Techniques and* Application, Piscataway, NJ, USA, Dec. 2010, pp. 320-327.

[2] B. Pogorelc, et al., “Automatic recognition of gai-related health problems in the elderly using machine learning,” *Multimedia Tools and Applications,* vol. 58, pp. 333-354, May 2012. [Online]. Available: https://link.springer.com/article/10.1007/s11042-011-0786-1 [Accessed: Jan. 22, 2019].

[3] DeepMind. *Kinetics*, deepmind.com. [Online]. Available: https://deepmind.com/research/open-source/open-source-datasets/kinetics/ [Accessed: Jan. 21, 2019].

[4] Kungliga Tekniska Högskolan. *Recognition of human actions*, www.nada.kth.se. [Online]. Available: http://www.nada.kth.se/cvap/actions/ [Accessed: Jan. 21, 2019].

[5] L. Gorelick, et al., “Actions as Space-Time Shapes,” *Transactions on Pattern Analysis and Machine Intelligence,* vol. 29, no. 12, pp. 2247-2253, December 2007. [Online]. Available: http://www.wisdom.weizmann.ac.il/~vision/SpaceTimeActions.html [Accessed: Jan. 21, 2019].

[6] R. Ebrahimzadeh and M. Jampour, “Efficient Handwritten Digit Recognition based on Histogram of Oriented Gradients and SVM,” *International Journal of Computer Applications,* vol. 104, no. 9, October 2014. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.800.5844&rep=rep1&type=pdf [Accessed: Feb. 9, 2019].

[7] N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” in *CVPR 2005. IEEE Conference on Computer Vision and Pattern Recognition, San Diego, CA, USA, Jun. 2005, pp. 886-893.*

[8] C. McCormick (2013, May 9). *HOG Person Detector Tutorial.* Chris McCormick [Online]. Available: http://mccormickml.com/2013/05/09/hog-person-detector-tutorial/ [Accessed: Feb. 9, 2019].

[9] A. Fathi and G. Mori, “Action recognition by learning mid-level motion features,” in *CVPR 2008. IEEE Conference on Computer Vision and Pattern Recognition,* Anchorage, AK, USA, Jun. 2008.

[10] M. S. Ryoo and J. K. Aggarwal, “Spatio-Temporal Relationship Match: Video Structure Comparison for Recognition of Complex Human Activities,” in *ICCV. 2009 IEEE 12th INternational Conference on Computer Vision,* Kyoto, Japan, Sep.-Oct. 2009.

[11] J. Liu, et al., “Recognizing realistic actions from videos ‘in the wild’”, in *CVPR 2009. IEEE Conference on Computer Vision and Pattern Recognition,* Miami, FL, USA, Jun. 2009.

[12] P. Dollár, et al., “Behavior Recognition via Sparse Spatio-Temporal Features,” in *VS-PETS. 2005 IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance,* Beijing, China, Oct. 2005.

[13] S. Brin and L. Page, “The Anatomy of a Large-Scale Hypertextual Web Search Engine,” *Computer networks and ISDN Systems,* vol. 30, pp. 107-117, April 1998. [Online]. Available: http://ilpubs.stanford.edu:8090/361/1/1998-8.pdf [Accessed: Feb. 4, 2019].

[14] S. Lawrence, et al., “Face Recognition: A Convolutional Neural-Network Approach,” *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98-113, January 1997. [Online]. Available: http://www.cs.cmu.edu/afs/cs/user/bhiksha/WWW/courses/deeplearning/Fall.2016/pdfs/Lawrence_et_al.pdf [Accessed: Jan. 23, 2019].

[15] U. Karn. “An Intuitive Explanation of Convolutional Neural Networks”. 11 Aug 2016. [Blog entry]. *The data science blog.* Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ [Accessed: 23 Jan 2019].

[16] D. Maturana and S. Scherer, “VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition,” in *IROS. 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems,* Hamburg, Germany, Sep. 2015, pp. 922-928.

[17] A. Karpathy, et al., “Large-scale Video Classification with Convolutional Neural Networks,” in *CVPR 2014. IEEE Conference on Computer Vision and Pattern Recognition,* Columbus, OH, USA, Jun. 2014, pp. 1725-1732.

[18] Y. Bengio, et al., “Learning Long-Term Dependencies with Gradient Descent is Difficult,” *IEEE Transactions on Neural Networks,* vol. 5, no. 2, pp. 157-166, March 1994. [Online]. Available: http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf [Accessed: Jan. 27, 2019].

[19] C. Olah (2015, Aug. 27). *Understanding LSTM Networks.* Github [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ [Accessed: Jan. 27, 2019].

[20] J. Y. H. Ng, et al., “Beyond Short Snippets: Deep Networks for Video Classification,” in *CVPR 2015. IEEE Conference on Computer Vision and Pattern Recognition,* Boston, MA, USA, Jun. 2015.

[21] J. Chung, et al., “Empircal evaluation of gated recurrent neural networks on sequence modeling,” in *NIPS 2014. Neural Information Processing Systems: Workshop on Deep Learning*, Montréal, Canada, vol. 1412.3555, Dec. 2014.

[22] K. Cook (2018, Nov. 5). *Top 4 Best Programming Languages For Image Recognition.* House of Bots [Online]. Available: https://houseofbots.com/news-detail/3948-4-top-4-best-programming-languages-for-image-recognition [Accessed: Jan. 28, 2019].

[23] K. Gupta (2017, Aug. 19). *The Best Programming Languages For Face Recognition.* FreelancingGig [Online]. Available: https://www.freelancinggig.com/blog/2017/08/19/best-programming-languages-face-recognition/ [Accessed: Jan. 28, 2019].

[24] S. Negahdaripour and C.-H. Yu, “A generalized brightness change model for computing optical flow,” in *ICCV 1993*. *IEEE 4th International Conference on Computer Vision*, Berlin, Germany, May 1993, pp. 2-11.

[25] D. J. Fleet and Y. Weiss, “Optical Flow Estimation,” *Handbook of Mathematical Models in Computer Vision,* pp. 237-257, 2006. [Online]. Available: http://www.cs.toronto.edu/~fleet/research/Papers/flowChapter05.pdf [Accessed: Feb. 11, 2019].

[26] R. Rojas, “Lucas-Kanade in a Nutshell.” Freie Universit at Berlinn, Dept. of Computer Science, Tech. Rep., 2010. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/tutorials/Lucas-Kanade2.pdf [Accessed: Feb. 11, 2019].

[27] MathWorks. *extractHOGFeatures,* uk.mathworks.com. [Online]. Available: https://www.mathworks.com/help/vision/ref/extracthogfeatures.html [Accessed: Feb. 16, 2019].

[28] L. van der Maaten, et al., “Dimensionality Reduction: A Comparative Review,” *Journal of Machine Learning Research,* vol. 10, pp. 66-71, January 2007. [Online]. Available: https://lvdmaaten.github.io/publications/papers/TR_Dimensionality_Reduction_Review_2009.pdf [Accessed: Feb. 16, 2019].

[29] MathWorks. *pca,* uk.mathworks.com [Online]. Available: https://www.mathworks.com/help/stats/pca.html [Accessed: Mar. 21, 2019].

[30] MathWorks. *fitcecoc*, uk.mathworks.com [Online]. Available: https://www.mathworks.com/help/stats/fitcecoc.html [Accessed: Apr. 1, 2019].

[31] MathWorks. *loss,* uk.mathworks.com [Online]. Available: https://uk.mathworks.com/help/stats/classificationecoc.loss.html [Accessed: Apr. 1, 2019].

[32] University of Central Florida. *UCF Sports Action Data Set,* www.crcv.ucf.edu [Online]. Available: https://www.crcv.ucf.edu/data/UCF_Sports_Action.php [Accessed: Apr. 15, 2019].